Verification and Performance Evaluation of AADL Models*

Marco Bozzano, Alessandro Cimatti, Marco Roveri,
Embedded Systems Group
Fondazione Bruno Kessler, Trento, Italy
{bozzano,cimatti,roveri}@fbk.eu

Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University, Aachen, Germany
{katoen,nguyen,noll}@cs.rwth-aachen.de

ABSTRACT

This paper reports on a model-based approach to system-software co-engineering which is tailored to critical on-board systems for the aerospace domain but is relevant to a much wider class of dependable systems. Our main contribution is a formal semantics for a greater part of standardised AADL, the Architecture Analysis and Design Language, and its Error Model Annex. It covers nominal and degraded hardware/software operations, hybrid (and timing) aspects as well as probabilistic faults, their propagation and recovery. The accompanying software toolset employs SAT-based and symbolic model checking techniques and probabilistic variants thereof. The precise nature of these techniques together with the formal semantics provide a trustworthy modelling and analysis framework to support, among others, assessment of functional correctness, evaluation of performance measures and automated derivation of dynamic fault trees, FMEA tables and observability requirements.

Categories and Subject Descriptors

B.6.3 [**Design Aids**]: Verification

General Terms

Design, Performance, Verification, Reliability

1. INTRODUCTION

The increasing complexity of safety-critical systems has driven industry to develop system-level languages like AADL and SysML, where the focus lies on capturing the overall system design, especially the interaction between hardware and software. An important paradigm here is the component-based design. It helps mastering the overall complexity while in addition allowing for reusability.

Safety-critical systems are also subject to hardware and software faults. The adequate modelling of faults, their likelihood of occurrence and the way a system can recover from faults are essential to a comprehensive system-level design of safety-critical systems. Although several formal verification approaches to component-based design have been recently reported in the literature (see references in [2]), error handling and modeling has received scant attention, if at all. Another shortcoming of many proposals is the lack of connection to a notation that is used and understood by design engineers. We propose the SLIM language in which we attempt to overcome the problems of the previous proposals by enriching a greater part of AADL and its Error Model Annex (§2) with a formal semantics (§3) based on networks of event-data automata.

The first version of the accompanying toolset has been developed last year and is currently being extended. It addresses the issues of correctness, safety, and performability through model checking. This toolset is being developed in the context of the European Space Agency project COMPASS (Correctness, Modelling, and Performance of Aerospace Systems) [1] and will be applied to several case studies by a major industrial developer of aerospace systems. A complete list of supported analyses and a sketch of the tool's architecture is described in §4.

In §5 we conclude this short paper on references to other papers where more information about the SLIM modelling language and the underlying techniques used by the toolset is provided.

*Funded by ESA/ESTEC under Contract 21171/07/NL/JD

ESEC-FSE'09, August 24–28, 2009, Amsterdam, The Netherlands.
ACM 978-1-60558-001-2/09/08.

2. SYSTEM-LEVEL MODELLING

Akin to AADL, the SLIM language distinguishes between *nominal models* and *error models*.

The *nominal model* describes the system under normal operation. It is expressed as a set of nominal components with their corresponding interactions. A nominal component represents software or hardware. Sets of interacting components can be grouped into a composite component. This facility enables the modeler to manage the system's complexity through introducing component *hierarchy*. A nominal component is further separated into a *type* and its *implementation*. The type describes the interface, i.e., the event and data ports, for interacting with other components. Events are transmitted in a rendez-vous manner. Data ports are typed (**`int`**, **`real`**, **`bool`**) and transmit data between components. The *implementation* describes how these interactions are performed by a finite state automaton.

The *error model* expresses how the system can fail, i.e., it models how faults may affect normal operation and may lead to a degraded mode of operation. It is modelled as a *probabilistic finite state automaton*. Transitions occur due to spontaneous **`events`** which may be annotated with a rate that indicates the expected number of occurrences per time unit. This information is leveraged for performing probabilistic analyses. Transitions can also occur because of *error propagations*. Inwards error propagations cause a transition,

outwards error propagation are generated by following the transition. These transitions are based on whether components access a common **bus**, or whether they are in a super-subcomponent relation. This implicit linking reflects the oblivious and pervasive nature of error models.

As error models bear no relation with nominal models, an error model does not influence the nominal model unless they are linked through *fault injection*. A fault injection consists of three parts: a state s in the error model, a variable d in the nominal model, and the fault effect given as the expression e. Semantically, the error and nominal model are concurrently active. On entering error state s, the assignment $d := e$ is carried out, i.e., the variable is assigned with the fault effect. Multiple fault injections between error models and nominal models are possible. The combined model, i.e., the integration of the nominal models, error models, and the fault injections, is called the *extended model*.

3. FORMAL SEMANTICS

The formal semantics provides the interpretation of SLIM specifications in a precise and unambiguous manner. The meaning of a nominal specification is defined on two levels, distinguishing between the local behavior of an active component and the interaction between active components via ports, which is referred to as the global behaviour.

The *local behaviour* emerges from the product composition of a nominal model and its corresponding error model. It is formalised by an *event-data automaton* (EDA), which is a tuple containing modes, initial mode, variables, default valuation of variables, mode invariants, events and transitions. The semantics of a EDA is given by a labelled transition system (LTS) where the states are *configurations*, i.e., pairs of modes and valuations of variables. Transitions model the passage of time, probabilistic events, updates to variables or triggers of events.

The *global behaviour* is captured by a *network of event-data automata* (NEDA). A NEDA tuple contains the EDAs, an activation mapping describing when EDAs are active, and the event/data connections between EDAs. Interaction in a NEDA is highly dynamic as local transitions can cause EDAs to become (in-)active and can change the topology of event and data port connections.

The *semantics of a NEDA* is described by a LTS as well, where the states are the product of configurations of active EDAs. Transitions occur due to the passage of time (and thereby affecting all the active EDAs), probabilistic transitions due to an active error model, internal transitions within active EDAs or communication between EDAs. This LTS describes overall system behaviour and captures probabilistic, discrete and continuous system evolution.

4. TOOLSET FOR FORMAL ANALYSES

We leverage NuSMV and MRMC to enable formal analyses of SLIM models. NuSMV employs symbolic model checking techniques and is known for coping with large transition systems. MRMC is a probabilistic model checker for Markov chains and computes performability characteristics. The design engineer does not interact with these tools directly as it requires extensive expertise in formal logics and low-level modeling formalisms. Instead, fully automatic translators from SLIM to (and between) the lower-level input formats have been developed to abstract away these operational details. The first version of the toolset works as of April 2009 and it is currently being extended for the final version. Therefore, we shall only sketch the possible analyses supported by the final toolset.

The *simulator* generates a trace describing a single interaction scheme between components. *Model checking* techniques can be employed to exhaustively verify a model against a property, i.e, the formalised requirement. The *probabilistic* counterparts of these compute probabilities. The properties are entered in a user-friendly manner via property patterns; a collection of often-used templates with accompanying high-level descriptions. Translators shall transform these templates to the underlying formal logics, like Linear Temporal Logic and Continuous Stochastic Logic.

To facilitate the analysis of safety and dependability aspects, *fault trees* can be generated via FSAP, an extension of NuSMV. Fault trees describe which faults could have caused the system to enter a particular mode, which is called the *top-level event*. If enough probabilistic information is given in the SLIM model, it is also possible to compute the probability of the top-level event using MRMC by transforming the fault tree to its underlying Markov chain. *Failure Mode and Effects Analysis* (FMEA) tables can also be automatically generated using FSAP as well. They describe which modes may be entered when a fault occurs.

For complex systems that implement *Fault Detection, Identification, and Recovery* (FDIR) strategies, it is also important to verify their effectiveness. We plan to leverage the above mentioned analyses for that. These analyses require the modelling of partial observability. The keyword **observable** is reserved for this in SLIM and can be attached to Boolean output data ports. A value change of an observable port may be considered as a fault detection means for a previously occurred fault. The **observable** attributes can also be used for diagnosability analysis, which concerns whether a component, regardless of its FDIR system, has proper, or overly, outgoing Boolean data ports from which a faulty state can be deduced.

5. CONCLUSIONS

This paper briefly sketched the SLIM modelling language, whose semantics can be considered as a (first) formal interpretation of AADL and its Error Model Annex. We currently are bringing our results to the AADL standardisation body. The first version of the formal verification toolset that supports SLIM specifications is working and it is currently being extended. For more information, we refer to [2], where an in-depth description of SLIM and its semantics are covered and [3], which comprehensively covers the COMPASS project.

6. REFERENCES

[1] Website of the COMPASS Project `http://compass.informatik.rwth-aachen.de/`

[2] M. Bozzano, A. Cimatti, M. Roveri, J.-P. Katoen, V.Y. Nguyen, T. Noll, *Codesign of Dependable Systems: A Component-Based Modeling Language* in 7th MEMOCODE. IEEE Computer Society, 2009.

[3] M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, M. Roveri, *The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems* in 28th SAFECOMP. Springer, 2009.