HAL
open science

Unifying Probabilistic and Traditional Formal Model Based Analysis

Matthias Güdemann, Michael Lipaczewski, Simon Struck, Frank Ortmeier

To cite this version:

Matthias Güdemann, Michael Lipaczewski, Simon Struck, Frank Ortmeier. Unifying Probabilistic and Traditional Formal Model Based Analysis. 8. Dagstuhl-Workshop MBEES 2012 - Model-Based Development of Embedded Systems, Feb 2012, Dagstuhl, Germany. hal-00665607

HAL Id: hal-00665607

https://inria.hal.science/hal-00665607v1

Submitted on 23 Nov 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Unifying Probabilistic and Traditional Formal Model Based Analysis

Matthias Güdemann, Michael Lipaczewski, Simon Struck and Frank Ortmeier

matthias.gudemann@inria.fr, {michael.lipaczewski|simon.struck| frank.ortmeier}@ovgu.de

INRIA Rhône-Alpes, FIN / ITI, CSE, Otto-von-Guericke University Magdeburg

Abstract: The increasing complexity of modern software-intensive systems makes their analysis much more difficult. At the same time, more and more of theses systems are used in safety-critical environment.

Model based safety analysis can help with this problem by giving provably correct and complete results, very often in a fully automatic way. Today, such methods can cope with logical as well as probabilistic questions. However, very often the models used in many model based approaches must be specified in different input languages depending on the chosen verification tool for the desired aspect, which is time consuming and often error-prone.

In this paper, we report on our experiences in designing a tool independent specification language (SAML) for model based safety analysis. This allows to use only one model and analyze it with different methods and different verification engines, while guaranteeing the equivalence of the analyzed models. In particular, we discuss challenges and possible solutions to integrate SAML in the development process of real systems.

1 Introduction

Software is becoming a major innovation factor in many different application domains. With increasing demands and more features, the complexity of systems is growing steadily.

The increased complexity of software-intensive systems and their increased application in safety critical domains makes the detection of possible malfunctioning more and more important, but unfortunately also more difficult. To tackle this problems, model-based safety analysis techniques have been developed. Their usage is today encouraged or even demanded by many domain specific norms like DO-178B for avionics, ISO26262 for automotive or the generic IEC61508.

In the last decade, different safety analysis methods have been developed. Some approaches focus on qualitative safety analysis i.e. the determination of critical combinations of component failures that can lead to a system hazard. However, all technical systems may fail some time. The remaining question is the occurrence probability of a hazard under known failure occurrence probabilities of the components. Thus there are different approaches to compute the quantitative safety aspects.

Safety analysis of industrial applications requires qualitative and quantitative methods. It is our experience that applying both methods at the same time requires a lot of unnecessary effort. One reason is the lack of tool support for convenient specification. Verification tools are improved continuously but actually the front-ends only improve slowly. And last but not least, both methods often require different tools and thus rely on different modeling languages.

To solve this problem, we defined the Safety Analysis Modeling Language (SAML). It allows for tool independent specification of models for safety analysis. For verification, these models are automatically transformed into model representations usable for different verification engines. The transformations are semantically well-founded and mathematically proven. This allows the decoupling of the model (and its safety properties) from the actual verification tools. In this paper, we will focus on our experiences an challenges for simplifying the process of qualitative and quantitative safety analysis.

In Sect. 2 we will briefly describe the idea of SAML and by using an example from a recent case-study describe the language and the different aspects. Different approaches for current and further work are discussed in Sect. 3. Some related approaches are discussed in Sect. 4, followed by an conclusion in Sect. 5.

2 Safety Analysis Modeling Language (SAML)

2.1 Tool-independent Specification

The basic concept of a model-based approach for safety analysis is that system designers and the safety engineers share a common system model [JMWH05]. This has the advantage, that safety analysis is conducted using a system model, which will also be used for a sketch/blueprint for implementation. Therefore all implicit (stochastic) dependencies are automatically considered and consistency of the analysis with the engineering model is guaranteed.

A key requirement for accurate model-based safety analysis is, that models must be capable of deterministic, non-deterministic and stochastic behavior. Deterministic behavior is typically (but not exclusively) necessary for modeling software functionalities. Non-determinism is useful for modeling system environment or software faults (for both very often no stochastic information exist). Physical failure modes are most often modeled using probabilistic models (for example known failure rates of components). There exist numerous tool-specific specification languages, where each tool focuses on a specific class of problems and provides a fitting specification language. Just to name a few examples: NuSMV [1] is one of the most powerful model checkers for Computational Tree Logic. Its input specification are in the form of finite automata. MRMC [2] is a very powerful stochastic model checker which can deal with discrete and continuous time Markov chains, but

[1] http://nusmv.fbk.eu
[2] http://www.mrmc-tool.org

requires the specification directly as the transition relation. On the other hand, PRISM [3] is a little less time-efficient than MRMC but allows for a much more convenient specification of models and also supports discrete time Markov decision processes.

As safety analysis often contains stochastic as well as logical analysis, we suggest using a tool-independent specification language and automatic model transformations to tool-specific languages of the verification engines. We defined the tool-independent specification language safety analysis and modeling language (SAML) to solve this problem (see Fig. 1).

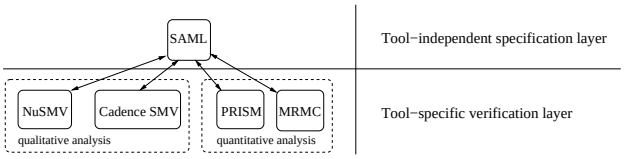

Figure 1: SAML overview

Tool-independent specification layer: Problems are of course tool-independent. All that is needed is a formalism, which is expressive enough to capture all joint model properties and simple enough to be understandable and usable. In this layer, functional behavior of the software and the controlled hardware are specified. It is also necessary to specify environment and failure mode models. However, modeling must not depend on the verification engine which will be used. We defined a tool-independent specification language – SAML – for usage on this layer. Precise syntax and semantics may be found in [GO10, Güd11], but a brief introduction and an application example may be found in the following section.

Verification Layer: SAML does not provide its own verification tool (nor shall it). Instead system models expressed in SAML are transformed into the input language of different formal analysis tools, depending on the type of the system property that should be checked. These transformations result in trace-equivalent models expressed in the input language of specified verification tools (proofs may be found in [Güd11]). The results of an analysis are either counterexamples of the analysis models mapped back to the SAML model or the probability of the checked properties. For this, the connection between the modeling artifacts of SAML and the analysis formalism is kept track of.

Such a separation offers two big benefits. Firstly, it is guaranteed that for example probabilistic analysis (maybe done with PRISM) and qualitative analysis (maybe done with NuSMV) are done using the same model. Therefore, the results are consistent with each other (in the sense that the same model was used). Secondly and maybe even more important, the separation allows to switch between different verification engines. For example, MRMC is much more time-efficient than PRISM for discrete time Markov chains and SPIN might be your first choice if you are interested in properties expressed in linear time temporal logic (instead of NuSMV). Such choice may now be done without changing/rebuilding the whole system model.

[3] http://www.prismmodelchecker.org

2.2 A SAML Example Model

Syntactically, a SAML model describes a set of finite state automata that are executed in a synchronous parallel fashion with discrete time-steps. The automata are described as modules that contain state variables which are updated according to transition rules. Transitions can contain both non-determinism and probabilistic choice. The syntax is very much inspired by the syntax of PRISM (as a matter of fact it is an extended subset). The underlying semantic model is a Markov Decision Process (MDP). In this paper we do not present full formal semantics and syntax but only explain the idea of using SAML on a small example. Full syntax and formal semantics may be found in [GO10].

Fig. 2 shows (parts of) the SAML code for the example of an airbag controller software. The case-study was initially presented in [KHE11]. The core idea of this system is, that a (software) controller is fed by two separate sensors, aggregates their measures over some time and then decides whether to ignite an airbag or not. Fig. 2 shows the model of an environmental component, of a specific failure mode and a part of the software called *sensor validator*, which aggregates sensor inputs (which is later used as input for the actual software controller).

```
01 constant double   p_magSensor_fail   := 2.78E-10;
02 constant int      Detection_interval := 5;
03 constant int      threshold          := 4;
04 formula           crash detected     := mechSensor_state > 0 & magSensor_state > 0;
[...]
05 module env_crashOccurrence
06 env_crashOccur :[0..1] init 0;
07 env_crashOccur = 0     ->  choice: ( 1: env_crashOccur'=1 )
                            + choice: ( 1: env_crashOccur'=0 );
08 env_crashOccur = 1     ->  choice: ( 1: env_crashOccur'=1 );
09 endmodule
[...]
10 module failure_magSensor
11 magSensor_faulty :[0..1] init 0;
12 magSensor_faulty = 0   ->  choice: (    p_magSensor_fail: magSensor_faulty'=1
                                      + 1-p magSensor_fail: magSensor_faulty'=0 );
13 magSensor faulty = 1   ->  choice: (  1                  : magSensor_faulty'=1 );
14 endmodule
[...]
15 module sw_SensorValidator
16 counter :[0 .. 6] init 0;
17 sum     :[0 .. 5] init 0;
18 //crash detected
19 crash detected & counter >=detection interval
                          ->  choice: ( 1: (counter'=0)         & (sum'=    1) );
20 crash detected & counter < detection interval & sum < sumLevel
                          ->  choice: ( 1: (counter'=counter+1) & (sum'=sum+1) );
21 crash detected & counter < detection interval & sum >=sumLevel
                          ->  choice: ( 1: (counter'=counter+1) & (sum'=sum  ) );
22 //crash not detected
[..]
23 endmodule
```

Deterministic decision | Non-deterministic choice | Probability distribution

Figure 2: Example SAML Model of Airbag Controller

Lines 1 to 3 contain definitions of **constants**, which may be used to define probabilities or other static parameters for global usage in the model. Valid types of constants in SAML are int, double and float. Line 4 shows the usage of a **formula**, which is often used as abbreviations for complex Boolean formulas.

Different components in SAML are described as modules (keywords: **module** and **endmodule**). Each module contains a set of state variables (see lines 6, 11, and 16-17) together with an initial value. The dynamic behavior is described by transition rules. A transition rule is a tuple of an activation condition [4] and a set of non-deterministic **choices** of *probability distributions* [5]. A probability distribution is a set of tuples, where the first part is a probability [6] and the second part is an assignment of a new value to the state variables of a model for the next time-step. For example the transition rule in line 12 is activated if "magSensor_faulty = 0" holds. This transition rule includes no non-deterministic choice (i.e. only one **choice** keyword). Therefore by defining a non-trivial distribution (line 12, bold border) with probability "p_magSensor_fail" the next state will be 1; otherwise the next state will be 0.

Different types of model components require different types of specification. For most software and hardware components behavior is typically deterministic as every possible situation leads to a single next situation. In SAML this leads to transition rules with only a single **choice** and a trivial probability distribution (i.e. exactly one assignment which is applied with a probability of 1). An example is the (software) module "sw_SensorValidator" starting in line 15. In contrast failure occurrences are often considered in a purely probabilistic way. An example is the failure of the magnetic sensor (see lines 10 to 14). In SAML this leads to transition rules with only a single choice but non-trivial distribution functions. Environmental processes are often specified using non-determinism. For example the occurrence of crashes (modeled in lines 5 to 9). In SAML, this leads to multiple choice in one transition rule (line 7, bold border). However, there also exist many situations where combinations are needed (for example software faults or more elaborate environment models). This is no problem because SAML allows the combination of probabilistic and non deterministic choices.

2.3 Current State of SAML

In the last 18 months we focused on a prototypical realization of SAML. In a first step, we implemented a parser for SAML using the ANTlr[Par07] parser generation framework. This proved to be a very efficient choice as it runs robust and allows relatively easy adaptations to the syntax. Next, we implemented a simulator for executing SAML models and first model transformations. We implemented model transformations to NuSMV, Cadence SMV and PRISM. A transformation to MRMC is also possible by using existing PRISM2MRMC model transformations. From an algorithmic point of view, we had to choose whether to translate on a semantic level (for example extraction of the Kripke structure out of the Markov Decision Process) or on a syntactic level (for example transla-

[4] For creating a well-founded model, it is necessary to have exactly one active transition for each possible state. This means all activation conditions must be pairwise contradicting and that their logical disjunction rewrites to true. This is necessary to guarantee existence of infinite trace (a pre-requisite for using CTL) as well as understandable probability spaces. If this restriction holds can only be decided on the semantic level. In the current implementation we rely on a special function of the PRISM model checker to evaluate this property.

[5] Non-determinism differs from a probability distribution by allowing every possible distribution

[6] which must together sum up to 1

tion of SAML modules to NuSMV automata). Both solutions are supposed to be difficult in realization and have both advantages and disadvantages. The first one is of course more charming and allows for more flexible transformations. However, it can become very costly as huge state spaces must be built, stored and handled. The second solution is much more performant (in terms of computation costs for the translation), but heavily depends on the input language of the chosen target tool. We chose the second option, in particular because it also yields human-readable intermediate models for the verification tools, which is a great help during debugging. More information on this choice and also the proofs for showing correctness maybe be found in [Güd11].

We now specified and analyzed about half a dozen case studies in SAML. We learned that the model transformations are working reliable and that specification in SAML is feasible and intuitive. In particular, qualitative and quantitative model-based safety analysis could efficiently be combined.

3 Next Steps and Outlook

During the exhaustive tests, we also learned that numerous open issues for improvements exist. This starts by extending the language with safety specific notions (i.e. failure modes). Another open issue is supporting specification with a state-of-the-art editor to increase performance. Finally, using SAML as an intermediate language between low-level verification tools and high-level CASE tools is a challenging topic. In this section, we will explain briefly our current work on these three topics.

3.1 Safety Specific Extensions

In the first version, SAML did not include any safety specific terms. This means that the grammar did not distinct between a software artifact, an environment specification and an failure mode. All are modeled as a simple *module*.

In IEC 61508 there exists the generic distinction between per-time and per-demand failure modes. A per demand failure mode may only appear if there is a demand to the safety critical system at the moment (for example physical breakdowns are usually only possible when an actuator is working). Such a failure mode occurrence is described with a failure probability. In contrast, a per-time failure mode may occur at any time and is generally specified using a failure rate. In an orthogonal dimension, one may distinguish between transient and persistent failure. Transient failures may arbitrarily appear and disappear, while persistent failures occur once and then stay in this mode.

A generic approach to model failure modes was already presented in [OGR07]. The key idea is to separate *when* a failure occurs and *what* its semantics are. This was extended for SAML models in [Güd11]. In the last months, we automated the modeling by adding new keywords to SAML for specific occurrence patterns (for example “hd-trans-failure” for an high-demand transient failure mode). This allows much more intuitive specification

of failure modes. Technically, the semantics of these keywords could be defined using only basic SAML constructs. This made it easy to extend the syntax in a prototypical implementation without changing the model transformations.

Another ongoing work is to extend SAML to families of MDPs. This makes sense if several (similar) design alternatives for a system exist (for example threshold for certain sensor values or alternative components). It is then interesting to compute, which thresholds/components yield the best results for a given goal. If all alternatives (i.e. the whole family of MDPs) can be expressed in one model, then it is possible to apply optimization algorithms to it. We were able to successfully demonstrate that such a combination is efficiently possible for combining a multi-dimensional genetic algorithm with stochastic model checking to find best solutions (with respect to some model checking property) in a family of MDPs[GO11]. Currently, we are working on extending SAML such that families of MDPs can be described (and be analyzed).

3.2 Specification Environment

By modeling and analyzing the case-studies it became obvious that using the current toolchain for modeling, transformation, checking and result viewing becomes quiet exhausting. For specification in SAML we had to use a text editor, model transformation required calling of LISP functions, model checking was done by command line and interpretation of the results was done by hand.

Therefore a single Safety Design Environment (SDE) would increase the usability of SAML and the model-based safety analysis techniques towards a full specification and modeling language. This SDE could not only integrate the transformation from SAML to the specific model checker language and all necessary transformations and program executions into a "push-the-button" approach[7], it would also give the developer a powerful editor with common interface and tools of other development frameworks.

Our current idea is to create a SAML SDE by using the Eclipse framework. Eclipse was chosen, as it already comes with a powerful plug-in mechanism and reusable components. We implemented SAML specific syntax highlighting, auto-completion, outline module and in-time error checking with customized failure messages for the Eclipse editor. The user can invoke the model transformations and verifications by the use of customized menu entries. The overall experience is similar to the a software development environments. The current prototypical realization of the SDE,the model transformations and the optimization are implemented in different programming languages. We are currently working on porting all parts to the Java virtual machine based runtime environment. Next steps will be the automatic generation of deductive-cause-consequence-analysis proof obligation and visualization of the analysis results.

[7] If the failure modeling extension are also integrated and DCCA[ORS06] is selected as analysis method.

3.3 SAML as Intermediate Language

Beside using SAML as independent high-level modeling and specification language as suggested before, using it as an intermediate language could be even more beneficial (see Fig. 3).

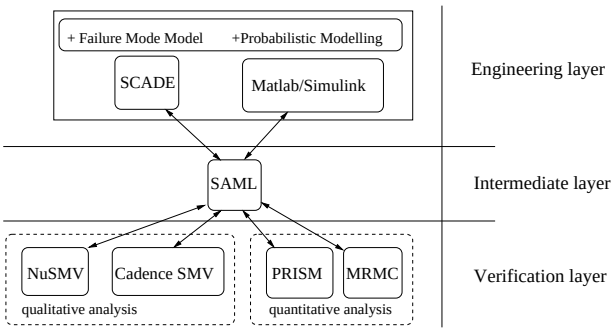

Figure 3: SAML as an intermediate language

In this scenario SAML would only be used as an intermediate layer. Specification would written in a high-level CASE tool like SCADE or Matlab and verified using NuSMV or PRISM. However there are numerous problems to be solved. Firstly, model transformation from the high-level to the intermediate level need to be specified and implemented. We did some experiments with SCADE resp. Lustre and learned this might be possible but difficult problems have to be solved. Some are of technical nature and others of computational nature. Secondly, in CASE tools typically only the software is modeled. Failure modes, environment and probabilistic behavior are usually not supported. So extensions need to be made at this point. Nevertheless, we think that such an approach is for sure worth further investigation.

4 Related Work

One prominent example of a formal language used for the development of safety critical applications is SCADE, which is based on the synchronous data-flow language LUSTRE. The SCADE suite [8] is developed by Esterel Technologies and includes a model-checker for safety properties. Nevertheless, this formal analysis is not well suited for more complex safety analysis as shown in [GOR07].

The language FIACRE [FGP+08] is included as intermediate language in the TopCased toolkit [VPF+06]. It is used to verify properties of the SysML models created in TopCased. Nevertheless, it turned out that the large number of supported SysML artifacts lead to very large formal models even for small case studies [FGP+08]. Therefore we designed SAML

[8] http://www.esterel-technologies.com

deliberately as simple as possible while retaining the necessary expressiveness for safety analysis.

A recent approach is developed in the COMPASS project [BCK+10]. Here, the already exiting FSAP-NuSMV/SA [BV03] framework is combined with the MRMC probabilistic model checker [KZH+10] to allow for the analysis of systems for Aerospace applications specified in the SLIM language, which is inspired by the Architecture Analysis and Design Language (AADL) and its error annex. The SLIM language describes the architecture and behavior of the system components, it allows for a combination of discrete and continuous (hybrid) variables. From the AADL error annex, exponentially distributed failure modes are supported and the effects can automatically be injected into the system specification. The hybrid behavior of the SLIM models and all internal transitions are removed by lumping and the resulting interactive Markov chain is analyzed with MRMC.

COMPASS is one of the few existing approaches which also combines qualitative and quantitative modeling and analysis. Nevertheless, the design is very much tool-dependent and exchanging the model-checking tools would not be easy. It is also not possible to combine hybrid and probabilistic modeling in the very same model.

In addition, none of the existing approaches allows for the integration of both per-time and per-demand failure modes as it is possible with SAML and our analysis approach. These two different probabilistic occurrence pattern are described in IEC 61508 [Lad08] as *high or continuous demand* and *low demand* failure modes. COMPASS and most other probabilistic safety analysis approaches use continuous time Markov chains models (CTMC) as system models which allows per-time failure mode modeling with failure rates. CTMCs are well suited for modeling asynchronous, interleaved systems but not for synchronous parallel systems [HKMKS00] as many safety critical system are.

5 Conclusion

The combination of quantitative and qualitative model-based safety analysis in a single framework has several advantages. It is ensured that the analyzed models are equivalent. Additionally the modeling overhead is minimized compared to the traditional approach to create a model for every safety analysis aspect. The combination and also the tool-independence of the model-based safety analysis approach based on SAML is technically possible and was successfully applied to several case studies. However the integration into a common modeling framework like the SAML SDE based on Eclipse is necessary to make the method accessible for broader acceptance in industrial contexts.

Further work will at first concentrate on well integration into the modeling tool. Afterwards the work on system variant optimization, goal specification language etc. will be further conducted as it is expected that more case studies are available.

Acknowledgments

Michael Lipaczewski is sponsored by the Deutschen Ministerium für Bildung und Forschung in the ViERforES project (BMBF, project-Nr.: 01IM08003C).

References

[BCK+10] M. Bozzano, A. Cimatti, JP. Katoen, VY. Nguyen, T. Noll, and M. Roveri. Safety, Dependability, and Performance Analysis of Extended AADL Models. *The Computer Journal*, 2010.

[BV03] M. Bozzano and A. Villafiorita. Improving System Reliability via Model Checking: the FSAP/NuSMV-SA Safety Analysis Platform. In *Proceedings of the* 22nd *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2003)*, pages 49–62. Springer, 2003.

[FGP+08] P. Farail, P. Gaufillet, F. Peres, JP. Bodeveix, M. Filali, B. Berthomieu, S. Rodrigo, F. Vernadat, H. Garavel, and F. Lang. FIACRE: an intermediate language for model verification in the TOPCASED environment. In *European Congress on Embedded Real-Time Software (ERTS 2008)*, http://www.see.asso.fr, janvier 2008. SEE.

[GO10] M. Güdemann and F. Ortmeier. A Framework for Qualitative and Quantitative Model-Based Safety Analysis. In *Proceedings of the* 12th *High Assurance System Engineering Symposium (HASE 2010)*, 2010.

[GO11] M. Güdemann and F. Ortmeier. Model-Based Multi-Objective Safety Optimization. In *Proceedings of the* 30th *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2011)*. Springer LNCS, 2011. to appear: 19.9.2011.

[GOR07] M. Güdemann, F. Ortmeier, and W. Reif. Using Deductive Cause Consequence Analysis (DCCA) with SCADE. In *Proceedings of the* 26th *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2007)*. Springer LNCS 4680, 2007.

[Güd11] M. Güdemann. *Qualitative and Quantitative Formal Model-Based Safety Analysis*. PhD thesis, 2011.

[HKMKS00] H. Hermanns, JP. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov Chain Model Checker. In *Proceedings of the* 6th *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2000)*, pages 347–362. Springer, 2000.

[JMWH05] A. Joshi, SP. Miller, M. Whalen, and MP. Heimdahl. A Proposal for Model-Based Safety Analysis. In *Proceedings of the* 24th *Digital Avionics Systems Conference (DASC 2005)*, Oct 2005.

[KHE11] J. Kloos, T. Hussain, and R. Eschbach. Failure-based testing of safety-critical embedded systems. 2011.

[KZH+10] JP. Katoen, IS. Zapreev, EM. Hahn, H. Hermanns, and DN. Jansen. The Ins and Outs of the Probabilistic Model Checker MRMC. *Performance Evaluation*, In Press, Corrected Proof:167–176, 2010.

[Lad08] PB. Ladkin. An Overview of IEC 61508 on E/E/PE Functional Safety, 2008.

[OGR07] F. Ortmeier, M. Güdemann, and W. Reif. Formal Failure Models. In *Proceedings of the* 1st *IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2007)*. Elsevier, 2007.

[ORS06] F. Ortmeier, W. Reif, and G. Schellhorn. Deductive Cause-Consequence Analysis (DCCA). In *Proceedings of the* 16th *IFAC World Congress*. Elsevier, 2006.

[Par07] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Programmers. Pragmatic Bookshelf, 2007.

[VPF+06] F. Vernadat, C. Percebois, P. Farail, R. Vingerhoeds, A. Rossignol, JP. Talpin, and D. Chemouil. The TOPCASED Project - A Toolkit in OPen-source for Critical Applications and SystEm Development. In *Data Systems In Aerospace (DASIA)*. European Space Agency (ESA Publications), 2006.