The Journal of Systems and Software 86 (2013) 2488–2501

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Efficient optimization of large probabilistic models

Simon Struck[a,*], Matthias Güdemann[b], Frank Ortmeier[a]

[a] AG Computer Systems in Engineering, Otto-von-Guericke University Magdeburg, Germany
[b] CONVECS, Inria Rhône-Alpes, Grenoble, France

ARTICLE INFO

Article history:
Received 30 July 2012
Received in revised form 7 March 2013
Accepted 20 March 2013
Available online 6 April 2013

Keywords:
Safety analysis
Formal methods
Multi-objective optimization
Safety optimization

ABSTRACT

The development of safety critical systems often requires design decisions which influence not only dependability, but also other properties which are often even antagonistic to dependability, *e.g.*, cost. Finding good compromises considering different goals while at the same time guaranteeing sufficiently high safety of a system is a very difficult task.

We propose an integrated approach for modeling, analysis and optimization of safety critical systems. It is fully automated with an implementation based on the Eclipse platform. The approach is tool-independent, different analysis tools can be used and there exists an API for the integration of different optimization and estimation algorithms. For safety critical systems, a very important criterion is the hazard occurrence probability, whose computation can be quite costly. Therefore we also provide means to speed up optimization by devising different combinations of stochastic estimators and illustrate how they can be integrated into the approach.

We illustrate the approach on relevant case-studies and provide experimental details to validate its effectiveness and applicability.

1. Introduction

In many domains, software has become a major innovation factor. Very often different systems use the same standard hardware, their difference is mainly due to different software implementations and features. Nowadays this trend not only holds for general computers, laptops and tablets, but also for embedded systems and cyber-physical systems.

Such systems are more and more used for safety critical applications in domains like avionics and automotive. Good examples are the various "X-by-wire" systems which increasingly replace traditional mechanical connections. For obvious reasons, one must ensure the dependability and reliability of such systems before they can be put in use.

Unfortunately the increasing complexity resulting from ever more functionality realized in software renders safety analysis more difficult. In contrast to mechanical systems, the effects of failures in software are not continuous. For physical devices, most often a small failure will have rather small effect; in contrast to that, even a small software error can have completely unpredictable effects. In addition, it is not sufficient to verify software in isolation of its environment, as it is done in classical software verification. In any case, the effect of possible hardware errors must be taken into account, as well as the physical environment where the system will be used. Finally, even if an accurate analysis and evaluation of an embedded system for a safety critical application is possible, it is not obvious if this is the best possible system variant with specific properties. In general, different and often even antagonistic objectives must be considered for the final system configuration.

A lot of work has been done to tackle the problem of qualitative safety analysis for complex embedded systems where component failures must be taken into account, *e.g.*, see Åkerlund et al. (2006), Bozzano et al. (2003), Bozzano and Villafiorita (2003), Ortmeier et al. (2005), Distefano and Puliafito (2007), Walker et al. (2007). These approaches allow for analysis of combinations of component failures which may cause a potentially dangerous system malfunction, called a hazard. For certification of a safety critical system, it is important to conduct a quantitative safety analysis computing the hazard occurrence probability. This is often achieved using rather approximations based on qualitative analysis results and requires assumptions of stochastic independence. Newer approaches like (Aljazzar et al., 2009; Bozzano et al., 2010; Güdemann and Ortmeier, 2010b) compute hazard probabilities directly, with a much higher degree of accuracy, but also with higher computational cost. There also exist some first approaches to optimize systems like (Meedeniya et al., 2010; Papadopoulos et al., 2010) which try to find a system variant which is optimized, in order to improve its reliability.

* Corresponding author. Tel.: +49 391 6719353.
E-mail addresses: simon.struck@ovgu.de (S. Struck), matthias.gudemann@gmail.com (M. Güdemann), frank.ortmeier@ovgu.de (F. Ortmeier).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.03.078

The approach presented in this article consists of the following: we propose to use the safety analysis and modeling language (SAML) (Güdemann and Ortmeier, 2010a) to model safety critical systems, allowing for qualitative and quantitative safety analysis on the same model, *i.e.*, without the need to construct different models for each of the different analyses. We extend SAML to support the specification of a set of possible systems and of different objective functions for optimization. Modern multi-objective optimization algorithms, including means to reduce computation time, are used to find the best possible system variants. All analyses are conducted fully automatic using different state-of-the art model checking tools. The approach is tool-independent, it uses model transformations to convert SAML models into the input specification language of analysis tools which allows for easy integration of new analysis tools. All this is integrated into an extensible framework based on the Eclipse platform. The work presented here is an extension of our first, rather ad-hoc approach to safety optimization, presented in Güdemann and Ortmeier (2011a). We now have developed an explicit notion of variability modeling and provide an interface for arbitrary objective functions and for arbitrary estimation methods. We also present the results of our experiments using different approaches to exploit estimation techniques to make the optimization process more efficient.

2. Modeling

Our approach is based on creating a model of the system with its intended functional behavior, the behavior of its surrounding physical environment and the occurrence pattern and effect of failure modes.

We use a rather low level modeling language to express our models. The safety analysis and modeling language (SAML) (Güdemann and Ortmeier, 2010a) is derived from the PRISM modeling language (Norman et al., 2010) and describes Markov decision processes (MDP) (de Alfaro et al., 2005). This formal model allows the combination of software modeling, where failure modes are often non-deterministic and physical component modeling, where failure modes are often probabilistic. It is also possible to combine per-time and per-demand failure mode modeling with high accuracy (Güdemann and Ortmeier, 2010b). Model transformations allow the analysis with different verification tools, dependent on the desired properties.

We chose SAML, as it is very close to the underlying formal model and therefore does not introduce much overhead into the state space. Nevertheless, it is possible to transform higher-level models to SAML for analysis. For an outline of that approach see Güdemann and Ortmeier (2011b) and some discussion of such possibilities in Section 6. The following description of SAML is adapted from Güdemann (2011).

2.1. MDP/SAML

The most important aspect in the development of SAML was the possibility to model the control software, the physical environment and possible failure modes. These are all relevant for embedded systems in a safety critical domain. A SAML model is then analyzed using state-of-the-art verification engines, using proven correct model-transformation. In most cases, a SAML model consists of a model of the nominal behavior, an accurate model of the behavior of its physical environment and probabilistic failure mode modeling. Non-deterministic behavior facilitates specification of software failure modes and environment modeling, where probabilities are not known or cannot be given. Probabilistic behavior often reflects well physical environment modeling with known probabilities and failure modes of physical components.

```
saml-model :  (constant | formula)* module+ ;
constant :  constant TYPE IDENT (:= value)?  ;
formula :  formula IDENT := condition ;

condition :  ( condition )
           | !  condition
           | condition & condition
           | condition | condition
           | term ;

term :  IDENT (= |< |> |>= |<=) state_expr
      | IDENT | true | false;

module :  module IDENT declaration+ update+ endmodule;
declaration :  IDENT : [ INT ..  INT ] init INT ;

update :  condition -> non-det_assigns
        | prob_assigns ;

non-det_assigns :  non-det_assign
                 (+ non-det_assign)* ;

non-det_assign :  choice ( prob_assigns ) ;
prob_assigns :  prob_assign (+ prob_assign)* ;

prob_assign :  probability :  nextstate_assign
                            (& nextstate_assign)* ;

probability :  IDENT | DOUBLE | arith_expr;
nextstate_assign :  ( IDENT' = state_expr ) ;

state_expr :  IDENT | INT | ( state_expr )
            | state_expr (+ | - | / | *) state_expr ;

arith_expr :  INT | DOUBLE | IDENT | ( arith_expr )
            | arith_expr (+ | - | / | *) arith_expr
```

Fig. 1. Basic SAML syntax.

Syntactically, SAML models describe sets of finite state automata with non-deterministic and probabilistic transitions. These are executed in a synchronous parallel fashion with discrete time-steps. SAML has been designed to be tool-independent and as simple as possible, while being expressible enough for convenient modeling of larger case-studies. It has been implemented using the ANTlr framework with an Eclipse-based specification front-end. Automatic model transformations for well-known model checkers (NuSMV, PRISM) also exist. For space restrictions, we do not present the complete formal syntax and semantics here. The complete definitions can be found here (Güdemann and Ortmeier, 2010a).

The grammar rules for the syntax of the most important part of SAML is shown in Fig. 1 in Extended Backus Naur Form (EBNF) notation.[1] The actual implementation is done using the ANTlr parser generator (Parr, 2007). The syntax is derived from the input language of the PRISM model checker (Kwiatkowska et al., 2002; Norman et al., 2010). The main differences to the PRISM language are the absence of synchronization labels and the explicit modeling of non-deterministic choices with the `choice` keyword. These changes were done to facilitate the correct constructions for formal safety analysis. It forces the user to adhere to the modeling guideline which results in SAML models for which the presented safety analysis is sound. In contrast, PRISM uses implicit non-determinism modeling, *i.e.*, overlapping activation conditions. In the case of only a partial overlap, the probability distributions must be normalized which potentially changes the intention of the user. Therefore we chose to make non-determinism modeling explicit and treat overlapping activation conditions as modeling error.

Fig. 2 shows an example SAML model, consisting of two modules, `A` and `B`. Module `A` contains the state variable `V_A` with a value

[1] Lexer rules are written uppercase, parser rules in lowercase. Bold font indicates keywords or literal symbols without explicit lexer rules.