Multi-Objective Optimization of Formal Specifications

Simon Struck, Michael Lipaczewski, Frank Ortmeier
Otto-von-Guericke University Magdeburg
Computer Systems in Engineering
Magdeburg, Germany
{simon.struck, michael.lipaczewski, frank.ortmeier}@ovgu.de

Matthias Güdemann
CONVECS
Inria Grenoble - Rhône-Alpes
Grenoble, France
matthias.gudemann@inria.fr

***Abstract*—Even in the domain of safety critical systems, safety and reliability are not the only goals and a developing engineer is faced with the problem to find good compromises wrt. other antagonistic objectives, in particular economic aspects of a system. Thus there does not exist a single optimal design variant of a system but only compromises each "best" in its own rights . With the rising complexity, especially of cyber-physical systems, the process of manually finding best compromises becomes even more difficult.**

To cope with this problem, we propose a model-based optimization approach which uses quantitative model-based safety analysis. While the general approach is tool-independent, we implement it technically by introducing well defined variation points to a formal system model. These allow enough variability to cover whole families of systems while still being rigorous enough for formal analysis. From the specification of this family of system variants and a set of objective functions, we compute Pareto optimal sets, which represent best compromises.

In this paper we present a framework which allows for optimization of arbitrary quantitative goal functions, in particular probabilistic temporal logic properties used for model-based safety analysis. Nevertheless, the approach itself is well applicable to other domains.

I. Introduction

The term "cyber-physical system" describes a system with a tight coupling between the computational units of a system and its physical components. Computational units are all kinds of programmable devices such as micro controllers, micro computers, embedded systems and computer systems. Physical components comprise all kind of sensors and actors, but also every mechanical interconnection between them. The overall system behavior cannot be determined by analyzing the program code or the physical components in isolation; only a combined analysis of both explains the system behavior.

To cope with the safety requirements of complex reactive systems, model-based safety analysis methods were developed [1], [2]. These methods rely on a formal model which consists of the software control, i.e., the behavior of the computational units, and its environment, i.e., all the physical components interconnected with the computational units. This model is then analyzed for safety aspects using deduction, most often based on automatic model checking techniques.

However, for most cyber-physical systems, safety is not the only important aspect, even in safety critical domains. Other aspects are for example system costs or performance. A system with very high dependability may simply be too expensive to build, or its performance might be below expectations or even requirements. On the other hand, it will often be a goal to build a system with the highest possible performance and lowest cost, but at the same time with an appropriate safety to allow for necessary certifications. Because of the often antagonistic nature of these different aspects, there is typically no single optimal possibility to develop a cyber-physical system. In general there exist different variants, where each is a best compromise in the sense that only worsening one aspect can augment another. The identification of such compromises is no trivial task. It is traditionally performed manually by engineers during system development.

In this paper we propose a systematic model-based optimization approach to find the set of best compromises for a cyber-physical system. We implemented the approach using a formal system modeling language (SAML) and multi-objective optimization algorithms to identify best system specifications.

The antagonistic nature of reliability, availability, maintainability and safety (RAMS) is also context of standards like IEC62278/EN50126. In general, this can be seen as a multi-objective optimization problem, i.e. the identification of best compromises among the four different objectives. Thus our approach is capable of targeting requirements according to RAMS.

The remainder of the paper provides an introduction to multi-objective optimization in Section II. Section III provides a brief introduction of SAML and introduces our formal analysis techniques. We describe our optimization approach of formal specifications in Section IV and provide an evaluation of our approach on a case study in Section V. Finally, Section VI provides some related approaches and Section VII gives an outlook on future work.

II. Multi-Objective Optimization

To optimize a system design wrt. different goals and to identify the best compromises, we use a very abstract view on a system. We consider the possible variations of a system as free parameters of an optimization problem and the different characteristics of the system as its objective functions. More precisely, we consider it to be a multi-objective optimization

problem (MOP). Every possible configuration of the parameters describes a specific system variant for which the objective functions are evaluated. One of these is the probabilistic safety analysis, which consists of the computation of the hazard occurrence probability.

In general a MOP is described as follows: All k parameters of a system are denoted as a vector of decision variables $\mathbf{x} = \{x_1, \ldots, x_k\} \in X$. The l objectives of the optimization are functions $i = 0 \ldots l, f_i : X \to \mathbb{R}$. All l objectives together forming the vector-valued function $\mathbf{f} = (f_1(\mathbf{x}), \ldots, f_l(\mathbf{x}))^T$, the MOP is defined as the minimization of the vector-valued objective function:

$$min\left(\mathbf{f}\left(\mathbf{x}\right)\right) = min \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_l(\mathbf{x}) \end{pmatrix} \tag{1}$$

For such vector functions, it is in general impossible to define a total order, in particular if the objective functions f_i are antagonistic to each other. Consider for example a system that moves goods from one point to another. While designing such a system, the engineer needs to determine the maximum allowed speed. In this example, there might be two objectives functions that need to be considered: The traveling time and the systems hazard probability. Obviously with a higher speed, the traveling time decreases and the hazard probability increases. It is clear that there is no single best solution considering the minimization of the hazard probability and the traveling time at the same time.

To cope with antagonistic objective functions we rely on the Pareto dominance criterion. It is a partial order which allows for optimality comparison of different solutions. Further details are for example in [3].

Definition *1 (Pareto Dominance):* A vector $u = (u_1, \ldots, u_n) \in \mathcal{R}^n$ dominates another vector $v = (v_1, \ldots, v_n) \in \mathcal{R}^n$ if and only if:

$$\forall i \in \{1, \ldots, n\} : u_i \leq v_i \wedge \exists j \in \{1, \ldots, n\} : u_j < v_i. \tag{2}$$

If u dominates v this is also denoted as $u \prec v$.

Given two vectors u and v, where $u \prec v$ means that all elements in u are less than or equal to the corresponding elements in v and at least one element in u is strictly smaller than the corresponding element in v. In other words: u is for at least one goal function strictly better than v *and* not worse in all others.

Applied to the example of the transportation system, the Pareto dominance considers a solution with a short traveling time and a high hazard probability to be equally good as a system with a long traveling time and a low hazard probability. Yet a solution with a high traveling time and a high hazard probability is not optimal. This fits the idea of the best compromises. All parameter settings $\mathbf{x}$ that lead to Pareto optimal solutions form the Pareto set:

Definition *2 (Pareto Set):* For a given multi-objective optimization problem, $\mathbf{f}(\mathbf{x})$ with $\mathbf{x} \in X$, the Pareto Set, $\mathcal{P}^*$, is defined as:

$$\mathcal{P}^* := \{\mathbf{x} \in X \mid \neg\exists \mathbf{x}' \in X : \mathbf{f}(\mathbf{x}') \prec \mathbf{f}(\mathbf{x})\}. \tag{3}$$

The set of images of the elements of the Pareto set defines the Pareto front, as the set of solutions that do not dominate each other, and are not dominated by any other available solution:

Definition *3 (Pareto Front):* For a given multi-objective optimization problem, $\mathbf{f}(x)$ with $\mathbf{x} \in X$, and its Pareto Set $\mathcal{P}^*$ the Pareto front $\mathcal{PF}^*$ is defined as:

$$\mathcal{PF}^* := \{u = \mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\} \tag{4}$$

Finding the Pareto front is a non-trivial problem even for analytic functions. In the past we did some experiments to approximate the Pareto set using evolutionary algorithms [4]. They are well applicable and are widely used for MOP. Besides sophisticated search algorithms, more pragmatic approaches are also feasible. In Section V we use brute force and non-dominated sorting to solve a MOP with a sufficiently small search space.

All elements in the Pareto optimal set are considered as equally good solutions. To select a single one for the realization of one system variant, further criteria are required. These could either be an expert's choice, or further heuristics. However, these are very domain specific and thus not discussed in this paper.

III. Model-Based Safety Analysis

Model-based safety analysis forms the basis for the proposed optimization approach. It starts from a formal description of a model, its physical environment and the possible failure modes and uses formal deduction techniques to analyze the system.

A. Specification

We express models in the safety analysis modeling language (SAML). Here, we only cover its syntax and semantics as far as it is required to understand the optimization approach. A complete definition is presented in [5], [6]. It is derived from the PRISM language [7], allows for modeling of finite state automata and uses a discrete time model. The state automata are executed in a synchronous parallel way, the transitions allow for the expression of combined probabilistic and non-deterministic behavior.

```
constant double prob := 0.2;
module myMod
    state : [1..3] init 1;
    state = 1 -> choice:( 1: (state' = 2));
    state = 2 -> choice:( prob: (state' = 1) +
                          (1-prob): (state' = 3));
    state = 3 ->choice:( 1: (state' = 1)) +
                 choice:( 1: (state' = 2));
endmodule
```

Fig. 1. A minimalistic SAML model with only one module

Every automaton description is included in a *module* that consists of one or more state variables and a set of update rules. Figure 1 shows a brief example. The module `myMod` contains one state variable named `state`, defined for the interval of naturals from one to three and the initial value one.

Transitions are specified in update rules. Every update rule has an activation condition and one or more non-deterministic choices. The activation condition is a propositional logic formula. It denotes in which system states the update rule is active. In the example there is one update rule for each of the three possible states. If the activation condition evaluates to true, one of the following non-deterministic choices (denoted by the keyword `choice`) is selected. In the example, the first update rule consists of one choice and a trivial probability distribution. The third update rule consists of two choices. Within every non-deterministic choice a discrete probability distribution is given. The second update rule in the example shows a non-trivial probability distribution. The constant (`prob`) denotes the probability of reaching one of the two possible successor states. For a variable v, v' marks the value in the next time step, i.e., the successor state. If the module uses more than one state variable, the assignment is done in parallel.

Constants are used as a generic concept to use named aliases for certain numbers. Constants may be of floating point and integer types. Additional to the usage in the example as probabilities (`prob`) they may also be used in the propositional expressions of the activation conditions and in next state assignments.

Besides constants and modules, SAML supports a third construct called *formulas* which are propositional logic expressions over the state variables. In Summary, a SAML model consists of a set of constants, a set of formulas and a set of modules:

Definition *4 (SAML Model):* A SAML model is syntactically defined as a tuple

$$\mathcal{S} = (\mathcal{C}, \mathcal{F}, \mathcal{M}), \tag{5}$$

where $\mathcal{C}$ is a set of constants, $\mathcal{F}$ is a set of formulas and $\mathcal{M}$ is a set of modules.

The semantics of a SAML model with one single module is a Markov decision process (MDP). An MDP consists of a finite set of states, one initial state and a non-empty set of probability distributions over the successor states. A labeling function assigns a set of atomic Boolean propositions to each state. For a detailed definition see, e.g. [8].

If a model contains more than one module and thus more than one state automaton, these are combined into one single module by parallel composition of all modules, creating the product automaton of all synchronous parallel automata.

Informally, a SAML model with a single module and a single state variable maps to an MDP in the following way: All possible values of the state variable form the set of states. The initial value (stated with the `init` keyword) denotes the initial state. The update rules of the module contain the probability distributions over the successor states. Multiple distributions can be assigned to every state. The exact semantics of SAML models can be found in [5].

The MDP for the SAML model listed in Figure 1 is depicted in Figure 2. Here every value of the single state variable represents one state. The transitions are labeled with $k : p$ where k is the index of the non-deterministic choice that is available in the originating state and p is the probability of this transition in the selected choice.

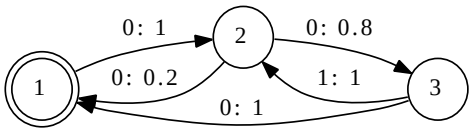

Fig. 2. The MDP for the SAML model in Figure 1

A valid SAML model must fulfill the following requirements: (1) All the probabilities in a distribution must sum to one. This ensures that the distributions are valid probability distributions. (2) The activation conditions in one module must be pairwise exclusive. This assures that normalization of the probabilities is not necessary, their intended value does not change and also that there is no unwanted non-determinism in the model. (3) The transition relation must be total. This assures that every modeled deadlock state is also deliberately integrated.

B. Analysis

For an actual analysis, SAML models are transformed into the input languages of state of the art model checkers using provably sound model transformations [5]. Currently implemented are NuSMV[1] and PRISM[2]. In this section we provide a brief introduction to model based safety analysis of SAML models. A complete explanation of model-based safety analysis is out of scope of this paper.

NuSMV is a symbolic model checker for Kripke structures. It performs qualitative analysis, for which all the probabilistic information in the SAML model is stripped and replaced with non-determinism. The qualitative safety analysis points out worst case scenarios. The properties are specified in computational tree logic (CTL) [9]. If the model checker finds a violation of a property in the given model, it presents a sequence of states (starting from the initial state) that demonstrates the violation.

The results of the qualitative analysis are not directly relevant for the optimization procedure, but can identify infeasible system variants. Still, they can provide very valuable information about the behavior of the model to the system engineer. Counter examples can be produced which show a possible sequence of states, for further analysis by a human analyst or engineer.

Qualitative analysis is very useful for safety analysis. The deductive cause consequence analysis (DCCA) is a qualitative

[1] http://nusmv.fbk.eu/

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

model-based safety analysis approach [10]. For DCCA, the analyst must identify unique failure modes of system components. The model is then extended with a set of automata which represents the possible occurrence for every failure mode. The functional part of the model is extended in such a way that every component (module) has an appropriate failure behavior if the corresponding failure automaton is in the failure state. It is important to notice that the failure extended model is path inclusion equivalent to the model without failure extension [11]. Based on the failure extended model and a specification of an hazard (a set of states which are considered as hazardous), the DCCA automatically calculates all possible failure combinations that can lead to a hazardous system state. In general, qualitative analysis is used in our optimization approach for identifying feasible solutions which satisfy a set of qualitative properties.

The quantitative safety analysis is performed using a probabilistic model checker. It is of fundamental importance for the optimization. Quantitative properties exploit the probabilistic information in the SAML models. Their result is not only a simple *yes* or *no* but a probability stating how likely it is that a trace of the model is chosen on which a certain temporal logic property holds . The specification of properties is expressed in PCTL probabilistic temporal logic [12]. For safety analysis we use the probabilistic deductive cause consequence analysis (pDCCA) [13], [5]. It computes the occurrence probability of the system hazard, if the occurrence probabilities of all possible failures are specified. The approach allows for a sensible combination of per-time failure modes specified by failure rates and per-demand failure modes specified by failure probabilities.

IV. Optimization of a Specification

A. Syntax and Semantics

Qualitative and quantitative analysis can be performed on SAML models. Using this, a system can be optimized for multiple goals by extending the specification with certain variation points. We now propose a modifier keyword (`param`) to SAML, used in the context of constants and modules. We named the new language optimizable-SAML (OSAML).

- **Param Module:** describes a set of interchangeable modules. Each of the modules must have the same signature in terms of the state space (i.e. Alternative modules must have the same state variables). However, the state transitions may differ. Thus variants of similar behavior can be expressed.
- **Param Constant:** describes constants in a model that may be changed. Throughout one analysis this value is constant, but another analysis may be performed with different values for the constant.

The idea behind *param module* is to model parts with a similar purpose but different architecture or technique. Consider the configuration of a car where several different tachometers are available; all do measure the speed, but one does this by measuring the rotational speed of the wheels and another is based on satellite navigation. Both will differ in accuracy, reliability and costs. In a formal model such a component might be modeled with a state variable representing the current reading. The rotational measurement can immediately present coarse results, while the satellite-based measurement presents very accurate results but with a slower update interval. In both cases the state variables are the same, but the update rules differ.

A very minimalistic example with one param constant and one param module is depicted in Figure 3. The first variant of the param module (`variantA`) is the same as the module depicted in Figure 1. The second variant (`variantB`) possesses the same definition for the state variable (`state : [1..3]`), but has other update rules. The constant used in the example from Figure 1 is now also turned into a param constant. This means, that it specifies no specific value but an interval of allowed/valid values.

```
param constant double p := [0.2 .. 0.5];
param module myMod
   module variantA
      state : [1..3] init 1;
      state = 1 -> choice:( 1: (state' = 2));
      state = 2 -> choice:( p: (state' = 1) +
                            (1-p): (state' = 3));
      state = 3 -> choice:( 1: (state' = 1)) +
                   choice:( 1: (state' = 2));
   endmodule
   module variantB
      state : [1..3] init 1;
      state = 1 -> choice:( p: (state' = 2) +
                            (1-p): (state' = 3));
      state = 2 -> choice:( 1: (state' = 3 ));
      state = 3 -> choice:( 1: (state' = 1));
   endmodule
endparam
```

Fig. 3. Minimalistic OSAML model

An OSAML model is a tuple of sets of all the elements. In addition to the SAML model there are two more sets for the param modules and the param constants:

Definition *5:* An OSAML model is syntactically defined as a tuple

$$\mathcal{OS} = \left(\mathcal{C}^O, \mathcal{PC}^O, \mathcal{F}^O, \mathcal{M}^O, \mathcal{PM}^O\right), \tag{6}$$

where $\mathcal{C}^O$ is a set of constants, $\mathcal{F}^O$ is a set of formulae, $\mathcal{M}^O$ is a set of modules, $\mathcal{PC}^O$ is a set of sets of param constants and $\mathcal{PM}^O$ is a set of param modules.

Here $\mathcal{PC}^O$ is a finite set of param constants. Every param constant may cover an infinite set of values. Similarly $\mathcal{PM}^O$ is a finite set of param modules where every param module contains a finite number of (normal) modules.

On the semantic layer, an OSAML model is a family of SAML models (which matches a family of MDPs). Based on the assumption that a family can contain good and bad candidates there is no point in analyzing a whole family. For a meaningful analysis one specific variant for each param module and a certain value for every param constant must be

chosen. Thus only members of the model family are analyzed. We call the process of selecting one specific SAML model from a family the instantiation of an OSAML model.

Definition *6 (Instance of an OSAML model):* An SAML model $\mathcal{S}$ is an instance of a OSAML model $\mathcal{OS}$ if and only if all of the following rules apply:

1) For every param constant in the OSAML model $\mathcal{OS}$ a constant (with the same name) exists in the SAML model $\mathcal{S}$.
2) The types of the param constants in the OSAML model $\mathcal{OS}$ must match with the corresponding constants in the SAML model $\mathcal{S}$.
3) The values of the constants corresponding to the param constants must fit in the domain of the param constants.
4) For every param module in the OSAML model $\mathcal{OS}$ one module exists in the SAML model $\mathcal{S}$ that equals one module in the corresponding param module group.
5) Every constant, formula and module in the SAML model $\mathcal{S}$ is also in the OSAML model $\mathcal{OS}$.
6) The SAML model $\mathcal{S}$ contains no additional constants, formulas and modules except the above mentioned.

We omit a complete formal definition because it is rather complicated to express, while the meaning is pretty straight forward. Instantiating an OSAML model is done in the following way: In a first step, all modules, constants and formulas are transferred from the OSAML model to an empty SAML model that is going to become the instance. In a second step, every param constant is turned into an appropriate constant of the same type and with a value from the domain of the param constant. The specific values of the generated constants depends on the desired instance. The newly generated constants are added to the list of constants in the SAML model. In the third step, one module is picked out of every param module and added to the list of modules in the SAML model. Obviously, now every instance of an OSAML model can be analyzed as every other SAML model.

We extended the existing SAML grammar [5] with rules for the *param* keyword and used the ANTLR [14] parser generator for the creation of a OSAML parser. After parsing all constants, formulas, modules, param constants and param modules they are stored in an abstract syntax tree consisting of instances as Java objects. Given the abstract syntax tree the instantiation of a OSAML model is then a trivial procedure.

B. Optimization Framework

We extended the existing implementation of the SAML framework towards optimization. The software concept is depicted in Figure 4. An OSAML model is covered by a *SamlSystemCluster* instance. The *generateInstance* method performs the instantiation. The resulting SAML model is represented by a *SamlSystem* object.

All objectives for the optimization must implement the *IObjective* interface. In Figure 4 this is exemplary shown for the PRISM model checker based safety analysis. This class (*PrismObjective*) contains the model transformation and also handles the invocation of the model checker.

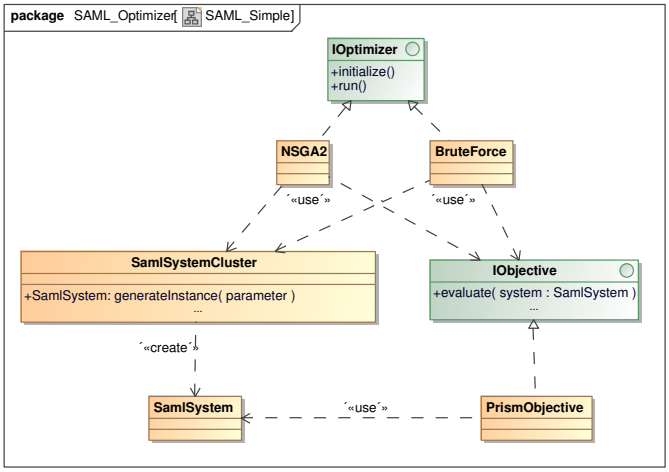

Fig. 4. Simplified software architecture of the SAML optimizer

All optimization algorithms must implement the *IOptimizer* interface. Currently there are implementations of a brute force enumeration algorithm and the fast non-dominated sorting genetic algorithm (NSGA2) [15] available. The optimizer implementations use the *SamlSystemCluster* to create instances of the OSAML model and the *IObjective* interface to evaluate the optimization objectives. In general, the architecture of the framework provides two ways to specify objectives for the optimization.

- Use one or more CTL/PCTL formulas with the (already existing) PRISM or NuSMV objectives.
- Generic objectives can be expressed in Java code that implements the *IObjective* interface. These objectives can access both the SAML instance and the parameter configuration used to generate the instance.

Both methods can be used in the same optimization problem. It is therefore possible to optimize two ore more PCTL properties and several Java based objectives at the same time.

V. Evaluation

We evaluated the approach on a case study of a redundant data processing system. The goal was to identify optimal design solutions by minimizing the hazard probability and the system costs at the same time.

A. Case Study

The following case study from model-based safety analysis is taken from literature. It was first presented in [16].

The case study consists of the measuring of an input signal, its computational processing, and the generation of an output signal. For higher reliability certain components are redundant. A block diagram is depicted in Figure 5. The two sensors $S1$ and $S2$ measure an input signal. The sensor values are then processed by the arithmetical units $A1$ and $A2$. The first arithmetical unit uses both sensor signals and the second one only uses one sensor signal. The second arithmetical unit is disabled by default. A monitor M observes $A1$ and activates

the second algorithmic unit if the first one fails. The output unit O selects the proper signal from the two redundant arithmetical units. A quantitative and qualitative analysis of the model may be found in [10] and [13].

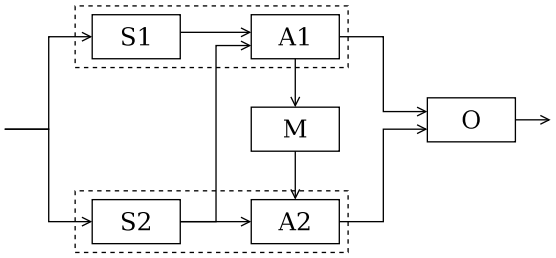

Fig. 5. A System with Redundant Components

For optimization, we introduced three parameters in addition to the original case study. This turns the specification into a family of specifications. All parameters are expressed in the OSAML model with *param constants* and *param modules*. The parameters are as follows:

- Instead of using a separate sensor $S1$ and arithmetical unit $A1$, one unit that combines the functionality of the sensor and the arithmetical unit can be used. In Figure 5 this is indicated by the dashed box around $S1$ and $A1$.
- Analogously to $S1$ and $A1$ also $S2$ and $A2$ may be combined into one unit. In the figure this is indicated with the second dashed box.
- For the monitor there are 5 different realizations available, which differ in the failure probability:. $1 \cdot 10^{-4}$, $5 \cdot 10^{-4}$, $1 \cdot 10^{-5}$, $5 \cdot 10^{-5}$, $1 \cdot 10^{-6}$ and $5 \cdot 10^{-6}$.

The failure of all other components is modeled with a probability of 10^{-5}. We did not assign a distinct time model to our formal state-based model. Thus all stated probabilities are per-step probabilities. If the analysis results shall be evaluated for a real system, a sampling interval Δt must be specified. The relation between failure rates and the per-step probability is defined in [5].

The first two parameters are realized with the *param module* feature in OSAML. The third parameter can be expressed as *param constant* or also as *param module*. A *param constant* would allow the setting of arbitrary values between the lower and upper bounds. Thus we used a *param module* to express the logarithmic scaling of the failure probability parameter.

B. Pareto Optimization of the Case-Study

For multi-objective optimization of the case study, we introduced two different objective functions. The first one is the occurrence probability of the hazard within k time-steps, where k is also referred as mission time. This hazard is defined as the state when the system is not able to generate a proper output signal. With H describing the hazardous state (i.e., when the system does not report a proper output signal) the failure probability objective using pDCCA is defined as PCTL formula[3]:

$$Pmax_k(H) := Pmax_{=?}[trueU^{\leq k}H] \tag{7}$$

This objective is evaluated with the PRISM model checker. For the optimization we assumed a mission time $k = 360$ steps.

The second objective calculates the costs of the system. It is based on the assumption that a system with less components is cheaper and that components with lower failure probability are more expensive. The combination of a sensor and an arithmetical unit into one component is assumed to cost 1, whereas the separation in two components costs 2. For the Monitor module the costs are 1, 2, 4, 8 and 16 in sequence of decreasing failure probability. The overall system costs are then defined as the sum of the costs of all chosen system components. This objective function is implemented directly as Java method.

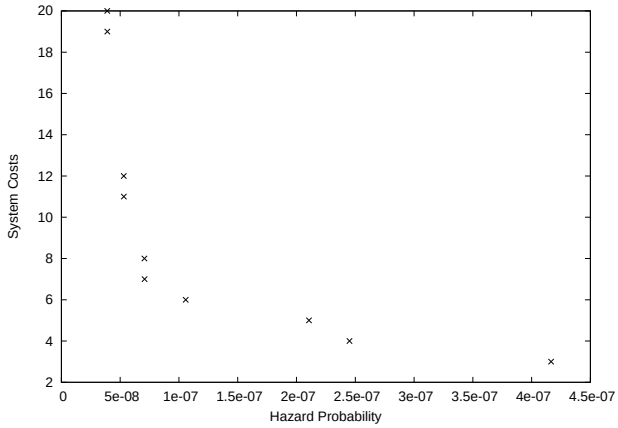

Fig. 6. The Pareto Front

This leads to two binary parameters and one parameter with up to five different values. All in all this gives a total of 20 different parameter configurations. The interesting question is, which of these configurations lead to Pareto optimal systems.

Due to the fact that the search space of the problem is rather small we decided to solve the problem with brute force via the enumeration of the design space. However, the current implementation also contains an evolutionary algorithm for dealing with larger or infinite families.

The Pareto front for the example is depicted in Figure 6, the failure probability on the x axis and the system costs on the y axis. All ten exact parameter configurations of the Pareto optimal solutions are listed in Table I. For the example, only half of all available parameter configurations lead to optimal systems. This means that for every non optimal parameter configuration a different configuration exists, that is both cheaper and has a lower hazard probability at the same time.

[3]The $=?$ in the formula belongs to PCTL syntax and applies to the quantitative aspect of PCTL. It denotes that the formula evaluates to the occurrence probability of the subsequent CTL formula.

Design Variant	$S1/A1$	$S2/A2$	P(Monitor Failure)	$Pmax_k(H)$	Costs
1	combined	combined	$1 \cdot 10^{-4}$	$4.1656893 \cdot 10^{-7}$	3
2	separate	combined	$5 \cdot 10^{-4}$	$2.1063764 \cdot 10^{-7}$	5
3	combined	combined	$5 \cdot 10^{-4}$	$2.4509865 \cdot 10^{-7}$	4
4	separate	separate	$1 \cdot 10^{-5}$	$7.0649944 \cdot 10^{-8}$	8
5	separate	combined	$1 \cdot 10^{-5}$	$7.0775668 \cdot 10^{-8}$	7
6	combined	combined	$1 \cdot 10^{-5}$	$1.0572783 \cdot 10^{-7}$	6
7	separate	separate	$5 \cdot 10^{-5}$	$5.3059304 \cdot 10^{-8}$	12
8	separate	combined	$5 \cdot 10^{-6}$	$5.3153742 \cdot 10^{-8}$	11
9	separate	separate	$1 \cdot 10^{-6}$	$3.8964419 \cdot 10^{-8}$	20
10	separate	combined	$1 \cdot 10^{-6}$	$3.9033802 \cdot 10^{-8}$	19

TABLE I
PARETO OPTIMAL PARAMETER COMBINATIONS AND RESULTING HAZARD PROBABILITY AND SYSTEM COSTS

By further analyzing the results in Figure 6 it seems like there are dominated points in the front. However, Table I clarifies that they are all non-dominated. It turns out that the affected pairs of design variants ($\{4, 5\}$, $\{7, 8\}$ and $\{9, 10\}$) only differ in the realization of $S2$ and $A2$. This leads only to a very small decrease in the hazard probability while causing a strong impact on the costs. All other parameter configurations differ rather strongly in both, the hazard probability and the system costs. This means that in theory, all these solutions must be considered equally good. In reality of course, one of the variants must be chosen for the implementation.

If there is only a minor impact on one objective while there is a strong impact on the other objective it is straight forward to pick the cheaper one. To choose a variant when a stronger compromise is required, further strategies are needed. One way is to prioritize the objective functions, e.g., setting a threshold for the hazard probability and then taking the system with the lowest cost which is below that threshold. Another approach is to identify regions where the increase of one objective function leads to a rather large deterioration in at least one other [17]. According to this rule, solution number 5 seems to be a good choice. There are of course other methods and heuristics to chose a single design variant from a Pareto set, often dependent on the nature of the problem domain.

VI. RELATED WORK

A first safety optimization approach which uses hazard probabilities as objectives was presented in [18]. This a-posteriori approach used an analytic mathematical model of all objectives for the optimization. Even though this method is computationally efficient, it relies heavily on stochastic independence which is not a very realistic assumption in more complex models.

The underlying idea of the SAML based multi-objective optimization was presented in [4]. In this paper the authors applied an evolutionary algorithm on a parametric SAML specification. To speed up the optimization process, artificial neural networks where proposed. However, it was on a very prototypical level without a grammar nor specific semantics of the parameters used in the specification. OSAML continues this initial idea, and introduces proper syntax and semantic as well as a proper implementation and a framework for the objective functions.

Another optimization approach of formal specifications is presented in [19]. In this case Markov reward models of embedded systems are optimized along reliability and energy consumption objectives. However, this approach is limited to quantitative analysis. By using SAML as specification language we can perform qualitative and quantitative analysis on the same model.

A completely different approach of analyzing parametric Markov models is presented in [20]. The parametric Markov model and a PCTL property is transformed into a polynomial formula, that states the dependency between the parameter values and the analysis result. However, translating a parametric model into a polynomial formula is a complex problem. Even if it is solvable for complex case studies the complexity still remains an issue. Also the approach presented by Hahn et al. does not fully support Markov decision processes (as SAML does).

Design space exploration (DSE) is another important topic for identifying ideal system configurations. DSE based approaches are presented for example in [21] and [22]. These approaches use either UML or proprietary modeling languages for system description and a explanatory algorithm to seek optimal solutions. In contrast to our Approach, DSE uses a set of variation rules to alter the model. The proposed modeling formalisms support the expression of complex and large structural models, but miss the support for the verification of functional aspects and non-deterministic behavior. From a theoretical point of view our approach should be able to cover the objectives aimed by the DSE approaches.

VII. CONCLUSION AND OUTLOOK

We proposed an approach to optimize formal specifications wrt. objective function expressed either as PCTL properties or directly via Java methods. As the objectives can be antagonistic, there exists no single optimal solution but only best compromises. We defined optimality in terms of minimization of a multi-objective optimization problem and used the Pareto dominance criterion to compare different design variants.

On the specification layer we used the SAML specification language. To express whole families of specifications in one model we extended the language with the new keyword *param*. The new version of the language is named *optimizable safety analysis modeling language* (OSAML). Parameters of

a specification family can be the variation of the values of constants or the specification of alternative modules.

On the analysis layer we rely on sound model transformations from SAML into the specification languages of NuSMV and PRISM [5]. This allows for using PCTL properties as objectives for the optimization process. Our software architecture allows the implementation of further Objectives in the Java programming language.

We evaluated our approach on a case study from model based safety analysis. Our optimization along costs and hazard probability identified 10 Pareto optimal candidates. Despite the rather small case study in this paper we also implemented a evolutionary algorithm to efficiently cover a large search space in more complex models.

Particularly motivated by the fact that probabilistic model checking is time consuming, we are going to further investigate optimization algorithms. Brute force enumeration, as used for the case study in this paper, is only feasible for very small search spaces. Nevertheless we believe that often a reduction of the search space is possible by exploiting meta information from the application domain of the optimization problem. Very often the parameters represent physical quantities of components in the system. Due to tolerances these cannot be manufactured with an arbitrary precision. Thus it is not necessary to operate with a finely grained search space, if this does not map to the parameters of the real system. A possible approach could be the specification of logarithmically scaled parameters (similar to the failure probability parameter of the monitor module in the case study).

A second approach for the reduction of model checking time is to integrate stochastic estimation algorithms into the optimization procedure. We are currently busy with the evaluation of artificial neural network and simulation based estimation of PCTL formulas. For artificial neural networks the basic idea (as laid out in [4]) is the usage of previous function evaluations to train the estimation network. It can then be used to estimate further function evaluations. Instead creating an independent estimation model we are evaluating the usage of the verification models in terms of Monte Carlo simulation as estimator. We expect a relevant speed-up due to the usage of estimation techniques.

ACKNOWLEDGMENTS

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

Simon Struck is sponsored by the German Research Foundation (DFG) within the ProMoSA project.

REFERENCES

[1] P. A. Abdulla, J. Deneux, G. Stalmarck, H. Agren, and O. Åkerlund, "Designing safe, reliable systems using SCADE," in *Proceedings of the* 1^{st} *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 04)*. Springer, 2004.

[2] M. Bozzano and A. Villafiorita, "Improving system reliability via model checking: the FSAP/NuSMV-SA safety analysis platform," in *Proceedings of the* $22^{n.d}$ *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2003)*. Springer, 2003, pp. 49–62.

[3] K. Miettinen, "Some methods for nonlinear multi-objective optimization," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, E. Zitzler, L. Thiele, K. Deb, C. Coello Coello, and D. Corne, Eds. Springer Berlin / Heidelberg, 2001, vol. 1993, pp. 1–20.

[4] M. Güdemann and F. Ortmeier, "Model-Based Multi-Objective Safety Optimization," in *Proceedings of the* 30^{th} *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2011)*. Springer LNCS, 2011, pp. 423–436.

[5] M. Güdemann and F. Ortmeier, "A framework for qualitative and quantitative model-based safety analysis," in *Proceedings of the* 12^{th} *High Assurance System Engineering Symposium (HASE 2010)*, 2010, pp. 132–141.

[6] M. Güdemann and F. Ortmeier, "Towards Model-driven Safety Analysis," in *Proceedings of the 3rd international Workshop on Dependable Control of Discrete Systems (DCDS 2011)*. IEEE, 2011, pp. 53–58. [Online]. Available: http://www.dcds11.uni-saarland.de/

[7] G. Norman, D. Parker, and M. Kwiatkowska. (accessed: June 2010) The PRISM language - semantics. [Online]. Available: http://www.prismmodelchecker.org/doc/semantics.pdf

[8] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga, "Model checking discounted temporal properties," *Theoretical Computer Science*, vol. 345, pp. 139–170, 2005.

[9] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 2000.

[10] M. Güdemann, F. Ortmeier, and W. Reif, "Computing ordered minimal critical sets," in *Proceedings of the* 7^{th} *Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2008)*, E. Schnieder and G. Tarnai, Eds., 2008.

[11] F. Ortmeier, M. Güdemann, and W. Reif, "Formal failure models," in *Proceedings of the* 1^{st} *IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2007)*. Elsevier, 2007.

[12] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, pp. 102–111, 1994.

[13] M. Güdemann and F. Ortmeier, "Probabilistic model-based safety analysis," in *Proceedings of the* 8^{th} *Workshop on Quantitative Aspects of Programming Languages (QAPL 2010)*. EPTCS, 2010, pp. 114–128.

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

[15] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Transaction on Evolutionary Computation*, pp. 181–197, 2002.

[16] M. Walker, L. Bottaci, and Y. Papadopoulos, "Compositional Temporal Fault Tree Analysis," in *Proceedings of the* 26^{th} *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2007)*, 2007.

[17] J. Branke, K. Deb, H. Dierolf, and M. Osswald, "Finding knees in multi-objective optimization," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. Lecture Notes in Computer Science, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervs, J. A. Bullinaria, J. Rowe, P. Tino, A. Kabn, and H.-P. Schwefel, Eds. Springer Berlin / Heidelberg, 2004, vol. 3242, pp. 722–731.

[18] F. Ortmeier and W. Reif, "Safety optimization: A combination of fault tree analysis and optimization techniques," in *Proceedings of the Conference on Dependable Systems and Networks (DSN 2004)*. Florence: IEEE Computer Society, 2004.

[19] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske, "Architecture-driven reliability and energy optimization for complex embedded systems," *Research into Practice–Reality and Gaps*, pp. 52–67, 2010.

[20] E. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "Param: A model checker for parametric markov models," in *Computer Aided Verification*. Springer, 2010, pp. 660–664.

[21] A. Hegedus, A. Horváth, I. Ráth, and D. Varró, "A model-driven framework for guided design space exploration," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*. IEEE, 2011, pp. 173–182.

[22] D. Knorreck, L. Apvrille, and R. Pacalet, "Formal system-level design space exploration," in *New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on*. IEEE, 2010, pp. 1–8.