Model-Based Optimization of Automotive E/E-Architectures*

Stefan Kugele
Institut für Informatik
Technische Universität München
Garching b. München, Germany
kugele@in.tum.de

Gheorghe Pucea
Institut für Informatik
Technische Universität München
Garching b. München, Germany
george.pucea@tum.de

ABSTRACT

In this paper we present a generic framework to enable constraint-based automotive E/E-architecture optimization using a domain-specific language. The quality of today's automotive E/E-architectures is highly influenced by the mapping of software to executing hardware components: the so-called *deployment problem*. First, we introduce a holistic architectural model facilitating a seamless model-based development from requirements management to deployment, which is the focus of this work. Second, we introduce our domain-specific constraint and optimization language **AAOL** (*Automotive Architecture Optimization Language*) capable to express a wide range of deployment-relevant problems. Third, we present a generic, i. e., solver-independent framework currently supporting multi-objective evolutionary algorithms (MOEA). We investigate the feasibility of the approach by dint of a case study taken from the literature.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Languages (e.g., description, interconnection, definition)*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*

General Terms

Design, Languages

Keywords

Model-based optimization, domain-specific languages, constraint satisfaction problem, automotive E/E-architecture

*This work is part of the Software Platform Embedded Systems (SPES_XT) project. SPES_XT is supported by the German Federal Ministry for Education and Research (BMBF) by 01IS12005.

CSTVA'14, May 31, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2847-0/14/05...$15.00
http://dx.doi.org/10.1145/2593735.2593739

1. INTRODUCTION

During the last decades, more and more software-controlled functions were introduced in automobiles. These functions are realizing a multitude of different tasks ranging from highly safety-critical and real-time control tasks, as for instance the airbag control system or advanced driver assistance systems, to comfort and infotainment systems. Later ones are less safety-critical but pose different kinds of requirements with respect to non-functional requirements, for instance usability.

Today's automotive E/E-architectures (electric/electronic) can be best characterized as historical grown networks of different bus systems which connect electronic control units (ECUs). We observe ad-hoc solutions rather than well-engineered future-prone architectures. As in the future, the number of software-controlled functions in cars will grow further, the challenge to grasp the enormous complexity involved in the development of such systems will dominate the engineer's daily work. Hence, new ways of system design had to be applied: (i) Model-driven development (MDD) helped to reduce the complexity apparent to the developer and (ii) the introduction of different levels of abstraction helped to structure the model as well as the development process into different stages. The approach of this work fits well in the model-centric way of thinking and conceptually extends previous work done by Haberl et al. [28,29] and Broy et al. [12].

Engineers have to decide on which of the available ECUs a function should be realized. This is similar to the question where to execute software components as we will see later in this paper. Traditionally, this question was answered with a lot of experience and gut feeling. Of course, on the one hand the influence of engineering experience should not be underestimated, on the other hand, however, these solutions are—if at all—optimal by accident.

During the development of software-intensive automotive systems, optimization steps are involved at different stages. We will exemplary point to some of them and go into depth at the already mentioned software to hardware allocation step, which is also referred to as deployment or partitioning in the literature. The aim is to develop a domain-specific language (DSL) capable to express likewise optimization objectives and constrains. Note, the aim is not to develop a general-purpose optimization and constraint language (cf. Section 2 for related work) but a specifically tailored language for the automotive domain. However, we think that it could easily be adopted—if necessary at all—to the special needs of for instance the avionics domain because both domains share to a large extend similar characteristics with respect to safety, real-time, and cost constraints for example. A central requirement for such a DSL is its appropriateness for the specific needs and to reach a high level of user satisfaction and thus acceptance. Typically, during system development—no matter of which domain—engineers are faced many requirements that are oftentimes conflicting, i. e., usually the goal to develop a car with a reduced carbon footprint conflicts with the goal of a car with maximized power. Therefore, the goal in such situations is to find the best compromise (trade-off) amongst the different goals (design criteria) [18].

This work provides the following contributions:

(i) We present a generic and flexible framework for model-based optimization of automotive E/E-architectures.

(ii) In this regard we suggest a domain-specific optimization and constraint language called **AAOL**.

(iii) A first implementation makes use of multi-objective evolutionary algorithms.

We developed a plugin for the Eclipse [2] platform that implements our methodology for architectural models and optimization objectives and constraints written in **AAOL**.

Outline.

The remainder of this paper is structured as follows: Section 2 refers this work to the state-of-the-art and state of practice. Next in Section 3 we introduce the proposed DSL embedded in a seamless development methodology. In Section 4 we show how the DSL is realized. Section 5 evaluates the presented approach by dint of a case study from the automotive domain. Finally, Section 6 concludes this article and sketches future research directions.

2. RELATED WORK

This work is basically related to two research directions: (i) *constraint programming* or constraint satisfaction problems (CSP) and (ii) *(multi-objective) optimization* of automotive E/E architectures. Hence, we will relate our work to both directions separately.

Very similar to a mathematical way of writing constraints and combinatorial problems is AMPL—A Mathematical Programming Language by Fourer et al. [21]. This high-level programming language is used to formalize optimization problems. Instead of solving the problems directly, AMPL calls external solvers for linear or nonlinear problems with discrete or continuous variables. For experts in the operations research (OR) domain, the fact that AMPL follows a mathematical notation could be beneficial [21]. However, this cannot be expected from engineers or system architects, who are our intended users. There are many other specification languages proposed in related work such as DEPICT [6], which is a high-level formal language for modeling constraint satisfaction problems, or ESSENCE [23], which is a constraint language for specifying combinatorial problems. ESSENCE was designed to write rigorous problem specifications in a combination of natural language and discrete mathematics such as those catalogued by Garey and Johnson [24], or Z [44], or NP-SPEC [13] to compile problem specifications into SAT. Chenouard et al. [14] propose to model constraint in a graphical fashion and then translate the model into different formats used by common solvers. We take a similar approach to [30] of letting the user specify *what* they want and not *how* to achieve it. In this paper a synthesis of Cyper-Phisical Architectural Models is performed, by specifying several real-time constraints. This synthesis is performed using a new approach called Integer linear programming Modulo Theories. However, the focus is more on specifying different kind of constraints from the aerospace domain and not on multi-objective optimization.

During the development of automobiles, historically optimization steps were performed at different stages and by various stakeholders. Questions concerning economic (e. g. cost models [8–10, 41]) and mechanical/electrical (e. g. car body optimization [43], hybrid powertrain [19], and EMC [11, 22, 42]) engineering oftentimes arose. In recent years, however, researchers were also concerned with reliability of automotive E/E architectures and an automatic process of mapping software components onto available ECUs, which is also referred to as deployment or allocation in the literature. Grunske et al. [27] give an outline of architecture-based methods for optimizing dependability of software-intensive systems. Similar to the presented approach, Grunske [26] propose to use evolutionary algorithms and multi-objective optimization strategies to find good architectural design alternatives with the tool ArcheOpterix [7]. Kugele et al. [32] use integer linear programming (ILP) to optimize deployment with respect to non-functional requirements in combination with an SMT-based (SAT modulo theories) scheduling scheme within the COLA tool-chain [28, 29]. Moreover, Meedeniyaa et al. [36] use a Genetic Algorithm (GA) in a reliability-driven deployment optimization of embedded systems. Also a GA is used by Kumar et al. [34] to perform a multilevel redundancy allocation. Streichert et al. [45] apply multi-objective evolutionary algorithms for topology optimization in networked embedded systems. This work is extended by Lukasiewycz et al. [35] to allow for concurrent topology and routing optimization in automotive networks. Their approach is based on *SAT decoding* that combines a Pseudo-Boolean (PB) solver and *Multi-Objective Evolutionary Algorithms*. The latter one is also applied in this paper. A further improvement is presented by Glass et al. [25]. They propose a new algorithm for multi-objective routing with a genetic encoding independent of the underlying network topology. Czarnecki and Olaechea suggest an optimization framework for Software Product line development [37]. The paper presents ClaferMoo, an optimization framework that is able to perform multi-objective optimization in context of Variability-rich software for Software Product line development. ClaferMoo uses an iterative algorithm that can list the Pareto optimal points during computation. The used modeling language called Clafer, allows specification of objectives and constraints. However, Clafer is a more abstract modeling language compared to **AAOL** which is specifically targeted for the automotive domain.

All the mentioned approaches, let it the CSP specification languages or the automotive E/E architecture optimization methods, have in common that they have their specific advantages and also disadvantages. The presented approach, however, tries to seamlessly integrate a simple but yet powerful DSL to be applied in the development of automotive E/E architectures—currently for software to hardware deployment—with multi-objective evolutionary algorithms to concurrently optimize different (conflictive) design objectives while satisfying a set of constraints.

3. APPROACH

This paper tries to establish the notion of *model-based optimization* for the development of embedded systems, in particular automotive E/E architectures. The notion is inspired by model-based design or engineering and assumes not only the use of design models (architectural models, cf. Section 3.1), but also the use of *optimization models* (cf. Section 3.2), allowing a seamless development process taking optimization goals into account. This combination of architectural model with its viewpoints and the newly introduced optimization model with its optimization viewpoint facilitates:

(i) *Reuse* optimization objectives and constraint models in future projects from a library.

(ii) Specify optimization goals *regardless* of the tool support.

(iii) Documentation and *traceability* of design decisions in the same model.

In the following, some more details about the mentioned models are given.

3.1 Architectural Model

To grasp the enormous complexity apparent to developers or architects of automotive E/E-architectures or similar complex systems in general, model-driven development was one methodology to push back the limits of practicable manageable system sizes. The ideas to structure and modularize systems are not new, but go back to seminal works by Dijkstra [17] and Parnas [39] in 1972. Decoupling, structuring, and hierarchization are some examples of simplifications in system design. But designers of complex, reliable, safety-critical, and networked embedded (automotive) systems still face complex problems. The task remains complex even though structuring techniques are used since the complexity is problem-inherent. However, what these methods *can* provide is to tame the complexity such that developers are able to manage the design process.

The framework of the presented approach proposes to consider the system design under different viewpoints, namely the *Requirements*, the *Functional*, the *Logical*, and the *Technical Viewpoint*. The last three mentioned are also used in the approaches proposed by Kugele et al. [33], Broy et al. [12], and all of them in the SPES 2020 (Software Platform Embedded Systems) [40] methodology.

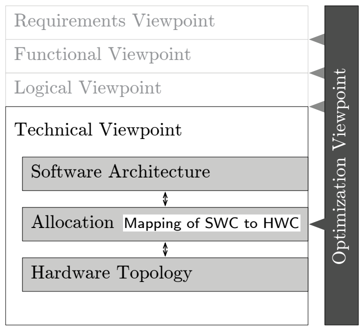


Figure 1: "Classical" viewpoints together with the orthogonal *Optimization Viewpoint*.

In the following, we give a brief description of the different viewpoints in so far as necessary for understanding the whole approach. However, we will concentrate on the *Technical Viewpoint*. Moreover, in addition to the mentioned approaches, we propose to add a so-called *Optimization Viewpoint*, which facilitates the specification of both *objective functions* and *constraints* to be considered at the respective point in the overall development process.

Figure 1 shows the mentioned viewpoints. In principle at each transition from a higher to the next lower level, whereas the *Requirements Viewpoint* is considered as the highest and the *Technical Viewpoint* as the lowest, respectively, optimizations can be performed. As indicated in the figure, we focus our attention in this work on the *Technical Viewpoint* and therefore on optimizations within this viewpoint. Nonetheless, also the other viewpoints are briefly explained next. For further details please refer to the mentioned literature.

Requirements Viewpoint.

Goals, derived and refined requirements are specified and managed within this viewpoint.

Functional Viewpoint.

All software-controlled functions of a system are defined and structured. Moreover, possible interactions—so-called *feature interactions*—are modeled here.

Logical Viewpoint.

The logical cause-effect relationships are modeled using for instance box and arrow diagrams and state charts. This viewpoint is still hardware-independent. The cause-effect from data sources, further processing, to data sinks is modeled.

Technical Viewpoint.

Both the software architecture and the targeted hardware topology are modeled within this viewpoint. The allocation realizes a mapping relation $\mapsto \subseteq S \times H$, where S is the set of software components of a software architecture and H is the set of hardware components of a given hardware topology that can be an allocation target. In **AAOL** $\mapsto$ is written as `->`. Basically the software architecture consists of a set of software components annotated with technical details like worst-case execution time (WCET), period, RAM, ROM etc. Moreover for each software component $s \in S$ a precedence relation is given, i. e., a set of other software components s depends on. We write $s' \prec s$ if s depends on the results of the execution of s'. The hardware topology consists of hardware components and hardware connections. Hardware components are electronic control units (ECUs), sensors, actuators, and gateways. Hardware connections are typically bus systems like CAN, LIN, MOST, or FleyRay. Hardware components are connected via buses. In **AAOL**, we write `SoftwareComponent s <> LIN l` when a software component s produces traffic on the LIN bus l, for example.

3.2 Optimization Model

As pointed out in Section 2, there are different optimization languages, mostly very mathematical and thus for non-mathematician hard to understand. Derived from this lack of simplicity and usability, we propose a DSL to express likewise optimization objectives and constraints. But before going into details, an easy example should motivate this need.

Example 1. *Assume we want to minimize* cost *and* weight *of an E/E-architecture simultaneously respecting that the resource capacities (CPU, ROM) are not exceeded when allocating software to hardware components. For this multi-objective optimization problem we have the following exemplary objectives and constraints:*

$$\min\left(f_1(\mathbf{x}), f_2(\mathbf{x})\right) \tag{1}$$

where $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ denote the functions (cost, weight) to be minimized and $\mathbf{x}$ denotes the set of decision variables. Such that the following constraints are satisfied:

$$\sum_{s\in\{s'\in S|s'\mapsto h_1\}} s^{ROM} \leq h_1^{ROM} \tag{2}$$

$$\sum_{s\in\{s'\in S|s'\mapsto h_2\}} s^{ROM} \leq h_2^{ROM} \tag{3}$$

$$\vdots$$

$$\sum_{s\in\{s'\in S|s'\mapsto h_m\}} s^{ROM} \leq h_m^{ROM} \tag{4}$$

where $h_1, h_2, \ldots h_m \in H$ and $m = |H|$ is the number of hardware components. Similarly, constraints for any other kind of restricted resource (e. g. CPU) have to be stated. In **AAOL** *this problem can be stated as follows:*

```
objectives:
  min Weight:
     forall (HardwareComponent h)
     where SoftwareComponent s -> h:
     sum(h.weight);
  min Cost:
     forall (HardwareComponent h)
     where SoftwareComponent s -> h:
     sum(h.cost);
constraints:
  const ROM:
     forall (ECU e)
     where SoftwareComponent s -> e:
     sum(s.rom) <= e.rom;
```

3.2.1 Objectives

In **AAOL** objectives are specified in order to express what aspects of a system (e. g. E/E-architecture) shall be optimized. Both *minimization* and *maximization* of system aspects are supported (see the example above).

3.2.2 Constraints

So far, we support the following constraints. Detailed examples using **AAOL** are given in the next Section 3.3.

fix-*constraints* are used to state that several design decisions are already made, i. e., they are *fixed*. For example when a certain software component shall be executed on a defined ECU.

```
const Fix:
  for(SoftwareComponent SWC_1, SWC_2):
  SWC_1 -> ECU_1, SWC_2 -> ECU_2;
```

matching-*constraints* are generally used to check if the hardware components on which software components are deployed have some matching property. For example, check for all software components whether the ECUs on which they are deployed are from the same vendor.

```
constraint Vendor:
  forall (SoftwareComponent s)
  where s -> ECU e:
  s.vendor == e.vendor;
```

capacity-*constraints* are usually used to restrict the amount of an available resource. Examples are memory, CPU-time, or network bandwidth.

```
const ROM:
  forall (ECU e)
  where SoftwareComponent s -> e:
  sum(s.rom) <= e.rom;
```

The support for network bandwidth requires more sophisticated considerations as discussed in Section 4.2.2.

compatibility-*constraints* are used to express compatibility relations. In the automotive context, for instance the ASIL classification (Automotive Safety Integrity Level) defined in the ISO 26262 [31] standard defines a hierarchy of safety levels.

```
const Asil:
  forall (SoftwareComponent s)
  where s -> ECU e:
  s.asil ~ e.asil);
```

3.2.3 Orders

We give our users the possibility to define custom orders. Custom ordering is needed because if our optimization model has multiple solutions we want to rank the solutions based on custom defined importance. Also if a compatibility constraint is defined, we have to know the ranking of all possible values of a property for being able to asses the compatibility of a deployment. This functionality is provided in the **orders** section of the optimization language.

Ranking of Solutions.

For our optimization model multiple solutions might be generated. Since it is unfeasible to present a set of random solutions, we are trying to help our users to pick the best solution for them.

We give the user the possibility to provide a custom ordering of the defined objectives. Based on the custom ordering we rank the solutions and present them to the user. Below we can see an example of such a custom ordering. We have defined three minimization objectives *Cost*, *Height*, and *Weight* and additionally defined, in the orders section, a custom ordering of the objectives.

```
objectives:
  min Cost: ...
  min Height: ...
  min Weight: ...
orders:
  order objective: [Cost > Weight]
```

Based on the objective ordering, we know that solutions with lower cost should be presented first, afterwards if the cost is the same for two solutions the weight objective is used as a further ranking field.

Expressing Compatibility Relations.

An example of compatibility relation was given in Section 3.2.2, where the ASIL level of all software components deployed on a hardware component should be compatible. In order to derive this compatibility relation we have to know the ordering of the ASIL levels.

```
orders:
  order ASIL: [ QM < A, A < B, B < C, C < D ];
```

The above example defines the critical order of the ASIL level. In case of an ASIL compatibility constraint, the generated deployments are checked for ASIL compatibility. This means that a software component with ASIL D can only be deployed on hardware components that have also level D, because from the defined ordering ASIL D is the most critical. Deploying a software component with ASIL B on a hardware component with ASIL A is not possible whereas deploying the same software component on a hardware component with ASIL D is possible. For the first case, an automatic ASIL decomposition is planed as future work.

For every compatibility constraint for which a custom property is checked for compatibility, we have to define a custom ordering as in the example above.

3.3 Syntax and Semantics of AAOL

In this section, we define both the syntax and the semantics of the *Automotive Architecture Optimization Language* (**AAOL**) by dint of a running example. **AAOL** facilitates the specification of constraints as well as the definition of optimization goals of E/E designs in a model-based development setting.

3.3.1 Syntax

The syntax (grammar) of **AAOL** is defined in Figure 2. The basic concepts of **AAOL** are: (i) *objectives*, (ii) *constraints*, and (iii) *orders*. For a longer example, please refer to the case study explained in Section 5.

3.3.2 Semantics

After having seen some examples and having defined the syntax of **AAOL**, the next step is to detail on its semantics. Therefore, we give the denotation of each language construct.

Structuring Commands.

The **AAOL** commands **objectives**, **constraints**, and **orders** are used to indicate the start of the respective sections within the textual model.

The **import** *Command.*

The **import** command is used to reference the model to use for optimization.

The **use** *Command.*

use S with H where **S** is the set of software architectures contained in the used model and **H** is the set of hardware topologies, respectively. For all combinations of software architectures and hardware topologies an optimal deployment is computed, i. e., the number of possible combinations is $|\mathbf{S}| \cdot |\mathbf{H}|$.

⟨model⟩	⊨	⟨import⟩⟨use⟩⟨lobj⟩[⟨lconst⟩][⟨lorders⟩]
⟨import⟩	⊨	`import` ⟨fname⟩;
⟨use⟩	⊨	`use {` ⟨swarch⟩ `} with {` ⟨hwtop⟩ `} ;`
⟨lobj⟩	⊨	`objectives :` ⟨obj⟩ +
⟨lconst⟩	⊨	`constraints:` ⟨const⟩ +
⟨lorders⟩	⊨	`orders:` ⟨order⟩ +
⟨obj⟩	⊨	⟨objtype⟩ ⟨identifier⟩ `:` ⟨objbody⟩
⟨objtype⟩	⊨	`minimize | min | maximize | max`
⟨objbody⟩	⊨	⟨forall⟩ ⟨where⟩ `:` ⟨mathexpression⟩
⟨const⟩	⊨	(`const | constraint`) ⟨identifier⟩ `:` ⟨constbody⟩
⟨constbody⟩	⊨	(⟨forall⟩ | ⟨for⟩) ⟨constwhere⟩ [`:` ⟨mathineq⟩]
⟨constwhere⟩	⊨	(`where | :`) ⟨wherecriteria⟩ +
⟨order⟩	⊨	`order` (⟨identifier⟩ | `objective`) `'['`⟨orderRelation⟩ + `']'`
⟨orderRelation⟩	⊨	⟨identifier⟩ ⟨reloperator⟩ ⟨identifier⟩ `,`?
⟨reloperator⟩	⊨	`< | > | <= | >=`
⟨for⟩	⊨	`for` (⟨swcdecl⟩ | ⟨hwcdecl⟩)
⟨forall⟩	⊨	`forall` (⟨swcdecl⟩ | ⟨hwcdecl⟩)
⟨swarch⟩	⊨	(⟨identifier⟩`,`?) +
⟨hwtop⟩	⊨	(⟨identifier⟩`,`?) +
⟨where⟩	⊨	`where` ⟨wherecriteria⟩ +
⟨wherecriteria⟩	⊨	(⟨swcdecl⟩ | ⟨swcvar⟩) (`-> | <>`) (⟨hwcdecl⟩ | ⟨hwcvar⟩)
⟨swcdecl⟩	⊨	`SoftwareComponent` ((⟨identifier⟩ | ⟨swcinswarch⟩) `,`?)+
⟨swcinswarch⟩	⊨	⟨identifier⟩ `(.)` ⟨identifier⟩
⟨swcvar⟩	⊨	⟨identifier⟩
⟨hwcdecl⟩	⊨	(`HardwareComponent | HardwareConnection | ECU | Actuator | Sensor | CAN | LIN | FlexRay | MOST | Ethernet`) ((⟨identifier⟩ | ⟨hwcinhwtop⟩) `,`?)+
⟨hwcinhwtop⟩	⊨	⟨identifier⟩ `(.)` ⟨identifier⟩
⟨hwcvar⟩	⊨	⟨identifier⟩
⟨variabilewithprop⟩	⊨	⟨identifier⟩ `(.)` ⟨identifier⟩
⟨double⟩	⊨	`('0'..'9')+ '.' ('0'..'9')+`
⟨fname⟩	⊨	⟨osfilename⟩
⟨mathexpression⟩	⊨	⟨expression⟩ ⟨mathop⟩ ⟨expression⟩
⟨expression⟩	⊨	⟨aggregatefunction⟩ | ⟨double⟩ | ⟨variabilewithprop⟩
⟨aggregatefunction⟩	⊨	⟨sum⟩
⟨sum⟩	⊨	`sum` (⟨mathexpression⟩ | ⟨variabilewithprop⟩)
⟨mathop⟩	⊨	`+ | - | + | /`
⟨mathineq⟩	⊨	⟨mathexpression⟩ ⟨ineqop⟩ ⟨mathexpression⟩
⟨ineqop⟩	⊨	`< | > | <= | >= | == | ~`

Figure 2: Syntax of AAOL

Specification of Objectives.

Within the **objectives** section, a set of minimization or maximization objectives is defined as follows: first it is specified whether an optimization function is to be minimized or to be maximized using the **minimize** (**min**) or **maximize** (**max**) keywords followed by an objective name. Then we have a **forall** (M) quantification over a set M of software or hardware components, which is restricted by a predicate in the **where** clause, yielding M^r. Currently only simple predicates are supported, but it will be extended in the future (cf. also Section 6):

(i) *Deployment predicate*:
There exists a software component $s \in S$ that is deployed onto a hardware component $h \in H$.
$\exists s \in S \; : \; (s, h) \in \mapsto$: **where** `SWC s -> h`.

(ii) *Bus communication predicate*:
There exists a software component $s \in S$ that sends data to another software component d in a way that bus communication over e. g. a LIN bus l is needed.
$\exists s, d \in S \; : \; (s, h_1), (d, h_2) \in \mapsto$ and $s \prec d$ and $h_1 \neq h_2$: **where** `s <> LIN l`

As one can see, the **where** clause in objectives is an existential quantification. Finally, a mathematical expression is evaluated over the restricted set M^r. For this purpose, **AAOL** supports so-called *aggregation functions*. These functions operate on sets—on M^r, to be precise. Such a function is for instance **sum**(M^r) intuitively computing $\sum_{i \in M^r} i$. Of course, standard mathematical operators and (in)equalities are supported as well and other aggregation functions can be easily integrated.

Specification of Constraints.

Within the **constraints** section, a set of constraints is defined as follows: a constraint begins with a keyword of the same name, **constraint**, followed by its name. Then it depends on whether we want to specify constraints for concrete instances of software or hardware components, then we use **for** `(.)`, or for all software or hardware components, then we use **forall** `(.)`. Again, analogous to the objective specification, but now an *optional* **where** predicate follows. Finally, a mathematical expression follows similarly to the objectives. However, there is a difference: the aggregate functions operate over the restricted set M^r in the objective case, whereas in the constraint case, they are applied to those components restricted in the **where** clause. The **forall** command is internally unrolled to a set of single constraints (cf. Section 4.2.1).

Specification of Orders.

Within the **orders** section, a set of relations is defined as follows: an order begins with the keyword **order**. There is a special order—the **objective order**—, which is used to define an order on the objectives. Deployment results are ranked according this order (cf. Section 3.2.3). For non-objective orders, e. g. ASIL classification, an order is assumed, whereas transitive elements are derived automatically and thus do not have to be specified. Moreover, these orders have a name.

3.4 Abstract Optimization Framework

We have build an abstract framework that supports the realization and evaluation of our problem-specific optimization model by using different solver implementations. Flexibility was one of the main factors that drove the design of the framework. Figure 3 provides a brief overview of our framework. The abstract optimization framework is viewed from three different perspectives marked by the rectangles in the lower right of the figure.

Problem-specific Perspective.

First we have to model an optimization problem, which is denoted by the *ProblemOptimizationModel 1..k*. An example of such a problem is one of the scopes of this paper, the definition of an abstract optimization problem for deploying software components to hardware components by optimizing a set of objectives at the same time. Each problem definition has its own solution model, marked in the Figure 3 by the *ProblemSolution* interface.

Solver-specific Perspective.

Second our model is easily adaptable to different solver implementations marked in the figure by the classes *Solver$_1$* to *Solver$_s$*. The optimization model has to be solved later, so that a solution is generated. There are several different solvers that can find solutions for various optimization problems. Dealing with solver-specific interfaces in a clean and separate manner was an important requirement for us. We have used the *Visitor* design pattern, to allow a clear and separate way to translate our optimization model to solver-specific interfaces. The solver-specific visitors are represented by the classes *Solver$_1$Visitor* to *Solver$_s$Visitor*, which both call the respective solver instances *Solver$_1$* to *Solver$_s$*.

Problem-specific and Solver-specific Perspective.

The concrete classes in Figure 3, *Solver$_1$ProblemSolution* to *Solver$_s$ProblemSolution* are first problem-specific because each optimization model has its own representation of a solution. But the solutions are also solver-specific because each solver represents solutions in its own way. We needed a clear abstraction to model separate needs of this two perspectives in the solution part.

The multi-objective *ProblemOptimizationModel is a OptimizationModel* and contains objectives, modeled by the *Objective* class, constraints modeled by the *Constraint* class, and orders modeled by the *Order* class, which are inherited from the abstract *OptimizationModel*.

The Abstract Optimization Framework is fully adaptable to different needs by providing different abstraction points. It is totally decoupled from the **AAOL** language constructs, letting the optimization model vary independently from the actual **AAOL** syntax. Separate problem-specific optimization models are supported by our framework. To add a new solver to our framework only means to define a new visitor-specific class, which translates the abstract optimization model into the solver-specific interface and also to define a solver-specific and problem-specific solution to view the results.

General Flow in the Framework.

The **AAOL** syntax is parsed in an abstract syntax tree (AST) which is later translated into the problem-specific Optimization Model. All objectives, constraints, and orders are translated into the abstract class hierarchy shown in Figure 3, which is then translated by the solver-specific visitors into solver-specific interfaces. The solver tries to find solutions, which are then returned back in problem-specific solution objects. The UI component is invoked, so that the results are displayed to the users.

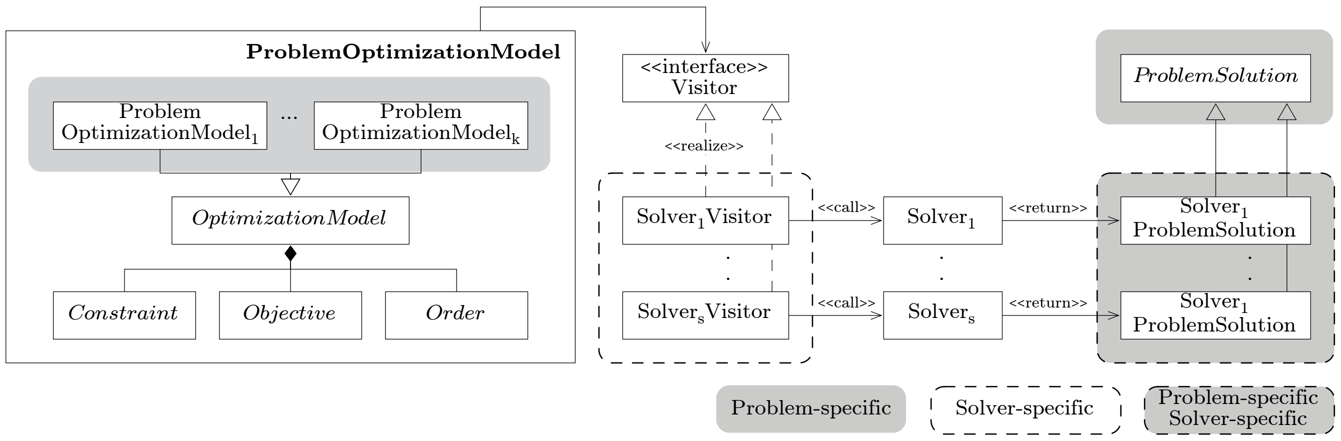


Figure 3: Abstract framework

4. REALIZATION

We have implemented the abstract optimization framework mentioned in Section 3.4 in Java. The whole project is realized as an Eclipse [2] plugin. We are using Xtext [5], an Eclipse-based framework for developing DSL languages that provides support for building the **AAOL** language artifacts.

Our solver implementation is based on a multi-objective evolutionary algorithm framework.

4.1 Concrete Framework Instantiation

Our approach is evaluated by instantiating the abstract optimization framework defined in Section 3.4, with the problem of deploying software components onto hardware components by optimizing various parameters.

We defined a concrete *DeploymentOptimizationModel* class which is a problem specific optimization model. The *DeploymentOptimizationModel* class contains several objectives, constraints, orders, and it also contains the software components and hardware components, for which a deployment will be generated.

An abstract class *DeploymentSolution* was defined to model the actual deployment. MOEA mentioned in Section 4.2 is used as a solver implementation. An *MOEAVisitor* is defined to translate the *DeploymentOptimizationModel* in the MOEA-specific interface. Because MOEA is generating the deployment in a specific way a specific implementation of the *DeploymentSolution* is defined, *MOEADeploymentSolution* class.

4.2 Multi-objective Optimization

One of the most powerful features of our **AAOL** language is the possibility to specify multi-objective optimization problems. Objectives can be self-contradictory because depending on the hardware topology and the software architecture in use, minimizing or maximizing two objectives at the same time, is not always possible. In the context of multi-objective optimization an optimal solution is rather different than in the context of single optimization.

Multi-objective Example.

To illustrate the challenges of multi-objective optimization we present a simple example. Suppose we want to deploy two software components SWC1 and SWC2 on hardware components. As a hardware topology we have two ECUs available, ECU1 and ECU2, properties like cost and weight are shown in Table 1.

Table 1: ECU properties

	ECU1	ECU2
Cost	15 EUR	20 EUR
Weight	850 g	750 g

Assumptions: for the sake of simplicity we assume that each hardware component has enough resources to satisfy the ROM/RAM needs of both software components at the same time. The software components can run independent of each other, they do not need to communicate. We also ignore the underlying hardware connections because we only want to emphasize the problem of multi-objective optimization. Two objectives are defined, one for minimizing the cost and the other objective for minimizing the weight. We want to minimize the cost while at the same time minimizing also the weight. Any constraints are omitted because they are not relevant for this example.

Our optimization problem, is composed then of two minimization objectives and has as an input the two software components SWC1 and SWC2 which have to be scheduled on the above mentioned hardware components. It is easy to see the two optimal deployments result, as a solution to our optimization problem. In the first deployment both software components SWC1 and SWC2 can be scheduled on ECU1, for a total cost of 15 EUR and weight 850 g. The second solution schedules both software components on ECU2 for a total cost of 20 EUR and weight 750 g.

We can see that the first solution is optimal from the point of view of cost and the second one is optimal from the point of view of weight. As opposed to simple optimization we cannot find a global optimal solution. If we choose the solution 2 instead of solution 1 we are minimizing the weight, but the cost is getting higher. The same happens if we choose solution 1, we improve the cost but the weight gets higher, we cannot improve one objective without worsening the other. These two solutions are called non-dominated or *Pareto optimal solutions.* [38]

The above deployments show that in the context of multi-objective optimization a set of solutions can be generated, which are all acceptable solutions from the optimization point of view.

MOEA Multi-objective Framework.

Starting with the optimization model defined in Section 3.2, we are using a genetic algorithm approach to generate all possible non-dominated deployments.

We have chosen MOEA [4] as our solver implementation. MOEA (multi-objective evolutionary algorithms) is an open source Java library that supports multi-objective optimization by the use of evolutionary algorithms. The framework implements several state-of-the-art genetic algorithms and provides an extensible interface for using them.

The **AAOL** language is parsed in our *DeploymentOptimizationModel*, after this the abstract objectives and constraints are transformed into the MOEA framework, with the help of the *MOEAVisitor* class implementation. A customized genetic algorithm based on NSGA-II [16] is called which generates one or multiple deployments. The algorithm starts with an initial population of 1000 deployments, these deployments are then gradually selected and mutated until a set or a single non-dominated solutions is generated. We have used a custom mutation operator to preserve defined fix-*constraint*s. These deployments are represented by the *MOEADeploymentSolution* objects. However, there might be a large set of solutions. Based on the user defined importance of objectives, see Section 3.2.3, the solver solutions are sorted according to the user-defined importance.

4.2.1 *Unrolling of Objectives and Constraints*

During the translation into solver-specific interfaces the abstract objectives and constraints of the optimization model are unrolled in more fine grained constrains and objectives.

Objectives Unrolling.

Objectives are functions that we want to minimize or maximize. Each defined objective in the **objectives** section is unrolled in exactly one function in the solver interface. After we generate a deployment, we also want the exact value of the minimized/maximized objective, because of the 1-to-1 mapping between defined objectives and unrolled objectives, the solver can generate out of the box the minimized/maximized value for the objective.

Constraints Unrolling.

In case of constraints that are defined using **for**, a single constraint instance is translated in the solver specific interface.

Constraint defined using **forall** will be unrolled in multiple constraints in the specific solver interface.

```
const RAM:
  forall (ECU e)
  where SoftwareComponent s -> e:
  sum(s.ram) <= e.ram;
```

In the above example we have a capacity-*constraint* that checks if on the ECUs on which software components are deployed the total RAM capacity is not exceeded. Since we are applying the **where** clause, a restricted set of ECUs is taken into account, the set M^r of ECUs is defined as in Section 3.3.2. This set of ECUs yields a set of constraints. The above constraint is unrolled in a set of constraints having the size equal to the size of the set M^r. Each of the unrolled constraint is checking whether the total RAM of a specific ECU is not exceeded.

4.2.2 *Minimization of Bus Load*

Depending on the deployment decision taken by engineers or the automatic approach presented in this work, we observe a different amount of load on the involved buses. Assume we have given a topology as depicted in Figure 4(a). There, different ECUs are communicating via two bus systems connected via a gateway. An exemplary communication relation between SWC 1 on ECU 1 and SWC 3 on ECU 3 is depicted as a dashed arrow. As one can see, a message sent from SWC 1 has to pass the gateway and causes bus loads on CAN 1 and CAN 2, respectively. Of course, in more complex topologies with several gateways and even more buses, complex communication patterns have to be considered. Therefore, we reduce the communication problem to the *All Pairs Shortest Path* (APSP) problem and use a breadth-first search (BFS) algorithm to find all paths between deployment targets—usually ECUs but also smart sensors and actuators—in a graph induced by the respective hardware topology, yielding a run-time complexity of $\mathcal{O}(n \cdot (|E| + |V|))$ where n is the number of ECU vertices (gray shaded), $|V|$ is the number of vertices, and $|E|$ is the number of edges in the graph. An example is depicted in Figure 4(b). Depending on the communication partners, different buses are used, illustrated in the table of Figure 4(c). We model the causal relationship among the software components to be deployed right within the model. Moreover an (early) approximate for the required payload on an (in-)directly connected bus is specified. This information together with that of the hardware topology is used to specify objectives as well as constraints over buses, e. g. bandwidth limitations:

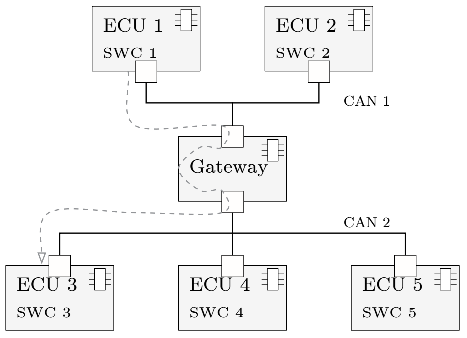
```
min Bandwidth:
  forall (SoftwareComponent s)
  where s <> CAN c:
  sum(s.payload / s.period);

const CAN:
  forall (CAN c)
  where SoftwareComponent s <> c:
  sum(s.payload / s.period) <= c.bandwidth;
```

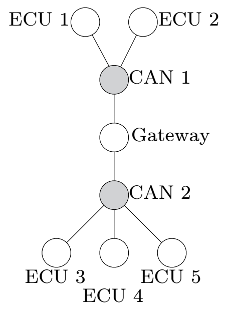
5. CASE STUDY AND EVALUATION

We evaluate the presented approach using a case study taken from the literature (cf. [20]): four power windows are modeled using both a software architecture and a hardware topology. We investigate the difference with respect to cost and bandwidth using two different hardware topologies. The alternatives are depicted in Figures 6(a) and 6(b), respectively. The two alternatives reflect the general design paradigms: federated vs. centralized architectures. In the first case, we use a dedicated ECU (door controller, DC) for each door (FR, FL, RR, RL), in the second case, we have a centralized computing platform (central controller, CC). The presented constraint and optimization language is well-suited to harmonize on the one hand the resource offer of the hardware topology and on the other hand the resources demanded by software components.

In the following, we first briefly describe the used software as well as hardware components along with their respective properties.



(a) Hardware Topology



(b) Connection graph

	E1	E2	E3	E4	E5
E1	-	C1	C1,C2	C1,C2	C1,C2
E2	C1	-	C1,C2	C1,C2	C1,C2
E3	C1,C2	C1,C2	-	C2	C2
E4	C1,C2	C1,C2	C2	-	C2
E5	C1,C2	C1,C2	C2	C2	-

(c) Communication matrix with bus involvement.

Figure 4: Example: (a) depicts a hardware topology and (b) the induced connection graph. A communication matrix is shown in (c). E*n* is short for ECU *n*, and C*n* for CAN *n*, respectively.

5.1 Software Architecture

The software architecture consists of the two basic software components *Power Window Control* (PWC) and *Power Window Control Coordinator* (PWCC). The PWC component is assumed to require 8 kB of RAM and 12 kB of ROM, the PWCC component 6 kB and 8 kB, respectively. Moreover in the second topology, we have additionally modeled software components deployed to sensors and actuators, because in this setting bus communication occurs, which can be modeled using communicating software components. For the sake of simplicity, we present cumulated signals between interacting software components. Please refer to Florentz and Huhn [20] for a detailed overview. Figure 5 depicts the software architecture together with the involved sensors and actuators as discussed above. The figure sketches the architecture for the front right power window. All others are dashed indicated.

As additional information we see the number of bits sent as well as the respective period. These information are used to compute the bus utilization. Note, in the figures we are using the graphical notation in compliance with the AUTOSAR [1] standard.

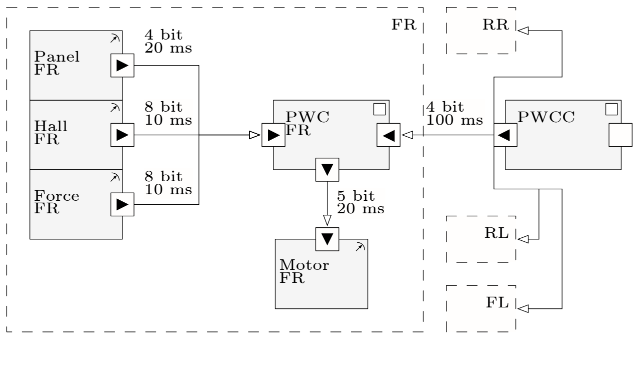


Figure 5: Considered software architecture

5.2 Hardware Topology

Table 2 contains information about the hardware attributes. For the first scenario, we assume sensors and actuators directly connected to the respective DC, whereas in the second scenario, (*smart*) sensors and actuators are directly connected to a LIN bus. Hence, bus load caused by sensor and actuator interaction only appears in the second scenario.

Table 2: Hardware component properties

	Cost	RAM	ROM
Door Controller (DC)	15 EUR	32 kB	32 kB
Panel	2 EUR	-	-
Hall sensor	1 EUR	-	-
Force sensor	1 EUR	-	-
Motor	5 EUR	-	-
Central Controller (CC)	20 EUR	64 kB	64 kB
Panel (smart)	4 EUR	-	-
Hall sensor (smart)	3 EUR	-	-
Force sensor (smart)	3 EUR	-	-
Motor (smart)	7 EUR	-	-

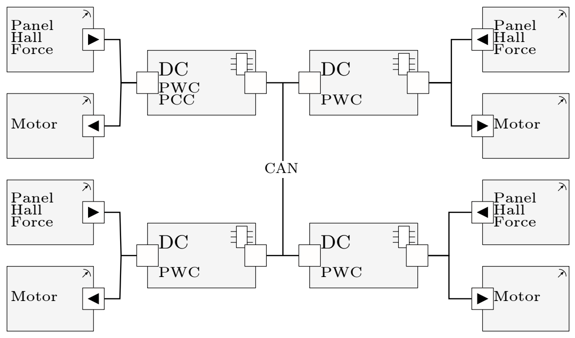
Table 3: Results

	Cost	Bandwidth
Alternative 1 (CAN)	96 EUR	120 bit/s
Alternative 2 (LIN)	88 EUR	8.200 bit/s

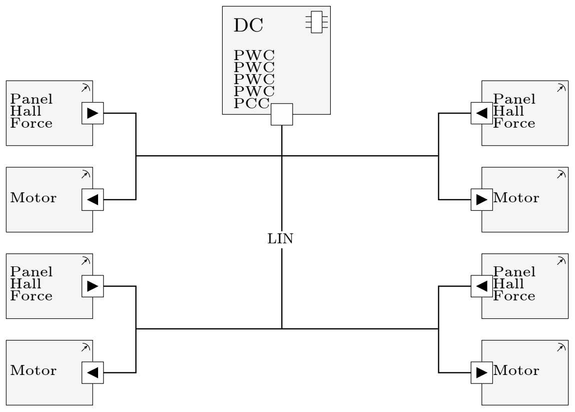
5.3 Evaluation

We modeled the described system using the tool in Section 4 and the Eclipse Modeling Framework (EMF). In both cases we wanted to minimize cost and bandwidth. For this example, however, there was not much space for optimization. In the federated case, at each door controller an instance of PWC is executed. PWCC could possibly be deployed to any of the identical (with respect to software and hardware) door controllers. In practice, door controllers might not be identical for instance in the case when on the front controllers software for an electrically foldable mirror function is additionally deployed. Then the full potential of the presented approach becomes visible.

For the presented two hardware topology alternatives we obtain after having specified the optimization model using the introduced DSL the following results in Table 3. Currently we only consider the net bandwidth, i. e., without message headers. But we already see that alternative two has a much more bandwidth usage while being cheaper. We use the following **AAOL** commands for the case study. Due to space limitations, only constraints for the door controller on the front right (FR) is shown.



(a) Hardware topology alternative 1



(b) Hardware topology alternative 2

Figure 6: (a) Federated architecture with a CAN bus. (b) Centralized architecture using a LIN bus. For the sake of clarity, the shown sensor components subsume the three sensors "Panel", "Hall", and "Force".

```
import CAN_LIN_optimization.deployment;
use {SWA_CAN, SWA_LIN} with {HWA_CAN, HWA_LIN};
objectives:
  minimize Cost:
     forall (HardwareComponent h)
    where SoftwareComponent s -> h: sum(h.cost);
  minimize BW_CAN:
     forall (SoftwareComponent s) where s<>CAN can:
    sum(s.payload / s.period);
  minimize BW_LIN:
     forall (SoftwareComponent s) where s<>LIN lin:
    sum(s.payload / s.period);
constraints:
  constraint Fix_CAN_FR:
    for(SoftwareComponent SWA_CAN.PWC_FR_CAN):
      PWC_FR_CAN -> CAN_Topology.DC_FR;
  constraint Fix_LIN_FR:
    for(SoftwareComponent SWA_LIN.PWC_FR):
      PWC_FR -> LIN_Topology.CC;
  constraint Fix_LIN_Panel:
    for(SoftwareComponent SWA_LIN.Panel_FR):
      Panel_FR -> LIN_Topology.Panel_FR;
  constraint Fix_LIN_Hall:
    for(SoftwareComponent SWA_LIN.Hall_FR):
      Hall_FR -> LIN_Topology.Hall_FR;
  constraint Fix_LIN_Force:
    for(SoftwareComponent SWA_LIN.Force_FR):
      Force_FR -> LIN_Topology.Force_FR;
  constraint Fix_LIN_Motor:
    for(SoftwareComponent SWA_LIN.Motor_FR):
      Motor_FR -> LIN_Topology.Motor_FR;
  constraint ROM:
     forall (ECU e) where SoftwareComponent s -> e:
    sum(s.rom) <= e.rom;
  constraint Ram:
     forall (ECU e) where SoftwareComponent s -> e:
    sum(s.ram) <= e.ram;
  constraint CAN_Com:
     forall (CAN can)
    where SoftwareComponent s <> can:
    sum(s.payload / s.period) <= can.bandwidth;
  constraint LIN_Com:
     forall (LIN lin)
    where SoftwareComponent s <> lin:
    sum(s.payload / s.period) <= lin.bandwidth;
orders:
  order objective: [BW_CAN < Cost];
```

6. CONCLUSION AND FUTURE WORK

We presented a generic framework for model-based optimization of automotive E/E-architectures—in particular the software to hardware deployment problem. We proposed to use a domain-specific language, which is particularly tailored towards the needs to express both objectives and constraints occurring in the mentioned setting. The use of multi-objective evolutionary algorithms allows to simultaneously optimize multiple objectives. We designed the framework in such a way that it is easily extendable with respect to other solvers or even other optimization problems which we are faced during the development process. A power window was used to evaluate the presented approach using two different hardware topologies, namely a federated and a centralized topology with a CAN bus in the first case and a LIN bus in the latter. The language has been specified with a close industrial collaboration. We implemented the outlined methodology as plugin for the eclipse platform.

We plan to extend our work in several directions. First by improving the syntax of **AAOL**, for example by adding more predicates and operators in the **where** clause and also by allowing more mathematical expression/aggregation functions when defining constraints and objectives. Logical operators between predicates in **where** clause will also be added, for building complex logical relations between predicates. Second, the solver part of our tool will be further extended. Since we defined an extensible framework, we will later easily explore new solver solutions like SMT (e. g. Z3 [15]) or ILP (e. g. CPLEX [3]) or a combination of SMT and MOEA, for providing more accurate results. We also plan to use a more complex case study to emphasize the power of multi-objective optimization of our framework, since in the present work the focus was more on constraints.

7. REFERENCES

[1] AUTOSAR: Automotive Open System Architecture. http://www.autosar.org. [Online; accessed 31-January-2014].

[2] Eclipse—The Eclipse Foundation. http://www.eclipse.org/.

[3] IBM ILOG CPLEX Optimizer. `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[4] MOEA Framework: A Free and Open Source Java Framework for Multiobjective Optimization. `http://www.moeaframework.org`.

[5] Xtext - Language Development Made Easy! - Eclipse. `http://www.eclipse.org/Xtext/`.

[6] A. Abbas, E. Tsang, and A. Nasri. Depict: A high-level formal language for modeling constraint satisfaction problems. In *AICCSA '06: Proceedings of the IEEE International Conference on Computer Systems and Applications*, pages 365–368, Washington, DC, USA, 2006. IEEE Computer Society.

[7] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya. Archeopterix: An extendable tool for architecture optimization of aadl models. In *Proceedings of the ICSE Workshop on Model-based Methodologies for Pervasive and Embedded Software*, volume 0, pages 61–71, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[8] N. Arunkumar and L. Karunamoorthy. An optimization technique for vendor selection with quantity discounts using genetic algorithm. *Journal of Industrial Engineering International Islamic Azad University, South Teheran Branch*, Jan 2007. [Online; accessed 07-September-2011].

[9] J. Axelsson. Cost models for electronic architecture trade studies. In *ICECCS*, pages 229–. IEEE Computer Society, 2000.

[10] J. Axelsson. Cost models with explicit uncertainties for electronic architecture trade-off and risk analysis. In *Proc. 16th International Symposium of the International Council on Systems Engineering*, July 2006.

[11] D. Beetner, H. Weng, M. Wu, and T. Hubing. Validation of worst-case and statistical models for an automotive emc expert system. In *Electromagnetic Compatibility, 2007. EMC 2007. IEEE International Symposium on*, pages 1 – 5, July 2007.

[12] M. Broy, M. Feilkas, J. Grünbauer, A. Gruler, A. Harhurin, J. Hartmann, B. Penzenstadler, B. Schätz, and D. Wild. Umfassendes Architekturmodell für das Engineering eingebetteter Software-intensiver Systeme. Technical Report TUM-I0816, Technische Universität München, 2008.

[13] M. Cadoli, G. Ianni, L. Palopoli, A. Schaerf, and D. Vasile. Np-spec: an executable specification language for solving all problems in np. *Comput. Lang.*, 26(2-4):165–195, 2000.

[14] R. Chenouard, L. Granvilliers, and R. Soto. Model-driven constraint programming. In S. Antoy and E. Albert, editors, *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP), July 15-17, 2008, Valencia, Spain*, pages 236–246. ACM, 2008.

[15] L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. Evolutionary Computation*, 6(2):182–197, 2002.

[17] E. W. Dijkstra. Chapter i: Notes on structured programming. In *Structured programming*, pages 1–82. Academic Press Ltd., London, UK, UK, 1972.

[18] M. Eisenring, L. Thiele, and E. Zitzler. Conflicting criteria in embedded system design. *IEEE Design & Test of Computers*, 17(2):51–59, 2000.

[19] R. Fellini, N. Michelena, P. Papalambros, and M. Sasena. Optimal design of automotive hybrid powertrain systems. In *Environmentally Conscious Design and Inverse Manufacturing, 1999. Proceedings. EcoDesign '99: First International Symposium On*, pages 400 –405, February 1999.

[20] B. Florentz and M. Huhn. Embedded systems architecture: Evaluation and analysis. In C. Hofmeister, I. Crnkovic, and R. Reussner, editors, *Quality of Software Architectures*, volume 4214 of *Lecture Notes in Computer Science*, pages 145–162. Springer Berlin / Heidelberg, 2006.

[21] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press, November 2002.

[22] S. Frei, R. Jobava, and D. Topchishvili. Complex approaches for the calculation of emc problems of large systems. In *Electromagnetic Compatibility, 2004. EMC 2004. 2004 InternationalSymposium on*, volume 3, pages 826–831, August 2004.

[23] A. M. Frisch, M. Grum, C. Jefferson, B. M. Hernández, and I. Miguel. The design of essence: A constraint language for specifying combinatorial problems. In M. M. Veloso, editor, *IJCAI*, pages 80–87, 2007.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability (A Guide to Theory of NP-Completeness)*. Freeman, San Francisco, 1979.

[25] M. Glaß, M. Lukasiewycz, R. Wanka, C. Haubelt, and J. Teich. Multi-Objective Routing and Topology Optimization in Networked Embedded Systems. In *Proceedings Int. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation (IC-SAMOS 2008)*, pages 74–81, Samos, Greece, July 2008.

[26] L. Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In L. J. Osterweil, H. D. Rombach, and M. L. Soffa, editors, *ICSE*, pages 849–852. ACM, 2006.

[27] L. Grunske, P. A. Lindsay, E. Bondarev, Y. Papadopoulos, and D. Parker. An outline of an architecture-based method for optimizing dependability attributes of software-intensive systems. In R. de Lemos, C. Gacek, and A. B. Romanovsky, editors, *WADS*, volume 4615 of *Lecture Notes in Computer Science*, pages 188–209. Springer, 2006.

[28] W. Haberl, M. Herrmannsdoerfer, S. Kugele, M. Tautschnig, and M. Wechs. One click from model to reality, 2009. accepted for presentation at SAASE '09: Symposium on Automotive/Avionics Systems Engineering.

[29] W. Haberl, M. Herrmannsdoerfer, S. Kugele, M. Tautschnig, and M. Wechs. Seamless model-driven development put into practice. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6415 of *Lecture Notes in Computer Science*, pages 18–32. Springer, October 2010.

[30] C. Hang, P. Manolios, and V. Papavasileiou. Synthesizing cyber-physical architectural models with real-time constraints. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 441–456, Berlin, Heidelberg, 2011. Springer-Verlag.

[31] International Organization for Standardization. Road vehicles–Functional safety (ISO 26262), 2011.

[32] S. Kugele, W. Haberl, M. Tautschnig, and M. Wechs. Optimizing automatic deployment using non-functional requirement annotations. In T. Margaria and B. Steffen, editors, *ISoLA*, volume 17 of *Communications in Computer and Information Science*, pages 400–414. Springer, 2008.

[33] S. Kugele, M. Tautschnig, A. Bauer, C. Schallhart, S. Merenda, W. Haberl, C. Kühnel, F. Müller, Z. Wang, D. Wild, S. Rittmann, and M. Wechs. COLA – The component language. Technical Report TUM-I0714, Institut für Informatik, Technische Universität München, sep 2007.

[34] R. Kumar, K. Izui, M. Masataka, and S. Nishiwaki. Multilevel redundancy allocation optimization using hierarchical genetic algorithm. *IEEE Transactions on Reliability*, 57(4):650–661, 2008.

[35] M. Lukasiewycz, M. Glaß, C. Haubelt, J. Teich, R. Regler, and B. Lang. Concurrent Topology and Routing Optimization in Automotive Network Integration. In L. Fix, editor, *DAC*, pages 626–629. ACM, 2008.

[36] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011.

[37] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside. Modelling and multi-objective optimization of quality attributes in variability-rich software. In *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, NFPinDSML '12, pages 2:1–2:6, New York, NY, USA, 2012. ACM.

[38] V. Pareto. *Cours d'Économie politique*, volume I and II. F. Rouge, Lausanne, 1896.

[39] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

[40] K. Pohl, H. Hönninger, R. Achatz, and M. Broy. *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology.* Springer, Berlin, 2012.

[41] C. Quigley, R. McMurran, R. Jones, and P. Faithfull. An investigation into cost modelling for design of distributed automotive electrical architectures. In *Automotive Electronics, 2007 3rd Institution of Engineering and Technology Conference on*, pages 1–9, June 2007.

[42] H. Rebholz and S. Tenbohlen. A fast radiated emission model for arbitrary cable harness configurations based on measurements and simulations. In *Electromagnetic Compatibility, 2008. EMC 2008. IEEE International Symposium on*, pages 1–5, August 2008.

[43] J. Sobieszczanski-Sobieski, S. Kodiyalam, and R. Y. Yang. Optimization of car body under constraints of noise, vibration, and harshness (nvh), and crash. *Structural and Multidisciplinary Optimization*, pages 295–306, Jan 2001.

[44] J. M. Spivey. An introduction to z and formal specifications. *Softw. Eng. J.*, 4:40–50, January 1989.

[45] T. Streichert, C. Haubelt, and J. Teich. Multi-objective topology optimization for networked embedded systems. In G. Gaydadjiev, C. J. Glossner, J. Takala, and S. Vassiliadis, editors, *ICSAMOS*, pages 93–98. IEEE, 2006.