On the Deployment Problem of Embedded Systems

Stefan Kugele*, Gheorghe Pucea*, Ramona Popa*, Laurent Dieudonné[†] and Horst Eckardt[‡]
* Institut für Informatik, Technische Universität München, Garching b. München, Germany
Email: kugele@in.tum.de, george.pucea@tum.de, ramona.popa@tum.de
[†] Liebherr-Aerospace Lindenberg GmbH, Lindenberg, Germany, Email: laurent.dieudonne@liebherr.com
[‡] Siemens AG, Corporate Technology, München, Germany, Email: Horst.Eckardt@siemens.com

***Abstract*—The quality of today's embedded systems e. g. in vehicles, airplanes, or automation plants is highly influenced by their architecture. In this context, we study the so-called *deployment problem*. The question is where (i. e., on which execution unit) to deploy which software application or which sensor/actuator shall be connected to which device in an automation plant. First, we introduce a domain-specific constraint and optimization language fitting the needs of our partners. Second, we investigate different approaches to tackle the deployment problem even for industrial size systems. Therefore, we present different solving strategies using (i) multi-objective evolutionary algorithms, (ii) SMT-based, and (iii) ILP-based solving approaches. Furthermore, a combination of the first two is used. We investigate the proposed methods and demonstrate their feasibility using two realistic systems: a civil *flight control system* (FCS), and a *seawater desalination plant*.**

I. Introduction

During the last decades, more and more software-controlled functions were introduced in embedded systems which we can find for instance in automobiles, airplanes, or very large automation plants. These functions are realizing a multitude of different tasks ranging from highly safety-critical and real-time control tasks (e. g. the flight control system (FCS) of airplanes or automation control of plants) up to comfort and infotainment systems found in modern cars. In the future, the number of software-controlled functions in networked embedded systems as well as the number of sensors and actuators in smart factories will grow further—especially in emerging cyber-physical systems (CPS). There, engineers have to decide on which of the available execution units (EXUs) a function (or software application) should be realized. Additionally, in the automation domain, also an optimal connection of so-called field devices (e. g. sensors and actuators) is required for instance to save cabling costs. Therefore, we are faced with at least two different kinds of deployment: (i) software to hardware, and (ii) hardware to hardware deployment. Traditionally, this question was answered with a lot of experience and intuition. 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. Due to the high number of deployable entities and deployment targets in practical application (e. g. 600 deployable entities and 216 deployment targets in the seawater desalination plant case study presented in Section IV-B) the combinatorial complexity is tremendous. This order of magnitude cannot be managed by hand anymore let alone be optimized. Experience has taught us that the design of embedded architectures has wide-ranging effects on cost, robustness, safety, and maintainability just to mention a few. Therefore, an optimal architectural design is always desirable.

Optimization steps are involved at different stages during the system development process. Typically, engineers are faced with many requirements that are oftentimes conflicting. In the avionic domain, for instance, one goal is to develop an airplane with a better ratio of kerosene consumed per aircraft-passenger, which is more expensive to achieve due to modern materials or jet-propulsion engines; this conflicts with the goal to be as cost-efficient as possible. Therefore, the purpose in such situations is to find the best compromise (trade-off) amongst the different goals (design criteria) [1]. Hence, these problems boil down to multi-objective optimization problems. However, systems engineers are usually no experts in mathematical optimization or operations research in general. Thus, they have to be supported by powerful modeling and engineering tools. However, hard-coded optimization rules (such as for instance in AutoFOCUS [2] or ArcheOpterix [3]) restrict the engineers' flexibility. Therefore, we propose that system modeling is accomplished by the specification of optimization goals and constraints using an easy yet concisely domain-specific optimization language. We introduce the *System Architecture Optimization Language* (**SAOL**) for both objectives and constraints, whereas special focus was put on simplicity and usability by engineers. **SAOL** considerably extends the possibility of the *Automotive Architecture Optimization Language* which we introduced in a previous work [4].

Besides the specification of pursued design goals and technical, regulatory, or other constraints, powerful solving strategies also have to be examined and realized. In this paper, we analyze three different solving approaches: (i) an extension to the NSGA-II multi-objective evolutionary algorithm (MOEA), (ii) satisfiability modulo theories (SMT)-based, and (iii) integer linear programming (ILP)-based approaches. Also a combination of (i) and (ii) is studied. The approaches are realized within an integrated Eclipse-based modeling and optimization tool for system architectures.

We perform the evaluation with two realistic industrial size case studies, namely a civil flight control system (software to hardware deployment) and a large-scale seawater desalination plant (hardware to hardware deployment) and investigate which approach works well in which scenario.

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.

This work provides the following contributions:

(i) We present language extensions tailored towards safety requirements demanded by regulatory compliances in the considered application domains.
(ii) We present and compare an extended MOEA, an SMT-, and an ILP-based approach to tackle the deployment problem. Moreover, the *Modified Guided Improvement Algorithm* is presented.
(iii) We conduct experiments to evaluate the feasibility of the presented approaches using two realistic industrial large-scale case studies.

Outline: The remainder of this paper is structured as follows: Section II relates this work to the state-of-the-art. Next, in Section III we present the main contributions of this work, followed by two case study descriptions and evaluation in Sections IV and V. Finally, Section VI concludes this work and points out possible future research directions.

II. Related Work

This work is basically related to two research directions: (i) *constraint programming* and (ii) *(multi-objective) optimization* of embedded systems architectures. Hence, we will relate our work to both directions separately.

AMPL by Fourer et al. [5] is a very mathematical notation to formulate constraints and objectives. 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 [5]. However, this cannot be expected from engineers or system architects, who are our intended users. The same applies to DEPICT [6], ESSENCE [7], or Z [8]. We take a similar approach to [9] and let the users specify *what* they want and not *how* to achieve it. In contrast to most of the above mentioned specification languages, we defined **SAOL** to easily yet concisely specify both objectives and constraints, whereas special focus was put on simplicity and usability by engineers rather than mathematicians.

Similar to Grunske et al. [3], [10], a part of this work studies to use MOEA strategies to find good architectural design alternatives. Moreover, Genetic Algorithms (GA) are used by Meedeniyaa et al. [11] in a reliability-driven deployment optimization of embedded systems. Also a GA is used by Kumar et al. [12] to perform a multilevel redundancy allocation. Czarnecki and Olaechea suggest an optimization framework for software product line development [13]. 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 (*Guided Improvement Algorithm*) that can list the Pareto-optimal points during computation. It is a general purpose algorithm for solving combinatorial many-objective optimization problems [14]. We present an extension of the algorithm using the Z3 [15] SMT and Gurobi [16] (M)ILP solver. Another interesting approach for solving multi-objective optimization problems by combining different solvers emerged also from software product line development. The approach is presented by Sayyad et al. [17] and consists of generating the initial population of evolutionary algorithm with different solvers for improving the scalability of the approach. Hilbrich and Dieudonné [18] use an ILP-based approach with a DSL for the deployment problem in the avionics domain.

III. Approach

Within this paper we use the following definition of a *deployment problem* to have a common understanding.

Definition 1 (Deployment Problem): We say that $\mathcal{D} = \langle \mathbf{S}, \mathbf{T}, \mathbf{O}, \mathbf{C} \rangle$ is a deployment problem, where $\mathbf{S}$ and $\mathbf{T}$ are non-empty sets of *source* and *target* objects. Let $\mathbf{O}$ be the possible empty set of *objectives* and $\mathbf{C}$ the non-empty set of *constraints*. A *feasible solution* ℓ of the deployment problem is an assignment of *truth values* to *decision variables* $x_{\text{source}\rightarrow\text{target}}$ ($|\mathbf{S}| \cdot |\mathbf{T}|$ many) such that all constraints $c \in \mathbf{C}$ are satisfied considering possible optimization objectives. For software to hardware deployment, software applications are *source* objects and hardware components (e. g. EXUs) are *target* objects. For hardware to hardware deployment, hardware components (e. g. sensors and actuators) are *source* objects and inverters or DIOs (distributed input/output units where sensors and actuators are connected to) are *target* objects.

A. Language **SAOL**

In a previous work [4], we reported on the syntax and semantics of an early version of **SAOL**, which is considerably extended in this work with respect to expressive power. In the following, we briefly give an introduction into **SAOL** and point out the new language capabilities.

*1) Short Introduction into **SAOL**:* The basic concepts of **SAOL** are that of (i) *objectives*, (ii) *constraints*, and (iii) *orders*. Objectives are used to specify (conflicting) goals that have to be optimized, i. e., minimized or maximized, simultaneously while satisfying the constraints. Finally, possible solutions of this (multi-criteria) optimization problem are ranked according to the orders. Moreover, orders can be given to specify compatibility relations, for instance when using safety classifications like the Design Assurance Level (DAL) from the DO-178C [19] standard.

The following example is used to illustrate and discuss the language capabilities.

Listing 1: Running **SAOL** example.

```
objectives:
  min Cost:   forall(EXU e) where SWA s -> e: sum(e.cost);
  min BW:     forall(SWA s) where s <> CAN c:
                                   sum(s.payload / s.period);
constraints:
  const Fix:  for(SWA swa_inst): swa_inst -> EXU exu_inst;
  const Mat:  forall(SWA s) where s -> EXU e:
                            e.fabricator == s.fabricator;
  const ROM:  forall(EXU e) where SWA s -> e:
                                   sum(s.rom) <= 0.8 * e.rom;
  const DAL:  forall(SWA s) where s -> EXU e: s.dal ~ e.dal;
  const CAN:  forall(CAN c) where SWA s <> c:
              sum(s.payload / s.period) <= c.bandwidth;
orders:
  order objective: [Cost > BW];
  order DAL: [ A > B, B > C, C > D, D > E ];
```

All attributes used in **SAOL** specifications e. g. `cost`, `payload`, or `dal`) refer to artifact attributes modeled in the meta-model. Their respective values are taken from the concrete model.

a) Objectives: Objectives are used to state the design goals of a system architecture. **SAOL** supports likewise minimization and maximization of objectives. Line 2 aims at minimizing the summed costs of used EXUs, i. e., the cost of all EXUs **where** software applications (SWA) are deployed to (indicated by `SWA s -> e`). The next objective in lines 3-4 aims at minimizing the busload caused by software applications that periodically send a payload on the bus. These objectives use on the right hand side the *aggregation function* **sum**`()`.

b) Constraints: Constraints are used to formulate for example technical or regulatory restrictions and thus to reduce the solution space.

(i) A fix-*constraint* in line 6 forces a specific source object instance (e. g. `SWA swa_inst`) to be deployed onto a specific target object instance (e. g. `EXU exu_inst`).
(ii) The matching-*constraint* in lines 7-8 states that valid deployment targets **forall** `SWA s` are only `EXU e` instances with the same fabricator attribute.
(iii) With the capacity-*constraint* in lines 9-10 usage of any *limited resource* can be restricted. The ROM usage, for instance, is limited to 80%. A second example is network bandwidth expressed in the lines 12-13.
(iv) We use compatibility-*constraint*s to express any kind of compatibility relation. In the example, a software application `s` with a certain design assurance level is only allowed to be deployed onto an EXU with a compatible level (cf. line 11). The notion of compatibility is expressed in line 16 and explained below.

c) Orders: In **SAOL** orders are used to express two things: first, the user can state how solutions are ranked and presented (cf. line 15). Second, orders are used to define compatibility relations. The design assurance levels (A = catastrophic, E = no effect) compatibility relation, for instance, restricts valid deployment decisions (cf. line 16).

2) Language Extensions: In the following, we list the new language features contributed in this work.

For our industrial partners, the reduction of used hardware components is a major objective in order to save weight, space, energy, and of course money. Moreover, safety considerations are a major topic.

*a) Aggregation Function **count**:* We introduce a new *aggregate function* **count**, which allows to count the number of used artifacts. If for instance the objective is to minimize the number of used EXUs, then this translates to:

```
min NoOfEXUs: forall(EXU e) where SWA s -> e: count(e);
```

The number of used EXUs, i. e., some software application is deployed onto it, is counted and minimized.

b) Optimization of Resource Requirements: As a further extension, **SAOL** supports the optimization of any numeric artifact attribute:

```
min ROM:     forall(EXU e) where SWA s -> e: sum(s.rom);
```

This is important, since resource usage may vary according to the deployment decision. Another example is the cost required for cables between different buildings due to placement decisions.

c) Logical Operators: The possibility to use the logical operators **and**, **or**, and **not** is a further powerful language extension allowing to express arbitrary complex nested properties in propositional logic. A more complex example is given in the avionics case study discussed in Section IV-A (Listing 4).

d) Arithmetic Operators: Now, it is possible to state complex nested expressions using the basic arithmetic operators ($+$, $-$, $*$, and $/$) and brackets. A simple example taken from the case study described in Section IV-B is given next.

```
const PowerConsumption:
  forall(Inverter i) where FieldActuator fa -> i:
      sum(fa.maxVoltage * fa.maxCurrent) <= i.maxPower;
```

Note, arithmetic expressions can also be used within aggregation functions and to combine them.

e) Dislocality and Dissimilarity Constraints: To meet the strict safety guidelines for civil aircrafts (cf. also [19]), we extended **SAOL** to support the concepts of *dissimilarity* and *dislocality*. These are generic safety concept which also apply in other domains. In the avionic context, dislocality requires two *dislocal* software applications `s1` and `s2` to be deployed on two *distinct* hardware components (usually EXUs) not at the same place. Even more details about the actual installation site (e. g. room, zone, cabinet, building, plant area, etc.) can be specified in the **where** clause, e. g. `e1.building != e2.building`. Hence, for each pair of dislocal software applications a respective constraint can be given. An even stronger concept is needed for highly safety-critical systems: that of *dissimilarity*. In addition to dislocality, two software applications `s1` and `s2` need to be deployed on target hardware of different type, from different suppliers, using different processors, cores, etc. These requirements are translated as follows:

```
const DISLOC:
  for(SWA s1, s2)where s1 -> EXU e1, s2 -> EXU e2: (e1!=e2);
const DISSIM:
  for(SWA s1, s2)where s1 -> EXU e1, s2 -> EXU e2:
                          (e1.fabricator != e2.fabricator);
```

f) Property Constraints: With property constraints, one can specify which SWAs are allowed to be deployed on a particular EXU by characterizing them using their *properties*.

```
const PROPERTY small:
  for(EXU small) where SWA s -> small: s.ram >= 0 and
                                       s.ram <= 100;
const PROPERTY big:
  for(EXU big) where SWA s -> big: s.ram > 100 and
                                   s.ram <= 300;
```

B. Solver Infrastructure

Our solver infrastructure consists of three different framework categories: (i) The MOEA framework [20], which implements several state-of-the-art evolutionary algorithms for multi-objective optimization. (ii) The Z3 [15] SMT solver, which can solve our constraint satisfaction problem and can be used to perform multi-objective optimization (cf. Section III-C1a). (iii) We support several built-in (Mixed) Integer Linear Programming solvers, namely LP_SOLVE [21], GLPK [22], and Gurobi [16]. We discuss different capabilities with respect to **SAOL** and report on the evaluation in Section V.

We use the state-of-the-art multi-objective evolutionary algorithm NSGA-II [23]. However, due to scalability issues of the default implementation of NSGA-II we added some extensions to the algorithm, presented later in this section.

Z3 is an efficient SMT solver by Microsoft Research. Due to the fact that SMT is a decision problem, Z3 allows us to encode the deployment problem as constraint satisfaction problem (CSP). However, the CSP will solve only half of our deployment problem. It will only generate deployments which satisfy all constraints. In order to generate solutions that are optimal, the objectives are transformed into constraints. The implementation is called *Modified Guided Improvement Algorithm* (MGIA) and presented in Section III-C1a. Similarly, we use the generic concept of MGIA to solve a multi-objective optimization problem using ILP solvers.

1) Extensions to MOEA: Multi-objective evolutionary algorithms start with an initial population, which is then evolved with the help of *crossover* and *mutation* operators so that it stepwise converges to an optimal solution. The main scope of these variation operators is to be able to introduce new solutions which are better that the old ones while satisfying all constraints.

a) Pre-assigned Initial Population: In order not to start with a totally random initial population $\mathbf{P}$, and thus to reduce the search space, we pre-assigned the indices of decision variables affected by fix-constraints by the required truth values.

b) Crossover Operator: Crossover operators (we use an extended *Hux*, half uniform crossover operator) are used to evolve the population towards a better one, by creating new offspring solutions from two existing solutions. A Hux operator takes two solutions and compares the decision variables at each position in the array of decision variables. If the two decision variables are different, then these positions are swapped. We implemented a new Hux operator which ensures that for fix-constraints no variation is introduced at the index of that particular decision variables to eliminate constraint violation.

c) Mutation Operator: Mutation operators (we use an extended *BitFlip* operator) are used to introduce variation into the population, too. This operator changes the decision variables to the negated values by picking random positions in the array of decision variables. We adapted this operator to only introduce variability in the array of decision variables, where the positions are not fixed. For example through a fix-*constraint*.

2) MOEA with IIPE: It turned out during our experiments that scalability is very sensitive with regard to the initial population: a bad initial population hampers finding valid solutions. Hence, we use an SMT-based approach to quickly find a (set of) valid solution(s) (Lines 7–8 in Listing 2) and implant it in the initial population $\mathbf{P}$ (Line 8) with size s. We call this approach *Iterative Initial Population Enumerating* (IIPE). Line 9 prevents this solution from being found again in the next iteration. Of course, we cannot guarantee that the found solutions are Pareto-optimal in any aspect, but they are a good starting point. Besides being the provider for an initial population, IIPE can also be used as a standalone procedure to quickly enumerate solutions of the deployment problem.

We assume that the set of constraints is encoded into an SMT formula ϕ. Let $\mathbf{P}$ be the set of the initial population with size $s = |\mathbf{P}|$.

Moreover we say that ℓ is a feasible solution, i. e., a satisfying assignment to the decision variables such that ϕ evaluates to true and write $\ell \models \phi$.

Listing 2: IIPE

```
1  function IIPE($\phi, s$)
2  begin
3      $\mathbf{P} := \emptyset$;
4      $c := 0$;
5      while $\exists \ell : \ell \models \phi \wedge c \neq s$ do
6      begin
7          $\ell :=$ getSolution();
8          $\mathbf{P} = \mathbf{P} \cup \{\ell\}$;
9          $\phi := \phi \wedge \neg \ell$;
10         $c := c + 1$;
11     end;
12     return $\mathbf{P}$;
13 end;
```

3) MOEA with MGIA: First we initialized the MOEA solver only with solutions that have all constraints satisfied. Now, we go one step further and implant not only valid solutions into the initial population, but Pareto-optimal solutions. Section III-C1 describes the algorithm to get such solutions. The "pure" MOEA and the first extension (MOEA+IIPE) do not guarantee to find Pareto-optimal solutions (but they can), but approximates for them. The last extensions (MOEA+MGIA), on the contrary, *guarantees* to find Pareto-optimal solutions.

4) ILP with Weighted Sum Approach: Linear programming solvers and the usually used simplex algorithm do per se not support multi-objective optimization. One common approach is to give weights for each objective and to sum over all weighted objectives: the so-called *weighted sum approach*. Hence, instead of optimizing several objectives at the same time, a single aggregated objective is optimized: $\min / \max \ \sum_{i=1}^{k} w_i \cdot o_i$, where w_i is the weight for objective $o_i \in \mathbf{O}$. The weighted sum approach, however, suffers under the necessity to provide weights. A good choice of weights is important and has a strong influence on the solution especially in the case of similar objective function evaluations. In contrast, if an objective function's evaluation is dominating the others, i. e., if its unit is an order of magnitude higher than the others, weights are less influential unless they are adapted accordingly.

5) ILP with MGIA: Similarly to Section III-B3 (MOEA+MGIA), we use MGIA (the modified guided improvement algorithm), to iteratively strengthen the constraints in order to push the solution towards Pareto-optimality. Therefore, we start with an arbitrary initial solution found by the ILP solver without any specific optimization function (e. g. *minimize* 0). The MGIA algorithm, described in Listing 3, is then used.

C. Encoding of the Deployment Problem

For a given deployment problem $\mathcal{D} = \langle \mathbf{S}, \mathbf{T}, \mathbf{O}, \mathbf{C} \rangle$, a translation for the respective solver has to be given. We use decision variables $x_{s \to t}$, where $s \in \mathbf{S}$ and $t \in \mathbf{T}$ are the sets of source and target objects and

$$x_{s \to t} = \begin{cases} \mathsf{true} & \text{if } s \text{ is mapped to } t \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

The natural representation for these decision variables is an n-by-m matrix where $n = |\mathbf{S}|$ and $m = |\mathbf{T}|$. The Z3 SMT solver directly supports matrices of decision variables encoded using the *BoolExpr* type. Both the evaluated ILP solvers and the NSGA-II algorithm of the MOEA framework do not support matrices. Therefore, we used a canonical linearization

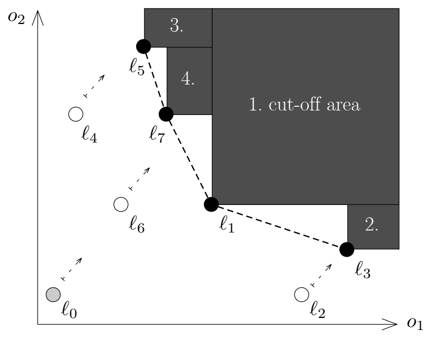


Fig. 1: Exemplary run of the MGI algorithm.

into arrays of size $n \cdot m$, which—for MOEA—is the natural representation when thinking of gene sequences.

1) Objectives: Multi-objective optimization is the strength of evolutionary algorithms such as NSGA-II. Therefore, no adaptions are necessary due to natural support. SMT and ILP solvers do not support multi-criteria optimization in general (a first attempt can be found in [24]). Therefore, we realized the weighted sum approach for ILP and a *modified* version of the *Guided Improvement Algorithm* (GIA) [14] which we call MGIA, for both SMT and ILP.

One way to improve solutions with respect to the quality, i. e., their evaluation with respect to the given objectives, is to guide the solver towards an optimal solution. The optimality criteria (objectives) are given in the **SAOL** specification. The modifications of MGIA are: (i) the parameter c in MGIA marks a maximum number of solutions that the algorithm should find, (ii) different from the original *Guided Improvement Algorithm* the MGIA speeds up the search for Pareto-optimal solutions by not finding solutions which have the same objective evaluation. We are interested in presenting diverse solutions to our users, hence we want deployments which have different objective evaluation.

a) The Modified Guided Improvement Algorithm (MGIA): Assume we have a set of objective (or metric) functions $\mathbf{O} = \{o_1, o_2, \ldots, o_k\}$ where $k = |\mathbf{O}|$ is the number of objectives. We apply each objective function to the solution ℓ to obtain an objective value: $o_i(\ell)$ for $1 \leq i \leq k$.

Listing 3: Modified Guided Improvement Algorithm

```
1  function MGIA(c)
2  begin
3    $\mathbf{P} = \emptyset$;
4    $counter := 0$;
5    $\ell_0$ := getSolution(); $\ell := \ell_0$;
6    while $counter < c$ do
7    begin
8      while $\exists \ell^* : \exists i, 1 \leq i \leq k : o_i(\ell^*) > o_i(\ell) \wedge$
             $\forall j \neq i, 1 \leq j \leq k : o_j(\ell^*) \geq o_j(\ell)$ do
9      begin
10       $\ell := \ell^*$;
11     end;
12     $\mathbf{P} := \mathbf{P} \cup \{\ell\}$;
13     $counter := counter + 1$;
14     if $\exists \ell^* : \exists i, 1 \leq i \leq k : o_i(\ell^*) > o_i(\ell)$ then
15     begin
16       $\ell := \ell^*$;
17     end else return $\mathbf{P}$;
18   end;
19   return $\mathbf{P}$;
20 end;
```

MGIA works as outlined in Listing 3 and visualized in Figure 1. Without loss of generality, we assume that each objective is to be maximized. Assume ℓ_0 is the initial random feasible solution satisfying all constraints (cf. Line 5 of Listing 3). As long as a new solution ℓ^* that is better in at least one objective and at least as good as the old one ℓ in any other objective can be found, update ℓ. This pushes the solution towards Pareto-optimality. If this is no longer possible a Pareto-optimal solution (ℓ_1 in the figure) is found (cf. Line 12). However there might be other solutions that are better with respect to at least one objective while worsening other objectives (cf. Line 14 and solution ℓ_2 in the figure). If so, we again push it towards the Pareto optimality. This is repeated until a defined number of Pareto-optimal solutions is found (cf. Line 6), or no other solution is possible (cf. Line 17). Each Pareto-optimal solution (ℓ_1, ℓ_3, ℓ_5, and ℓ_7 in the example) cuts off parts of the search space.

The search iteratively reduces the search space: for each objective, the solution ℓ is evaluated and a new constraint is added that requires a possible new solution to be better or as good as the old one. This is a generic algorithm and thus can be used in different realizations. We use it for SMT- as well as ILP-based optimization.

2) Constraints: All constraints defined in the **SAOL** specification are translated into the respective input of the used framework. Note, the syntax of **SAOL** is restricted in a way, that all constraints are translated into linear logic. Thus the SMT solver uses linear logic as background theory and the ILP solvers have to solve a system of only linear (in)equalities. When using SMT/Z3, all constraint types can be translated straightforward using linear combinations of the decision variables and Boolean connectors.

Currently, the disjunction in capacity-*constraints* (using **or**) is not supported when applying ILP-based techniques, because the evaluation cannot be done statically and thus has to be evaluated by the ILP solver. Most common ILP solvers, however, use the simplex algorithm working on convex polyhedrons. Because of disjunctions, convexity cannot be guaranteed.

IV. Case Studies

To evaluate the feasibility of the presented approach, we conducted two realistic large-scale case studies in cooperation with our industrial partners. Therefore, we first describe each of them in detail.

A. Study 1: Flight Control System

The avionic case study is an extended version of [18]. Due to certification and performance reasons, there is oftentimes still a *federated* architecture preferable rather than a centralized IMA architecture (integrated modular avionics). Therefore, the problem is to reduce the number of needed execution units (referred to as EXUs) as much as possible compared to today's "one execution unit per system function/software application" paradigm.

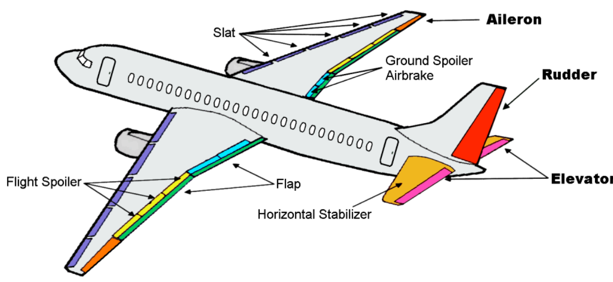


Fig. 2: Flight control system case study.

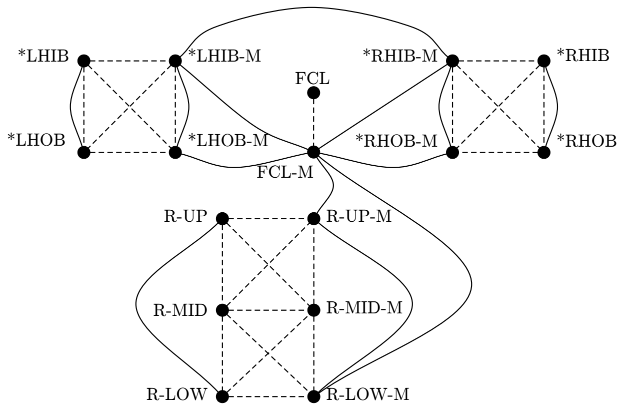


Fig. 3: Solid arcs connect two dislocal software applications, dashed arcs connect two dissimilar software application. The upper part (* in the names) has to be duplicated for each surface controlled by two actuators, the lower part is a template for those controlled by three (e. g. rudder).

1) Functional Description: The basic functionality of a flight control system of an airplane is to control the position of the flight surfaces depicted in Figure 2: aileron, elevator, and rudder are referred to as *primary* flight control surfaces, horizontal stabilizer, flap, slat, flight spoiler, and ground spoiler/air brake to as *secondary* flight control surfaces. The surfaces' position is calculated according to the information sent by the pilot via the cockpit control (e. g. side stick units and pedals) and synchronized in coordination with the other surfaces. The coordination uses so-called *flight control laws* that translate cockpit commands for the three axes (roll, pitch, and yaw) into corresponding surface movements respecting a stable, safe, and comfortable flight.

2) Software and Hardware Aspects: Due to safety requirements, each surface is controlled by at least two independent actuators. In the case of the rudder surface we refer to the *lower*, *middle*, and *upper* actuator. All others are referred to as *in board* (IB), i. e., next to the aircraft fuselage, and *out board* (OB), i. e., distant to the fuselage for each side of the aircraft (right hand (RH), and left hand (LH)).

TABLE I: Act. and SWAs

Function	Act.	SWAs
Rudder	3	6
Aileron	4	8
Elevator	4	8
FCL	-	2
CMS	-	1
Flight Spoiler	4	8
Ground Spoiler	4	8
Flaps	4	8
Slats	4	8
Horiz. Stab.	4	8
Total:	31	65

Each actuator is controlled by a software application which in turn is watched by a monitoring component (M). In case of a malfunction, the monitor deactivates the actuator control. As pointed out above, the translation of cockpit control commands into surface positioning is done by the flight control law software (FCL) application, which is also monitored. Less safety-critical information is stored and displayed to the pilot using a centralized maintenance system software application (CMS). Table I summarizes the number of actuators (devices) and (corresponding) software applications.

Note that the number of software applications is always twice as the number of actuators as there is always a primary software application and its monitoring component (except for the CMS component). Each of the 65 software applications has to be executed on an execution unit satisfying all given constraints. Today, in the avionics domain there is no fixed hardware topology. That means that executing units are connected using bus systems—usually CAN and ARINC 664 (AFDX)—*after* the software to hardware partitioning. This is different for instance to the automotive domain, where relatively fixed hardware topologies are common.

3) Avionic Safety Aspects: Due to safety considerations, we use redundant software-controlled actuators as well as software applications with both *dislocality* and *dissimilarity* constraints. Figure 3 depicts an example visualizing both dissimilarity (dashed arcs) and dislocality (solid arcs) constraints. The upper part of the image shows the constraints for software applications controlling redundant actuators (2 pieces) on both sides of the airplane. The star (*) in the names means, that this part has to be copied for each surface with two actuators, i. e., all surfaces except the rudder, which has three actuators. Dislocalities and dissimilarities for the rudder are depicted in the lower part of the figure. Moreover, in some cases software applications pose stricter DAL classifications than offered by the available execution platforms. In that case, less strict platforms are allowed if a complex condition (please refer to Listing 4) consisting of the number of years in use and the number of hours safety and other applications were executed on that platform without errors, i. e., the so-called *Product Service Experience* (PSE).

4) Problem Formulation: In traditional (classical) avionic architectures, a dedicated execution unit (single-core board) for each SWA was used. That would be 65 dedicated execution units for the presented case study. We examine, whether an architecture with less executing units compliant with the strict avionics safety requirements is possible. Less execution units help to reduce cost, weight and therefore the kerosene consumption per aircraft-passenger. Thus, we aim at optimizing the following objectives simultaneously:

O1 Minimize the number of used execution units.
O2 Maximize the number of *certified* execution units, i. e., those which are specially marked.
O3 Minimize the total hardware cost.

Listing 4: Safety considerations for COTS hardware components taken from the European Aviation Safety Agency Certification Memorandum [25]. The given PSE-values represent hours of aircraft application running time.

```
const PSE: forall (EXU e) where SWA s -> e: (s.DAL ~ e.DAL) or
  ((s.DAL== "A" ) and (e.yearsInUse >= 2.0) and                                                    /* DAL A */
    ((e.PSESafetyApplications + e.PSEOtherApplications) > 10^6) and
    ((e.PSESafetyApplications > 10^6) or ((e.PSESafetyApplications > 10^5) and (e.PSEOtherApplications > 10^7))))
  or
  ((s.DAL == "B") and (e.yearsInUse >= 2.0) and                                                    /* DAL B */
    ((e.PSESafetyApplications + e.PSEOtherApplications) > 10^6) and
    ((e.PSESafetyApplications > 10^5) or ((e.PSESafetyApplications > 10^4) and (e.PSEOtherApplications > 10^7))))
  or
  ((s.DAL == "C") and ((e.PSESafetyApplications + e.PSEOtherApplications) > 10^5));                /* DAL C */
```

TABLE II: Software and hardware properties. DAL and fabricator are considered but subject to confidentiality.

(a) Software properties

SWA	Period [ms]	WCET [ms]	RAM [kB]	ROM [kB]	DAL
*LHIB	3	1.5	50	200	X
*LHIB-M	3	1.5	100	100	X
*RHIB	3	1.5	50	200	X
*RHIB-M	3	1.5	100	100	X
R-UP	3	1.5	50	200	X
R-UP-M	3	1.5	100	100	X
R-MID	3	1.5	50	200	X
R-MID-M	3	1.5	100	100	X
R-LOW	3	1.5	50	200	X
R-LOW-M	3	1.5	100	100	X
FCL	10	5	250	450	X
FCL-M	8	5	500	100	X
CMS	100	50	150	400	X

(b) Hardware properties: We modeled 5 instances of each EXU type.

EXU	Fab.	RAM [kB]	ROM [kB]	DAL	PSE-Saf. [h]	PSE-Oth. [h]
EXU1	Fab1	1536	8192	X	$1.50 \cdot 10^4$	$1.00 \cdot 10^5$
EXU2	Fab2	512	8192	X	$1.25 \cdot 10^5$	$5.20 \cdot 10^6$
EXU3	Fab3	9216	2048	X	$8.50 \cdot 10^4$	$3.50 \cdot 10^6$
EXU4	Fab4	6144	4048	X	$1.55 \cdot 10^4$	$3.50 \cdot 10^5$
EXU5	Fab5	512	8192	X	$1.55 \cdot 10^5$	$1.03 \cdot 10^7$

At the same time the following constraint categories have to be considered:

C1 Capacity constraints for ROM and RAM
C2 Dissimilarity and dislocality
C3 DAL and PSE consideration
C4 Simple EDF scheduling test for each EXU e, i. e.,

$$\sum_{s \in \mathbf{S}_e} \frac{s.wcet}{s.period} \leq 1,$$

where $\mathbf{S}_e$ denotes the set of SWA allocated on EXU e.

B. Study 2: Seawater Desalination Plant

The second study was conducted with our partner Siemens. The modeled desalination plant has a typical configuration as found similarly in the seawater desalination plant Al Hidd in Bahrain [26].

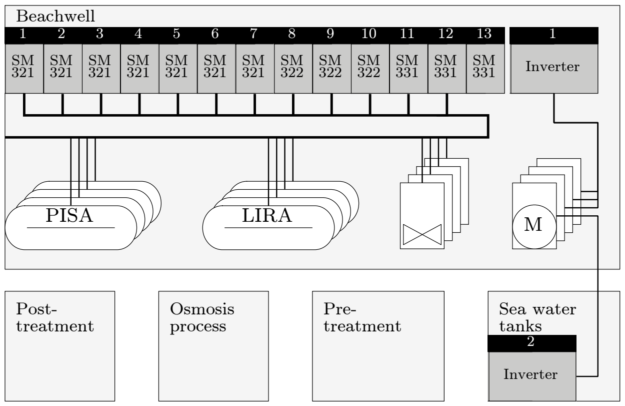


Fig. 4: Desalination plant consisting of several buildings, DIO groups, sensors, valves, inverters, and pumps. The beachwell part is exemplarily expanded.

1) Functional Description: The basic functionality of a desalination plant is to desalinate sea water gained using beachwells at sandy coastal areas. Pumps feed sea water from the wells into a complex automation plant. Through an osmosis process fresh water is produced. Many pressure and level sensors and valves control the transport and desalination process using low-power pumps.

2) Software, Hardware, and Spatial Aspects: The total automation system consists of the central PLC (programmable logic controller) structure and 25 groups of field I/O devices (DIOs (distributed input/output devices), sensors and actuators) spread over five buildings or plant areas, respectively. These are (i) the beachwell, (ii) the sea water tanks, (iii) the pre-treatment, (iv) the osmosis process, and (v) the post-treatment. Relevant attributes and parameters of the DIO groups and their spatial distribution are listed in Table III. Each sensor and valve uses 4 connection interfaces of a certain type: input or output, whereas each DIO (SM 321 and SM 322) offers 16 interfaces of a single type. Figure 4 depicts an exemplary plant configuration. There, sensors (PISA pressure and LIRA water level), valves, and pumps are connected to a group consisting of several I/O devices of different types. Inverter components are used to supply low-power pumps. As indicated, a power cable to another building or plant area is allowed if it is beneficial from both an electrical as well as economic point of view (with cost of 20 EUR per meter). Therefore, also the spatial plant distribution has to be considered. In contrary,

TABLE III: Seawater desalination plant configuration.

									Distances [m] to				
Name	ID	DIO groups	sensors	valves	pumps	inverters	SM321	SM322	1	2	3	4	5
Beachwell	1	2	16	24	8	2	8	6		200	250	250	300
Sea water tanks	2	5	40	60	20	6	18	15			70	90	100
Pre-treatment	3	6	48	72	24	5	21	18				20	60
Osmosis process	4	8	64	96	32	9	46	24					40
Post-treatment	5	4	32	48	16	6	20	12					5
Sum		25	200	300	100	28	113	75					

sensors and valves have to be connected to DIO groups within the same building. Note that the number of DIOs within a DIO group may vary depending on the used sensors and actuators. In our example only digital DIOs, i. e., SM 321 and SM 322, were used. To connect analogue sensors or actuators, the DIOs SM 331 and SM 332 are available. In practice configurations with 50% to 100% of the maximal number of DIOs are common.

3) Problem Formulation: We want to examine whether a desalination plant with a reduced number of DIOs and inverters compared to the existing reference automation plant is possible. We aim at optimizing the following objectives simultaneously:

O1 Minimize the number of used DIOs and thus minimize the DIO costs.
O2 Minimize the number of inverters.
O3 Minimize the power cable cost, i. e., when a pump is connected to an inverter located in another building.

At the same time the following constraint categories that are typical for industry automation have to be considered:

C1 Digital/analogue sensors/actuators shall be connected to compatible DIO types.
C2 Sensors and valves shall be connected to DIO within the same building or plant area.
C3 The number of available DIO/inverter interfaces shall be considered and must not be exceeded.
C4 The inverter components shall not exceed their designed power consumption.

In addition to the flight control system case study, also spatial distribution in the form of cable lengths is considered.

V. Evaluation

After having described the necessary details of both case studies, this chapter summarizes the experiments and their results. For the different solver categories (MOEA, SMT-based, and ILP-based), we asked ourselves several research questions:

RQ 1 What is the influence of having valid solutions in the initial population $\mathbf{P}$ and what do we gain of Pareto-optimal solutions in $\mathbf{P}$ in the MOEA approach?
RQ 2 Is it preferable to have more than one valid and accordingly Pareto-optimal solution in $\mathbf{P}$ and what are recommendable parameter settings for NSGA-II?
RQ 3 To what extent are the solving strategies applicable with respect to the case studies and how do they perform?
RQ 4 Which solver category is recommendable with regard to objectives and constraint types?

For the experiments, we use the Z3 [15] SMT solver and the Gurobi (M)ILP solver [16]. It turned out that Gurobi is many times faster than GLPK [22] and LP_SOLVE [21]. We used Eclipse and EMF to define our meta-model and Xtext for the DSL. Together, we realized an integrated modeling and optimization tool which we used to model the case studies.

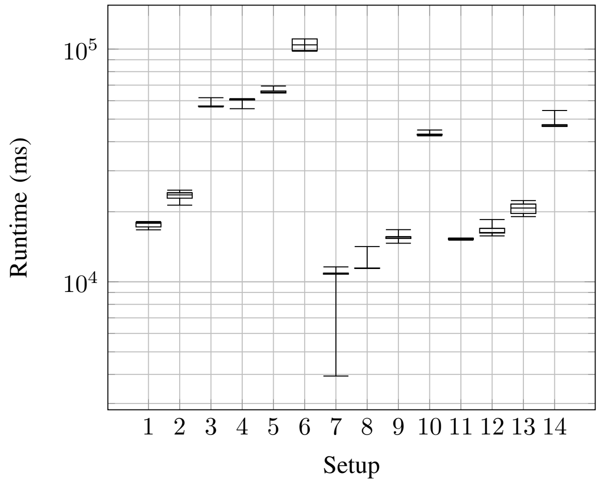
A. Study 1: Flight Control System

1) MOEA: During the experiments[1], we used a mutation probability of 0.01 and crossover probability of 1.0 for the NSGA-II algorithm. As the MOEA approach did not perform well even for the smaller FCS, we did not continue to pursue it for the second study. The NSGA-II algorithm did not find any valid solution (even without scheduling constraint **C4** and PSE consideration). Therefore, we implanted valid seedlings in the initial population $\mathbf{P}$.

a) Experiment 1: MOEA with IIPE: We varied the size of the initial population π, the number of implanted valid solutions ι, and the number of evaluations max_e and measured the runtimes five times each. Figure 5 depicts the settings and runtimes (min, max, median, lower and upper quartile). We can immediately answer **RQ 1**: Implanting valid solution seedlings into the initial population allows MOEA to return valid solution(s). However, we did not obtain any new solution but only the implanted, which we did not expect. We assumed that we would get new solutions, i. e., valid offspring solutions of the provided. When inserting more valid solutions (compare settings 7-10 with 11-14), the probability to insert “better” solutions increases. However, this implies longer runtimes for the SMT solver. Thus, **RQ 2** cannot be answered in a general way. For problem instances that can be solved by an SMT solver quickly, more valid solutions are preferable. But this has to be examined on a case by case basis. It did not pay off to evaluate more solution candidates, i. e., to have many offspring generations, hence **RQ 2** can be answered with respect to max_e: less evaluations are sufficient.

b) Experiment 2: MOEA with MGIA: In Section III-C1a, we proposed the MGI algorithm to find Pareto-optimal solutions using SMT solvers by guiding the solver towards Pareto-optimality. The found Pareto-optimal solutions were used as initial population similar to experiment 1. Again, MOEA did not find new Pareto-optimal solutions besides the

[1]MacBook Pro, OS X 10.10, 2.66 GHz Intel Core i7 dual core, 8 GB RAM



Setup	1	2	3	4	5	6	**7**	**8**	**9**	**10**	11	12	13	14
ι	10	10	10	10^2	10^2	10^2	1	1	1	1	10	10	10	10
π	10^2	10^2	10^2	10^2	10^2	10^2	10	10	10	10	10	10	10	10
max_e	10^3	10^4	10^5	10^3	10^4	10^5	10^2	10^3	10^4	10^5	10^2	10^3	10^4	10^5

Fig. 5: Parameter settings and corresponding runtimes.

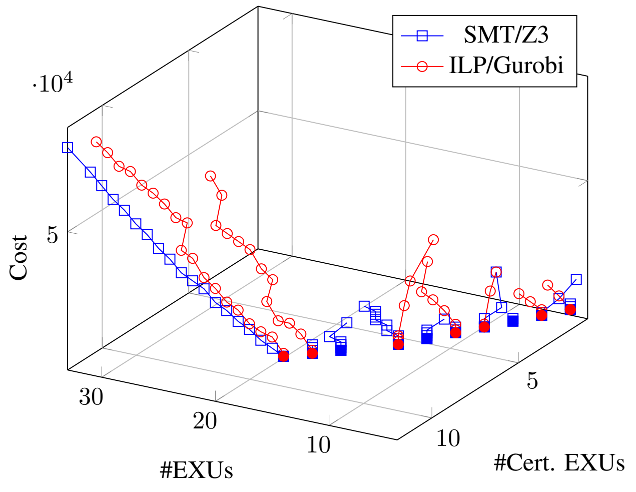


Fig. 6: Pareto-optimal solutions for Study 1. Solid marks depict Pareto-optimal solutions. The paths leading to those points indicate the iterative convergence towards Pareto-optimality.

already inserted ones. Research questions **RQ 1** and **RQ 2** are thus answered as follows: Pareto-optimal solutions do not have any impact on other solutions—at least in our experiments. MOEA only returns the already inserted ones. Hence, one should only use the SMT-based approaches without MOEA for large system models.

2) SMT- vs. ILP-based Approach: For the experiments[2], we set the maximum number of solutions to 10 and a timeout (TO) of one hour. The results are given in Table IV. Different constraint combinations were investigated, since not all are

TABLE IV: Runtimes and the number of solutions in brackets for different settings.

Constraints	SMT+MGIA [min] (#)	ILP+MGIA [min] (#)	ILP+WS [min] (#)
C1, C2	52.7 (10)	1.34 (7)	0.08 (1)
C1, C2, C3	45.8 (10)	—	—
C1, C2, C3, C4	TO	—	—
C1, C2, C4	TO	0.02 (1)	0.006 (1)

supported by ILP (—). The modified guided improvement algorithm was able to find Pareto-optimal solutions for ILP and SMT in the setting C1, C2. Figure 6 depicts the found solutions. When considering the experimental results, we can answer research questions **RQ 3** and **RQ 4**: when having logical expressions in capacity-*constraints* in the **SAOL** specification, currently SMT-based approaches are inevitable; without them, ILP-based approaches offer a clear performance advantage. MGIA-based approaches are preferable when diverse Pareto-optimal solutions are desired.

B. *Study 2: Seawater Desalination Plant*

For this study we only report on results of the ILP-based approach, because the SMT-based approach did not find a Pareto-optimal solution within three weeks (non-optimal solutions were found within minutes).

a) Experiment 1: ILP with Weighted Sum Approach: **Setting**: We set the weights as follows: $w_{NoOfDIO} = 0.005$ (O1), $w_{NoOfInverter} = 0.005$ (O2), and $w_{CableCost} = 0.99$ (O3).

Result: It took 5 seconds[2] to find an optimal (with respect to the given weights) solution: 125 DIOs, 25 inverters, and 5.200 monetary units for cabling. Hence, we could find a solution which is 26% cheaper compared to the initial model of the desalination plant (not even considering possible cabling cost for the initial model).

b) Experiment 2: ILP with MGIA: Objective function weights are not of relevance for the MGIA, thus not used. We found a Pareto-optimal solution for the desalination plant within 11.6 minutes. This solution is a result of 104 solver calls, which incrementally reduce the search space. Actually this is the same solution as in experiment 1, thus in this case both algorithms found a Pareto-optimal solution. In contrast to experiment 1, which found a solution quickly, ILP with MGIA took much longer. However, in cases of conflictive objectives, this approach yields several *diverse* solutions. For system architects, this is a very helpful basis of decision-making.

VI. Conclusion and Future Work

In this paper, we presented language extensions to the *System Architecture Optimization Language* fitting industrial needs. **SAOL** allows for a clear, precise, unambiguous, and at the same time understandable specification of design goals and domain-specific conditions. Moreover, we introduced the *Modified Guided Improvement Algorithm* using both SMT and ILP solvers to find Pareto-optimal solutions for multi-criteria design space exploration problems. Furthermore, we reported

[2]MacBook Pro, OS X 10.10, 2.8 GHz Intel Core i7 quad core, 16 GB RAM

that—in our experimental study—the pure NSGA-II multi-objective evolutionary algorithm never returned a satisfiable solution to the deployment problem. With implanted satisfiable seedlings, however, MOEA did find solutions, but never better ones than the implanted. In conclusion, the MGIA approaches turned out to be the best choice for the conducted studies when the designer is interested in a variety of different solutions. In both studies, we were able to find deployment solutions satisfying all constraints with less needed hardware and thus being more cost-effective. As a lesson learned, we recommend not to use MGIA with a "150%" model, but rather find first valid solution(s) using IIPE in order to get a feeling of how much capacity is needed and then reduce the model according to these insights. This reduces the design space and makes practical use possible. Currently, the use of the logical **or** in capacity-*constraints* is only supported by the SMT-based approaches. Hence, a consequent next step is to investigate and realize support of disjunctive constraints in integer linear problems. There is no silver bullet for the deployment problem for all application areas. Deciding which approach is best suited, depends on the size of the model, type of constraints, and whether there are conflictive objectives or not. The SMT and ILP-based approaches seem to be most promising for industrial size models, while even for smaller sized models MOEA performs rather poor. All approaches were implemented in an integrated Eclipse-based modeling and engineering tool facilitating the optimization of embedded system architectures.

A threat to the validity of our result is given by the limited number of conducted studies. However, in instead of having many small studies, we took two big industrial size studies from different engineering domains, which should be good representatives for embedded systems.

In the future, we want to evaluate both the **SAOL** language expressiveness and the solver capabilities by conducting further experiments with examples from other application domains. A comparison between the presented approach and specifically crafted solution algorithms would be interesting, too.

References

[1] M. Eisenring, L. Thiele, and E. Zitzler, "Conflicting criteria in embedded system design," *IEEE Design & Test of Computers*, vol. 17, no. 2, pp. 51–59, 2000.

[2] F. Hölzl and M. Feilkas, "Autofocus 3: a scientific tool prototype for model-based development of component-based, reactive, distributed systems," in *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems*, ser. MBEERTS'07. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 317–322. [Online]. Available: http://dl.acm.org/citation.cfm?id=1927558.1927576

[3] 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*, vol. 0. IEEE Computer Society, 2009, pp. 61–71.

[4] S. Kugele and G. Pucea, "Model-based optimization of automotive e/e-architectures," in *Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis, CSTVA 2014, Hyderabad, India, May 31, 2014*. ACM, 2014, pp. 18–29.

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

[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*. IEEE Computer Society, 2006, pp. 365–368.

[7] 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 *IJCAI*, M. M. Veloso, Ed., 2007, pp. 80–87.

[8] J. M. Spivey, "An introduction to Z and formal specifications," *Softw. Eng. J.*, vol. 4, pp. 40–50, January 1989.

[9] 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*, ser. CAV'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 441–456.

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

[11] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske, "Reliability-driven deployment optimization for embedded systems," *Journal of Systems and Software*, vol. 84, no. 5, pp. 835–846, 2011.

[12] R. Kumar, K. Izui, M. Masataka, and S. Nishiwaki, "Multilevel redundancy allocation optimization using hierarchical genetic algorithm," *IEEE Transactions on Reliability*, vol. 57, no. 4, pp. 650–661, 2008.

[13] 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*. New York, NY, USA: ACM, 2012, pp. 2:1–2:6.

[14] D. Rayside, H.-C. Estler, and D. Jackson, "A guided improvement algorithm for exact, general purpose, many-objective combinatorial optimization," MIT Computer Science and Artificial Intelligence Laboratory, Tech. Rep. MIT-CSAIL-TR-2009-033, 2009.

[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*, ser. TACAS'08/ETAPS'08. Springer-Verlag, 2008, pp. 337–340.

[16] I. Gurobi Optimization, "Gurobi optimizer reference manual," http://www.gurobi.com, 2015.

[17] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable product line configuration: A straw to break the camel's back," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 465–474.

[18] R. Hilbrich and L. Dieudonné, "Deploying safety-critical applications on complex avionics hardware architectures," *Journal of Software Engineering & Applications*, vol. 6, no. 5, 2013.

[19] RTCA DO-178C, "Software considerations in airborne systems and equipment certification," 2011.

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

[21] M. Berkelaar, K. Eikland, and P. Notebaert, "lp_solve 5.5, open source (mixed-integer) linear programming system," Software, May 1 2004. [Online]. Available: \url{http://lpsolve.sourceforge.net/5.5/}

[22] "GNU Linear Programming Kit, Version 4.41," http://www.gnu.org/software/glpk/glpk.html.

[23] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[24] N. Bjørner and A.-D. Phan, "νZ - Maximal Satisfaction with Z3," in *SCSS 2014*, ser. EPiC Series, T. Kutsia and A. Voronkov, Eds., vol. 30, 2014, pp. 1–9.

[25] EASA: European Aviation Safety Agency, http://easa.europa.eu/system/files/dfu/certification-docs-certification-memorandum-EASA-CM-SWCEH-001-Issue-01-Rev-01-Development-Assurance-of-Airborne-Electronic-Hardware.pdf.

[26] Siemens AG, "Seawater Desalination Plant Al Hidd, Bahrain," http://w3.siemens.com/mcms/water-industry/en/reference-center/Documents/Seawater_Desalination_Plant_Al_Hidd_Bahrain.pdf, 2009.