Optimal Product Line Architectures for the Automotive Industry

Tobias Wägemann[1], Ramin Tavakoli Kolagari[2], Klaus Schmid[3]

Abstract:

The creation of product line architectures is a difficult and complex task. The resulting architectures must support the required system variabilities as well as further quality attributes. In the automotive domain, product lines of software-intensive system models have a great diversity of products, which leads to vast design spaces. Finding optimal product line architectures as part of the system design process requires the consideration of a variety of trade-offs. In practice, this challenge cannot be solved manually for all but the smallest problems, therefore an automated solution is required. Our contribution is the generation of a sound mathematical formalization of the problem. This formalization makes the product line optimization problem accessible to various established multi-objective optimization techniques. The applicability of the chosen approach is shown by means of applying a commercial tool for multi-criteria decision making.

Keywords: architecture optimization; multi-objective; variability; software product lines; automotive

1 Introduction

An important activity in product line engineering is the development of an adequate product line architecture [CN01, Sc03, Ti12]. Due to the great diversity of products that must be taken into account in this process, finding an optimal product line architecture is a complex and error-prone task. This is particularly the case when performing optimization of product line architectures of software-intensive systems (as opposed to product line architectures of software systems) as this must also account for objectives like weight or production cost. Creating automated support for architecture optimization in such a context can be extremely helpful, as it supports architects in navigating the complex and vast design space which constitutes the basis for a multi-objective decision making problem.

In this paper, we present an approach for product line architecture optimization of software-intensive systems based on EAST-ADL [Bl13], a domain-specific architecture description

[1] Technische Hochschule Nürnberg, Keßlerplatz 12, 90489 Nuremberg, Germany tobias.waegemann@th-nuernberg.de

[2] Technische Hochschule Nürnberg, Keßlerplatz 12, 90489 Nuremberg, Germany ramin.tavakolikolagari@th-nuernberg.de

[3] Universität Hildesheim, FB 4, Institut für Informatik, Universitätsplatz 1, 31141 Hildesheim, Germany schmid@sse.uni-hildesheim.de

language for the automotive industry. As opposed to other work in this area [RRV16], we provide a full mathematical formalization of the optimization problem. In a previous publication we presented a concept for formalizing product line variability [WW15] as a basis for exploring the design space. Here, we extend our previous work by including design goals and variable realization elements (system components) in the formalization. We also created a prototypical implementation of our approach that uses an off-the-shelf optimization tool for solving the formalized problems. Since our approach builds on EAST-ADL system models as the basis for optimization, we must also note that creating such a model in the first place (including the aggregation of all relevant data) is a significant challenge in itself. However, the language and respective system models are already in industrial use and there are concerted efforts to further promote the industrial application of EAST-ADL by the automotive industry.

While the ideas behind our approach are in principle generic and can be transferred to other layered approaches for representing product line architectures, our implementation focuses on the use of models defined in EAST-ADL. The EAST-ADL language was not developed for (product line) architecture optimization in particular, but provides modeling techniques for automotive systems, including techniques for variation modeling. As a consequence, it contains constructs for variability representation, but not for describing an optimization space. We therefore have to explicitly differentiate between actual *product line variability* and the *architectural degrees of freedom* (i.e., the design space of the product line architecture, cf. Section 3.3). Since both are represented using the same language constructs in EAST-ADL, we introduce a method to differentiate between the two kinds of variation.

The paper is structured as follows: Section 2 gives an overview of the related work and shows the differences between our approach and existing work. Section 3 describes the domain-specific architecture description language (EAST-ADL) used in our work. Section 4 discusses the encoding of the architecture optimization problem as a search problem. The method used for deriving solutions is described in Section 5. Section 6 illustrates our approach by means of a small case study. Finally, Section 7 concludes the paper and describes our plans and ideas for future work.

2 RELATED WORK

There is a range of other work being conducted on the multi-objective optimization of product line system architectures and search-based system design in general. In regard to optimization approaches based on the EAST-ADL language specifically, to our knowledge only one other approach has been realized and published. Walker et al. [Wa13] present an optimization approach based on multi-objective genetic algorithms which considers system dependability, timing concerns and a simple cost metric. The approach uses HiP-HOPS[4] for fault tree analysis and MAST[5] for response time computation and is tightly coupled to these

[4] http://hip-hops.eu
[5] http://mast.unican.es

external solutions for the evaluation of objectives. A similar optimization approach for cost and dependability is presented by Mian et al. [Mi15] for the AADL[6] language, also using HiP-HOPS for fault tree analysis.

Thüm et al. [Th12] present a classification framework for product line analysis strategies to provide systematic access and guidance to the research in this particular field. They divide the analysis strategies into four different categories: product-based, family-based and feature-based analysis, as well as techniques that use combinations of these three. Our approach operates solely on domain artifacts of the product line and can be classified as a family-based analysis strategy in regard to the classifications introduced by this work.

Colanzi et al. present a number of publications in the field of search-based product line design by means of multi-objective evolutionary algorithms. Their work includes an exploratory study of applying search-based design methods to the SPL-context [CV12], the introduction of a novel search-based approach for PLA-design based on PLA-specific metrics by the name of MOA4PLA [Co14], as well as the introduction of a feature-driven crossover operator for PLA optimization using genetic algorithms [CV16].

Aleti et al. [Al13] give a broad overview of common architecture optimization methods used in published work and present a taxonomy for the classification of existing research, based on the three categories Problem, Solution, and Validation. Lopez-Herrejon et al. [LHLE15] present a systematic mapping study on research regarding the application of search-based software engineering methods to the realm of software product lines. The study focuses on the type of employed SBSE techniques, the stage of the affected SPL life-cycle, commonly used validation methods and the specific forums for publication. Ramírez et al. [RRV16] present a comparative study of multi-objective evolutionary algorithms for software architecture optimization. This publication therefore centers on the internals of evolutionary architecture optimization, including an empirical exploration of the behavior of a set of selected multi- and many-objective algorithms in regard to predefined optimization problems with up to nine objectives.

Our approach integrates aspects from three different research fields: software product lines, system architecture modeling and mathematical optimization. In light of related research in this area, our approach is distinct by a combination of the following characteristics: (a) The result of our optimization is not an optimal product but a product line architecture with optimal architectural decisions. (b) The use of multi-objective integer linear programming (MOILP) as a rigorous mathematical formulation of the optimization problem. (c) Adaptability towards tools for optimization and multi-criteria decision making (MCDM) (cf. Section 5).

[6] http://www.aadl.info

3 ARCHITECTURE MODELING APPROACH

EAST-ADL is a domain-specific architecture description language with a focus on capturing all relevant information to represent variant-rich software-intensive systems in a standardized way. The language was developed in a series of European research projects with strong participation of the automotive industry[7] and applied/enhanced by a number of national and international research projects[8]. Today the EAST-ADL is managed by the EAST-ADL association[9]. The language has been tailored towards compatibility with the well-established AUTOSAR standard[10], which in turn serves as an integral part of the EAST-ADL language by realizing one of its abstraction layers.

This section gives an overview of the EAST-ADL; details about the language can be found in the EAST-ADL white paper [Bl13] and in the language specification[11]. Section 3.1 provides an overview of EAST-ADL language; Section 3.2 examines the EAST-ADL variability modeling approach, and Section 3.3 explains the distinction between product line variability and architectural degrees of freedom.

3.1 EAST-ADL—an Architecture Description Language

EAST-ADL defines a language for modeling automotive systems; the implementation of these systems is then managed by AUTOSAR in the aforementioned tight coupling with EAST-ADL. A major advantage of the EAST-ADL language is the organization of the system model along predefined abstraction levels (cf. Figure 1): Abstraction is not only supported in principle—as is often the case for ADLs—but is enforced by the system model with defined semantics for each level of abstraction. As a consequence, engineering information is structured in accordance to a reference methodology based on the V-Model that is widely used in the automotive domain. On each level of abstraction the system is complete from a given perspective: from a very abstract representation at higher levels to increasingly detailed representations at lower levels.

The system model is complemented by several extension packages that allow for modeling requirements, variability, timing, and dependability (cf. Figure 1). Most of the extensions are applicable on all levels of abstraction and are adapted to specific modeling needs of the automotive domain in general and EAST-ADL in particular; while the extensions heavily rely on the "core" system model, the system model itself is independent of the extensions,

[7] ITEA EAST-EEA (http://www.itea3.org/project/east-eea.html), ATESST, ATESST2 (http://www.atesst.org), MAENAD (http://www.maenad.eu)

[8] Artemis CESAR (http://www.cesarproject.eu), ITEA2 SAFE (http://www.safe-project.eu), Artemis MBAT (http://www.mbat-artemis.eu), to name a few

[9] http://www.east-adl.info

[10] http://www.autosar.org

[11] The EAST-ADL meta-model is published by the EAST-ADL Association: http://east-adl.info/Specification/V2.1.12/html/

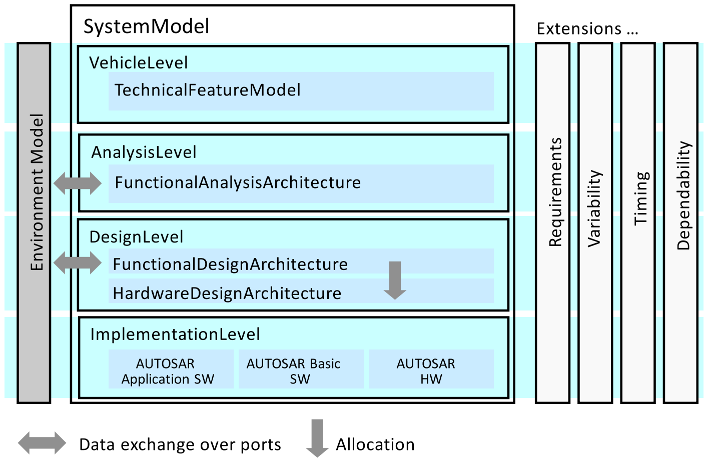

Fig. 1: EAST-ADL System Model [Bl13, p. 4]

such that extension modeling may be plugged in and out as required. Traceability with full support of SysML semantics is supported for all modeling entities, i.e., between different levels of abstraction as well as within a given abstraction level and to the extensions.

3.2 Variability Modeling Concepts in EAST-ADL

Managing variability is at the core of software product line engineering, it occurs because some aspect of a system may change from one variant of a system to another. EAST-ADL, as a decidedly automotive-specific language, takes these challenges into account and offers variability modeling techniques applicable for the car manufacturer and for the supplier. Its variability management starts on Vehicle Level (cf. Figure 1), where the external (i.e., user-centric/abstract) variability, as well as the product-line-strategy variability is described by cardinality-based feature models [CHE05]. The impact of the described variability is then defined on the Analysis and Design Levels, respectively. Traceability links the abstract/root view on Vehicle Level to the variability impact on lower levels of abstraction, i.e., the artifact levels. While the details of how variability is actually realized in the system are largely suppressed on the Vehicle Level, they are the focus of attention when managing variability on the artifact levels. The artifact levels are Analysis Level, Design Level and the extensions, where variability may occur as well. For example, one may think about the different requirements for a rain sensor with and without a rain light sensor. Variability is described in two ways on the artifact levels:

Feature models used on Analysis and Design Level get a much more concrete meaning as compared to the feature model on Vehicle Level in order to reflect detailed internal variability as concrete implementation of the variability on the Vehicle Level. Configuration decisions [RTW09] link feature models to one another with the purpose of defining configuration, both within one abstraction level and across abstraction levels. As a consequence of linking artifact level variation to Vehicle Level feature models by means of configuration decisions, all major variability configuration is essentially controlled by the Vehicle Level.

Explicit variation is used to denote that modeling entities may be optional, i.e., be deleted from the system model. Dependencies among variable entities are captured in terms of variation groups. This integrated way of modeling variability can also be linked to configuration decisions such that the (partial) configuration of this variability is guided by feature models on a hierarchy level one step higher and ultimately by the feature models on Vehicle Level.

3.3 Product Line Variability versus Architectural Degrees of Freedom

For the purpose of this paper it is essential to understand the distinction between two kinds of (architectural) variabilities: 1. *Product line variability* [MP07, CN01] describes the variations regarding components (or modules) of proper products that are well-formed with respect to the product line design space. 2. *Architectural degrees of freedom* refer to potential alternatives for designing the product (line) architecture. In other words, the result of configuring all architectural degrees of freedom would be a product line architecture (PLA), whereas the result of configuring all product line variability would be a product. The architectural degrees of freedom are the basis for our optimization process, which intends to produce an optimal product line architecture with respect to (multiple) criteria chosen by an architect. Therefore, at the end of our optimization process, no architectural degrees of freedom remain in the system model and the remaining architectural variability is governed only by the product line design space.

Although product line variability and architectural degrees of freedom indeed both describe variability, their purpose and their role, e.g., in an architecture optimization process, differ considerably. It would therefore be useful for an architecture description language to manage these variablities differently. As the EAST-ADL does not support this distinction (similar to all other established modeling languages we are familiar with), we use the variability modeling concepts of EAST-ADL for both and differentiate them as follows: we define all artifact variability that can be traced up to the Vehicle Level by means of configuration decisions as product line variability (i.e., all variability that is rooted on Vehicle Level is product line variability). Artifact variability without traceability to the Vehicle Level (i.e., variability that is introduced only at artifact levels) stands for architectural degrees of freedom; such variability is not configured as part of a product line configuration, but is instead decided at the time of system design. Identifying optimal design decisions for the architectural degrees of freedom is the primary objective of our optimization approach.

The chosen approach of distinguishing between product line variability and architectural degrees of freedom is advantageous in that it can be applied to any (variability) modeling approach that supports abstraction, which is indeed common. In the context of the EAST-ADL the chosen approach is especially elegant because of the predefined root abstraction level, i.e., the Vehicle Level.

4 TRANSLATION INTO AN OPTIMIZATION PROBLEM

A general definition of an optimization problem is the problem of finding the best solution in regard to specific criteria from all feasible solutions. Mapped to our problem domain, this definition translates to finding the best product line architectures within the optimization space defined by the architectural degrees of freedom as described by the EAST-ADL system models. The use of EAST-ADL models for architecture optimization requires to formalize all optimization-relevant system information in a way that is sufficient for optimization purposes.

In this section we present our formalization approach for variant-rich EAST-ADL system models, which involves (a) the identification of all model elements relevant for the optimization process, (b) an evaluation of the characteristics of these elements in regard to the intended optimization goals and (c) a generation of a mathematical formulation, which constitutes the basis of our optimization process. We also describe how we handle optimization problems with multiple objectives as part of our approach. It is not the goal of our approach to fully automate architecture definition.

4.1 Multiple Design Objectives

When considering multiple design objectives in a non-trivial optimization process, there is usually no solution that is truly optimal for each of the objectives simultaneously. This is caused by conflicts among the objectives, e.g., making the system more lightweight will likely increase costs, etc. There are two different ways of handling this issue, resulting in two different approaches to multi-objective optimization.

One possibility is to turn the initial multi-objective problem into a pseudo-single-objective problem, by aggregating all considered objectives into a single weighted objective function. The resulting single solution of this approach is optimal in regard to the predefined weights used for the design objective aggregate, which have to be determined before the optimization. This kind of approach is called scalarization or weighted normalization [GR06].

The other possibility, and the one we use in our approach, is to consider all design objectives simultaneously in a specialized optimization process called *Pareto* optimization. The result of a Pareto optimization is not a single solution, but a set of Pareto-optimal solutions, called the *Pareto front*. Pareto optimality is based on the concept of dominance; a solution is called non-dominated—and is thus part of the Pareto front—if there are no other solutions that are better in at least one objective without degrading one or more of the other objectives [BK05, p. 414ff].

All solutions that are part of the Pareto front, i.e., all dominant solutions, are in principle equally optimal. Selecting the most suitable solution from the Pareto set is therefore subject to a trade-off analysis. This step is dependent on the expertise and the end goals of the user, typically an architect intending to find the architecture that best suits a specific set of requirements. From the perspective of an architect, our approach is therefore a means to efficiently explore the design space of system models, which guarantees that a chosen architecture is Pareto-optimal in regard to the considered design objectives.

4.2 Formalization Approach

In order to establish a sound basis for the exploration of an architecture optimization space, all optimization-relevant information of a given variant-rich system model must be formalized into a rigorous mathematical form. Our optimization problems have binary decision variables and multiple design objectives. Therefore, our problem domain is that of multi-objective integer linear programming (MOILP) with all variables $\in \{0, 1\}$. In order to translate our optimization problems into MOILP form, we first assign all relevant variable elements to numbered decision variables $x_1 ... x_n$. We can then formulate the program as follows:

$$\begin{aligned} \text{Minimize} \quad & Cx \\ \text{subject to} \quad & Ax \geq a_0 \\ & x \in \{0, 1\}^n \end{aligned} \tag{1}$$

where C is a (m, n)-Matrix of design objective values, A and a_0 are a (p, n)-matrix and a p-vector representing a set of constraints which maps the optimization space and x is an n-vector of binary decisions variables; with m being the number of design objectives, n being the number of decision variables and p being the number of program constraints. The matrix Cx translates to a set of linear objective functions $F(x) = (f_1(x), f_2(x), ..., f_m(x))^T$, which represent the pursued design objectives.

First of all, a formalization approach must be able to distinguish between product line variability and the system's architectural degrees of freedom (cf. Section 3.3). The intended output of our optimization approach is a product line with Pareto-optimal architectural decisions, not a Pareto-optimal configuration of the product line, i.e., not a product. Therefore, it must be possible to omit all product-line-related variability from the constraint formalization, so that it doesn't get resolved as part of the optimization process. Our approach accomplishes this distinction between product line variability and architectural degrees of freedom by an evaluation of the language traceability across the abstraction levels of the EAST-ADL model. If a variation point is part of the system's product line variability, it must be possible to trace its origin up to the model's Vehicle Level, where the product line design space is defined by means of feature models. If however the trace ends below the Vehicle Level, the variation point must necessarily be part of the architectural degrees of freedom instead. Using this distinction, we assign decision variables to the variable elements of the architectural degrees of freedom.

In order to generate the objective functions from our variant-rich EAST-ADL models, we parse a type of native EAST-ADL language annotations called *GenericConstraints*. Using predefined *GenericConstraintKinds* like *weight* or *piece cost*, GenericConstraints can be used to annotate quantifiable quality attribute information to elements—including variable elements—of the model. The set of possible objectives is therefore defined by the available GenericConstraintKinds in the EAST-ADL language specification. To generate the objective functions for these annotations, we allocate the numeric values of the GenericConstraints (of one specific GenericConstraintKind) to the previously introduced decision variables. In doing so, we produce linear objective functions in the form of $f(x) = c^T x$, where c^T is the transposed vector of the numeric values of the GenericConstraints for the variable elements associated with the decision variables x. Performing this step for all pursued design objectives produces a set of linear objective functions.

Next, we formalize the variability information into program constraints. Having established a way of filtering out the (for the formalization process) unwanted product line variability, the formalization of desired variability (i.e., architectural degrees of freedom) into constraints for our MOILP is done by applying a set of transformation rules based on an intermediate conversion into propositional logic. These transformation rules were presented in detail in one of our former publications [WW15], which was focused solely on a method for generating propositional constraints from variability descriptions. In this paper we incorporate this method into a fully-fledged multi-objective optimization approach. Table 1 gives a summary of the rules and Section 6.2 demonstrates their application by means of a case study. With the formalization of (a) quality attributes into design objectives and (b) variability information into program constraints in place, we can now assemble full MOILP representations of optimization problems for variant-rich EAST-ADL models. Our implementation generates these MOILPs in the standard formats of OPL[12] and AMPL[13].

5 SOLVING OUR OPTIMIZATION PROBLEM

We use our formalization of the optimization problem as input for third-party optimization tooling. Our tool of choice is the commercial optimization software FINNOPT[14], which provides a human decision maker with an interactive process for finding the most preferred compromise among all Pareto-optimal solutions of a multi-objective optimization problem. The FINNOPT approach is inherently iterative and allows the user to guide the process towards preferred solutions as part of a trade-off analysis. FINNOPT is based on the

[12] https://www-01.ibm.com/software/commerce/optimization/modeling
[13] http://ampl.com
[14] http://www.finnopt.com

Variability		Propositional Logic	Program Constraints
Feature Tree	Feature has parent	$f \longrightarrow f_{parent}$	$f_{parent} - f \geq 0$
	Feature is excluded	$!f$	$(1 - f) = 1$
	Feature Group	for all m: $f_{parent} \longrightarrow M_m(f_1, .., f_n)$	for all m: $M_m(f_1, .., f_n) - f_{parent} \geq 0$
Feature Link	needs	$f_{start} \longrightarrow f_{end}$	$f_{end} - f_{start} \geq 0$
	optional alternative	$!(f_{start} \wedge f_{end})$	$f_{end} + f_{start} \leq 1$
	mandatory alternative	$f_{start} \oplus f_{end}$	$f_{start} + f_{end} = 1$
Variation Group	needs	$f_1 \longrightarrow (f_2 \wedge f_3 \wedge \ldots \wedge f_n)$	$\bigwedge_{k=2}^{n} (f_k - f_1 \geq 0)$
	optional alternative	for all m: $M_m(f_1, .., f_n)$	$f_1 + f_2 + \ldots + f_n \leq 1$
	mandatory alternative	$f_1 \oplus f_2 \oplus \ldots \oplus f_n$	$f_1 + f_2 + \ldots + f_n = 1$
Configuration Decisions		*criterion* $\longrightarrow$ *effect*	*effect* – *criterion* ≥ 0

Tab. 1: Overview of our transformation rules for EAST-ADL system variability [WW15].

IND-NIMBUS [Mi06] software that was developed by the Industrial Optimization Group of the University of Jyväskylä, Finland[15]. The tool integrates an external ILP-solver and utilizes it as part of its process. For this purpose we use the commercial solver CPLEX that is part of the IBM ILOG CPLEX Optimization Studio[16] for mathematical optimization.

FINNOPT is well-suited to the task of identifying solutions that are both Pareto-optimal in regard to a preferred emphasis on specific design objectives and useful for a system architect. The software is able to handle large and complex optimization problems and has a user interface that is well suited to analyzing trade-offs for system architectures. However, since we generate our problem formalization in standardized formats (OPL, AMPL), the FINNOPT-based tool setup can in principle quite easily be exchanged with alternative optimization tools for multi-criteria optimization and decision making.

[15] http://www.mit.jyu.fi/optgroup
[16] http://www.ibm.com/software/products/en/ibmilogcpleoptistud

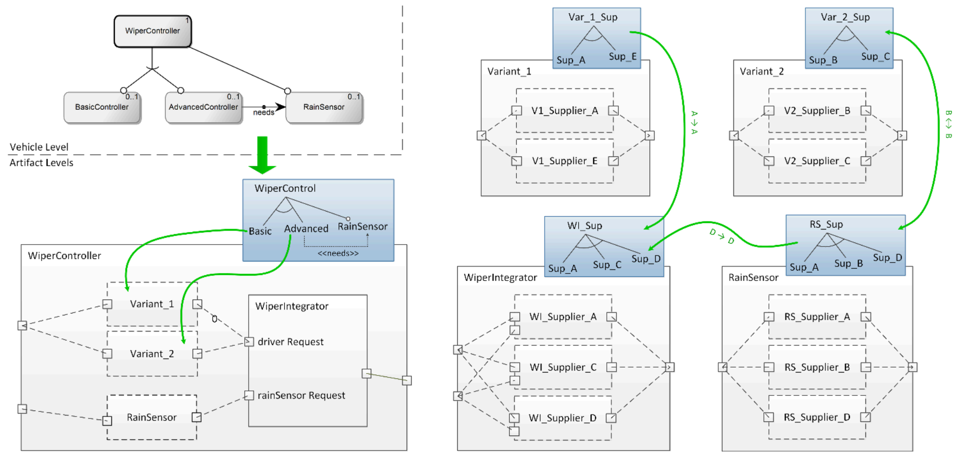

Fig. 2: The wiper control system demonstration model.

6 CASE EXAMPLE

We demonstrate the application of our architecture optimization approach by means of an example. The demonstration model is an extended version of an existing EAST-ADL example, which was previously used for showcasing the variability language concepts of the language in the MAENAD project[17]. While the demonstration model is smaller than real-world system models, it adequately serves the purpose of demonstrating our approach in the context of this paper. We will first give an overview of the model's structure and its contained variabilities. We then demonstrate our MOILP-based formalization approach, before the formalized program is used to explore the optimization space. The resulting Pareto-optimal product line architectures are used for a discussion of the correctness and consistency of the solutions and the adequacy of our optimization approach in general.

6.1 The Demonstration Model

The model used for this demonstration is illustrated in Figure 2. The system shows a product line architecture for the control electronics of a windscreen wiper. To keep the example concise, the demonstration model does not show the full EAST-ADL realization, but gives an overview of all elements that are relevant to the optimization process and makes a clear distinction between elements on the Vehicle Level and those on the artifact levels of the model. The level of abstraction used here also coincides with the level of abstraction used in the source document of the model. In this representation, variable elements are indicated by dashed lines, public feature models of containers are shown at the top right of the containers,

[17] MAENAD Concept Presentation on EAST-ADL Variability: http://www.maenad.eu/public/conceptpresentations/6_Variability_EAST-ADL_Introduction_2013.pdf

	weight / g	*cost / €*	*power consumption / W*
V1_Supplier_A	122	3.30	2.80
V1_Supplier_E	131	3.25	2.75
V2_Supplier_B	154	5.32	3.10
V2_Supplier_C	154	5.90	3.15
WI_Supplier_A	155	3.54	2.60
WI_Supplier_C	167	3.51	2.55
WI_Supplier_D	158	3.58	2.50
RS_Supplier_A	143	10.65	5.50
RS_Supplier_B	150	10.82	5.50
RS_Supplier_D	126	11.46	5.45

Tab. 2: (Excerpt of) quality attribute values for system components from different suppliers.

and (non-obvious) configuration decisions are represented by arrows from public feature models to configuration targets.

The system can be configured in either a basic or an advanced configuration, which toggles between two different system variants V1 and V2. The system further includes an optional rain sensor that is mandatory for the advanced configuration. These variation points are part of the product line design space.

In addition, the demonstration model also contains variabilities that describe degrees of freedom for the architecture. In this example, these variability descriptions represent alternative—but functionally identical—components from different suppliers and the interdependencies among them. The basic variant V1 and the advanced variant V2 each have two alternative components, the rain sensor and the wiper integrator each have three. The alternative components from different suppliers have varying weight, cost and power consumption (cf. Table 2). The following constraints exist:

- The basic variant V1 from supplier A needs the wiper integrator from supplier A; e.g., because it is a built-in feature of the wiper integrator.
- The advanced variant V2 from supplier B and the rain sensor from supplier B need each other; e.g., because they are sold as a set and not as separate modules.
- The rain sensor from supplier D needs the wiper integrator from supplier D; e.g., because supplier D uses a non-standard proprietary connector.

6.2 Problem Formalization

In order to conduct an automated optimization of the architecture, we first need to formalize all optimization-relevant data of the demonstration model. The first step of formalization is to assign ordered decision variables to all variable components of the model: $x_1 = V1\_Supplier\_A$, $x_2 = V1\_Supplier\_E$, $x_3 = V2\_Supplier\_B$, $x_4 = V2\_Supplier\_C$, $x_5 = WI\_Supplier\_A$, $x_6 = WI\_Supplier\_C$, $x_7 = WI\_Supplier\_D$, $x_8 = RS\_Supplier\_A$, $x_9 = RS\_Supplier\_B$, and $x_{10} = RS\_Supplier\_D$.

Next, we formalize the variability annotations of the model. The formalization follows the general process described in Section 4, more specifically the transformation rules shown in Table 1. This step produces the set of equality and inequality constraints for the integer linear program, i.e., the optimization space. The formalization of the degrees of freedom of the demonstration model results in the following set of constraints (with all decision variables $x_i \in \{0, 1\}$):

$$\begin{aligned} &x_1 + x_2 = 1 && x_5 - x_1 \geq 0 \\ &x_3 + x_4 = 1 && x_7 - x_{10} \geq 0 \\ &x_5 + x_6 + x_7 = 1 && x_3 - x_9 = 0 \\ &x_8 + x_9 + x_{10} = 1 && \end{aligned} \tag{2}$$

Our goal is to find a Pareto-optimal product line architecture, not a Pareto-optimal product. Therefore, we only formalize the architectural degrees of freedom, not the product-line-related variability (which will still be present in the resulting architectures). In our demonstration model, the product line variability from the Vehicle Level traces to the WiperControl variation point on the artifact levels, which was thus omitted from the formalization. All other variabilities (i.e., supplier choices and dependencies) have no traceability to the Vehicle Level; they are thus architectural degrees of freedom and are included in the formalization.

The formalization of quality attributes into objective functions naturally use the same assignment of decision variables. The quality attribute values defined in Table 2 result in the following linear objective functions for weight, cost and power consumption, in this order:

$$\begin{aligned} \text{MIN}\quad & 122 * x_1 + 131 * x_2 + 154 * x_3 + 154 * x_4 + \\ & 155 * x_5 + 167 * x_6 + 158 * x_7 + 143 * x_8 + \\ & 150 * x_9 + 126 * x_{10} \\ \text{MIN}\quad & 3.30 * x_1 + 3.25 * x_2 + 5.32 * x_3 + 5.90 * x_4 + \\ & 3.54 * x_5 + 3.51 * x_6 + 3.58 * x_7 + 10.65 * x_8 + \\ & 10.82 * x_9 + 11.46 * x_{10} \\ \text{MIN}\quad & 2.80 * x_1 + 2.75 * x_2 + 3.10 * x_3 + 3.15 * x_4 + \\ & 2.60 * x_5 + 2.55 * x_6 + 2.50 * x_7 + 5.50 * x_8 + \\ & 5.50 * x_9 + 5.45 * x_{10} \end{aligned} \tag{3}$$

Having generated both objective functions and program constraints means that we now have a proper multi-objective integer linear program (cf. Equation 1). This enables us to

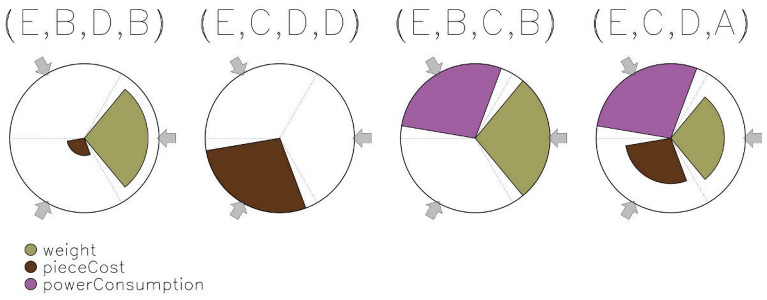

Fig. 3: Petal diagram of the Pareto-optimal solutions.

explore the optimization space and to identify Pareto-optimal solutions for the given quality attributes.

6.3 Discussion of Optimization Results

Even for the small optimization space of our demonstration model, finding Pareto-optimal solutions by hand can already be a tedious exercise. When scaling up the size of the problem to real-world models, finding solutions by hand quickly goes from tedious to impractical to outright impossible. In order to demonstrate our approach, we apply commercial tooling (cf. Section 5) for solving the formalization of our optimization problem and thereby identify four Pareto-optimal solutions. Regarding the consistency of our results, it should be noted that each solution directly corresponds to a real product line architecture for the given system, since all product line variability is still present in the resulting models. In other words, the product line variability space remains intact.

The optimization solutions can be described using a notation of 4-tuples in the form of $(V1, V2, WI, RS)$; e.g., (A, C, A, A) is a solution where V1, the wiper integrator and the rain sensor use components from supplier A, and V2 from supplier C. Using our optimization tooling, we identified the Pareto solutions (E, B, D, B), (E, C, D, D), (E, B, C, B) and (E, C, D, A). Figure 3 shows a Petal diagram of these solutions, where the petals depict the relative quality of criteria per solution, thereby giving a rough overview of the characteristics of each solution. Petal diagrams are one of the visualization options of the FINNOPT tool (cf. Section 5). Such visualizations can be used (among other methods) to conduct a trade-off analysis of the Pareto-optimal alternatives for a system architecture. Pareto analyses for large and complex real-world industrial models are a non-trivial task that requires considerable system expertise from the conducting architect or engineer.

The Pareto set for our case study example has a particularly interesting characteristic: out of all products that can be configured from the given product line design space, the most lightweight possible product (277g, V1 from supplier A, WI from supplier A, no rain sensor) is not part of the most lightweight product line architecture (E, C, D, D) (its most lightweight product has 289g). Effects like this are a consequence of family-based analyses like our product-line-aware optimization and must be taken into account when considering the adequacy of Pareto-optimal product line architectures for specific use cases.

7 CONCLUSIONS

In this paper we introduced a novel approach for product line architecture optimization and demonstrated its application by means of a case study. The approach defines a complete mathematical formalization as a multi-objective integer linear program that comprises the architectural degrees of freedom, the implementation components and the design objectives (cf. Section 4). We demonstrated that our approach has characteristics that differentiate it both from our previous work and from other research in this area. Furthermore, our approach is not merely theoretical, but has been implemented with the help of off-the-shelf optimization tooling (cf. Section 5).

As future work, we plan to compare our approach to other optimization frameworks in terms of efficiency and optimality of the results. We also plan to address variants of the given problem domain like an (optional) inclusion of specific border conditions, e.g., optimal product variants that shall always be part of a resulting product line architecture. Furthermore, we plan to identify useful application scenarios for the so far unused formalization of product line variability in addition to our current approach of optimizing only the architectural degrees of freedom. We plan to continuously evaluate our efforts with the aid of industry experts from the automotive domain. A first round of evaluations has already been concluded and yielded positive feedback.

References

[Al13] Aleti, A.; Buhnova, B.; Grunske, L.; Koziolek, A.; Meedeniya, I.: Software architecture optimization methods: A systematic literature review. IEEE Transactions on Software Engineering, 39(5):658–683, 2013.

[BK05] Burke, E. K.; Kendall, G.: Search methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer, 2005.

[Bl13] Blom, H.; Lönn, H.; Hagl, F.; Papadopoulos, Y.; Reiser, M.-O.; Sjostedt, C.-J.; Chen, D.-J.; Tavakoli Kolagari, R.: , White Paper Version 2.1.12: EAST-ADL - An Architecture Description Language for Automotive Software-Intensive Systems, 2013.

[CHE05] Czarnecki, K.; Helsen, S.; Eisenecker, U.: Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice, 10(2):143–169, 2005.

[CN01] Clements, P.; Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Professional, 2001.

[Co14] Colanzi, T. E.; Vergilio, S. R.; Gimenes, I.; Oizumi, W. N.: A search-based approach for software product line design. In: Proceedings of the 18th International Software Product Line Conference-Volume 1. ACM, pp. 237–241, 2014.

[CV12] Colanzi, T. E.; Vergilio, S. R.: Applying search based optimization to software product line architectures: Lessons learned. In: International Symposium on Search Based Software Engineering. Springer, pp. 259–266, 2012.

[CV16] Colanzi, T. E.; Vergilio, S. R.: A feature-driven crossover operator for multi-objective and evolutionary optimization of product line architectures. Journal of Systems and Software, 121:126–143, 2016.

[GR06] Grodzevich, O.; Romanko, O.: Normalization and other topics in multi-objective optimization. In: Proceedings of the Fields–MITACS Industrial Problems Workshop. The Fields Institute, pp. 89–102, 2006.

[LHLE15] Lopez-Herrejon, R. E.; Linsbauer, L.; Egyed, A.: A systematic mapping study of search-based software engineering for software product lines. Information and software technology, 61:33–51, 2015.

[Mi06] Miettinen, K.: IND-NIMBUS for demanding interactive multiobjective optimization. Multiple Criteria Decision Making ’05, 1:137–150, 2006.

[Mi15] Mian, Z.; Bottaci, L.; Papadopoulos, Y.; Sharvia, S.; Mahmud, N.: Model transformation for multi-objective architecture optimisation of dependable systems. In: Dependability Problems of Complex Information Systems, pp. 91–110. Springer, 2015.

[MP07] Metzger, A.; Pohl, K.: Variability management in software product line engineering. In: Companion to the proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society, pp. 186–187, 2007.

[RRV16] Ramírez, A.; Romero, J.; Ventura, S.: A comparative study of many-objective evolutionary algorithms for the discovery of software architectures. Empirical Software Engineering, 21(6):2546–2600, 2016.

[RTW09] Reiser, M.-O.; Tavakoli Kolagari, R.; Weber, M.: Compositional Variability: Concepts and Patterns. In: 42nd Hawaii International Conference on System Sciences. pp. 1–10, 2009.

[Sc03] Schmid, K.: A Quantitative Model of the Value of Architecture in Product Line Adoption. In: Proceedings in the 5th International Workshop on Product Family Engineering. 2003.

[Th12] Thüm, T.; Apel, S.; Kästner, C.; Kuhlemann, M.; Schaefer, I.; Saake, G.: Analysis strategies for software product lines. School of Computer Science, University of Magdeburg, Tech. Rep. FIN-004-2012, 2012.

[Ti12] Tischer, Christian; Boss, Birgit; Müller, Andreas; Thums, Andreas; Acharya, Rajneesh; Schmid, Klaus: Developing Long-Term Stable Product Line Architectures. In (de Almeida, E. Santana; Schwanninger, C.; Benavides, D., eds): Proceedings of the 16th International Software Product Line Conference (SPLC ’12). volume 1. ACM, pp. 86–95, 2012.

[Wa13] Walker, M.; Reiser, M.-O.; Tucci-Piergiovanni, S.; Papadopoulos, Y.; Lönn, H.; Mraidha, C.; Parker, D.; Chen, D.-J.; Servat, D.: Automatic optimisation of system architectures using EAST-ADL. Journal of Systems and Software, 86(10):2467–2487, 2013.

[WW15] Wägemann, T.; Werner, A.: Generating Multi-objective Programs from Variant-rich EAST-ADL Product Line Architectures. In: GI-Jahrestagung. pp. 1673–1685, 2015.