PerOpteryx: Automated Improvement of Software Architectures

Axel Busch, Dominik Fuchß, and Anne Koziolek

Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: {busch, koziolek}@kit.edu
dominik.fuchss@student.kit.edu

Abstract—**The design phase of software development processes is a decisive part of the quality characteristics of the later software system. If errors are not or only late discovered in early phases of the development, this often has strong effects on quality and costs of the project. Software architecture models systematically help to prevent errors in early phases, such as the design phase. On the basis of models, software architectures and their quality properties can be evaluated and optimized at design time. PerOpteryx supports the systematic process of evaluating and optimizing software architecture models early considering quality attributes, such as performance, reliability, costs. The approach automatically generates architecture candidates based on several degrees of freedom of component-based software architectures. PerOpteryx then automatically evaluates and optimizes these architecture candidates with regard to the quality requirements of the software system. PerOpteryx is the first automated tool for systematic quality optimization of component-based software architectures that combines quantitatively modeled knowledge with qualitatively modeled architecture knowledge. Including qualitatively modeled knowledge enables quality characteristics to be taken into account in the optimization process that would have been too cost-intensive to model quantitatively.**

I. Introduction

In recent years, more and more software systems have found their way into our lives in the course of digitalization. Digital systems have become irreplaceable both in the private sector and in the business world. A multitude of business models today even depend to a large extent on reliable and high-quality IT systems. A failure of such critical systems quickly has a negative impact on the business result of a company and can endanger human lives in the worst case due to the ever-increasing use of critical applications. High-quality software is therefore essential for the efficient support of business processes as well as for the seamless and secure support of modern life.

The dilemma in the planning and implementation of high-quality software, however, lies in the fact that uncompromising improvement of quality attributes is usually not possible, or at least mostly involves unrealistic costs. For the best possible implementation of functional requirements and quality requirements, these must be carefully weighed against each other. Every design decision leads to different realization of the requirements in a software architecture. It has already been shown [1] that the structure of the software architecture can have a major influence on the quality attributes to be expected later. In component-based software development processes, this structure is composed of the choice of software components, more complex subsystems, the placement of these components and subsystems on hardware, and the specification of hardware resources. More complex algorithms (assembled in a software component), for example, require a larger number of CPU cycles, from which higher average response times can be expected. In return, it can have a positive influence on other quality attributes or lower initial costs. However, achieving such a result is very time-consuming without a tool-supported method and systematic process.

Especially within the component-based software architecture paradigm, there are already approaches for the automatic evaluation of software architectures according to different quality attributes, such as performance, reliability, and maintainability. The Palladio [1] approach supports the analysis and simulation of software architecture models. Such an approach allows the evaluation of design decisions already before the actual implementation and make the software development process more efficient by avoiding mistakes in early phase. However, by using this approach, searching for an optimal software architecture is a manual process. Neither alternatives of software components, third-party subsystems nor their configuration within the software architecture can be automatically evaluated and optimized. Further questions, such as regarding the effects of implementing functional requirements or operationalized quality requirements on any other quality attributes of the software architecture remain unconsidered. In addition, only quality attributes can be evaluated that are considered by explicit evaluation functions. Informal architecture knowledge cannot be used in these approaches to improve the software architecture.

PerOpteryx addresses the aforementioned open problems by providing automatic optimization of software architectures. PerOpteryx allows to optimize component-based software architectures by several architectural degrees of freedom according to the project requirements regarding quality attributes. It has been applied in several industrial case studies [2], [3].

There are existing solutions for supporting design decisions with the quality attributes of software architectures being taken into account, such as ArcheOpteryx [4]. However, PerOpteryx allows to evaluate more degrees of freedom, the automatic

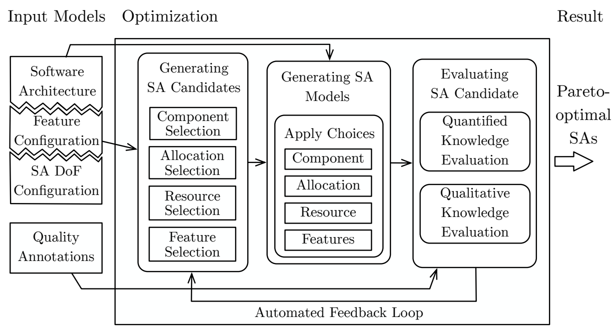

Fig. 1. PerOpteryx software architecture optimization workflow

exchange of features, trade-off decisions, and considering qualitative modeled knowledge.

II. Background: Palladio Component Model

PerOpteryx requires a software architecture model to describe the software architecture to be optimized. As model language, we use the Palladio component model (PCM). PCM models the structural information necessary to evaluate the required quality attributes. In addition, Palladio models entities to describe internal processes of software components. We use these as a basis for evaluating several quality attributes, such as performance or reliability.

The Palladio Component Model [1] is part of the Palladio approach. PCM is a meta model for component-based software architectures and also provides a set of analysis tools for performance, reliability, and cost evaluation.

PCM is specifically designed for component-based systems. It strictly separates parametrized component performance models from the composition models and resource models. Thus, the PCM naturally supports architectural degrees of freedom (e.g., substituting components, changing component allocation, etc.) for enabling automated design space exploration.

A PCM model can be decomposed into several parts namely repository, assembly, allocation, and usage. Each part covers a specific view on the modeled software architecture.

The repository gathers all components represented by the meta class `PCM:RepositoryComponent` and interfaces represented by the meta class `PCM:Interface`. A component can require or provide a given interface, which is modeled by a role (`PCM:RequiredRole` or `PCM:ProvidedRole`) referencing the interface. If a component provides a specific interface with a set of offered services, it has provide an abstract behavior description, called service effect specification (SEFF), for each offered service. SEFFs model the abstract control flow through a component service in terms of internal actions (i.e., resource demands accessing the underlying hardware) and external calls (i.e., accessing connected components). Modeling each component behavior with separate SEFFs enables us to quickly exchange component specifications without the need to manually change system-wide behavior specifications (as required in e.g. UML sequence diagrams). For performance annotations, component developers can use the extended resource-demanding service effect specifications (RDSEFFs). Using RDSEFFs, developers specify resource demands for their components (e.g., in terms of CPU instructions to be executed).

III. PerOpteryx: Optimizing Software Architectures

PerOpteryx[1] carries out the optimization of software architectures (SA) in three parts. In the center of Figure 1 we demonstrate the three main parts of the optimization process. PerOpteryx carries out the optimization process fully automatically. The three-part process is repeated iteratively until the abort criterion is reached.

- **Generating software architecture candidates:** Degrees of freedom span all possibilities within one dimension, while a degree of freedom *instance* spans all possibilities within one dimension that can be performed on one particular architecture model of a particular software system, such as component exchange. We assume, that every change in the software architecture leads to changes in the quality properties. Degrees of freedom instances can either be (partially) automatically derived from the initial software architecture model or pre-defined by the software architect. For the automatic derivation there is a meta model and a corresponding rule-set.
- **Generating the software architecture model:** PerOpteryx generates a corresponding architecture candidate model basing on the choice out of the dimensions of the degree of freedom space. The model can then be used to evaluate the relevant quality attributes.
- **Evaluating software architecture candidates:** PerOpteryx evaluates the architectural model with the help of evaluation functions. For each quality attribute to be considered, PerOpteryx determines the corresponding quality property to compare different architecture candidates with each other. Dominant candidates are further optimized, while it does not keep track on suboptimal candidates.

A. Preliminaries

1) Degrees of Freedoms: PerOpteryx supports several degrees of freedom [5] basing on component-based software architectures. The **component selection** degree of freedom automatically exchanges functionally equivalent components against each other. These components usually differ in their quality attributes. For example, a component can have a better performance, but may be more expensive, than an alternative component, which is slower but cheaper in development.

The **component allocation** degree of freedom varies the placement of components on system resources. For example, there are differences in performance or security when components are mainly allocated on only one physical resource of the system, compared to distributing them across several resources.

The **resource scaling** degree of freedom enables the automatic adaptation of hardware resources, such as CPU clock rates or IOPS of hard disks or solid state drives. Software

[1] https://github.com/PalladioSimulator/Palladio-Addons-PerOpteryx

architects should expect effects on the performance. More powerful systems must be weighed against cheaper systems in terms of higher acquisition costs or costs per operation.

The **feature inclusion** degree of freedom enables the automated evaluation of the effects of using complex subsystems in a target system. Intrusion detection systems, access control systems or logging frameworks represent several examples for such complex subsystems. Such subsystems typically provide many features that can be reused.

The presence of features opens a new degree of freedom, namely the **solution** degree of freedom. Features can be implemented in several different ways. The solution degree allows the optimization of the choice of different implementations (on model level) of the same feature. Based on a model that describes the placement of the features, the corresponding software components (to fulfill the functionality) are woven into the software architecture. Thus software architects can evaluate which concrete solution of the different available manufacturers is optimal regarding the considered quality attributes. The use of these systems at different locations in the target system can strongly influence quality characteristics. Software architects can use this mechanism to select different locations within the target software architecture where design decisions regarding features should be evaluated.

2) Input models: PerOpteryx requires four different input models: the initial **software architecture model** describes the structure of the software system. More specifically, it defines the components used, their assembly, their allocation to hardware resources, hardware resource configuration, and the usage profile for the basic software system [1]. The software architecture model represents the basis for evaluation and optimization. If design decisions regarding features and different solutions should be evaluated, a **features configuration** model for the description of the features to be evaluated and their configuration must be modeled in the target system [6]. The degrees of freedom instances to explore can be configured using the **SA DoF configuration** model or automatically derived. It describes the degrees of freedom to be considered and their dimension for the software architecture model to be evaluated. The **quality annotation model** adds quality information to the software architecture model [7], [8]. Individual components can be annotated as well as broader relationships that affect multiple components and hardware containers. The appearance of the model depends on the concrete quality attributes considered.

B. Generating Software Architecture Candidates

The given degree of freedom instances span the search space for the optimization. Each degree of freedom defines several possible choices. The number of choices can range from a few to continuous numbers. The combination of several degrees of freedom quickly results in several million architecture candidates. Therefore, in most cases the evaluation of the entire search space is not possible. In particular, evaluating the whole design space is often not possible for more complex systems with a larger number of components, resource containers, and various features.

Thus, to find (near) optimal architecture candidates in the design space defined by the degree of freedom instances, PerOpteryx uses a genetic algorithm. Particularly, we use the NSGA-II evolutionary algorithm [9] that is implemented in the Opt4J framework [10].

Genetic algorithms commonly distinguish between genotypes and phenotypes. The genotypes encode the possible solutions in terms of choices for each degree of freedom instance. In our case, each architecture candidate is encoded as a vector of choices, one for each degree of freedom instance. The actual software architecture candidate model that is called phenotype can be generated from the genotype (as described in the following).

C. Generating the SA Models

Based on a genotype selected by the genetic algorithm, PerOpteryx generates the corresponding software architecture candidate model, i.e. the corresponding phenotype, using model transformations. These model transformations range from the simple change of attributes or associations (for e.g. allocation degrees of freedom) to weaving in entire subsystems (for feature choices). In the latter case, the software architecture model is extended by additional components that were previously not included in the SA model. These new components realize the new features to be included in the target SA model. To make the components compatible with the components in the target SA model we generate adapters that make interfaces compatible. As an alternative, the internal processes of components are extended to integrate the new functionalities. These additional components are distributed to the resource containers after assembly. To apply these changes, we use the *Weaving Engine* of PerOpteryx [11], which implements the feature choices specified in the phenotype as model transformations.

D. Evaluating Architecture Candidates

The evaluation of the quality attributes of the generated software architecture candidates is divided into two parts. In the first part, PerOpteryx calculates the quality attributes using quantitative methods. The second part processes qualitatively modeled quality attributes, which are either not quantitatively calculable at the current state of research or the effort for this type of modeling would have been too high.

1) Quantified Knowledge Evaluation: PerOpteryx allows any evaluation function of various quality attributes to be integrated into the optimization. Currently the tool supports the quality attributes performance, reliability, and monetary cost. Further functions for quantitative evaluation are in process, such as the calculation of maintenance costs for change scenarios of software architectures or quantitative security analyses. We achieve performance evaluations by several simulation-based approaches, such as SimuCom [1] or SimuLizar [12], or alternatively calculated by layered queuing networks (LQN) [13]. Reliability can be calculated using Markov processes [14]. First attempts with the analyses of maintenance costs are calculated using the KAMP framework [15]. First tests for the quantitative determination of security properties are shown in [16].

2) Qualitative Knowledge Evaluation: For the evaluation of unstructured architectural knowledge and for joint consideration with quantitative evaluation methods, we have adapted the concept of qualitative reasoning for the analysis in component-based software architecture models [8]. This method can be used to model and evaluate effects on the quality attributes of individual components on other components or quality attributes of the overall system at any level of granularity. Coarse granularity of *has a positive effect* or *has a negative effect* up to complex dependencies between different quality attributes are possible.

E. PerOpteryx Results

PerOpteryx results in Pareto-optimal architecture candidates describing the architecture itself and evaluation results regarding the quality properties[2]. Various architecture decisions regarding component selection, allocation, resources, complex feature selection, and feature configurations can be automatically evaluated based on these results. Software architects then prioritize their quality requirements and finally select the most suitable candidate [3]. Prioritization is necessary because it depends on the requirements to decide whether performance is more important than costs or a certain level of security features. The combination of quantitative evaluation functions and initially unstructured architectural knowledge allows new and interesting questions to be answered. We can now evaluate whether the use of certain components or features improving the level of usability justifies the higher costs and higher average response times of the system [17] even we do not quantify the usability but evaluate them using qualitative modeled knowledge. Transitive effects between components across different quality attributes can also be analyzed [8]. We also used PerOpteryx in the context of self-adapting systems [18]. PerOpteryx automatically calculated Pareto-optimal architecture candidates under dynamically changing conditions.

Figure 2 shows example results for an optimization run, where we optimized for performance, costs (both quantitatively), security, and usability (both qualitatively modeled). In this experiment, PerOpteryx performed 200 iterations evaluating 2586 architecture candidates automatically. 63 of these resulted in Pareto-optimal architecture candidates. The evaluation considered several degrees of freedom namely component exchange, resource scaling and component allocation.

IV. Conclusion

In this paper, we describe PerOpteryx, our tool to the automatic evaluation and optimization of software architectures considering their quality attributes. The generation of the required models can be a time-consuming process. However, being able to evaluate critical decisions and long-term planning before a cost-intensive implementation can be worthwhile in the long term. Using PerOpteryx, software architects can optimize software architecture quality attributes based on models at design time. PerOpteryx automatically considers different degrees of freedom relevant to the component-based software development processes. On the basis of these degrees of freedom PerOpteryx automatically applies, evaluates, and optimizes software architecture candidates. As a result, software architects achieve a set of Pareto-optimal results used to select the optimal candidate according to the requirements.

[2]Raw evaluation results and tool screenshots can be found on https://sdqweb.ipd.kit.edu/wiki/PerOpteryx

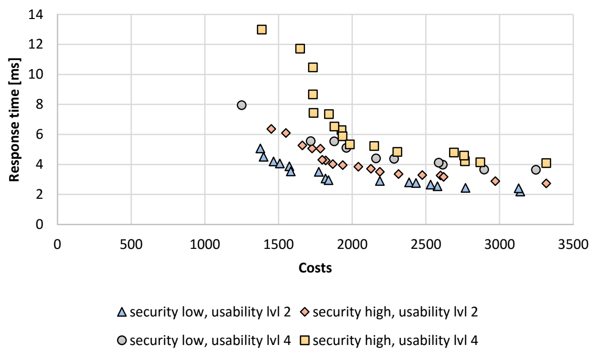

Fig. 2. Example results from [17]. Each point represents a Pareto-optimal architecture candidate

References

[1] R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann, *Modeling and Simulating Software Architectures: The Palladio Approach*. The MIT Press, 2016.

[2] H. Koziolek, B. Schlich *et al.*, "An industrial case study on quality impact prediction for evolving service-oriented software." ACM, 2011.

[3] T. de Gooijer, A. Jansen, H. Koziolek, and A. Koziolek, "An ind. case study of performance and cost design space explor." ser. ICPE '12, 2012.

[4] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya, "Archeopterix: An extendable tool for architecture optimization of aadl models," in *MOMPES '09*, 2009.

[5] A. Martens, H. Koziolek *et al.*, "Automatically improve sw models for perform., reliab. and cost using genetic alg." ser. WOSP ICPE'10, 2010.

[6] A. Busch, Y. Schneider, A. Koziolek, K. Rostami, and J. Kienzle, "Modelling the Structure of Reusable Solutions for Architecture-based Quality Evaluation," ser. CloudSPD'16. IEEE.

[7] S. Becker, H. Koziolek, and R. Reussner, "The palladio comp. model for model-driven perf. prediction," *Journal of Systems and Software*, 2009.

[8] Y. Schneider, A. Busch, and A. Koziolek, "Using Informal Knowledge for Improving SW Quality Trade-off Decisions," ser. ECSA'18, 2018.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[10] M. Lukasiewycz, M. Glaß, F. Reimann, and S. Helwig, "Opt4J - The Optimization Framework for Java," http://www.opt4j.org, 2010.

[11] M. Scheerer, A. Busch, and A. Koziolek, "Automatic Evaluation of Complex Design Decisions in Component-based Software Architectures," ser. MEMOCODE'17. ACM, 2017.

[12] M. Becker, S. Becker *et al.*, "SimuLizar: Design-Time Modelling and Performance Analysis of Self-Adaptive Systems," ser. LNI, 2013.

[13] G. Franks, T. Omari *et al.*, "Enhanced modeling and solution of layered queueing networks," *IEEE Trans. on Software Engineering*.

[14] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner, "Architecture-based reliability prediction with the palladio component model," *IEEE Transactions on Software Engineering*, 2012.

[15] K. Rostami, J. Stammel, R. Heinrich, and R. Reussner, "Architecture-based assessment & planning of change requests," ser. QoSA '15, 2015.

[16] A. Busch, M. Strittmatter, and A. Koziolek, "Assessing Security to Compare Architecture Alternatives of Component-Based Systems," ser. QRS '15. IEEE Computer Society, 2015.

[17] A. Busch and A. Koziolek, "Considering Not-quantified Quality Attributes in an Automated Design Space Exploration," ser. QoSA'16, 2016.

[18] N. Huber, A. van Hoorn, A. Koziolek *et al.*, "Modeling Run-Time Adaptation at the Sys. Arch. Level in Dynamic SO Env." *SOCA*, 2014.