# Using Quality of Service Bounds for Effective Multi-objective Software Architecture Optimization

Qais Noorshams, Anne Martens, Ralf Reussner
Karlsruhe Institute of Technology, Karlsruhe, Germany
Email: qais.noorshams@student.kit.edu, {martens,reussner}@kit.edu

## ABSTRACT

Quantitative prediction of non-functional properties, such as performance, reliability, and cost, of software architectures supports systematic software engineering. Even though there usually is a rough idea on bounds for quality of service, the exact required values may be unclear and subject to trade-offs. Designing architectures that exhibit such good trade-off between multiple quality attributes is hard. Even with a given functional design, many degrees of freedom in the software architecture (e.g. component deployment or server configuration) span a large design space. Automated approaches search the design space with multi-objective metaheuristics such as evolutionary algorithms. However, as quality prediction for a single architecture is computationally expensive, these approaches are time consuming. In this work, we enhance an automated improvement approach to take into account bounds for quality of service in order to focus the search on interesting regions of the objective space, while still allowing trade-offs after the search. To validate our approach, we applied it to an architecture model of a component-based business information system. We compared the search to an unbounded search by running the optimization 8 times, each investigating around 800 candidates. The approach decreases the time needed to find good solutions in the interesting regions of the objective space by more than 35% on average.

## Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; C.4 [**Computer Systems Organization**]: Performance of Systems—*modeling techniques*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

## General Terms

Design, Performance, Reliability

## Keywords

Optimization, Performance, Quality Attribute Prediction, Reliability, Software Architecture


*QUASOSS'10,* October 4, 2010 Oslo, Norway


## 1. INTRODUCTION

The design of software architecture is crucial to exhibit good quality of service (cf. [3]), e.g. performance and reliability. Model-driven, quantitative architecture evaluation approaches help the software architect to reason about the architecture and predict its quality attributes and cost. However, even though there usually is a rough idea of requirements for the non-functional properties, the exact required values may be unclear and subject to trade-offs. For example, the decision of how much response time of the system is acceptable may depend on the cost to achieve this response time and is subject to negotiation between stakeholders. Still, they may agree on bounds specifying the worst acceptable values of the quality attributes, e.g. the mean response time of the system should not exceed 15 seconds. A system that violates any bounds is declared infeasible, i.e. useless for the stakeholders.

Designing architectures that provide optimal trade-offs between multiple quality attributes is difficult. Even with a given functional design, the many degrees of freedom in the software architecture (e.g. component deployment or server configuration) still span a large design space.

Automated approaches support the software architect to improve their architectural designs and find good trade-offs between quality attributes. They search the design space with multi-objective metaheuristics such as evolutionary algorithms to find many Pareto-optimal candidates. However, as quality prediction for a single architecture is computationally expensive, these approaches are time consuming since many possible candidates need to be evaluated.

In this work, we present an approach to include rough bound estimations on quality of service requirements into an automated improvement approach to make the search for optimal trade-offs focus on interesting regions of the objective space. We extend our previous approach PerOpteryx [14] by three aspects: First of all, we add a modeling notation for quality requirements. We extended the existing *Quality of service Modeling Language (QML)* [11] to enable the specification of optimization goals. Second, we translate the QML requirements to constraints in an optimization problem. Finally, we use the *constraint domination* strategy of Deb et al. [10] to make the search focus on the feasible space.

The contribution of this paper is a novel approach that, to the best of our knowledge, is the first to combine multi-criteria architecture optimization and quality of service bounds so that the search can focus on feasible regions of the search space. With this extension, the time needed to find valuable solutions for the software architects can be reduced.

We have implemented the approach in the PerOpteryx tool. Using this tool, we demonstrate the benefits of our approach in a case study. Our extension was able to find solutions in the interesting regions of the objective space in average more than 35% faster than the old, unconstrained approach.

This paper is structured as follows: Section 2 presents related work to our approach. Section 3 gives background on the architecture evaluation approach Palladio that we use in this work. Section 4 discusses the required metamodel for modeling requirements in this context, and motivates our choice of QML. Section 5 then presents our architecture optimization process, which makes use of the specified bounds to focus the search on the feasible architecture candidates. A case study in Section 6 shows the feasibility of our work by applying the process to an example architecture and comparing the effect of the requirements consideration. Finally, Section 7 concludes.

## 2. RELATED WORK

Our approach is based on performance prediction [2], reliability prediction [12], multi-objective metaheuristic optimization [5], and constraint handling in evolutionary algorithms [6].

In summary, several other approaches to automatically improve software architectures for one or several quality properties have been proposed. Most approaches improve architectures by either applying predefined improvement rules, or by applying metaheuristic search techniques. All except one approach do not support trade-off between quality attributes after the search. In addition, all approaches do not allow to specify quality requirements for quality attributes that should be optimized, thus, they do not allow to focus on interesting regions of the objective space.

Xu et al. [18] present a semi-automated approach to improve performance. Based on a layered queueing network (LQN) model, performance problems (e.g., bottlenecks, long paths) are identified in a first step. Then, mitigation rules are applied. The search stops as soon as specified response time or throughput requirements are met. The approach is limited to performance only.

The ArchE framework (McGregor et al. [15]) assists the software architect during the design to create architectures that meet quality requirements. It provides the evaluation tools for modifiability or performance analysis, and stepwise suggests modifiability improvements depending on the yet unsatisfied requirements. The search stops as soon as specified requirements are met.

Canfora et al. [8] optimize service composition cost using evolutionary algorithms while satisfying service level agreement (SLA) constraints. They implement constraint handling with dynamic penalty functions.

Menascé et al. [16] generate service-oriented architectures that satisfy quality requirements, using service selection and architectural patterns. They model the degree of requirement satisfaction as utility functions. Then, a weighted overall system utility is optimized in a single-objective problem using random-restart hill-climbing. Thus, preferences for quality attributes and importance of requirements have to be specified in advance.

Aleti et al.[1] present a generic framework to optimize architectural models with evolutionary algorithms for multiple arbitrary quality properties, thus enabling trade-off after the search. In addition, the framework allows to specify constraints for the search problem, for example available memory consumption. However, the constraint handling is relatively simple: Infeasible candidates are just discarded. Quality requirements are mentioned, but not included in the optimization.

## 3. PALLADIO COMPONENT MODEL

Generally, our concepts can be used for different software architecture models. To a certain extent, service-oriented architectures can be regarded as a specialization of component-based software architectures. As a consequence, we focus the scope of our work on component-based software architectures.

We apply our approach to the Palladio Component Model (PCM) [4], a modeling language for component-based software architectures with an UML-like syntax. The PCM enables the explicit definition of the i) components , ii) architecture, iii) allocation, and iv) usage of a system in respective artifacts, which comprise a PCM instance (cf. Figure 1):

1. *Component specifications* contain an abstract, parametric description of components. Furthermore, the behavior of the components is specified using an UML activity diagram similar syntax.
2. An *assembly model* defines the software architecture.
3. The resource environment and the allocation of components to resources are specified in an *allocation model*.
4. The *usage model* specifies *usage scenarios*. For each user, one of the scenarios applies defining the frequency and the sequence of interactions with the system, i.e. which system functionalities are used with an *entry level system call*.

Using model transformations, the PCM instance can be analyzed or simulated to predict performance (response time and throughput) [4], reliability (probability of failure on demand (POFOD)) [7], and cost [14] of a system.

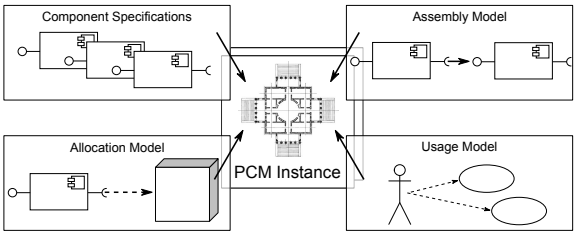


**Figure 1: Artifacts of a PCM instance.**

Figure 2 illustrates an example PCM instance of the so-called business reporting system (BRS) using annotated UML. The BRS provides statistical reports about business processes and is loosely based on a real system. The system consists of 9 components and is allocated to 4 servers. The behavior description (incl. CPU demands) of one component is here illustrated by an activity diagram. Having only one usage scenario, a user interacts with the system every 5s requesting a sequence of reports and views.

## 4. QUALITY REQUIREMENT MODEL

To deal with quality requirements, i.e. requirements on quality attributes (e.g. throughput or availability), their formal representation is essential. For our work, we identified

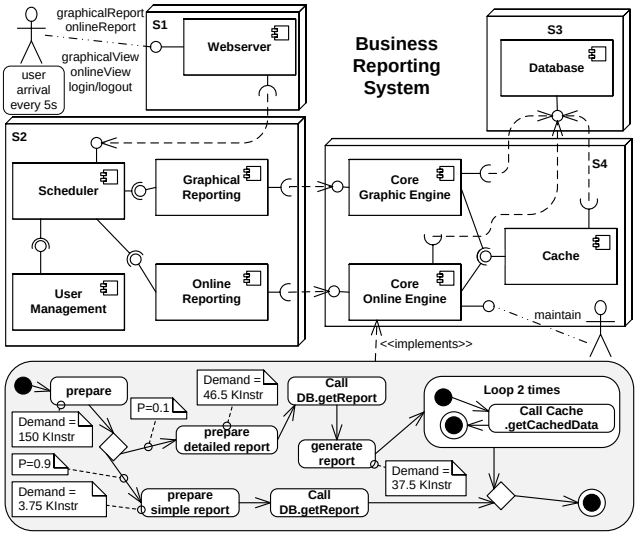


**Figure 2: PCM instance of the BRS.**

the following requirements for the representation of quality requirements:

1. Independent quality attribute definition – Quality attributes and their general information should be definable independent of their use or their requirements. E.g. different projects can have different requirements that refer to the same definitions.
2. Definition of objectives and requirements – Objectives (what is to be improved?) and requirements (what is a feasible system?) should be flexibly definable on quality attributes. E.g. a project could aim at improving response time with a cost limit. Another project could have a bound for reliability yet searching for improvement of this quality attribute.
3. Different aspects of attributes – Objectives and requirements can be evaluated deterministically and/or stochastically. E.g. the exact cost of a service could be determined in advance whereas the response time of this service could be guaranteed stochastically. Thus, the model needs to be able to differentiate between different *evaluation aspects* of a quality attribute.
4. Fine-grained requirement application – Requirements can also apply for parts of a system, e.g. a sequence of system functionalities or single functionalities.

We focused on reusing the *Quality of service Modeling Language (QML)* created by Frølund and Koistinen [11] as it meets almost every requirement. QML allows the specification of quality attributes as well as the specification of quality requirements. QML is defined in Extended Backus-Naur Form (EBNF). To exploit tool support of the Eclipse Modeling Framework [17], we metamodeled QML using the Ecore metamodel. We extended the language to be able to define objectives and explicitly distinguish between objectives and requirements. Furthermore, we modified the language to associate the requirements with usage scenarios or entry level system calls, as QML is normally bound to service interfaces defined with e.g. the interface definition language (IDL). The following section will introduce the QML metamodel. For a better understanding, the sub-metamodels are simplified. Our extensions to QML are marked with ⊛.

QML has three main levels: A *contract type* defines quality categories, a *contract* specifies quality requirements or quality objectives on a contract type, and a *profile* binds a contract to an entity, e.g. a method of an interface.

Illustrated in Figure 3, a QML contract type can have multiple *dimensions*, each with an optional *unit* and an obligatory *dimension type*. A dimension type specifies the possible values of a dimension, which can be numeric or consist of user defined elements (this is omitted in the figure). The *relation semantics* specifies if the dimension improves with *increasing* or *decreasing* values. As an example, the contract type 'performance' can have the dimensions 'throughput' and 'response time'. 'Throughput' has an increasing numeric dimension type and is measured in 'number of jobs per second'. 'Response time' has a decreasing numeric dimension type and is measured in 'seconds'.

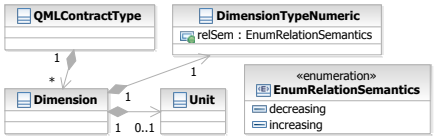


**Figure 3: Metamodel of a QML contract type.**

Illustrated in Figure 4, a QML contract can define *criteria* on dimensions of a QML contract. Deviating from standard QML where only constraints are definable, a criterion is either an *objective* or a *constraint*. An *objective* specifies an evaluation aspect of a dimension aimed for improvement. A *constraint* defines a *restriction* on an *evaluation aspect*. An *evaluation aspect* can be *deterministic* or *stochastic*. A *deterministic* aspect refers to the determined value(s) of a dimension. A *stochastic* aspect could e.g. be the mean value of a dimension. Using a contract, it is possible e.g. to define the *mean* response time as objective and a maximum *variance* as constraint. Note that it is also possible to define constraints on objectives by defining an objective and a constraint on the same evaluation aspect of a dimension.

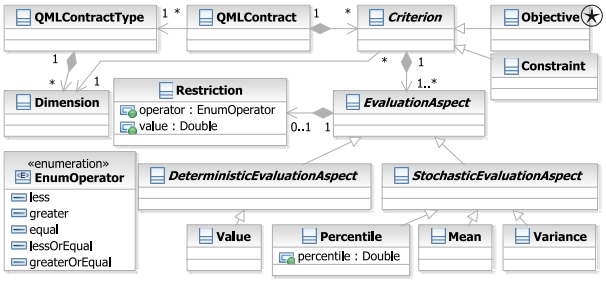


**Figure 4: Metamodel of a QML contract.**

Illustrated in Figure 5, a QML profile binds a contract to an entity by defining *requirements*. In our context, the entity is either an usage scenario or an entry level system call. Semantically, a contract is either required from a specific entity or from every entity of the usage model.

## 5. FINDING SATISFACTORY ARCHITECTURES

The goal of our work is to optimize component-based software architectures. To achieve this, we use metaheuristic techniques, particularly the multi-objective evolutionary algorithm (MOEA) NSGA-II developed by Deb et al. [10]. A disadvantage of a MOEA is that it may spend too much time

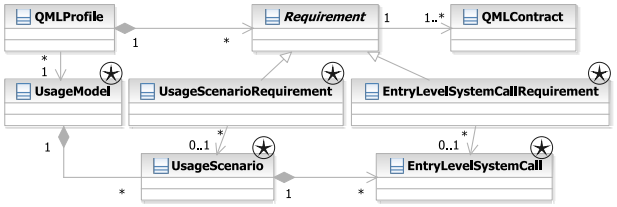


**Figure 5: Metamodel of a QML profile.**

exploring uninteresting regions of the objective space. Integrating quality requirements into the search aims at improving this algorithm due to the following advantages identified by Branke [6]:

1. Focus – MOEAs are approximate and non-deterministic. Quality requirements can be used to focus the search and identify particularly interesting alternatives.
2. Speed – Focusing the search avoids wasting computational effort on irrelevant regions of the search space.
3. Gradient – With increasing number of objectives, MOEAs are unable to determine the most promising search direction (*gradient*). Quality requirements provide additional information ensuring optimization progress.

Figure 6 illustrates the optimization process as a whole with four main steps:

1. The system to be optimized is modeled with the PCM. Additionally, the degrees of freedom, i.e. the possibilities to influence the non-functional properties of a system without changing its functional properties, are specified. In a component-based context, the degrees of freedom of a system can be e.g. component selection, component deployment, and hardware configuration (cf. [14] for details).
2. The quality requirements of the system are modeled using QML as described in Section 4. The link between PCM model and QML model is a QML profile.
3. With our tool PerOpteryx, the models are used to optimize the system. The optimization starts with one or more *initial candidates*, i.e. predefined system configurations, which can also be created randomly. Optimizing quality attributes and minimizing cost is pursued using NSGA-II with consideration of the *constraint domination* principle of Deb et al. as described in [10].
4. As solving multi-objective optimization problems results in a set of solutions rather than one single solution [9], the set of *Pareto-optimal*[1] architecture configurations feasible with respect to the quality requirements is presented. Finally, the software architect makes the trade-off decision and chooses one of the solutions.

To integrate requirements into this process, we extended the Opt4J framework [13], which implements NSGA-II, by the *constraint domination* strategy. Constraint domination enables the comparison of candidates on the basis of constraint violations. For the comparison of feasible candidates, i.e. candidates fulfilling all requirements, the strategy has no effect. The comparisons are essential for evolving candidates. The more a candidate violates the bounds, the less the probability of examining this candidate more closely becomes, i.e. the feasible objective space is most likely to be examined more closely. We chose this means of constraint handling because of the following advantages: First, it distinguishes explicitly between feasible and infeasible solutions and declares all feasible solutions superior to infeasible solutions as opposed to e.g. approaches based on penalty functions. Second, additional parameters are not required as many approaches are sensitive to parameter changes. Finally, it neither requires a specific number of constraints nor assumes a relation between objectives and/or constraints.

[1]A solution $x$ is *Pareto-optimal* if no other solution $y$ is better than $x$ w.r.t. all considered attributes (cf. [9]).

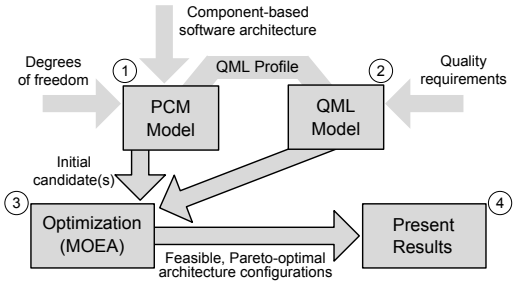


**Figure 6: Process Overview.**

## 6. CASE STUDY

This section describes a case study demonstrating the benefit of the consideration of requirements during the optimization process. The system under study is the business reporting system (BRS) described in Section 3. The software architect has to choose a candidate that minimizes mean response time, probability of failure on demand (POFOD), and cost. As degrees of freedom, the components can be allocated to up to nine different servers. Additionally, each server has a continuously variable CPU rate between 0.75 GHz and 3 GHz. The cost of the servers depends on the processing rate and the cost model is derived from Intel's CPU price list. A power function is chosen to be fitted to the data resulting in a cost model of $cost = \sum_i cost(i) = \sum_i 0.7665\ p_i^{6.2539}$ [monetary units (MU)] with a coefficient of determination $R^2 = 0.965$ and the processing rate of each server $p_i$ [GHz]. In terms of reliability, the servers have a mean time to failure (MTTF) of 17520 hours and a mean time to repair (MTTR) of 6 hours. We assume that for the quality attributes and cost the following requirements have been roughly identified[2]: cost $<$ 2000 MU, POFOD $<$ 0.2%, and mean response time $<$ 3.5 s.

After modeling the system, the requirements are modeled with our metamodel of QML. Note that for our optimization, cost is modeled as a quality attribute as we only consider the cost of the system. Figure 7 illustrates the model. First, the QML contract type defines the PCM contract type having three dimensions: cost, POFOD, and response time. On this contract type, the objectives and the constraints are defined in QML contracts (the objectives contract is similar to the constraints contract without `Restriction`s, thus omitted in the figure). The evaluation aspects of the cost and POFOD constraints are defined as `Value` as the dimensions are evaluated as single values that the constraints refer to, whereas the constraint on response time refers to the mean value of the dimension.

Figure 8 illustrates the QML profile associating the requirements with the PCM model. Using `UsageScenario-`

[2]The aspect of requirements engineering is beyond the scope of this paper.

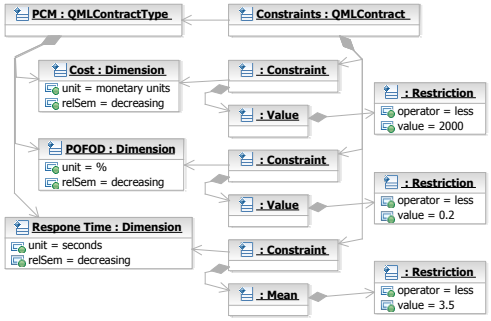


**Figure 7: Simplified model of the QML contract type and the QML contract.**

`Requirements`, the objectives and constraints are bound to the usage scenario of the usage model of the BRS.

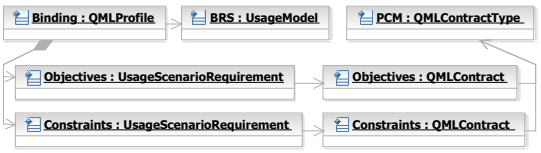


**Figure 8: Simplified model of the QML profile.**

With this presetting, the system is optimized using PerOpteryx. In total, 8 runs have been performed, each initialized with the same set of 20 randomly created candidates. For comparison reasons, 4 runs integrated the requirements in the optimization (*type* $\alpha$), 4 runs filtered the infeasible candidates after the run and were unconstrained during the optimization (*type* $\beta$). The $n$-th run of optimizations of type $\alpha$ and type $\beta$ will be denoted as $\alpha_n$ and $\beta_n$ respectively. In 200 iterations, every run analyzed around 800 candidates. Figure 9 illustrates the result of a type $\alpha$ optimization. As an example, the quality attributes of one optimal candidate are shown. Visually, the evaluated candidates are concentrated around the area of feasible solutions, i.e. the solutions respecting the bounds. However, due to the metaheuristic nature of the optimization, which "may incorporate mechanisms to avoid getting trapped in confined areas of the search space" [5], there is still spread in the search.

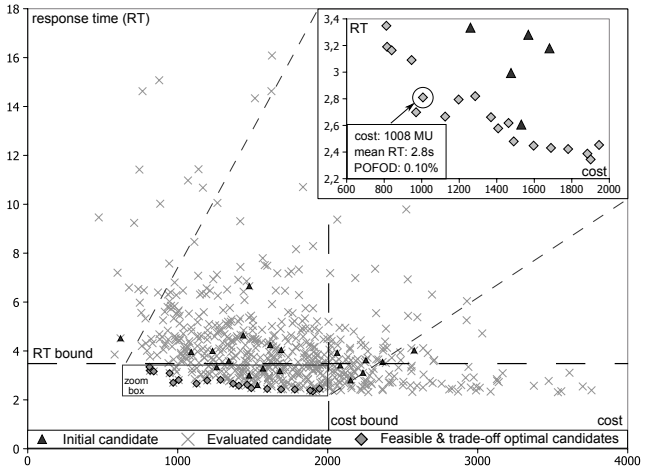


**Figure 9: Result of an optimization run with incorporated constraint handling.**

To make a qualitative statement, $\alpha_n$ will be compared with $\beta_m$ for all $n, m$. We compare the feasible, Pareto-optimal set $P$ of each run $n$ after $i$ iterations, denoted as $P(\alpha_n, i)$ and $P(\beta_n, i)$ respectively. For this reason, the *coverage metric* of [19] is slightly modified: Let $A$ and $B$ be *non-dominated sets*[3] and $Q \subseteq A \cup B$ be the non-dominated set of $A \cup B$. The coverage metric $\mathcal{C}$ is defined as $\mathcal{C}(A, B) := \frac{|A \cap Q|}{|Q|}$ ($\in [0, 1]$). If $\mathcal{C}(A, B) > 0.5$ then $A$ is considered better than $B$ as $A$ has a higher contribution to $Q$ than $B$.

With 4 runs per type, we have 16 combinations for comparisons of feasible, Pareto-optimal sets of solutions. In 13 combinations, the run with constraint handling was superior to the run without constraint handling. One type $\alpha$ run was inferior to three type $\beta$ runs. For each combination, the coverage metric for type $\alpha$ runs is calculated over the iterations. Figure 10 illustrates the maximum, minimum and mean of the combinations $\mathcal{C}(P(\alpha_n, i), P(\beta_m, i))$ with $n, m = 1, 2, 3, 4$ over all iterations $i$. The data can be interpreted as follows: In the first 45 iterations, the non-determinism causes a high variance in the progress with type $\alpha$ runs getting a better start as the candidates are pushed into the feasible region. After 55 iterations, the standard deviation stabilizes at approximately 0.11 as the optimizations converge. Additionally, the coverage of type $\alpha$ runs increases continuously until most near global solutions have been found reaching a mean coverage of approximately 0.61 at the end. Except for the first few iterations, the type $\alpha$ runs are superior to the type $\beta$ runs for every iteration considering the mean coverage value.

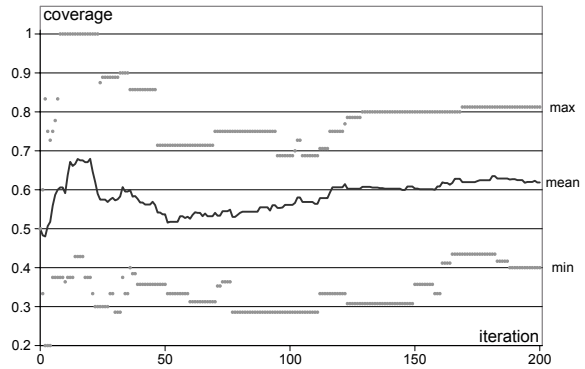


**Figure 10: Type $\alpha$ run coverage of type $\beta$ runs $\mathcal{C}(P(\alpha_n, i), P(\beta_m, i))$ with $n, m = 1, ..., 4$ and $i = 0, ..., 200$.**

Furthermore, we examined after how many iterations type $\alpha$ runs find solutions equivalent to the final result of type $\beta$ runs, i.e. formally stated, we first find the smallest $j$ for $\mathcal{C}(P(\beta_n, j), P(\beta_m, 200)) = 0.5$, then we find the smallest $i$ for $\mathcal{C}(P(\alpha_n, i), P(\beta_m, j)) = 0.5$. In other words, we compare the runs with constraint handling with the earliest iteration of runs without constraint handling where there is no change in solutions w.r.t. the final iteration. As an example, we compare $\alpha_3$ and $\beta_2$. $\beta_2$ has the last changes in the solution set in iteration 193. $\alpha_3$ has an equivalent solution set after 161 iterations, i.e. the run with constraint handling found equivalent results $\frac{193-161}{193} = 16.6\%$ faster. Table 1 shows all combinations demonstrating that we achieve a maximum time saving of 89.2%. Overall, we achieve a mean time saving of 38.8%. In verification, a Student's t-test considering all combinations of the time savings confirms that the difference is statistically significant (p-value = 0.0006685).

[3] In a non-dominated set, the elements are pairwise non-dominated (cf. [9]).

| | $\beta_1$ | $\beta_2$ | $\beta_3$ | $\beta_4$ |
|---|---|---|---|---|
| $\alpha_1$ | 72 (38.1%) | 76 (39.4%) | 170 (87.2%) | 120 (60%) |
| $\alpha_2$ | 87 (46.0%) | 172 (89.1%) | 174 (89.2%) | 130 (65%) |
| $\alpha_3$ | 13 (6.9%) | 32 (16.6%) | 83 (42.6%) | 124 (62%) |
| $\alpha_4$ | -76 (-38.2%) | -61 (-30.7%) | 102 (52.3%) | -8 (-4.0%) |

**Table 1: Absolute and relative time savings in iterations when using constraint handling during the optimization.**

# 7. CONCLUSION

This paper presents a novel extension of multi-criteria architecture optimization to consider bounds for quality requirements so that the search can focus on feasible regions of the search space. We extended the existing Quality of service Modeling Language to enable the specification of optimization goals and quality requirements. We translate the QML requirements to constraints in an optimization problem. Finally, we use existing constraint domination strategy to make the search focus on the feasible space.

With this extension, software architects can reduce the time needed to find valuable solutions. We demonstrated this ability in a case study. Our extension found solutions in the interesting regions of the objective space in average more than 35% faster than the old, unconstrained approach.

The application of this approach can be interesting in different phases of the software architecture design process. First, the approach can be applied after a first phase of creating an architecture with focus on functional requirements (definition of components and interfaces). This architecture can be used as an input for the optimization to improve the non-functional properties. Second, the optimization could already be used to support decisions during the architectural design: When making a more high level decision, the optimization can be used to assess the potential of the different alternatives. Finally, by modeling more high level decisions as transformations, these decisions could be included in the optimization process as degrees of freedom, thus letting the optimization explore different combinations of decisions.

We plan to extend the approach by also adding bounds for quality attributes that can express that a certain quality is enough and we are not interested in trading other qualities for extra improvement in this aspect. These bounds are not considered at the current stage, as we do not want to treat these bounds as infeasible regions when optimizing architectures. Additionally, we plan a more extensive validation that allows more detailed statistical conclusions on the effects of this work.

# 8. REFERENCES


[1] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya. Archeopterix: An extendable tool for architecture optimization of AADL models. *International ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES)*, pages 61–71, 2009.

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

[3] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, Second Edition.* Addison-Wesley Professional, 2003.

[4] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *JSS*, 82:3–22, 2009.

[5] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[6] J. Branke. Consideration of partial user preferences in evolutionary multiobjective optimization. In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, pages 157–178, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner. Parameterized Reliability Prediction for Component-based Software Architectures. In G. Heineman, J. Kofron, and F. Plasil, editors, *Proc. of the 6th Int'l Conference on the Quality of Software Architectures*, LNCS. Springer, 2010. To Appear.

[8] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for qoS-aware service composition based on genetic algorithms. In H.-G. Beyer and U.-M. O'Reilly, editors, *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005*, pages 1069–1075. ACM, 2005.

[9] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* Wiley, Chichester, 1st edition, 2001.

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

[11] S. Frølund and J. Koistinen. Qml: A language for quality of service specification. Tech. report hpl-98-10, Hewlett-Packard Laboratories, 1998.

[12] S. S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Trans. on Dependable and Secure Computing*, 4(1):32–40, 2007.

[13] M. Lukasiewycz. Opt4j - the optimization framework for java. http://www.opt4j.org, 2009.

[14] A. Martens, H. Koziolek, S. Becker, and R. Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *Proc. WOSP/SIPEW*, pages 105–116, New York, NY, USA, 2010. ACM.

[15] J. D. McGregor, F. Bachmann, L. Bass, P. Bianco, and M. Klein. Using arche in the classroom: One experience. Technical Report CMU/SEI-2007-TN-001, Software Engineering Institute, Carnegie Mellon University, 2007.

[16] D. A. Menascé, J. M. Ewing, H. Gomaa, S. Malex, and J. a. P. Sousa. A framework for utility-based service oriented design in SASSY. In *Proc. of WOSP/SIPEW*, pages 27–36. ACM, 2010.

[17] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework.* Addison-Wesley Longman, 2nd edition, 2009.

[18] J. Xu. Rule-based automatic software performance diagnosis and improvement. *Performance Evaluation*, In Press, Corrected Proof:–, 2009.

[19] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* PhD thesis, ETH Zurich, Switzerland, 1999.