

Focussing Multi-objective Software Architecture Optimization Using Quality of Service Bounds

Anne Kozirolek, Qais Noorshams, and Ralf Reussner

Karlsruhe Institute of Technology, Karlsruhe, Germany
{anne.kozirolek, noorshams, reussner}@kit.edu
<http://sdq.ipd.kit.edu/>

Abstract. Quantitative prediction of non-functional properties, such as performance, reliability, and costs, 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. We compare two different constraint handling techniques to consider the bounds. To validate our approach, we applied both techniques to an architecture model of a component-based business information system. We compared both techniques to an unbounded search in 4 scenarios. Every scenario was examined with 10 optimization runs, each investigating around 1600 architectural candidates. The results indicate that the integration of quality of service bounds during the optimization process can improve the quality of the solutions found, however, the effect depends on the scenario, i.e. the problem and the quality requirements. The best results were achieved for costs requirements: The approach was able to decrease the time needed to find good solutions in the interesting regions of the objective space by 25% on average.

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

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 costs. 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 costs 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, 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 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 the PEROPTERYX approach [15] by two aspects: First, we translate requirements specified with the *Quality of service Modeling Language (QML)* [12] into constraints in an optimization problem. Second, we use two constraint handling strategies [10, 11] to focus the search 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.

This paper extends a previous publication [19] by providing (1) the integration of a second constraint handling technique [11] and (2) more sound evaluation including the second technique, using more optimization runs and four different quality requirement scenarios. The evaluation leads to results with higher statistical significance and a more differentiated interpretation of the approaches' effects. We found that the constraint handling is beneficial in scenarios with strict quality bounds (i.e., where many candidates are infeasible). In these scenarios, our extension was able to find solutions in the interesting regions of the objective space in average 25.6% 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 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 5 shows the feasibility of our work by applying the process to an example architecture and comparing the effect of the requirements consideration. Finally, Section 6 concludes.

2 Related Work

Our approach is based on performance prediction [2], reliability prediction [13], multi-objective metaheuristic optimization [8], and constraint handling in evolutionary algorithms [9, 5]. A survey of constraint handling techniques is omitted here for brevity, but can be found in [18].

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 approaches except one do not support trade-off between quality attributes after the search. In addition, none of the approaches allows specifying 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. [20] 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. [16]) 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. [7] optimize service composition costs using evolutionary algorithms while satisfying service level agreement (SLA) constraints. They implement constraint handling with dynamic penalty functions.

Menascé et al. [17] 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 modelling 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 artefacts, which comprise a PCM instance (cf. Figure 1):

1. *Component specifications* contain an abstract, parametric description of components. Furthermore, the behaviour of the components is specified using a syntax similar to UML activity diagrams.
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 analysed or simulated to predict performance (response time and throughput) [4], reliability (probability of failure on demand (POFOD)) [6], and costs [15] of a system.

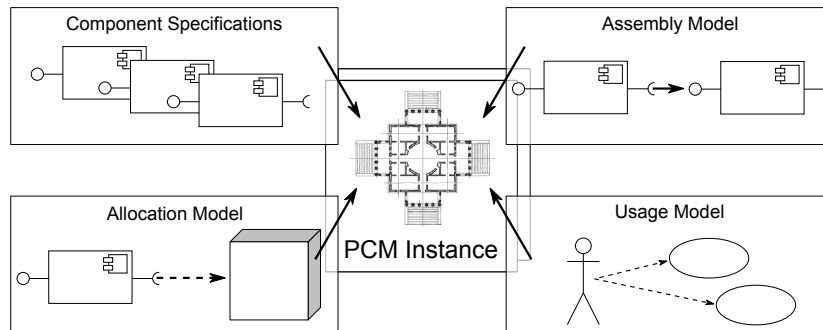


Fig. 1. Artefacts of a PCM instance.

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 behaviour description (incl. CPU demands) of one component is illustrated here by an activity diagram. Having only one usage scenario, a user interacts with the system every 5s

requesting a sequence of reports and views. User requests take different paths through the system based on passed parameters, expressed here as probabilities.

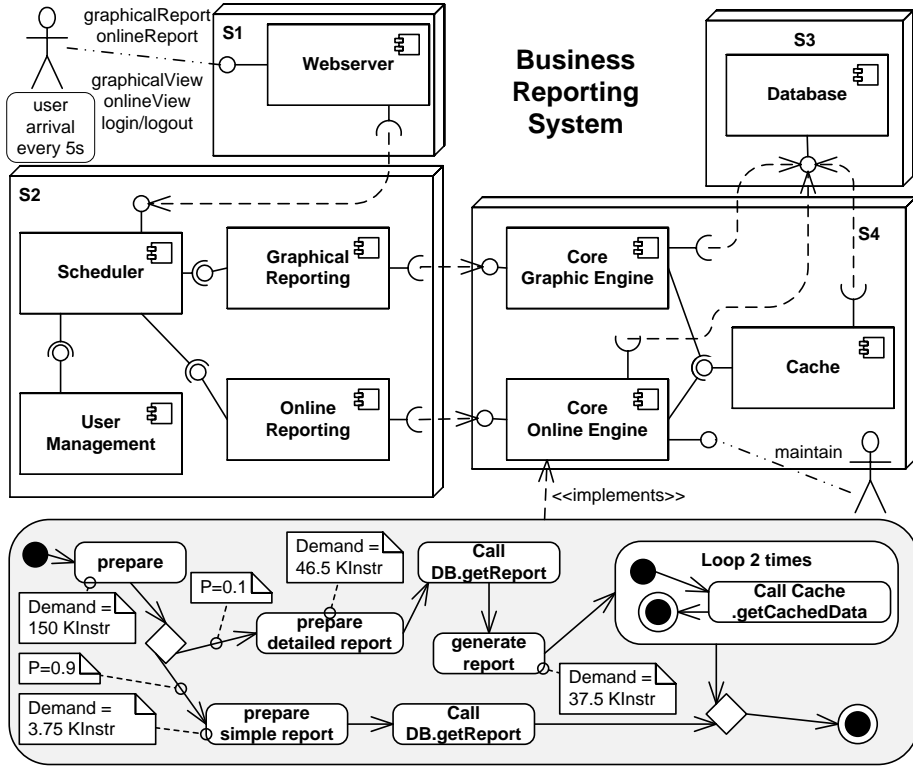


Fig. 2. PCM instance of the BRS (more detail in [15]).

4 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 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 [5]:

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.

4.1 Constraint Handling

To integrate requirements into this process, we extended the Opt4J framework [14], which implements basic NSGA-II without constraint handling, by the *constrained tournament* method (a.k.a. constrained NSGA-II) [10] and the *goal attainment* method [11], both described briefly below.

Requirements are transformed into constraints. If a solution violates any constraint, it is infeasible, i.e. useless for the user. Otherwise, it is feasible, thus a possible candidate to solve the problem. In their *constrained tournament* method (*C*), Deb et al. [10] handle constraints by modifying the dominance relation during the mating and the environmental selection of NSGA-II. Infeasible solutions are ranked according to their degree of infeasibility and declared inferior to feasible solutions.

In their *goal attainment* method (*G*), Fonseca et al. [11] define a goal value for each objective and aim at satisfying all goals by prioritizing objectives not fulfilling goals. Figuratively speaking, the Pareto-based comparison of two solutions is modified, such that before applying Pareto-dominance, the solutions are mapped on the goal value in the objectives that already fulfil the goal. Consequently, the objectives not fulfilling the goal have the impact on which solution dominates the other. Objectives for which no requirements exists are assigned a goal value of $+\infty$ when minimizing or $-\infty$ when maximizing.

We chose these two methods for constraint handling because of the following advantages: First, they explicitly distinguish between feasible and infeasible solutions and declare all feasible solutions superior to infeasible solutions as opposed to e.g. methods based on penalty functions. Second, no additional parameters are required (an advantage because many other methods are sensitive to parameter changes). Finally, they neither require a specific number of constraints nor assume a relation between objectives and/or constraints.

The difference of both methods is how solutions that violate the same constraints are treated: The constrained tournament technique uses a distance measure and favours solutions that are closer to the required values. In contrast, the goal attainment method uses standard Pareto-dominance if two solutions satisfy the same objectives.

4.2 Process

Figure 3 illustrates the optimization process as a whole with four main steps:

1. The system to be optimized is modelled 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. [15]), but this list is extendable to more and custom degrees of freedom.
2. Quality attributes (e.g. mean response time mrt) and quality requirements of the system (e.g. $mrt < 5\text{sec}$) are modelled using QML as described in [19]. Requirements are attached to a PCM usage scenario using 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 costs is pursued using NSGA-II with consideration of the requirements, using either *constrained tournament* or *goal-attainment*.
 4. As solving multi-objective optimization problems results in a set of solutions rather than one single solution [9], the set of feasible *Pareto-optimal*¹ architecture configurations with respect to the quality requirements is presented. Finally, the software architect makes the trade-off decision and chooses one of the solutions.

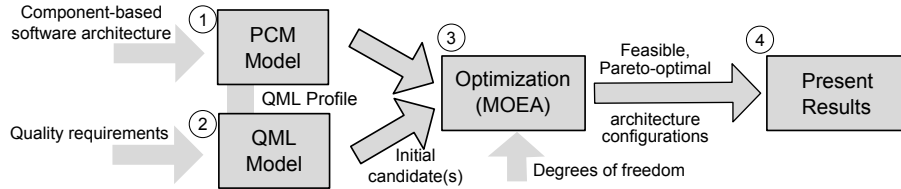


Fig. 3. Process Overview.

5 Case Study

This section describes a case study demonstrating the benefit of the consideration of requirements during the optimization process. The goal of this case study is to evaluate the benefits of the two constraint handling methods in different quality requirement scenarios.

5.1 Setup

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 costs.

As degrees of freedom, the components can be allocated to up to nine different servers. Three different webserver implementations with different costs, performance and reliability can be chosen. Additionally, each of the nine servers has a continuously variable CPU rate between 0.75 GHz and 3 GHz. The costs of the servers depend on the processing rate and the costs model is derived from Intel’s CPU price list. A power function is fitted to the data resulting in a costs

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

model of $costs = \sum_i costs(i) = \sum_i 0.7665 p_i^{6.2539}$ [monetary units (MU)] with the processing rate of each server p_i [GHz]. The coefficient of determination is $R^2 = 0.965$. Compared to [19], we used more realistic reliability values: the servers have a mean time to failure (MTTF) of 43800 hours and a mean time to repair (MTTR) of 3 hours. We get 19 (9+1+9) degree of freedom instances that can be independently varied.

To study the effects of different quality requirement values on the results, we ran the optimization for four different levels of requirements (weak, i.e., only few candidates are excluded from the Pareto front, to strict, i.e., many candidates are excluded). Table 1 shows the four different scenarios. The requirements are modelled with our metamodel of QML [19]. For each scenario $s \in \{1, 2, 3, 4\}$, we optimized the system once for each constraint handling technique $c \in \{C, G\}$, resulting in 8 optimization settings S_s^c , $1 \leq s \leq 4$. As a baseline, we optimized the system without constraint handling (setting S_0).

Scenario	costs	POFOD	mean response time
(S_1) Weak requirements	3000	0.00175	5.0 sec
(S_2) Medium requirements	2000	0.0015	3.0 sec
(S_3) Strict requirements	1500	0.0015	2.5 sec
(S_4) Only costs requirements	1000	∞	∞

Table 1. Quality Bound Scenarios

For each of the 9 settings, the system is optimized using PEROPTERYX. For statistical validity, we ran the optimization 10 times for every setting (runs r , $0 \leq r \leq 9$), so that in total, 90 runs rS_s^c have been performed. To exclude disturbing effects from differently generated random start populations, we randomly generated 10 start populations with 20 candidates each, and used these 10 start populations to run every setting (so that the r th run of any setting A starts with the same start population as the r th run of any setting B). Each optimization was stopped after 200 iterations.

5.2 Evaluation Measures

To compare the performance of the different settings, quality indicators have been suggested in the literature. Due to the trade-off nature of multi-objective optimization, there is no single quality indicator that objectively assesses an optimization run’s performance [21]. The coverage metric $\mathcal{C}(A, B)$ [22] is a useful measure to compare two optimization runs A and B’s results independent of the scaling of the objectives. However, the metric may be misleading if the Pareto fronts overlap each other with varying distances to the true optimal Pareto front. Additionally, both directions $\mathcal{C}(A, B)$ and $\mathcal{C}(B, A)$ have to be considered to assess the difference of the fronts. To overcome both problems, we (1) measure size of the dominated space $\mathcal{S}(A)$ [22] to assess the quality of each Pareto front A separately and (2) modify the coverage metric $\mathcal{C}(A, B)$ to make it symmetric.

Additionally, we include the quality bounds in the coverage metric, resulting in the following definition: Let A and B be *feasible, non-dominated sets*² and

² In a non-dominated set, the elements are pairwise non-dominated (cf. [9]).

$Q \subseteq A \cup B$ be the feasible, 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 because A has a higher contribution to Q than B .

The size of the dominated space $\mathcal{S}(A)$ measures the volume (in the three dimensional case) of the objective space weakly dominated by a Pareto front A . For minimisation problems, this measure requires a reference point to define the upper bounds of this volume. Here, we use the quality of service bounds and thus measure the size of the *feasible* space covered by A : $\mathcal{S}^*(A)$. For setting (4), which does not define upper bounds for response time and POFOD, we use the maximum values in all evaluated candidates of all runs as the upper bounds. Because the scale of the objectives are very different (POFOD ranges from 0, ..., 1, costs from 500 to 3500), and different upper bounds are used in the different settings, we normalize the objective values before determining the volume and, as a result, we cannot compare the absolute volumes across different settings.

We analyse the coverage \mathcal{C}^* of optimization runs with constraint handling over the basic optimization S_0 and compare the size of the dominated space \mathcal{S}^* . We study the effect of the constraint handling separately for each scenario $1 \leq s \leq 4$. To study the development of the optimization runs, we plot the coverage measure over the course of the optimization, i.e. determine it for each iteration $0 \leq i \leq 200$, written as $\mathcal{C}^*(A(i))$ for a run A . Similarly, we compare the size of the dominated feasible space over the course of the optimization runs. The size of the feasible space dominated by the basic approach is determined anew for each scenario $1 \leq s \leq 4$ with respect to the quality bounds of this scenario. Then, for each scenario s and each method $c \in \{C, G\}$, we aggregate the measures $\mathcal{C}^*({}^r S_s^c, {}^r S_0(i))$, $\mathcal{S}^*({}^r S_s^c(i))$, and $\mathcal{S}^*({}^r S_0(i))$ over all 10 runs r to account for the indeterministic nature of the optimization.

5.3 Results

Figure 4 illustrates the result of the optimization run ${}^0 S_3^C$ with medium constraints using the constrained tournament method C . 7 Pareto-optimal candidates that satisfy all three bounds were found and are marked with triangles.

We present the results in the following by scenario. Figures 5 and 6 show the coverage measure and the size measure for scenario 1. The coverage measure is around 0.5 in average over most of the iterations for both constraint handling methods C and G . With both measures, thus, no improvement towards the basic approach is visible. The size of the dominated feasible space grows similarly for all approaches, too.

Figures 7 and 8 show the coverage measure and the size measure for scenario 2. For both the coverage measure and the size measure, the runs with constraint handling start well (coverage > 0.5 and size larger than size of basic approach). However, the basic approach catches up: At iteration 200, all approaches perform equally well (G has a slightly better coverage, C a slightly larger dominated space, so none performs better than the other).

Figures 9 and 10 show the coverage measure and the size measure for scenario 3 with strict quality requirements. Here, we see an improvement of the search:

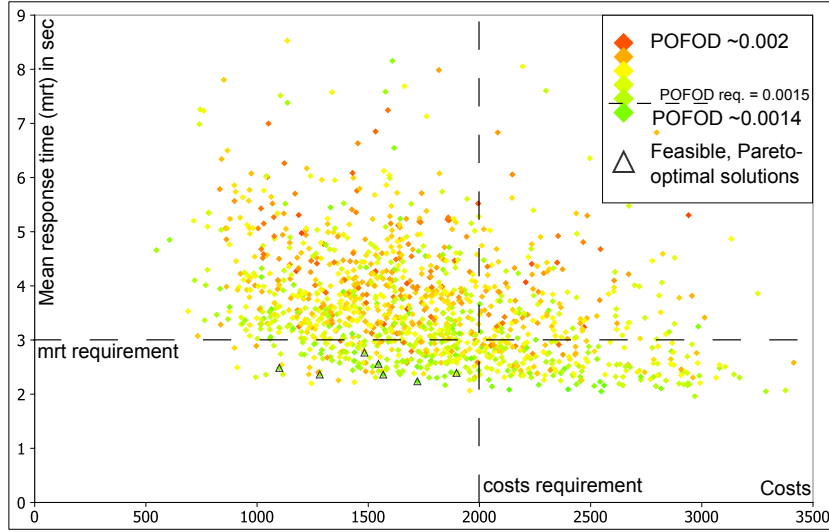


Fig. 4. Result of an optimization run ${}^0S_3^C$ with medium requirements $s = 3$ and the constrained tournament method $c = C$.

The coverage measure of method C is higher than 0.5 during all iterations, and the size measure is significantly larger than for the basic approach, too. Method G does not perform as well, even has a coverage < 0.5 at the beginning while still having a better size measure than the basic approach.

Finally, figures 11 and 12 show the results for the common case of a budget-only limitation. While both constraint handling methods do not perform well in the first 75 iterations, they catch up and provide better results in the last iterations, both regarding coverage and size measure.

To summarise, we observe that the quality bounds have almost no effect in lowly constrained scenarios 1 and 2. In scenario 3, the constrained tournament method C performs well in both coverage and even more so regarding the size of the dominated feasible space. The goal attainment method is less successful. In scenario 4, both constraint handling methods perform well. We conclude that using quality bounds to focus the search is only effective if a large portion of the search space are excluded by the quality bounds, such as given in scenarios 3 and 4. In the two first scenarios, fewer solutions on the Pareto-front are infeasible, so that the constraint handling is seldom used and thus cannot steer the search well. Because it is not necessarily known in advance whether given requirements are strict or lax, the constraint handling methods should always be used, as they do not worsen the performance of the search.

Furthermore, we examined after how many iterations type runs with constraint handling find solutions equivalent to the final result of basic approach runs based on both quality indicators \mathcal{C}^* and \mathcal{S}^* . For both indicators, we first find the smallest j for $\mathcal{C}^*(rS_s^c(j), rS_0(200)) = 0.5$ or $\mathcal{S}^*(rS_s^c(j)) > \mathcal{S}^*(rS_0(200))$, then we find the smallest i for $\mathcal{C}^*(S_0(i), S_0(200)) = 0.5$ or $\mathcal{S}^*(rS_0(i)) > \mathcal{S}^*(rS_0(200))$. In other words, we compare the runs with constraint handling with the earliest

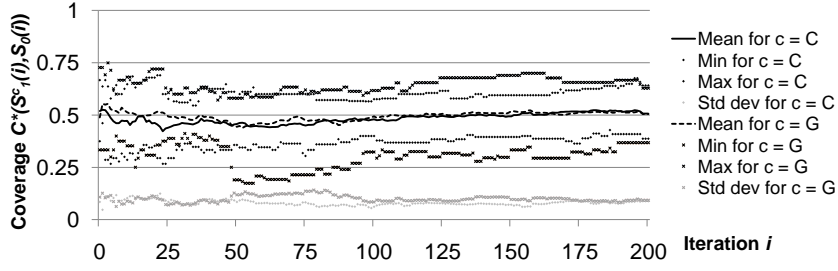


Fig. 5. Coverage measure $C^*(S_1^c(i), S_0(i))$ in scenario 1, aggregated over runs r

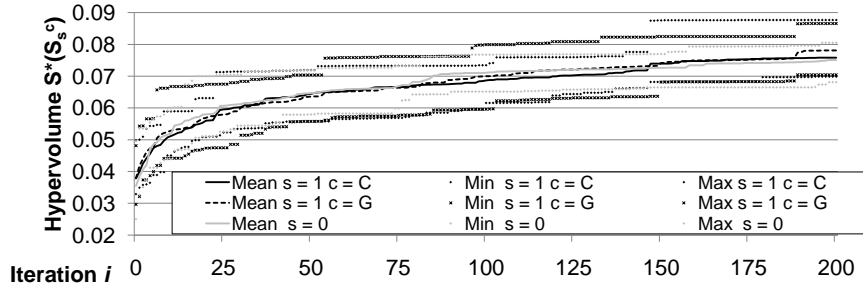


Fig. 6. Size of the dominated space $S^*(S_1^c(i))$ in scenario 1, compared to the basic scenario $S^*(S_0(i))$, aggregated over runs r

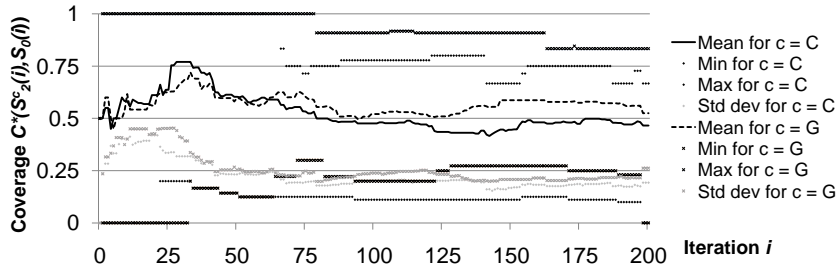


Fig. 7. Coverage measure $C^*(S_2^c(i), S_0(i))$ in scenario 2, aggregated over runs r

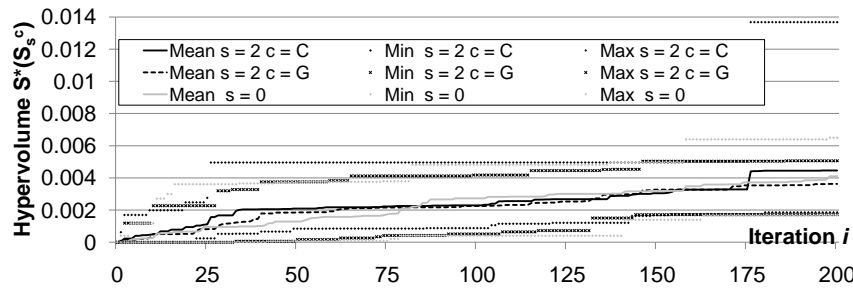


Fig. 8. Size of the dominated space $S^*(S_2^c(i))$ in scenario 2, compared to the basic scenario $S^*(S_0(i))$, aggregated over runs r

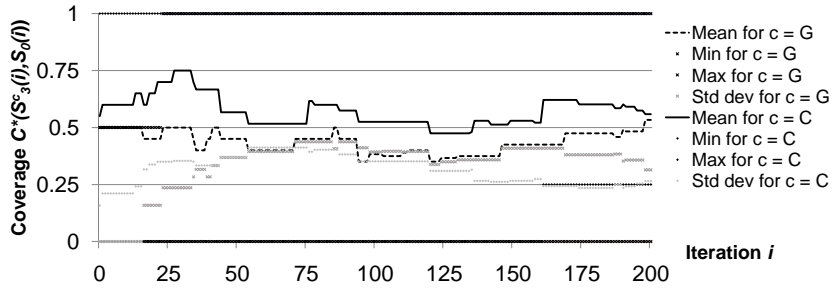


Fig. 9. Coverage measure $C^*(S_3^c, S_0(i))$ in scenario 3, aggregated over runs r

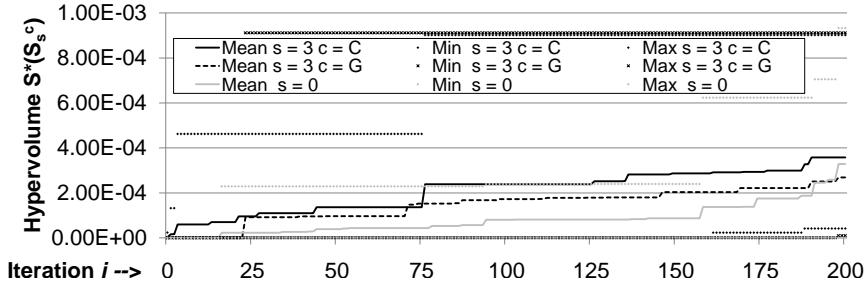


Fig. 10. Size of the dominated space $S^*(S_3^c(i))$ in scenario 3, compared to the basic scenario $S^*(S_0(i))$, aggregated over runs r

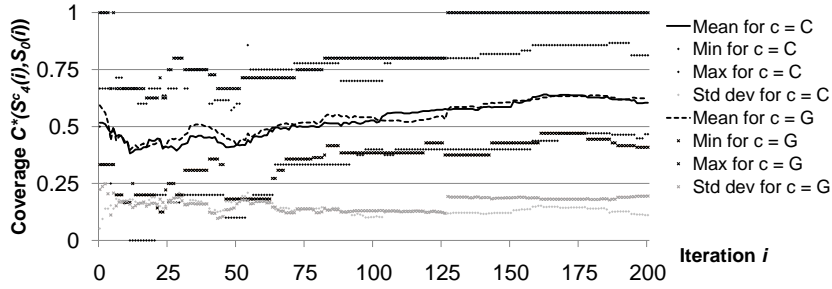


Fig. 11. Coverage measure $C^*(S_4^c, S_0(i))$ in scenario 4, aggregated over runs r

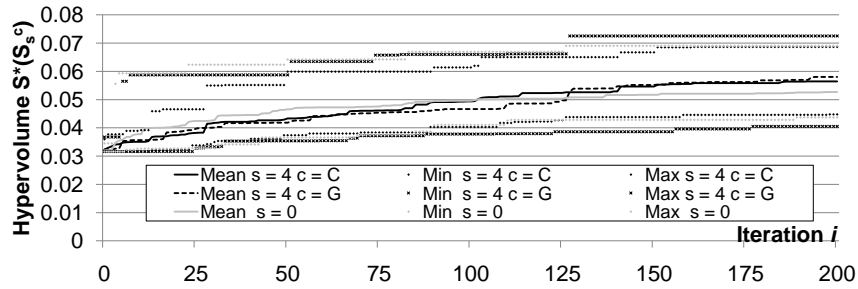


Fig. 12. Size of the dominated space $S^*(S_4^c(i))$ in scenario 4, compared to the basic scenario $S^*(S_0(i))$, aggregated over runs r

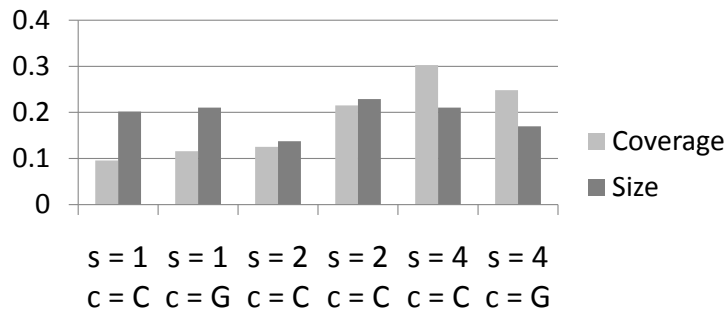


Fig. 13. Time Savings

iteration of basic approach runs where there is no change in solutions w.r.t. the final iteration. We measure the relative time saving $t = \frac{i-j}{\max(j,i)}$. As an example, we compare ${}^2S_s^c$ and 2S_0 regarding the coverage \mathcal{C}^* . ${}^2S_s^c$ has an equivalent solution set than ${}^2S_s^c(200)$ after 171 iterations. 2S_0 has the last changes in the solution set in iteration 192. Thus, the run with constraint handling found equivalent results $\frac{192-171}{192} = 10.9\%$ faster.

Figure 13 shows the relative time savings for scenarios 1, 2, and 4. In scenario 3, too few solutions were feasible and Pareto-optimal at the end, so that a sensible assessment of the time saving is not possible. We observe that for all scenarios, the constraint handling methods can find an equivalent front faster than the basic approach. The average time saving is 11.1% with respect to \mathcal{C}^* and 11.8% with respect to \mathcal{S}^* , and with the most time saving in scenario 4 with the constrained tournament method (30.3% for \mathcal{C}^* and 21.0% for \mathcal{S}^* , average 25.6%).

In further experiments [18], we have also studied to add lower bounds indicating that a quality values is good enough so that further improvement does not bring additional benefit, i.e. that other quality properties should not be traded off for more improvement of this value. However, we found that including such lower bounds does not significantly improve the optimization performance, neither in isolation nor in combination with upper bounds as presented in this work. Note that while we assume minimisation problems in this work, maximization problems can be inverted and handled as well, so that minimal allowed values are translated to upper bounds in our approach.

6 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 translate the quality requirements modelled in QML to constraints in an optimization problem. Then, we use existing constraint handling strategies to make the search focus on the feasible space. We compared the performance of two constraint handling strategies, namely constrained tournament methods and

the goal attainment method, in several scenarios in a case study. We found that constraint handling, especially the constrained tournament method, improves the efficiency of the search if strict requirements are used, i.e. if a significant portion of the objective space is defined to be infeasible. Additionally, we found that the constrained tournament method was superior to the goal attainment method in our setting.

With this extension, software architects can reduce the time needed to find valuable solutions. Our extension found solutions in the interesting regions of the objective space in average between 15% and 30% faster than the old, unconstrained approach for scenario 4 with strict requirements.

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 modelling 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.

As future work, we could investigate the effect of constraint handling if other metaheuristic optimization approaches than NSGA-II are used. Additionally, we plan to integrate quality attribute tactics into the search, to allow the search to improve a given candidate using domain knowledge, e.g. by balancing the load on the used servers to improve response time. In combination with bounds, tactics could be used to more directly steer the search towards feasible regions, which could be especially beneficial in highly constrained problems.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable feedback.

References

1. Aleti, A., Björnander, S., Grunske, L., Meedeniya, I.: Archeopterix: An extendable tool for architecture optimization of AADL models. In: Proc. of ICSE 2009 Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES). pp. 61–71. IEEE Computer Society (2009)
2. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering* 30(5), 295–310 (May 2004)
3. Bass, L., Clements, P., Kazman, R.: *Software Architecture in Practice*, Second Edition. Addison-Wesley, Reading, MA, USA (2003)
4. Becker, S., Koziolok, H., Reussner, R.: The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 82, 3–22 (2009)
5. Branke, J.: Consideration of partial user preferences in evolutionary multiobjective optimization. In: *Multiobjective Optimization: Interactive and Evolutionary Approaches*. pp. 157–178. Springer-Verlag, Berlin, Heidelberg (2008)

6. Brosch, F., Koziolok, H., Buhnova, B., Reussner, R.: Parameterized Reliability Prediction for Component-based Software Architectures. In: Proc. of QoS. LNCS, vol. 6093, pp. 36–51. Springer (2010)
7. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO). pp. 1069–1075. ACM (2005)
8. Coello Coello, C.A., Dhaenens, C., Jourdan, L.: Multi-objective combinatorial optimization: Problematic and context. In: Advances in Multi-Objective Nature Inspired Computing, Studies in Computational Intelligence, vol. 272, pp. 1–21. Springer, Heidelberg (2010)
9. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester, UK (2001)
10. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm : NSGA-II. *Evolutionary Computation*, IEEE Transactions on 6(2), 182–197 (August 2002)
11. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: ICGA. pp. 416–423. Morgan Kaufmann (Jun 1993)
12. Frølund, S., Koistinen, J.: QML: A Language for Quality of Service Specification. Tech. Report HPL-98-10, Hewlett-Packard Laboratories (1998)
13. Gokhale, S.S.: Architecture-based software reliability analysis: Overview and limitations. *IEEE Trans. on Dependable and Secure Computing* 4(1), 32–40 (2007)
14. Lukasiewicz, M., Glaß, M., Reimann, F., Helwig, S.: Opt4J - The Optimization Framework for Java. <http://www.opt4j.org> (2010)
15. Martens, A., Koziolok, H., Becker, S., Reussner, R.: Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: Proc. of Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW). pp. 105–116. ACM, New York, NY, USA (2010)
16. McGregor, J.D., Bachmann, F., Bass, L., Bianco, P., Klein, M.: Using arche in the classroom: One experience. Tech. Rep. CMU/SEI-2007-TN-001, Software Engineering Institute, Carnegie Mellon University (2007)
17. Menascé, D.A., Ewing, J.M., Gomaa, H., Malex, S., Sousa, J.a.P.: A framework for utility-based service oriented design in SASSY. In: Proc. of Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW). pp. 27–36. ACM (2010)
18. Noorshams, Q.: Focusing the Optimization of Software Architecture Models Using Non-Functional Requirements. Master’s thesis, Karlsruhe Institute of Technology, Germany (2010)
19. Noorshams, Q., Martens, A., Reussner, R.: Using quality of service bounds for effective multi-objective software architecture optimization. In: QUASOSS ’10: Proceedings of the 2nd International Workshop on the Quality of Service-Oriented Software Systems. pp. 1:1–1:6. ACM, New York, NY, USA (2010)
20. Xu, J.: Rule-based automatic software performance diagnosis and improvement. *Performance Evaluation* 67(8), 585–611 (2010), special Issue on Software and Performance
21. Zitzler, E., Knowles, J., Thiele, L.: Quality Assessment of Pareto Set Approximations, LNCS, vol. 5252, pp. 373–404. Springer-Verlag, Berlin (2008)
22. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evolutionary Computation* 3(4), 257–271 (1999)