A Hybrid Approach for Multi-Attribute QoS Optimisation in Component Based Software Systems

Anne Martens[1], Danilo Ardagna[2], Heiko Koziolek[3],
Raffaela Mirandola[2], and Ralf Reussner[1]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
{martens, reussner}@kit.edu
[2] Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milano, Italy
{ardagna, mirandola}@elet.polimi.it
[3] ABB Corporate Research, Ladenburg, Germany
heiko.koziolek@de.abb.com

Abstract. Design decisions for complex, component-based systems impact multiple quality of service (QoS) properties. Often, means to improve one quality property deteriorate another one. In this scenario, selecting a good solution with respect to a single quality attribute can lead to unacceptable results with respect to the other quality attributes. A promising way to deal with this problem is to exploit multi-objective optimization where the objectives represent different quality attributes. The aim of these techniques is to devise a set of solutions, each of which assures a trade-off between the conflicting qualities. To automate this task, this paper proposes a combined use of analytical optimization techniques and evolutionary algorithms to efficiently identify a significant set of design alternatives, from which an architecture that best fits the different quality objectives can be selected. The proposed approach can lead both to a reduction of development costs and to an improvement of the quality of the final system. We demonstrate the use of this approach on a simple case study.

1 Introduction

One of the today issues in software engineering is to find new effective ways to deal intelligently with the increasing complexity of software-intensive computing system. In this context a crucial role is played by the achievement of quality requirements, such as performance and availability.

In recent decades, software architecture (SA) has emerged as an appropriate level for dealing with software qualities [10, 31] and several efforts have been devoted to the definition of methods and tools able to evaluate quality at SA level (see, for example, [3, 25, 15, 31]). However, each method usually addresses a single quality attribute (e.g., performance or availability), while a major challenge in system development is finding the best balance between different, possibly conflicting quality requirements that a system has to meet and cost constraints (e.g., maximize performance and availability, while minimizing cost).

For these multi-attribute problems, there is usually no single global solution, and a promising way to deal with them is to exploit multi-objective optimisation [16, 6] where the objectives represent different quality attributes. The aim of these techniques is to devise a set of solutions, called Pareto optimal solutions or Pareto front [16], each of which assures a trade-off between the conflicting qualities. In other words, while moving from one Pareto

solution to another, there is a certain amount of sacrifice in one objective(s) to achieve a certain amount of gain in the other(s). This activity is time consuming, thus the software architect needs an automated method that efficiently explores the architectural design space with respect to the multiple quality attributes. Previous approaches in this direction use evolutionary algorithms [27], however, the derived optimisation process is time-consuming.

To overcome these drawbacks, this paper proposes a method where different design alternatives are automatically generated and evaluated for different quality attributes, providing the software architect with a powerful decision making tool enabling the selection of the SA that best fits multiple quality objectives. The proposed approach is centered around a hybrid approach, where an initial SA of the system (fulfilling its functional requirements) is taken as input. Based on this initial solution, a search problem is formulated by defining "degrees of freedom". The identification of a significant set of design alternatives is then based on a combined use of analytical optimisation techniques and evolutionary algorithms [6]. This hybrid approach extends the work presented in [27], introducing a step based on analytical optimisation whose goal is to derive very efficiently an approximated Pareto front with respect to a simplified search space. The obtained results are used as input candidates for an evolutionary optimisation of the original search problem. In this way, more accurate estimates for availability and performance metrics and a larger Pareto optimal solution set can be obtained. The advantages of this hybrid approach are shown in a case study.

The proposed method can lead both to a reduction of development costs and to an improvement of the quality of the final system, because an automated and efficient search is able to identify more and better design alternatives.

The remainder of the paper is organized as follows. Section 2 introduces the adopted architectural model and quality prediction techniques. Section 3 describes the optimisation process. Experimental results are presented in Section 4. Section 5 reviews other literature proposals. Conclusions are finally drawn in Section 6.

2 Background: Architecture Modelling and Analyses

In this section, we present the architectural model and the existing quality analyses methods our approach is based on. To quickly convey our contributed concepts to the reader, we introduce an example system.

Our approach requires a component-based architecture model with performance, availability, and costs annotations as input. Balsamo et al. [3] and Koziolek [25] have surveyed many different methods for specifying performance models, and Smith and Williams [31] have provided a set of guidelines on how to obtain performance models during early development stages and on how to refine such models as the implementation progresses. For reliability, Gokhale [17] provides a survey.

In our approach, we adopt the Palladio Component Model (PCM) [5], but our approach could be extended to consider other architectural performance and availability models and analysis methods. The PCM is beneficial for our purposes as it is specifically designed for component-based systems. Thus, the PCM naturally supports many architectural degrees of freedom (e.g., substituting components, changing component allocation, etc.). Additionally, the model-driven capabilities of the PCM allow an easy automated generation of alternative architecture candidates.

The example system in this paper is the so-called business reporting system (BRS), which lets users retrieve reports and statistical data about running business processes from

a data base. It is loosely based on a real system [33]. Fig. 1 shows some parts of the PCM model of the BRS visualised using annotated UML diagrams. It is a 4-tier system consisting of several software components. In an open workload usage scenario, requests arrive according to a Poisson process with a rate equal to 0.2 req/sec. Users issue three types of requests, that lead to varying execution paths in the system.

Components are annotated with software cost (*Cost*) in K€. The initial system is deployed to four servers annotated by costs (*HCost*) in K€, availability (*HA*) and processing rate (*PR*) in GHz. Fig. 1 also shows an excerpt of the behaviour in the lower half of the figure. The behaviour contains the CPU resource demands (*demand* in sec on a 2.6GHz CPU, log normally distributed with coefficient of variation equal to 2) and failure probabilities (*FP*) for actions. `ExternalCallActions` model calls to other components. The components are allocated on four different servers. The complete model can be found at [36].

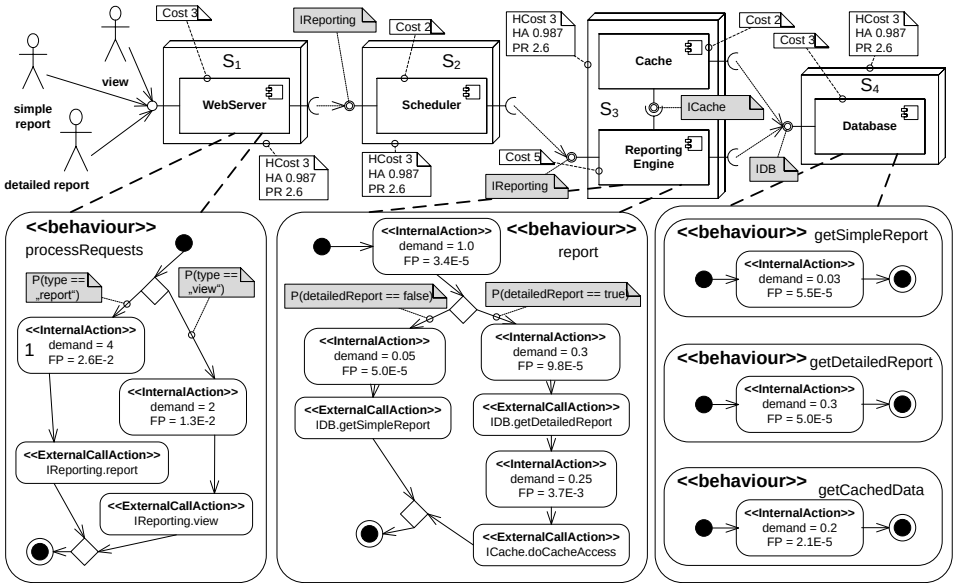

Fig. 1. Business Reporting System: PCM instance of the case study system

In order to provide SAs with a priori performance and availability guarantees, if the application include any loops, they are annotated with a discrete probability distribution and an upper bound for their number of execution exists. In the following, we briefly explain the analysis methods for the considered quality criteria performance, availability, and costs:

– **Performance:** For the analytic optimisation, we model the software system by introducing an M/G/1 queue for each physical server. For the evolutionary optimisation, we use an automated transformation of PCM models into a discrete-event simulation (SimuCom [5]) to derive response times. The performance model is in the class of extended queueing networks, so that we can analyse models containing resource demands specified as arbitrary distribution functions. However, the simulation can be time-consuming to derive stable results.

– **Availability:** For the analytic optimisation, we consider the well known serial/parallel formula [2] applied to the PCM model. In particular, the availability is evaluated by considering the set of components involved, the physical servers supporting the execution, and the probability of invocations. For the evolutionary optimisation, we use an automated transformation of PCM models into absorbing discrete time Markov chains (DTMC) and solve them with the PCM Markov solver [8].
– **Costs:** We annotate constant costs to each component and each server configuration. The software architect can choose whether costs values represent procurement costs, total costs of ownership, or other. If a server specified in the model is not used, i.e., no components are allocated to it, its costs do not add to the overall costs. The goal of this simplistic model is to allow costs to be considered, not to provide a sophisticated costs estimation technique. For the latter, existing costs estimation techniques such as COCOMO II [7] could be integrated here to obtain more accurate values.

Although the example in Fig. 1 is not particularly complicated, it is not obvious how to change the architectural model efficiently to improve the quality properties. For example, the software architect could increase the processing rate of server $\mathtt{S}_1$, which would result in better performance but higher costs. The software architect could also change the component allocation ($4^5 = 1024$ possibilities) or incorporate other component specifications with different QoS attributes.

The design space for this example is huge. Manually checking the possible design alternatives in a trial-and-error approach is laborious and error-prone. The software architect cannot easily create design alternatives that are even locally optimal for all quality criteria. Finding global optima is practically impossible because it requires modelling each alternative. In practice this situation is often mitigated by overprovisioning (i.e., incorporating fast and expensive hardware resources), leading to unnecessarily high costs.

3 Optimisation Process

To present our hybrid optimisation approach, we first give an overview in Section 3.1. In Section 3.2, we describe the search problem. Then, we describe in detail the analytical optimisation (Section 3.3) and the evolutionary optimisation (Section 3.4).

3.1 Overview

Our approach starts considering as input an initial architectural model of the system, named *initial candidate* in Fig. 2. In our case, this is a complete PCM model instance as shown in Fig. 1. The optimisation process starts with the *search problem formulation*. In this work, we consider three degree of freedom types: (1) allocation of components, (2) server configuration, and (3) component selection. The result of this step is a set of system-specific degrees of freedom that describe the search problem.

In the second step, a simplified version of the search problem is optimised using analytic techniques. The impact of a degree of freedom (or a combination of them) is evaluated by analytic models, and the Pareto optimal candidates are derived very efficiently by solving a mixed integer linear programming problem. The result of this step is a set of candidates that are globally Pareto-optimal with respect to the simplified search space. In the third step, the results of the analytic optimisation are used as input candidates for an evolutionary optimisation of the original search problem. Evolutionary optimisation is more time consuming, but it can consider the whole search space and obtain more accurate estimates for availability and performance metrics.

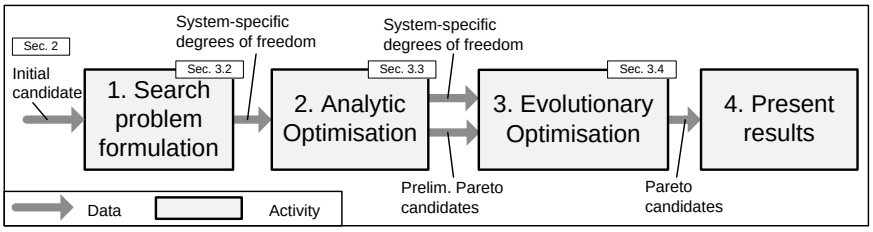

Fig. 2. Hybrid Optimisation Process Overview

The results of the evolutionary optimisation phase is a set of Pareto-optimal candidates. The Pareto-optimal candidates are presented to the software architect, who can study the remaining optimal trade-offs between possibly conflicting objectives.

3.2 Search Problem Formulation

Candidate solutions can be evaluated for optimal trade-offs, i.e. for Pareto-optimality [16]. A candidate architecture is Pareto-optimal, if it is superior to all other candidate in at least one quality criterion. More formally: Let a be a candidate solution, let DS be the set of all possible candidates, and let q be a quality criterion with a value set D_q, an evaluation function $f_q : DS \rightarrow D_q$ so that $f_q(c)$ denotes the quality property of a $c \in DS$ for the quality criterion q, and an order $\leq_q$ on D_q so that $c_1 \leq_q c_2$ means that c_1 is better than or equal to c_2 with respect to quality criterion q. Then, a candidate solution a is Pareto-optimal iff $\forall b \in DS \; \exists q : f_q(a) \leq_q f_q(b)$. If a candidate solution is not Pareto-optimal, then it is Pareto-dominated by at least one other candidate solution in DS that is better or equal in all quality criteria. The optimisation problem can be formulated as follows for a set of quality criteria $Q = \{q_1, ..., q_m\}$: $\min_{c \in DS} [f_{q_1}(c), ..., f_{q_m}(c)]$. In this work, we consider three quality criteria: $q_1 = T =$ mean response time, $q_2 = A =$ availability measured as the probability of success of each request, and $q_3 = C =$ cost.

In our approach, the following degrees of freedom can be considered:

Allocation of components to available servers: The mapping of components to servers can be changed. This is an integral part of most performance-prediction models and has large effects on the performance of a system. When reallocating components, the number of servers can change as well. In our example, the `Scheduler` component could be allocated to $\texttt{S}_1$, so that $\texttt{S}_2$ could be removed and its cost can be saved. The software architect can specify the maximum number of servers to be considered.

Server configuration: The available hardware resources (CPU, HDD, ...) can be changed in a certain range. In this work, we model a discrete set of servers with different CPU processing rates and costs. Thus, components can be allocated to servers with different processing rates.

Component selection: If functionally-equivalent components with different non-functional properties are available, they can be exchanged. Currently, we deem that a component B can replace a component A if B provides (i.e., implements) all interfaces provided by A and if B requires at most the interfaces required by A.

More degrees of freedom that could be considered in an automated approach are described in [27]. In the search problem formulation step, the initial candidate model is automatically analysed for instantiations of these degrees of freedom, called system-specific degrees of freedom. The found set of system-specific degrees of freedom defines the search space. If desired, the software architect can also manually remove some of them.

3.3 Analytical Optimisation

The analytical optimization step starts by evaluating the quality metrics of each component i included in the initial candidate by means of M/G/1 and availability formula. Then, a binary decision variable x_j is introduced for each "atomic" design alternative which can be obtained from the degrees of freedom. x_j is equal to 1 if the corresponding design alternative is implemented in the system, and 0 otherwise. The optimization problem which can be introduced in this way is combinatoric in nature, since a Pareto optimal solution can be obtained by selecting a combination of atomic design alternatives.

For example, S_1 alternative configurations for the reference system in Fig. 1 can be modelled introducing the binary variables x_1 (CPU downgrade to 2.4 GHz), x_2 (CPU upgrade to 2.8 GHz), x_3 (CPU upgrade to 3 GHz). Down/upgrades of servers S_2, S_3, and S_4 can be modelled analogously with variables x_4 to x_{12}. Likewise, the alternative components selection can be modelled by introducing two binary variables x_{13} and x_{14} equal to 1 iff the `WebServer` is replaced by alternative `WebServer2` or `WebServer3` implementation.

The limit of the analytical optimisation with respect to the evolutionary search is in the evaluation of the *allocation* of components to servers degree of freedom. The analytical problem formulation cannot remove a server from the system if no components are allocated to it. The aim of the analytical optimisation is to derive quickly an approximated Pareto front which will be further refined by the evolutionary search. As it will be discussed in Section 4, providing approximated Pareto solutions to the evolutionary search allows to improve the whole analysis process. For the sake of simplicity in the following we assume that the application under study includes a single initial component and a single end component. Furthermore, loops are peeled and transformed into a number of branches with varying number of repetitions according to the annotated probability distribution [1]. In this way the application PCM model is transformed into a Directed Acyclic Graph (DAG).

Let us denote with $\mathcal{I}$ the set of indexes of the system components and with $\mathcal{J}$ the set of indexes for the atomic design alternatives arising from the degrees of freedom definition. Let us denote by $\tilde{C}$ the cost of the initial candidate and let δ_j^c be the cost variation of the initial candidate for implementing the design alternative j.

In the following optimization problem formulation we will consider only the average response time performance metric, availability optimization can be formalized similarly. Let us denote with $\tilde{t}_i$, the average response time for component i invocation in the initial candidate and let $\delta_{j,i}^t$ be the variation of the response time (evaluated by means of M/G/1 formula) for component i if the design alternative j is implemented. For example, if S_1 CPU frequency is raised to 2.8 GHz (x_2 design alternative), then the `WebServer` service demands for the two invocations and S_1 utilization are initially equal to 4 sec, 2 sec and 0.52 respectively, are reduced by a factor $2.8/2.6 = 1.08$. Thus, the initial response times equal to 8.33 sec, and 4.16 sec become 7.18 sec and 3.59 sec and hence the deltas are equal to -1.15 sec, and -0.57 sec.

Some of the atomic design alternatives could be in conflict. For example, since only one server CPU can be changed at one instance, the following constraint has to be introduced for S_1:

$$x_1 + x_2 + x_3 \leq 1$$

Formally, we introduce an exclusive set es_k for each combination of atomic design alternatives which are in conflict among each other, because they concern the same software component and/or the same physical server where components are deployed. A parameter

$es_{k,j} = 1$ is introduced indicating that the atomic design alternative j is in the exclusive set k, while $es_{k,j} = 0$ otherwise.

Note that the size of exclusive sets could grow exponentially, since taking into account all of the atomic choices is also combinatorial in nature. However, since the number of possibly conflicting atomic design alternatives is usually significantly lower than the number of degrees of freedom, the analytic problem can be formulated and solved efficiently, as it will be shown in Section 4.2.

If we denote by t_i the execution time of component i according to the selection of atomic design choices, we have:

$$t_i = \tilde{t}_i + \sum_{j \in \mathcal{J}} \delta^t_{j,i} x_j, \forall i; \qquad \sum_{j \in \mathcal{J}} es_{k,j} x_j \leq 1, \forall k$$

Let us denote with π_i the probability of execution of component i which can be derived from the sum of the transition probabilities of the paths in the DAG from the initial component to i. The execution time T of the whole application can then be computed as $T = \sum_{i \in \mathcal{I}} \pi_i \cdot t_i$, while the cost C corresponding to a given combination of atomic choices is given by $C = \tilde{C} + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \delta^c_j \cdot x_j$.

If $\overline{T}$ and $\overline{C}$ denote a bound for the application execution and system cost respectively, than the Pareto-optimal solutions can be obtained by solving iteratively the problems shown in Fig. 3 according to Algorithm 1.

(P1) $\min C$

subject to:

$$C = \tilde{C} + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \delta^c_j \cdot x_j$$

$$t_i = \tilde{t}_i + \sum_{j \in \mathcal{J}} \delta^t_{j,i} x_j, \ \forall i$$

$$\sum_{j \in \mathcal{J}} es_{k,j} x_j \leq 1, \ \forall k$$

$$\sum_{i \in \mathcal{I}} \pi_i \cdot t_i \leq \overline{T}$$

(P2) $\min T$

subject to:

$$t_i = \tilde{t}_i + \sum_{j \in \mathcal{J}} \delta^t_{j,i} x_j, \ \forall i$$

$$T = \sum_{i \in \mathcal{I}} \pi_i \cdot t_i$$

$$\sum_{j \in \mathcal{J}} es_{k,j} x_j \leq 1, \ \forall k$$

$$\tilde{C} + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \delta^c_j \cdot x_j \leq \overline{C}$$

Fig. 3. The Analytic Optimisation Problems for Performance and Cost

Algorithm 1 requires as input the upper $\overline{T}^{upper}$ and lower bound $\overline{T}^{lower}$ response time for the application under study, which can be computed easily by considering the maximum and minimum $\delta^t_{j,i}$ for each component i. Then, the Algorithm starts minimizing the system cost with the goal to provide a response time lower than $\overline{T}^{upper}$ (i.e., solving problem (P1), see step 4). Let x^* be the corresponding optimum solution (i.e., the set of atomic design alternatives to be implemented) and C^* be the corresponding cost. Then, the first Pareto solution is obtained by solving (P2) setting $\overline{C} = C^*$ (see step 6). Let T^* be optimum response time obtained. Indeed, no other atomic design alternative combination can lead to a better response time with a lower cost, hence x^* computed at step 6 is a Pareto global optimum solution. The process is then iterated by solving (P1) again and setting as constraint $\overline{T} = T^* - \epsilon$, where $\epsilon > 0$ is any sufficiently small constant. $IC + x^*$ at step 7 denotes the

solution obtained by applying to the initial candidate IC the set of atomic design alternatives x^*.

input : $\overline{T}^{upper}, \overline{T}^{lower}$
output: $Paretos$
1 $\overline{T} \leftarrow \overline{T}^{upper}$;
2 $Paretos \leftarrow \emptyset$;
3 **while** $\overline{T}^{lower} \leq \overline{T}$ **do**
4 Solve (P1). Let be x^* the optimum solution found and C^* its cost ;
5 $\overline{C} \leftarrow C^*$;
6 Solve (P2). Let be x^* the optimum solution found and T^* the application execution time ;
7 $Paretos \leftarrow Paretos \bigcup \{IC + x^*\}$;
8 $\overline{T} \leftarrow T^* - \epsilon$
9 **end**
10 **return** $Paretos$;

Algorithm 1: Analytical Pareto-optimality Algorithm

For the availability analysis the analytical problem formulation can be derived similarly. The main difference is that the delta values have to be derived for independent application execution paths (i.e., each path from the source to the sink) and the optimization has to be iterated for each execution path. The set of initial candidates provided to the evolutionary optimization is obtained as union of the analytical solutions of individual execution paths. It can be shown that (P1) and (P2) are NP-hard, since they are equivalent to a knapsack problem. The solution complexity grows exponentially with the number of binary variables. However, current solvers are very efficient and (P1) and (P2) solutions can be computed very quickly for realistic design problems of reasonable size.

3.4 Evolutionary Optimisation

If all degrees of freedoms presented in Section 3.2 have to be considered, then the analytical optimization model becomes a mixed integer non-linear problem and we cannot rely on efficient solvers as for the linear case to determine software architectures quality trade-offs. For this type of problems, metaheuristic optimisation techniques have been successfully applied in software engineering [19]. In this work, we use evolutionary optimisation (see, e.g. [6, p. 284]), as it has been considered useful for multi-objective problems [12]. Other metaheuristics could be used as well. More details on this choice can be found in [27].

Fig. 4 shows the main steps of our evolutionary search. The method is described here exemplary for our current realisation in the PERFOPTeryx tool [36] with the NSGA-II evolutionary algorithm [14] as implemented in the Opt4J framework [26] with an extended reproduction step.

The process starts with an input population derived from the analytical optimisation step. Individuals are then modified along system-specific degrees of freedom (see Section 3.1). As the software model contains all required annotations, all steps of the search can be completely automated. The population size n and the number of iterations i can be configured. If the input population size $|Paretos|$ is less than n, additional $n - |Paretos|$ random candidates are generated. The evolutionary search then iterates the following steps:

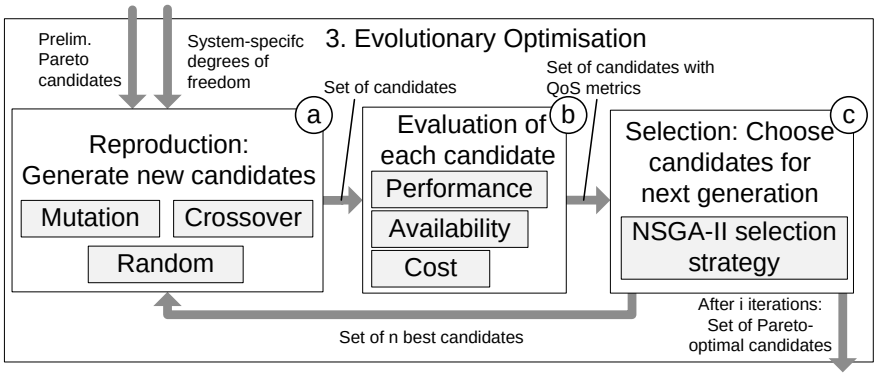

Fig. 4. Evolutionary optimisation process

(a) **Reproduction:** Based on the currently available candidates in the population, new candidate solutions are derived by "mutation" or "cross-over" or they are randomly created. With mutation, one or several design options are varied. With cross-over, two good candidate solutions are merged into one, by taking some of each candidates design option values for the cross-over. In addition to the original NSGA-II, in order to diversify the search, duplicate candidates are removed from the population and are replaced by candidates randomly generated based on the available design options.

(b) **Evaluation:** Each yet unevaluated candidate is evaluated for each quality attribute of interest. In our case, performance, availability and/or costs metrics are predicted as described in Section 2. As a result, each candidate is annotated with the determined quality properties (i.e. mean response time, availability, and/or cost).

(c) **Selection:** After the reproduction phase, the population has grown. In the selection phase, the population is again reduced by just keeping the n most promising candidates based on the NSGA-II selection strategy. After i iterations, the search ends here and returns the Pareto-optimal candidates found so far.

More details on the evolutionary optimisation (such as the genetic encoding) can be found in [27]. Over several iterations, the combination of reproduction and selection lets the population converge towards the real front of globally Pareto-optimal solutions. The result of the optimisation is a set of Pareto-optimal candidates with respect to all candidates evaluated before. If the search also keeps a good diversity of candidates, the result set can be near to the global optima. However, in general, evolutionary optimisation cannot guarantee globally Pareto-optimal solutions [27].

4 Experimental Results

This section reports the results of the quality optimisations performed for the BRS system to demonstrate the applicability and usefulness of our approach and is organised as follows. Section 4.1 describes the degrees of freedom adopted. Sections 4.2 and 4.3 summarize the analytical optimisation and evolutionary optimisation. Finally, Section 4.4 presents and discusses the results of the hybrid approach.

Notice that we do not compare our prediction results from the models with actual measurements from the system implementation. For our validation, we assume that the underlying modelling and prediction methods are sound and deliver accurate prediction results as discussed in other papers [5, 8].

4.1 Search Problem Formulation

We defined two separate search problems: (1) optimise performance and cost, (2) optimise availability and cost. The following degrees of freedom are considered:

Component allocation: For the evolutionary optimisation of (1), all components can be freely allocated to up to five servers. For the analytic optimisation of (1) the problem can be defined considering only one allocation degree of freedom: The `Cache` component can be allocated to server S_3 or S_4. For (2), we do not consider component allocation as a degree of freedom. With free component allocation, the optimal solution would be to deploy all components on one server, so that only this server's availability affects the system. However, the inevitable performance degradation would not be taken into account.

Component selection: The `Webserver` can be realised using third party components. The software architect can choose among three functional equivalent implementations: `Webserver2` with cost 4 and `Webserver3` with cost 6. Both have less resource demand than the initial `Webserver`. `Webserver2` has better availability for the requests of type "view", while `Webserver3` has better availability for the requests of type "report".

Server configuration: Four different server configurations C_1 to C_4 with varying processor speed (PR in GHz), hardware availability (probability HA), and cost $HCost$ available.

The exact values considered in the case study can be found at [36]. A performance and availability optimisation has been omitted for space reasons, but works analogously. In principle, it is also possible to optimise all three quality criteria at once to determine a three-dimensional Pareto candidate set.

4.2 Analytic Optimisation

The degrees of freedom are mapped into an optimization problem including 24 binary variables: x_1-x_3, x_4-x_6, x_7-x_9, and x_{10}-x_{12} specify physical servers up/downgrades. x_{13} and x_{14} are introduced to model the two `WebServer` component alternative implementations. x_{15} is associated with the allocation of the `Cache` component to S_4. x_{16}-x_{18} model S_1 down/upgrades joint with the `WebServer2` implementation. Similarly x_{19}-x_{21} model S_1 down/upgrades joint with the `WebServer3` implementation. Finally x_{22}-x_{24} model the `Cache` component allocation on S_4 with the joint down/upgrades of servers S_3 and S_4. Four exclusive sets have to be introduced which are defined as follows:

- es_1: prevents conflicting design alternatives for server S_1 down/upgrades and the different implementation of the `WebServer` component (i.e., includes variables x_1-x_3, x_{13}-x_{14}, and x_{16}-x_{21}).
- es_2: avoids conflicting design alternatives associated with server S_2 down/upgrades (i.e., includes variables x_4-x_6).
- es_3: enumerates conflicting design alternatives for server S_3 down/upgrades and the different allocations of the `Cache` component (i.e., includes x_7-x_9, x_{15}, and x_{22}-x_{24}).
- es_4: prevents conflicting design alternatives associated with server S_4 down/upgrades and the different allocations of the `Cache` component (i.e., includes variables x_{10}-x_{12}, x_{15}, and x_{22}-x_{24}).

The analytical optimization step is performed by running *CPLEX* [20], a state of the art integer linear programming solver based on the branch and cut technique [32]. At each iteration of Algorithm 1, the solver identifies the global optimum solution of problems (P1) and (P2). The initial Pareto front can be determined very efficiently in 0.14 sec for the performance vs. cost analysis and 0.54 sec for the availability vs. cost analysis on a single core of an Intel Nehalem @2.6 GHz.

4.3 Evolutionary Optimisation

For performance prediction, we use the SimuCom simulation [5]. A stop criterion based on the confidence of the mean response time T was used in all but one evaluation. The simulation of candidate stopped when the 90% confidence interval $Conf$, determined with the batching algorithm of [9], was within +/-10% of the mean value: $Conf \subset [0.9T, 1.1T]$. Before calculating the confidence interval, the samples had to pass an independence test ("run test algorithm" [24]). For availability and cost prediction, we use the PCM Markov solver [8] and the PCM costs solver, respectively.

For the evolutionary optimisation, our prototype PerOpteryx tool follows the process described in Section 3.4. The number or candidates per iteration was set 25% higher than of optimal candidates found by the previous, analytical step to leave room for more optimal solutions. Table 1 shows the statistics of the optimisation runs (cand. = candidate(s), it. = iteration) which have been performed single threaded on on a single core of an Intel Core 2 T7200 CPU @ 2GHz. The stop criterion of the search was a manually chosen maximum number of iterations. The results had to be inspected to determine whether the search had converged up to then.

Search problem	input cand.	cand. per it.	cand. total	optimal cand.	iter-ations	dura-ation d	mean d per cand.
Performance and Cost	19	25	151	16	10	46 min	18.3 sec
Availability and Cost	12	15	130	10	15	5 min	2.3 sec

Table 1. Statistics of the Evolutionary Optimisation with Analytic Input

4.4 Results

The results of the performance and cost optimisation for the BRS system are presented to the software architect as shown in Fig. 5. Based on this set of Pareto-optimal candidates, software architects can make well-informed trade-off decisions and choose one candidate based on their quality requirements. One resulting solution with a good trade-off is shown in Fig. 6. It is superior to the initial candidate both in average response time ($T = 4.9sec$) and cost ($C = 22$). Thus, the hybrid optimisation could successfully improve the BRS system's architecture.

More detailed results for the optimisation of performance and cost are shown in Fig. 7. The series $\square$ marks the 19 candidates of the analytic optimisation as evaluated with the analytic approach. The series $\diamond$ marks the same 19 candidates as evaluated with SimuCom (thus, every candidate has the same cost, but updated mean response time). We observe that all analytic result values deviate from the simulation results by 40% on average (25% to 97% percent) but are always conservative. The deviation is larger for lower costs. Still, the form of the Pareto-curve is preserved and can serve as a valuable input for the evolutionary search. The series $\triangle$ marks the 9 new optimal candidates found by the evolutionary optimisation. They dominate 12 of the analytic candidates. The 143 further candidates evaluated by the evolutionary optimisation are not shown.

Hence, the hybrid approach is superior to the analytic optimisation alone, because the Pareto-front can be refined and additional Pareto solutions can be found. To assess the benefit of the hybrid approach to an evolutionary optimisation, we compared the results to a evolutionary optimisation with the same number of iterations from random candidates. The

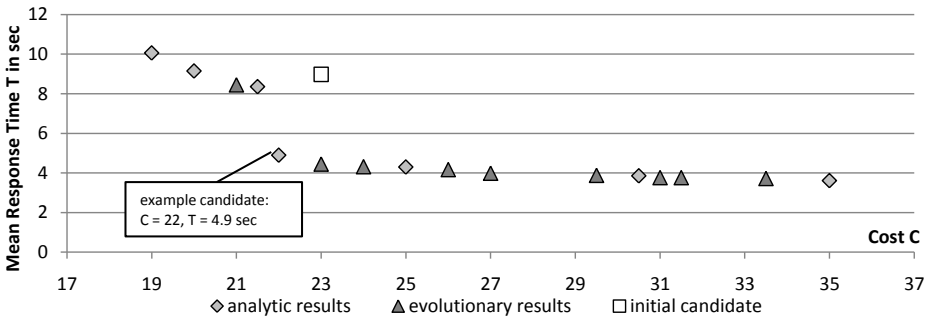

Fig. 5. Performance and cost optimisation: Results and comparison to initial candidate

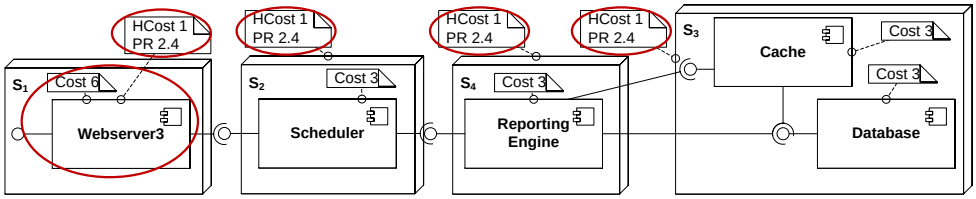

Fig. 6. Performance and cost optimisation: Example PCM Model for one Pareto-optimal candidate. Circles mark the changes compared to the initial candidate.

hybrid approach finds a superior Pareto-front (see Fig. 8), because 1) more optimal candidates are found and 2) all Pareto-optimal candidates found by the evolutionary optimisation from random candidates are dominated by the results of our hybrid approach. Thus, the evolutionary optimisation from random candidates would require more iterations to find a Pareto-front of similar quality, which is more time-consuming.

The results for the optimisation of availability and cost are shown in Fig. 9. The series $\diamond$ marks the 12 optimal solutions found by the analytical optimisation (some overlap each other), as evaluated with the PCM Markov solver. The series $\times$ marks the additional 118 candidates evaluated by the evolutionary optimisation. Three new optimal candidates have been found. All analytical optimal solutions stay undominated. Again, the hybrid results is superior to the analytic results alone. The results of a comparison with an evolutionary optimisation from random candidates can be found at [36]. The evolutionary optimisation from random candidates only finds an inferior Pareto-front in the same number of iterations.

5 Related Work

Our approach is based on software performance prediction [31, 3, 25], architecture-based software availability analysis [17], and search-based software engineering [19]. We categorize closely related approaches into: (i) Scenario based SA analysis, (ii) rule-based approaches, and (iii) metaheuristic-based approaches.

Scenario based SA analysis approaches: The definition of a SA model can embody not only the software qualities of the resulting system, but also the trade-offs decisions taken by designers [4, 11, 35]. The efforts to explore such trade-offs have produced the so-called

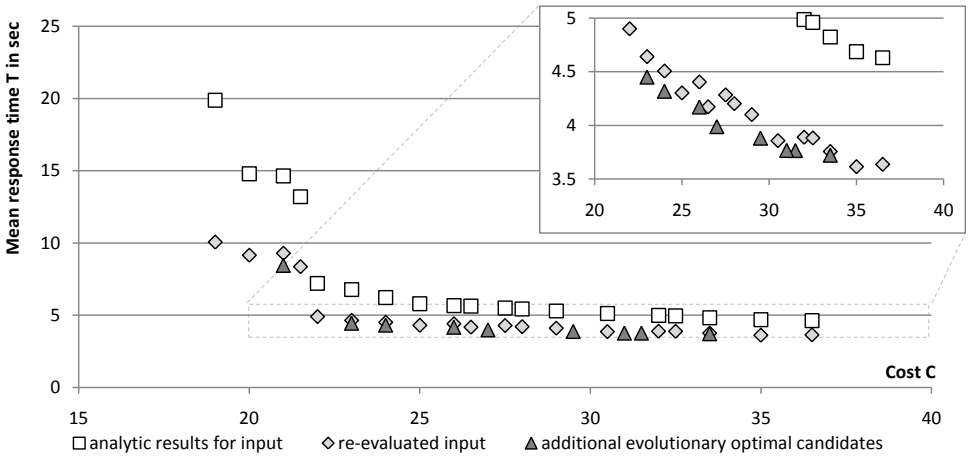

Fig. 7. Performance and cost optimisation: Analytic and simulation performance results.

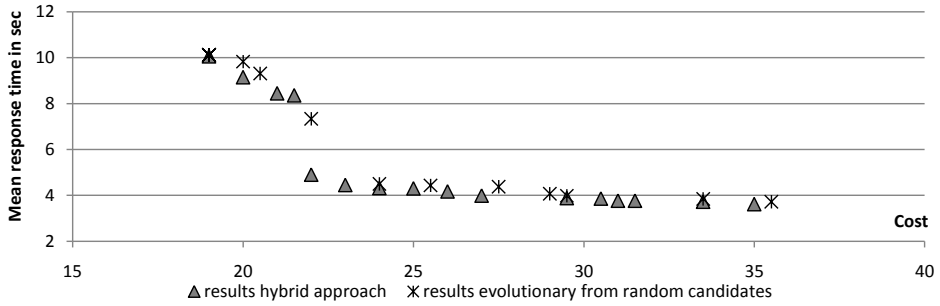

Fig. 8. Performance and cost optimisation: Hybrid vs. evolutionary search comparison.

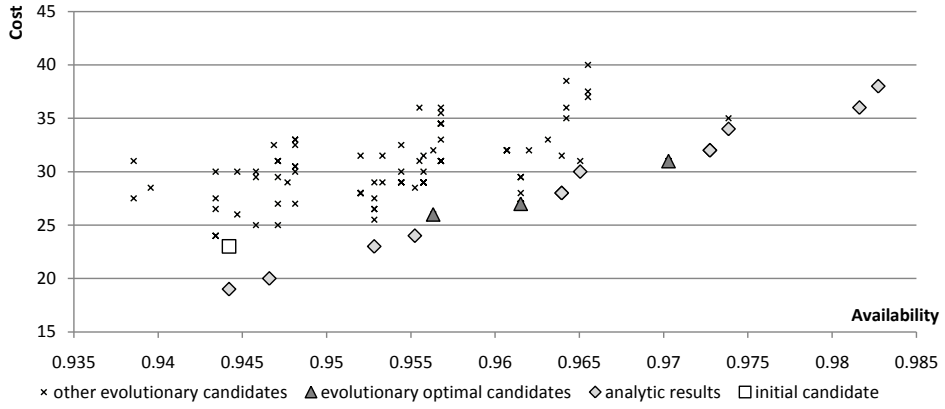

Fig. 9. Availability and cost optimisation: Results and comparison to initial candidate

scenario-based architecture analysis methods, such as SAAM and ATAM [22, 23] and others reviewed in [15]. These methods analyze the SA with respect to multiple quality attributes exploring also trade-offs concerning software qualities in the design. The outputs of such analysis include potential risks of the architecture and the verification result of the satisfaction of quality requirements. These methods provide qualitative results and are mainly based on the experience and the skill of designers and on the collaboration with different stakeholders. With respect to these works, our goal is to provide the software architect with a tool able to analyze the multiple objective problem in a quantitative way by allowing the automatic generation of several design architectures.

Rule-based approaches: Xu et al. [34] present a semi-automated approach to find configuration and design improvement on the model level. Based on a LQN model, performance problems (e.g., bottlenecks, long paths) are identified in a first step. Then, rules containing performance knowledge are applied to the detected problems.

McGregor et al. [28] have developed the ArchE framework. ArchE assists the software architect during the design to create architectures that meet quality requirements. It helps to create architectural models, collects requirements (in form of scenarios), collects the information needed to analyse the quality criteria for the requirements, provides the evaluation tools for modifiability or performance analysis, and suggests improvements.

Cortellessa et al. [13] propose an approach for automated feedback generation for software performance analysis, which aims at systematically evaluating performance prediction results using step-wise refinement and the detection of performance problem patterns. However, at this point, the approach is not automated.

Parsons et al. [30] present a framework for detecting performance anti-patterns in Java EE architectures. The method requires an implementation of a component-based system, which can be monitored for performance properties. Then, it searches for EJB-specific performance antipatterns in this model.

Kavimandan et al.[21] present an approach to optimise component allocation in the context of distributed real-time embedded component-based systems. They use heuristic rules to deploy components together that have a compatible configuration. In total, only allocation is considered as a degree of freedom, but the authors also mention that their approach could be combined with other approaches.

All rule-based approaches share a common limitation. The model can only be changed as defined by the improvement rules. However, especially performance is a complex and cross-cutting quality criterion. Thus, optimal solutions could lie on search paths not accessible by rules.

Metaheuristic-based approaches: Grunske [18] studies the improvement of two quality criteria, namely availability and costs, to allow well-informed trade-off decisions. Evolutionary computing is applied to search for good design solutions. However, only redundancy of components is studied as a degree of freedom to improve availability.

Menascé et al. [29] have developed the SASSY framework for generating service-oriented architectures based on quality requirements. Based on an initial model of the required service types and their communication, SASSY generates an optimal architecture by selecting the best services and potentially adding patterns such as replication or load balancing. As the allocation of components is irrelevant in SASSY's service architecture, the quality evaluations are simpler and allocation degrees of freedom cannot be considered. Thus, the approach is not suitable for component-based architectures in general.

6 Conclusions

In this paper, a hybrid approach for multi-attribute QoS optimisation of component based software systems has been proposed. The approach is promising. Both for performance vs. cost and availability vs. cost analyses, the hybrid approach is able to exploit the approximated analytical Pareto front providing a larger number of solutions with a more accurate estimate of performance and availability metrics. At the same time, the hybrid approach is less time-consuming than a evolutionary optimisation starting from random candidates. Hence, the integration of the analytical and evolutionary approaches is effective.

The proposed approach can lead both to a reduction of development costs and to an improvement of the quality of the final system, because an automated and efficient search is able to identify more and better design alternatives, and allows the software architect to make optimal trade-off decisions.

Future work will extend the analytical problem formulation in order to consider applications with parallel components execution and/or which can be modelled by means of closed queueing networks. Furthermore, the evolutionary search will be implemented as a parallel algorithm and an automated stop criterion will be developed. Ongoing work focuses on the integration of the analytic technique in the PCM software suite and on the QoS analyses of real industrial case studies.

Acknowledgments. This work reported in this paper has been partially supported by the EU FP7 Q-ImPrESS project.

References

1. D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. on Soft. Eng.*, 33(6):369–384, June 2007.
2. A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11 – 33, Jan.-March 2004.
3. S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. on Software Engineering*, 30(5):295–310, May 2004.
4. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley, Reading, MA, USA, 2003.
5. S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82:3–22, 2009.
6. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
7. B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
8. F. Brosch and B. Zimmerova. Design-Time Reliability Prediction for Software Systems. In *International Workshop on Software Quality and Maintainability*, pages 70–74, 2009.
9. E. J. Chen and W. D. Kelton. Batching methods for simulation output analysis: a stopping procedure based on phi-mixing conditions. In *Winter Simulation Conference*, pages 617–626, 2000.
10. P. C. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. SEI Series in Software Engineering. Addison-Wesley, 2001.
11. P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, Aug. 2001.

12. C. A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1:269–308, 1999.
13. V. Cortellessa, A. Di Marco, R. Eramo, A. Pierantonio, and C. Trubiani. Approaching the model-driven generation of feedback to remove software performance flaws. In *EUROMICRO Conf. on Softw. Engineering and Advanced Applications*, pages 162–169. IEEE Computer Society, 2009.
14. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature PPSN VI*, volume 1917/2000, pages 849–858. Springer, 2000.
15. L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *IEEE Trans. on Software Engineering*, 28(7):638–653, Jul 2002.
16. M. Ehrgott. *Multicriteria Optimization*. Springer, 2005.
17. S. S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Trans. on Dependable and Secure Computing*, 4(1):32–40, January-March 2007.
18. L. Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In *Intl. Conf. on Softw. Engineering*, pages 849–852. ACM, 2006.
19. M. Harman. The current state and future of search based software engineering. In L. C. Briand and A. L. Wolf, editors, *Workshop on the Future of Softw. Engin.*, pages 342–357. IEEE, 2007.
20. IBM ILOG. IBM ILOG CPLEX, 2010. http://www-01.ibm.com/software/integration/optimization/cplex/about/.
21. A. Kavimandan and A. S. Gokhale. Applying model transformations to optimizing real-time QoS configurations in DRE systems. In *Quality of Softw. Architectures*, pages 18–35. Springer, 2009.
22. R. Kazman, L. Bass, G. Abowd, and M. Webb. SAAM: A method for analyzing the properties of software architectures. In *Intl. Conf. on Softw. Engineering*, pages 81–90. IEEE, May 1994.
23. R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and S. Carrière. The architecture tradeoff analysis method. In *Intl. Conf. on Engineering of Complex Computer Systems*, pages 68–78. IEEE, 1998.
24. D. E. Knuth. *The Art of Computer Programming Vol. 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 1969.
25. H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, In Press, Corrected Proof:–, 2009.
26. M. Lukasiewycz. Opt4j - the optimization framework for java. http://www.opt4j.org, 2009.
27. A. Martens, H. Koziolek, S. Becker, and R. H. Reussner. Automatically improve software models for performance, reliability and cost using genetic algorithms. In *WOSP/SIPEW International Conference on Performance Engineering*. ACM, 2010.
28. 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.
29. D. A. Menascé, J. M. Ewing, H. Gomaa, S. Malex, and J. P. Sousa. A framework for utility-based service oriented design in SASSY. In *WOSP/SIPEW International Conference on Performance Engineering*, pages 27–36. ACM, 2010.
30. T. Parsons and J. Murphy. Detecting performance antipatterns in component based enterprise systems. *Journal of Object Technology*, 7(3):55–90, Mar. 2008.
31. C. U. Smith and L. G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2002.
32. L. Wolsey. *Integer Programming*. John Wiley and Sons, 1998.
33. X. Wu and M. Woodside. Performance Modeling from Software Components. *SIGSOFT Softw. Eng. Notes*, 29(1):290–301, 2004.
34. J. Xu. Rule-based automatic software performance diagnosis and improvement. In *International Workshop on Software and Performance*, pages 1–12. ACM, 2008.
35. J. Yang, G. Huang, W. Zhu, X. Cui, and H. Mei. Quality attribute tradeoff through adaptive architectures at runtime. *Journal of Systems and Software*, 82(2):319–332, 2009.
36. Details on case study for the hybrid optimisation approach. https://sdqweb.ipd.kit.edu/wiki/PerOpteryx/Hybrid_Optimisation_Case_Study, 2010.