Optimization of Complex QoS-aware Service Compositions

Dieter Schuller[1], Artem Polyvyanyy[2], Luciano García-Bañuelos[3], and Stefan Schulte[1]

[1] Technische Universität Darmstadt, Multimedia Communications Lab
{dieter.schuller,stefan.schulte}@kom.tu-darmstadt.de
[2] Hasso Plattner Institute at the University of Potsdam, Germany
Artem.Polyvyanyy@hpi.uni-potsdam.de
[3] Institute of Computer Science, University of Tartu, Estonia
luciano.garcia@ut.ee

Abstract. In Service-oriented Architectures, business processes can be realized by composing loosely coupled services. The problem of QoS-aware service composition is widely recognized in the literature. Existing approaches on computing an optimal solution to this problem tackle structured business processes, i.e., business processes which are composed of XOR-block, AND-block, and repeat loop orchestration components. As of yet, OR-block and unstructured orchestration components have not been sufficiently considered in the context of QoS-aware service composition. The work at hand addresses this shortcoming. An approach for computing an optimal solution to the service composition problem is proposed considering the structured orchestration components, such as AND/XOR/OR-block and repeat loop, as well as unstructured orchestration components.

Keywords: Service composition, Quality of Service, Optimization, Structured and unstructured orchestration components

1 Introduction

To support and enable agile business processes, the Service-oriented Architectures (SOA) paradigm is often recommended [1]. One of the key features of SOA is that (IT supported) business processes and, respectively, workflows are realized by composing loosely coupled services – a practice known as service composition. These services autonomously provide a more or less coarse-/fine-grained functionality [2]. Following the vision of the Internet of Services, multiple service providers offer various services at different service marketplaces. If multiple services, which are equally appropriate to accomplish certain task, are available at service marketplaces, enterprises can choose to compose those services which meet cost and Quality of Service (QoS) constraints best. This service composition problem (SCP), which forms an optimization problem, is widely recognized in the literature and has been discussed by several authors, [3,4,5,6,7]. An optimal solution to the SCP constitutes an execution plan, i.e., a set of selected services, which achieves an efficient business process execution with respect to specified cost and QoS requirements.

As real world business processes do not solely consist of structured orchestration components, it is necessary to account for complex structures when composing services. However, existing approaches aiming at computing an optimal solution to the SCP consider complex structures only insufficiently. They usually examine all execution paths resulting from conditional branchings or repeat loops, which leads to significant drawbacks, cf. Section 2. Our approach does not need every possible path of a business process to be examined separately and, thus, does not need all execution paths to be known. We are able to account for OR-blocks. Moreover, our approach goes beyond structured orchestration components by considering unstructured components, viz. unstructured Directed Acyclic Graphs (DAG[4]). To the best of our knowledge, DAGs have not been addressed previously in the context of QoS-aware service composition.

Thus, the work at hand significantly extends the related work in the field of computing optimal solutions to the SCP. We initially formulate the SCP as a non-linear optimization problem and transform it into a linear one. The linear optimization problem is then optimally solved by applying Integer Linear Programming (ILP) techniques from the field of operations research [8,9].

The remainder of the paper is structured as follows: In Section 2, we distinguish our approach from related work. The orchestration models and components considered in the work at hand are introduced in Section 3. We discuss QoS aggregation functions in Section 5 after having presented the applied system model in Section 4. Based on the aggregation functions, the SCP is formulated as an optimization problem in Section 6 and our solution to this problem is evaluated in Section 7. Finally, Section 8 draws conclusions and outlines next steps in our research.

2 Related Work

As already mentioned, the SCP is widely recognized in the literature. A survey of current approaches to the SCP can be found in [7]. The related work in this area can be broadly divided into two groups: heuristic suboptimal SCP methods, e.g., [10,11,12,13,5,14], and optimal SCP methods, e.g., [15,4,16,6].

In [10], the authors present and evaluate (basic) heuristic algorithms, such as Greedy and Pattern-wise Selection. A hill-climbing approach is proposed in [5]. In the same vein, [11,12,13] tackle the optimization problem with Genetic Algorithms. In all the above cases, the input orchestration is assumed to be structured.

As for optimal SCP methods, the common approach is to analyze every possible execution path. Zeng et al. [15] compute an optimal solution for every execution path. They require a merging step afterwards to account for situations where different services have been selected for the same task in different execution plans. Thus, it is not guaranteed that the merged execution plans still provide the optimal solution as the authors do not consider probabilities of XOR-blocks for the optimization. Anselmi et al. [4] consider all possible execution paths in a single optimization problem; however, they consider probabilities for the different

[4] In the following, we omit explicitly stating that DAG components are unstructured

execution paths only in the objective function. Thus, they fail to integrate probabilities into the process restrictions, which leads to a worst-case analysis. In other words, the result of ignoring probabilities regarding different branchings for the process restrictions is that, effectively, only the worst path of an XOR-block is considered although all possible execution paths are required to be integrated into the optimization. Our approach allows probabilities for the specification of process restrictions to be considered. This enables an average-case analysis in addition to the worst-case analysis. In [6], Huang et al. consider only one execution path for the optimization – the worst one. Hence, they also fail to consider probabilities of conditional branchings. Ardagna et al. [16] do not consider conditional branchings at all, but they account for repeat loops, cf. Section 3.2. They unfold the cyclic structure and consider all the resulting execution paths. Similar to Anselmi et al., Ardagna et al. integrate all execution paths into a single optimization problem. Thus, the number of constraints grows with the number of considered repeated executions. Furthermore, such an approach does not allow for limiting behavior considerations for a repeat loop.

The work at hand addresses computing an optimal solution to the SCP. In contrast to the related work mentioned above, we do not need to identify all possible execution paths of a given orchestration model. Our approach allows for the consideration of probabilities when specifying constraints for conditional branchings. Thus, we account for all execution possibilities directly. Furthermore, we can perform limiting behavior considerations regarding repeat loops. In addition, our approach accounts for OR-blocks and DAGs. In the next section, we provide a detailed description of all the considered orchestration components.

3 Orchestration Models & Components

3.1 Orchestration Models

An (*orchestration*) *model* is a directed graph consisting of edges (n_1, p, n_2), such that n_1 and n_2 are nodes (the source and target of the edge) and p is the edge probability, i.e., probability of taking the edge assuming that the execution of the orchestration model has reached node n_1. Nodes in an orchestration model are of two types: *tasks* and *gateways*. Tasks represent units of work that are accomplished by atomic services. Gateways encode the routing logic of the orchestration model. Gateways are of three types, cf. [17]: XOR gateways represent conditional branching (XOR-split) or merging of exclusive branches (XOR-join). AND gateways represent parallel forking (AND-split) or synchronization points (AND-join). OR gateways represent multiple choice (OR-split) or general synchronizing merge

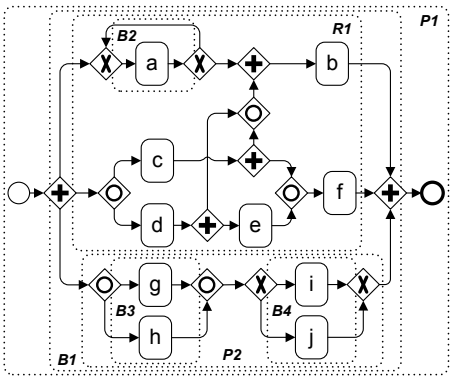


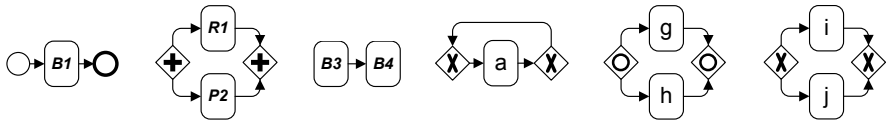
Fig. 1. An orchestration model

(OR-join). A split gateway is the one with a single incoming edge and multiple outgoing edges, while a join gateway is the one with multiple incoming edges and a single outgoing edge.

An orchestration model must be *well-formed*. A well-formed orchestration model meets the following requirements: (i) An orchestration model has a single *source* node, i.e., a node with no incoming edges, and a single *sink* node, i.e., a node with no outgoing edges. (ii) Every node is on a path from the source to the sink. (iii) Every task node has at most one incoming and at most one outgoing edge. (iv) Every gateway is either a split or a join. (v) The sum of the probabilities attached to the outgoing edges of an XOR-split gateway is 1. (vi) The sum of the probabilities attached to the outgoing edges of an OR-split gateway is larger or equal to 1. (vii) An edge whose source is neither an XOR- nor an OR-split gateway has a probability of 1. Fig. 1 shows a well-formed orchestration model using BPMN notation (without edge probabilities).

3.2 Orchestration Components

An orchestration model can be parsed into a hierarchy of (*orchestration*) *components*, each with a single entry and single exit node. Such orchestration components constitute logically independent units of work in the orchestration model. The result of the parsing procedure is a *parse tree*, which is the containment hierarchy of orchestration components of the orchestration model.



(a) P1 (b) B1 (c) P2 (d) B2 (e) B3 (f) B4

Fig. 2. Orchestration components

The *Refined Process Structure Tree* (RPST) is a technique for workflow graph parsing [18,19], i.e., for discovering the structure of a workflow graph. The RPST of an orchestration model is the set of all its canonical orchestration components. An orchestration component is canonical, if it does not overlap (on edges) with any other orchestration component of the orchestration model.

The set of all canonical orchestration components of a model clearly forms a hierarchy that can be represented as a tree. The parent of an orchestration component is the smallest component that contains it. The root of the tree captures the entire orchestration model. A leaf of the tree is an edge of the model. Orchestration components can be classified based on their structure, cf. [19] for details. In the following, we assume that orchestration models are composed of the following structural classes of orchestration components:

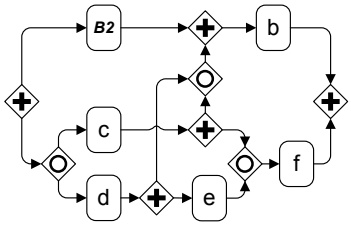


Fig. 3. DAG component R1

sequence, AND-block, XOR-block, OR-block, repeat loop, and DAG. Fig. 2 shows all structured orchestration components of the model in Fig. 1. Note that in the following, when referring to a component, we abstract from the internal logic of its child components. In the figure, $P1$ and $P2$ are sequences, $B1$ is AND-block, $B4$ is XOR-block, $B3$ is OR-block, and $B2$ is repeat loop component. Finally, Fig. 3 shows the only DAG component of the orchestration model.

4 System Model

This section describes the system model. We label the set of all tasks with I, $i \in I = \{1, ..., i^{\#}\}$. Referring to Fig. 1, the task numbers i correspond to the identifiers of the tasks, i.e., to "a", "b", etc. The order of the task numbers is not important as long as the respective sets are defined properly. Each task is required to be accomplished by exactly one service $j \in J_i = \{1, ..., j_i^{\#}\}$. Whether or not service j is selected for task i is indicated by the decision-variables $x_{ij} \in \{0, 1\}$. In this work, we take execution time e (duration to execute a service in seconds), reliability r (probability of successful service execution), and throughput d (number of service requests the service is able to serve within a certain time interval) as QoS parameters, as well as cost c (charge for a service invocation in cent considering a pay-per-use pricing model) into account. With these parameters – in fact, even with a subset of these parameters – the aggregation types summation, multiplication, and min/max-operator are covered. Thus, the integration of further QoS parameters into the optimization problem is straightforward. We label bounds for the QoS parameters with b_e, b_r, b_d, b_c.

Regarding branchings, the set L of paths is specified as $L = \{1, ..., l^{\#}\}$. Thereby, $l \in L$ indicate the path numbers within a branching. To give an example, we refer to Fig. 2(b). There are two paths l within the AND-block, thus $L = \{1, 2\}$. In order to distinguish multiple sets of paths from each other, we utilize additional indices, i.e., L_a, L_x, L_o, L_g for AND/XOR/OR-blocks and DAGs, whenever necessary and refer to them as branching L_a and L_x, respectively. The set $IW_L \subseteq I$ represents the set of tasks within a branching and $IW_l \subseteq IW_L$ the set of tasks within path $l \in L$. The remaining tasks, which are not located within a branching, are covered in the set $IS = I \setminus (IW_l \mid l \in L)$. The probability of executing a certain path l is indicated by p_l. When it comes to repeat loops, we label the probability that a task i is repeated with ρ_i. To give an example for such a repeat situation, assume a task that tries, e.g., to start an engine or to initialize a resource. The probability that the engine is started, is indicated by $1 - \rho_i$. In case we require the engine to be started or the resource to be initialized, we have to repeat the starting process until we achieve that aim.

The above described system model is used to develop the aggregation functions which are proposed in the next section.

5 Aggregation Functions

In this section, we describe QoS aggregation functions. These functions are required to specify the objective function and the constraints of the SCP. In

order to aggregate the QoS values of the considered candidate services for the whole orchestration model, the regarded orchestration components as well as the respective aggregation type of each QoS parameter have to be taken into account. Table 1 indicates aggregation functions for sequence, AND-block, and XOR-block. In a sequence, the QoS of all services has to be aggregated according to the respective aggregation type. Regarding an AND-block, we have to take the path with the highest aggregated execution time – the *critical path* – into account for execution time. For the other QoS parameters, all services within the AND-block are aggregated. For the XOR-block, we perform an *average-case analysis* by considering possible paths l according to their probabilities p_l in contrast to a worst-case analysis, where the worst of the alternative paths is considered for service selection. Further details on these aggregation functions are available in [20]. For the sake of clarity, we define e_s, e_a, e_x, etc. in Table 1 to represent the respective aggregation function in Section 6.

Table 1. Aggregation Functions

QoS	Sequence	AND-block	XOR-block
e	$e_s := \sum_{i \in IS} \sum_{j \in J_i} e_{ij} x_{ij}$	$e_a := \max_{l \in L}(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij})$	$e_x := \sum_{l \in L} p_l \sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij}$
c	$c_s := \sum_{i \in IS} \sum_{j \in J_i} c_{ij} x_{ij}$	$c_a := \sum_{l \in L} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$	$c_x := \sum_{l \in L} p_l \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$
r	$r_s := \prod_{i \in IS} \sum_{j \in J_i} r_{ij} x_{ij}$	$r_a := \prod_{l \in L} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$	$r_x := \sum_{l \in L} p_l \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$
d	$d_s := \min_{i \in IS}(\sum_{j \in J_i} d_{ij} x_{ij})$	$d_a := \min_{l \in L}(\min_{i \in IW_l} (\sum_{j \in J_i} d_{ij} x_{ij}))$	$d_x := \sum_{l \in L} p_l \min_{i \in IW_l} (\sum_{j \in J_i} d_{ij} x_{ij})$

Regarding repeat loops, we propose to perform limiting behavior considerations taking the mentioned probability ρ_i into account. Thus, we exchange e_{ij} for $e^*_{ij} = \frac{1}{1-\rho_i} e_{ij}$, c_{ij} for $c^*_{ij} = \frac{1}{1-\rho_i} c_{ij}$, and r_{ij} for $r^*_{ij} = \frac{(1-\rho_i) r_{ij}}{1-\rho_i r_{ij}}$, cf. [20] for further explanations. Throughput d_{ij} is not affected by a repeat loop.

The application of these aggregation functions implies a sequential arrangement of the process steps within a split and join [20]. To overcome this shortcoming and in order to account for recursive interlacings of the regarded patterns, we propose to abstract from the interlacing by adding a new service, which replaces the interlacing [21]. This new service as well as the computation of its QoS parameters is then considered for the optimization.

In the following, we propose aggregation functions to account for an OR-block and for a DAG component.

5.1 OR-block

Regarding the pattern multi-choice (OR-split), not only one (as in XOR-block) and not necessarily all (as in AND-block), but any subset of paths can be executed. In an OR-block, the execution of one, two, three, etc. or even all paths is allowed. In fact, we also allow that none of the alternative paths are executed. All the selected paths are executed in parallel. When combined with an OR-join (as assumed here), a synchronizing merge is carried out. Note that it is not known

before execution, which of the alternative paths will be executed – leading to different aggregation functions when considering the average- or worst-case. In the average-case, only started paths are considered for QoS aggregation, whereas in the worst-case, all paths are executed in parallel. In the latter case, the aggregation functions for an AND-block, cf. Table 1, can be applied.

Regarding the average-case, we have to consider the started paths. Therefore, we are required to take all possible combinations of paths l into account. Let $L = \{1, ..., l^{\#}\}$ be the set of all paths l. If only one path is started, we have $\binom{l^{\#}}{1}$ possibilities to select one of them; if two paths are to be started, there are $\binom{l^{\#}}{2}$ alternative paths. For three paths, it would be $\binom{l^{\#}}{3}$ and so on and so forth. Altogether, we have $h^{\#} := \sum_{l \in L} \binom{l^{\#}}{l}$ possible combinations. We define $H = \{1, ..., h^{\#}\}$, $h \in H$, as the set that contains an index number h for the possible path combinations. In addition, $L^h = \{l \mid l \in L \wedge h \in H\}$ specifies the set containing the selected paths for path combination h.

To make this clear, we refer to Fig. 2(e). As there are two alternative paths after the OR-split (executing g or h), $l^{\#} = 2$ and $L = \{1, 2\}$. There are $\binom{2}{1} = 2$ possibilities to select exactly one path and $\binom{2}{2} = 1$ possibility to select exactly two paths. Adding these possibilities leads to $h^{\#} := \sum_{l \in L} \binom{2}{l} = \binom{2}{1} + \binom{2}{2} = 3$ possible path combinations. H would be $H = \{1, 2, 3\}$, and the sets L^h are the following: $L^1 = \{1\}$, $L^2 = \{2\}$, $L^3 = \{1, 2\}$.

We label the probability that a certain combination h of paths is executed with p_h. Thereby, p_0 represents the probability that none of the paths l is executed. We assume $p_0 + \sum_{h \in H} p_h = 1$. For the sake of simplicity and in order to make sure that the execution does not reach a deadlock situation, we set $p_0 = 0$. As the selected paths are executed in parallel, the aggregation functions are based on the functions for an AND-block. We extend these functions by integrating the selection of the respective paths l. The resulting aggregation functions for an OR-block component are depicted in (1) to (4).

$$e_o := \sum_{h \in H} p_h \max_{l \in L^h} \Big(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \Big) \tag{1}$$

$$c_o := \sum_{h \in H} p_h \sum_{l \in L^h} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij} \tag{2}$$

$$r_o := \sum_{h \in H} p_h \prod_{l \in L^h} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij} \tag{3}$$

$$d_o := \sum_{h \in H} p_h \min_{l \in L^h} \Big(\min_{i \in IW_l} \big(\sum_{j \in J_i} d_{ij} x_{ij} \big) \Big) \tag{4}$$

5.2 Directed Acyclic Graph

In order to account for DAGs, we firstly identify all possible *runs* of a DAG component. A run is a (potentially concurrent) execution path in the DAG along with the probability of the occurrence of this run. Thus, the original DAG can be rewritten in the form of a XOR-block that combines all possible runs – with their

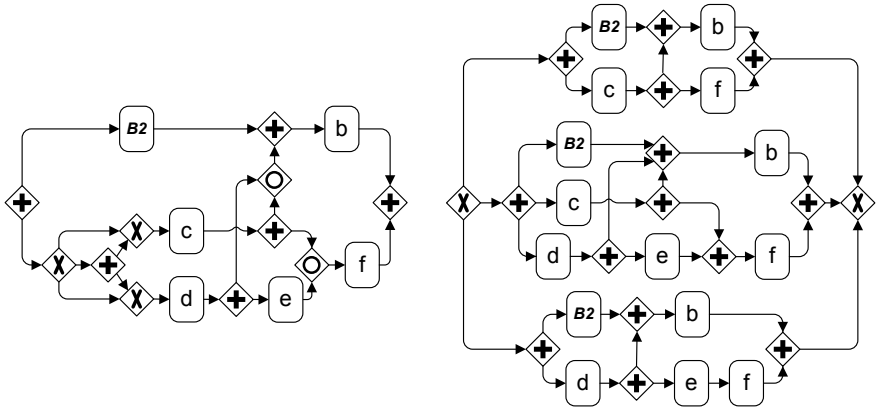


Fig. 4. (a) $R1$ with rewritten OR-splits, and (b) runs of $R1$ combined in a XOR-block

respective aggregated probabilities. An algorithm to compute runs is given in [22] but only considers DAG components composed of XOR and AND gateways and, therefore, has to be extended to account for OR gateways. To this end, every OR-split is transformed into an XOR-split followed by AND-splits according to the path combinations H as described in Section 5.1. To illustrate this, consider the DAG component shown in Fig. 4(a), which is the transformed version of the DAG presented in Fig. 3. Having removed OR-splits, we can now compute runs with a slightly modified version of the algorithm in [22], so as to replace OR-joins by AND-joins where required. Fig. 4(b) presents the XOR-block with the runs computed for $R1$. At this stage, we can apply our aggregation function for XOR-blocks from Table 1.

As the tasks in each of the identified runs and the execution paths of the XOR-block respectively, are not arranged sequentially, we again apply our recursive pattern interlacing technique to abstract from the complex N-structure [23] within the runs. This way, we create one new service j_{run} for each path l of the outer XOR-block. To compute the QoS values for each of these new services, we apply the aggregation functions for an AND-block from Table 1 and specify e_{run}, c_{run}, r_{run}, d_{run} in (5)–(8). Please note that we ignore the complexity of the respective N-structures for the QoS parameters c, r, d by applying the respective functions for AND-blocks, as each of the services is executed only once. This is indicated by utilizing L_a instead of L for the set of paths within the N-structure. Regarding the execution time e, the path with the highest aggregated execution time has to be taken into account, as the paths are executed in parallel, cf. [20] for additional explanations.

$$e_{run} := \max_{l \in L} \Big(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \Big) \tag{5}$$

$$c_{run} := \sum_{l \in L_a} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij} \tag{6}$$

$$r_{run} := \prod_{l \in L_a} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij} \tag{7}$$

$$d_{run} := \min_{l \in L_a} (\min_{i \in IW_l} (\sum_{j \in J_i} d_{ij} x_{ij})) \tag{8}$$

Having performed this abstraction step, we utilize the aggregation function for XOR-blocks from Table 1 to aggregate the QoS for each path within the XOR-block according to its respective probability. This is done in (9)–(12) resulting in the aggregation functions for the DAG considered here (indexed with g).

$$e_g := \sum_{l \in L} p_l e_{run_l} \tag{9}$$

$$c_g := \sum_{l \in L} p_l c_{run_l} \tag{10}$$

$$r_g := \sum_{l \in L} p_l r_{run_l} \tag{11}$$

$$d_g := \sum_{l \in L} p_l d_{run_l} \tag{12}$$

6 Optimization Problem

As mentioned in the introduction, the SCP describes the problem of selecting and composing those services (from sets of services equally appropriate to accomplish certain tasks) which meet cost and QoS constraints best. In this section, we describe the steps to model the SCP as a linear optimization problem. We, therefore, initially formulate a non-linear optimization problem in Section 6.1 based on the aggregation functions presented in Section 5. In order to obtain the linear optimization problem, in Section 6.2, we conduct adaptations of Model 1 from Section 6.1. Finally, in Section 6.3, we additionally describe a heuristic solution method based on our approach.

6.1 Non-Linear Optimization Problem

In order to formulate the non-linear optimization problem in Model 1, we specify an objective function in (13), which is aimed at minimizing the overall cost of the selected services, as well as a set of restrictions for the aggregated QoS values in (14)–(19). We perform an average-case-analysis by applying the aggregation functions described in Section 5. For readability reasons, we utilize variables e_a, c_a, r_a, etc. in Model 1 to represent these aggregation functions. Regarding repeat loops, we exchange the QoS parameters e, c, r for the adapted expression e^*, c^*, r^*, as described in Section 5; i.e., if there is a loop at task i, then the respective QoS values of the candidate services j_i appropriate to realize task i are adjusted. Otherwise, the respective QoS values are not modified.

Model 1 depicts the optimization problem in a general form to account for sequences, repeat loops, AND/XOR/OR-blocks, and DAGs that are not interlaced. In (14)–(17), the restrictions for the regarded QoS parameters are

Model 1 Generic Service Selection Problem

Objective Function

$$\text{minimize } F(x) = c_s + c_a + c_x + c_o + c_g \tag{13}$$

so that

$$e_s + e_a + e_x + e_o + e_g \leq b_e \tag{14}$$

$$c_s + c_a + c_x + c_o + c_g \leq b_c \tag{15}$$

$$r_s \cdot r_a \cdot r_x \cdot r_o \cdot r_g \geq b_r \tag{16}$$

$$\min(d_s, d_a, d_x, d_o, d_g) \geq b_c \tag{17}$$

$$\sum_{j \in J_i} x_{ij} = 1 \qquad \forall i \in I \tag{18}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, \forall j \in J_i \tag{19}$$

specified by aggregating the considered aggregation functions and restricting them to be lower/greater or equal to the respective bounds b_e, b_c, b_r, b_d for the QoS parameters. For reasons of clarity, we omit defining e_s, e_a, etc. in Model 1, but it has to be noted that their respective equations (from Table 1) are also part of the optimization problem and, therefore, of Model 1. Restriction (18) ensures that every task is accomplished by exactly one service and restriction (19) indicates the integrality restriction.

In order to account for the orchestration model shown in Fig. 1, we utilize the identified orchestration components, cf. Fig. 2 and Fig. 3, and formulate the respective optimization problem in Model 2. Regarding Fig. 2(a) and Fig. 2(b), we apply the aggregation functions for an AND-block. As the orchestration components $R1$ and $P2$ do not belong to the structural class "sequence" (as required for the application of the AND-block formulae), we abstract from their actual structure by creating new services j_{R1} and j_{P2} and use these services for the optimization, i.e., we apply the mentioned AND-block aggregation function for j_{R1}, j_{P2} in (21)–(24).

In order to calculate the QoS for j_{P2}, we apply the aggregation functions for an OR-block ($B3$) and XOR-block ($B4$) in (27)–(30) with respect to Fig. 2(c).

$$e_o + e_x = e_{P2} \tag{27}$$

$$c_o + c_x = c_{P2} \tag{28}$$

$$r_o \cdot r_x = r_{P2} \tag{29}$$

$$\min(d_o, d_x) = d_{P2} \tag{30}$$

In order to account for DAG $R1$ in Fig. 3, we compute the corresponding choice component as described in Section 5.2 and apply the respective aggregation functions in (5)–(12) to calculate the QoS for j_{R1}. We thereby utilize e^*_{ij}, c^*_{ij}, and r^*_{ij} to account for the repeat loop at $B2$. In analogy to Model 1, the equations

Model 2 Optimization Problem for Orchestration Model in Fig. 1

Objective Function

$$\text{minimize } F(x) = c_{R1} + c_{P2} \tag{20}$$

$$\max(e_{R1}, e_{P2}) \leq b_e \tag{21}$$

$$c_{R1} + c_{P2} \leq b_c \tag{22}$$

$$r_{R1} \cdot r_{P2} \geq b_r \tag{23}$$

$$\min(d_{R1}, d_{P2}) \geq b_d \tag{24}$$

$$\sum_{j \in J_i} x_{ij} = 1 \qquad \forall i \in I \tag{25}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, \forall j \in J_i \tag{26}$$

for the aggregation functions e_o, e_x, etc. as well as (27)–(30) and (5)–(12) are part of the optimization problem, but are omitted in Model 2 for clarity reasons.

6.2 Linearization of the Non-Linear Optimization Problem

In (16), (17), (23), (24), the decision-variables x_{ij} are multiplied and aggregated, respectively, using the min/max-operator, i.e., the decision-variables are aggregated in a non-linear way. As we aim to solve the SCP by applying ILP, we have to adapt these non-linear aggregations.

Regarding the max-operator, e.g., in (21), or in the aggregation function for AND-blocks in Table 1, it has to be noted that if the maximum of a set has to be lower or equal to an upper bound, each element of this set has to fulfill this constraint. Thus, we exchange the term with the max-operator for a new variable, e.g., e_a^{max}, and restrict each element in the max-operator to be lower or equal to e_a^{max}. To make this clear, we exemplify the linearization of e_a, cf. Table 1, in (14) for Model 1. Here, we exchange e_a for e_a^{max} and add restriction (31) to Model 1. To replace the min-operator, we analogously specify variables d^{min} and add appropriate restrictions for each min-operator in (17) to Model 1, cf. [21].

$$\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \leq e_a^{max} \qquad \forall l \in L \tag{31}$$

To linearize restrictions (16), (23), where the decision-variables are multiplied with each other, we utilize the approximation in (32) for the aggregation of QoS parameters r, which is very accurate for parameter values z_{ij} that are very close to 1, such as reliability. We thereby avoid having to multiply the reliabilities of the alternative services and allow for summing up the respective approximated reliabilities instead. This way, we avoid the multiplication of decision-variables in all aggregation functions with respect to the QoS parameter r leading to linear aggregation functions and restrictions. For further details, we refer to [20].

$$\prod_{i \in I} \sum_{j \in J_i} z_{ij} x_{ij} \approx 1 - \sum_{i \in I} (1 - \sum_{j \in J_i} z_{ij} x_{ij}) \tag{32}$$

$$\prod_{i \in I} \sum_{j \in J_i} z_{ij} x_{ij} = 1 - \sum_{i \in I} (1 - \sum_{j \in J_i} z_{ij} x_{ij}) + \epsilon \tag{33}$$

Applying the described linearization steps results in a linear optimization problem in Models 1 and 2. An optimal solution can be computed by applying ILP, if a solution exists. But, as we have utilized the approximation in (32) instead multiplying the reliabilities of the alternative services, we inserted an error into the optimization problem. The larger the set of tasks I, the higher is this error. Thus, in order to ensure that our approach actually computes the optimal solution, we compute the value of this error, labeled with ϵ, for the current solution and account for this error by considering its value explicitly in (33). Afterwards, we recompute the optimal solution taking (33) into account. If the resulting execution plan does not change, we obviously found the optimal solution. Otherwise, we recalculate the error and recompute the optimal solution taking this new error into account. This way, we run the optimization at least two times, but in the end, we guarantee that the solution is the optimal one.

6.3 Scalability

Applying our approach, computing the optimal solution to the SCP requires increased computational effort with a growing number of tasks and candidate services per task. To address scalability issues, we propose to relax the integrality restrictions (19) and (26), and to compute a solution by applying mixed integer linear programming (MILP) without considering the error ϵ. This probably results in an invalid solution to the SCP with no explicit indication which service to select for a certain task, as the decision-variables x_{ij} may contain values between 0 and 1 and not exactly 0 or 1. Afterwards, in order to obtain a valid but probably not optimal solution, i.e., $x_{ij} \in \{0, 1\}$, we apply a heuristic selection strategy. Based on the values of the decision variables x_{ij}, we randomly select services which satisfy the constraints. The performance of this heuristic solution method compared to the optimal solution is depicted in Fig. 5 and Fig. 6.

Alternatively, we could interpret the x_{ij} values as probabilities to select respective services for the accomplishment of a certain task. This way, if a business process is executed not only once but multiple times (as is assumed to be the normal case), the business process execution can be seen as realization of a random experiment – selecting respective services based on their probabilities – with minimal average cost satisfying the constraints *in average*.

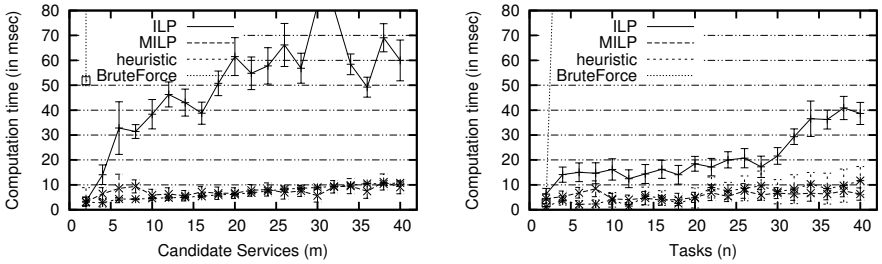
7 Evaluation

As a proof of concept, we implemented our approach to the SCP using the linear programming solver CPLEX[5]. In order to evaluate the efficiency and the solution quality of our approach, i.e., the time for computing the execution plan and its cost, we conducted a series of experiments to compare our approach, which

[5] http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

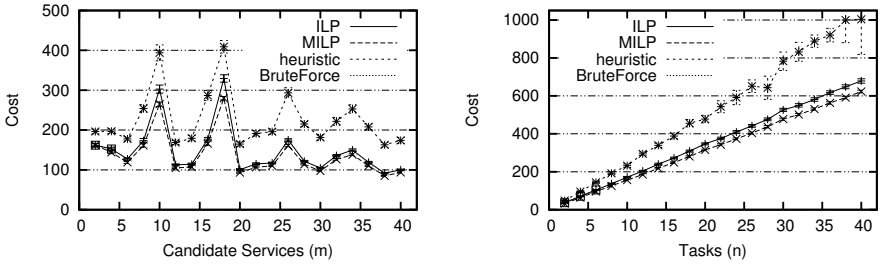
we label with ILP, to the heuristic solution method mentioned in the previous section and to a BruteForce algorithm, which iterates through all possible service combinations. Thus, BruteForce computes the optimal solution *per definitionem*. Our proposition to interpret the values of the decision variables as execution probabilities corresponds to the label MILP in Fig. 5 and Fig. 6. The experiments were performed on an Intel Core 2 Quad processor at 2.66 GHz, 4 GB RAM, running Microsoft Windows 7.

In order to evaluate the influence of the number m of candidate services j_i per task i, we varied m in Fig. 5(a) and Fig. 6(a) from 2 to 40 with step 2 for the orchestration model in Fig. 1, which is composed of the orchestration components sequence, AND-block, XOR-block, OR-block, repeat loop, and DAG. Regarding the influence of the number n of tasks, we varied n from 2 to 40 with step 2 considering 10 candidate services per task. The resulting orchestration models thereby only comprise of sequences.



(a) Impact of candidate services (b) Impact of tasks

Fig. 5. Evaluation of computation time



(a) Impact of candidate services (b) Impact of tasks

Fig. 6. Evaluation of cost

Regarding the efficiency of our ILP approach, we observe that the computation time increases with the number of tasks and candidate services. However, the computation time remains lower than 100 msec. For the heuristic approach as well as for MILP, the computation times increase only slightly. As indicated in Fig. 5(a) and Fig. 5(b), the computation time using the BruteForce algorithm is greatly increasing. Regarding Fig. 5(a), BruteForce requires $63,883.92$ msec for $m = 4$, which is not displayed in Fig. 5(a). For $n = 4$ and $n = 6$, which would be

the next plots for BruteForce in Fig. 5(b), the algorithm requires 209.62 msec and 29,261.48 msec, respectively.

With respect to the solution quality, the ILP approach computes an optimal solution, which is indicated in Fig. 6(a) and Fig. 6(b) by comparing the cost of the ILP solution to the cost of BruteForce. As the MILP algorithm does not create integer values for the decision variables, it must not be compared to ILP regarding cost.

The evaluation results show that ILP requires more time than the heuristic method for computing a solution, but the heuristic does not achieve the solution quality of ILP, i.e., the cost for execution plans computed by the heuristic are always higher than cost for execution plans computed by ILP. Compared to the BruteForce algorithm, which also computes the optimal solution, ILP's computation time is rather small.

8 Conclusion

The problem of selecting services based on their QoS – the QoS-aware SCP – is widely recognized in the literature and has been discussed recently by several authors. In the work at hand, we addressed the SCP for orchestration models composed of sequences, AND-blocks, XOR-blocks, OR-block, repeat loops, and DAGs, which has, to date, been insufficiently considered in the literature, cf. Section 2. We thereby aim to compute an optimal solution to the SCP. In our future work, we will focus on considering further structural classes of orchestration components such as loops with multiple entry and/or multiple exit points. We further aim to consider stochastic QoS values for the SCP.

Acknowledgment

This work is supported in part by E-Finance Lab e. V., Frankfurt am Main, Germany (`http://www.efinancelab.com`).

References

1. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: Web Information Systems Engineering (WISE), IEEE Computer Society (2003) 3–12
2. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR, Upper Saddle River, NJ, USA (2004)
3. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Heuristics for QoS-aware web service composition. In: International/European Conference on Web Services (ICWS), IEEE Computer Society (2006) 72–82
4. Anselmi, J., Ardagna, D., Cremonesi, P.: A QoS-based selection approach of autonomic grid services. In: Service-Oriented Computing Performance (SOCP), ACM (2007) 1–8
5. Menascé, D.A., Casalicchio, E., Dubey, V.K.: A heuristic approach to optimal service selection in service oriented architectures. In: Workshop on Software and Performance (WOSP), ACM (2008) 13–24

6. Huang, A.F.M., Lan, C.W., Yang, S.J.H.: An optimal QoS-based web service selection scheme. Information Sciences (ISCI) **179**(19) (2009) 3309–3322
7. Strunk, A.: QoS-aware service composition: A survey. In: European Conference on Web Services (ECOWS), IEEE Computer Society (2010) 67–74
8. Hillier, F., Lieberman, G.: Introduction to Operations Research. 8 edn. Mc Graw Hill, Boston (2005)
9. Taha, H.: Operations Research – An Introduction. 8 edn. Pearson Prentice Hall, London (2007)
10. Jaeger, M.C., Mühl, G., Golze, S.: QoS-aware composition of web services: An evaluation of selection algorithms. In: OTM Conferences. Volume 3760 of Lecture Notes in Computer Science., Springer (2005) 646–661
11. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Genetic and Evolutionary Computation Conference (GECCO), ACM (2005) 1069–1075
12. Gao, C., Cai, M., Chen, H.: QoS-aware service composition based on tree-coded genetic algorithm. In: International Computer Software and Applications Conference (COMPSAC), IEEE Computer Society (2007) 361–367
13. Lécué, F.: Optimizing QoS-aware semantic web service composition. In: International Semantic Web Conference (ISWC). Volume 5823 of Lecture Notes in Computer Science., Springer (2009) 375–391
14. Mabrouk, N.B., Georgantas, N., Issarny, V.: A semantic end-to-end QoS model for dynamic service oriented environments. In: Workshop on Principles of Engineering Service-oriented Systems (PESOS), IEEE Computer Society (2009) 34–41
15. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering (TSE) **30**(5) (2004) 311–327
16. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Transactions on Software Engineering (TSE) **33**(6) (2007) 369–384
17. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases (DPD) **14**(1) (2003) 5–51
18. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data & Knowledge Engineering (DKE) **68**(9) (2009) 793–818
19. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: Web Services and Formal Methods (WS-FM). Volume 6551 of Lecture Notes in Computer Science., Springer (2010) 25–41
20. Schuller, D., Eckert, J., Miede, A., Schulte, S., Steinmetz, R.: QoS-aware service composition for complex workflows. In: International Conference on Internet and Web Applications and Services, IEEE Computer Society (2010) 333–338
21. Schuller, D., Miede, A., Eckert, J., Lampe, U., Papageorgiou, A., Steinmetz, R.: QoS-based optimization of service compositions for complex workflows. In: International Conference on Service Oriented Computing (ICSOC). Volume 6470 of Lecture Notes in Computer Science. (2010) 641–648
22. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate quality of service computation for composite services. In: International Conference of Service-oriented Computing (ICSOC). Volume 6470 of Lecture Notes in Computer Science., Springer (2010) 213–227
23. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On structured workflow modelling. In: Conference on Advanced Information Systems Engineering (CAiSE). Volume 1789 of Lecture Notes in Computer Science., Springer (2000) 431–445