Dynamically Selecting Composition Algorithms for Economical Composition as a Service

Immanuel Trummer and Boi Faltings

Artificial Intelligence Laboratory
Ecole Polytechnique Fédérale de Lausanne
{immanuel.trummer,boi.faltings}@epfl.ch

Abstract. Various algorithms have been proposed for the problem of quality-driven service composition. They differ by the quality of the resulting executable processes and by their processing costs. In this paper, we study the problem of service composition from an economical point of view and adopt the perspective of a Composition as a Service provider. Our goal is to minimize composition costs while delivering executable workflows of a specified average quality. We propose to dynamically select different composition algorithms for different workflow templates based upon template structure and workflow priority. For evaluating our selection algorithm, we consider two classic approaches to quality-driven composition, genetic algorithms and integer linear programming with different parameter settings. An extensive experimental evaluation shows significant gains in efficiency when dynamically selecting between different composition algorithms instead of using only one algorithm.

Keywords: Quality-Driven Service Composition, Composition as a Service, Dynamic Algorithm Selection

1 Introduction

Over the last years, large scale, public registries for Web services have been emerging. These include domain-specific registries (e.g. biology[1], geospatial Web services[2]) as well as general purpose registries such as Seekda![3] which currently advertises over 28.000 Web services. Due to the large number of available services, a common situation is that several services are able to fulfill the same functionality. In order to select between them, non-functional properties such as service availability and response time can be taken into account. This issue is at the heart of quality-driven service composition [12] (QDSC). In QDSC, tasks of an abstract workflow are associated with sets of functionally equivalent services which differ in their non-functional properties. The goal is to select one service for every task such that the aggregated quality properties of the workflow are optimized while certain minimum requirements are fulfilled. Various algorithms have

[1] http://www.biocatalogue.org/

[2] http://services.eoportal.org/

[3] http://webservices.seekda.com/

been proposed for QDSC. Some of them produce optimal executable workflows but have high resource requirements, others sacrifice optimality for efficiency. In this paper, we propose to select different composition algorithms for different workflow templates in order to maximize the overall performance. The selection should consider structural properties of the template as well as workflow priority. Classifying workflow templates according to structural properties allows to predict the behavior of composition methods more accurately. Considering workflow priority allows to select high-quality composition algorithms for high-priority workflows and high-efficiency algorithms for low-priority workflows.

The original scientific contributions of this paper are *i)* an algorithm that maps workflow templates to composition algorithms, minimizing the overall processing costs for a specified average target quality, and *ii)* an extensive experimental evaluation of our algorithm in comparison to naive approaches. The remainder of the paper is organized as follows. In Sect. 2, we present a motivating scenario, in Sect. 3 the corresponding formal model. We review related literature in Sect. 4. In Sect. 5, we describe our approach in detail, followed by the experimental evaluation in Sect. 6. We conclude with Sect. 7.

2 Motivating Scenario

We adopt the perspective of a fictive Composition as a Service provider as described and motivated by Rosenberg et al. [10] and Blake et al. [4]. Fig. 1 shows an overview of the corresponding architecture. Clients are companies with a portfolio of business processes corresponding to different products and services (presumably more than one). Clients submit their whole portfolio as set of composition requests to the composition service. Every request is associated with a specific workflow template, minimum requirements on the QoS of the executable process, and a utility function weighting between different QoS of the executable process. Clients subscribe and pay for regularly receiving executable processes corresponding to their requests. It is necessary to repeat the composition regularly since the set of available services may change. We consider the processing cost for the provider to be proportional to the running time of the used composition algorithms (this is the case if an Infrastructure as a Service offer like Amazon EC2 [1] is used). The 80/20 rule predicts strong variations in the relative importance of different products and services in industry [8]. It is plausible that this translates to different priorities of the workflows within the portfolio. We will use the number of workflow executions per time unit as priority measure while different measures could be applied as well. Clients specify the expected number for every workflow (eventually using a rough estimate first and refining it later). The composition provider can exploit this information and select computationally cheap composition algorithms for less frequently executed workflows. These cost savings can in part be passed on to the clients. We assume that the composition provider has set a target average quality for the resulting compositions and assigns requests to algorithms in order to minimize the processing cost while guaranteeing this average quality.

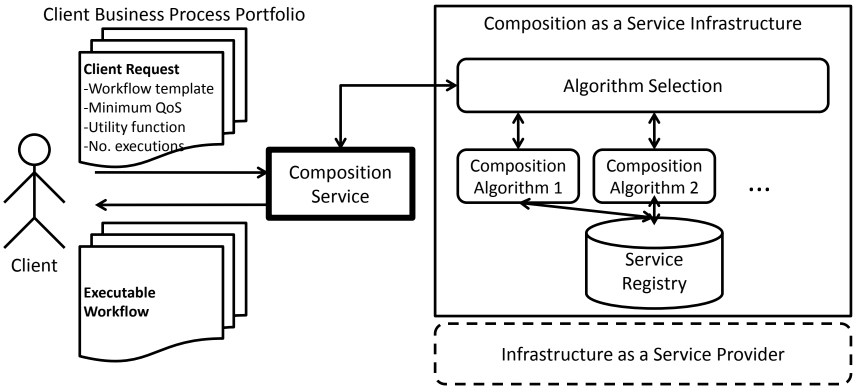


Fig. 1. Architectural overview

3 Formal Model

Our model is similar to the one presented by Zeng et al. [12] and makes the same fundamental assumptions. QDSC starts from an abstract workflow W. Every task of W is associated with a set of services which fulfill the required functionality. Those services expose different non-functional properties. We denote the set of relevant quality properties by A and by $QoS(s,a)$ the value of attribute $a \in A$ for service s. A binding is a function that maps every workflow task to one service in its associated set. The selected binding will determine the aggregated quality properties of the workflow as a whole. We denote by $QoS(W,B,a)$ the value of attribute a for workflow W and binding B. Zeng et al. [12] have shown how to map QoS values to the interval $[0,1]$ such that 1 corresponds to best quality. Hence, we have $QoS(W,B,a) \in [0,1]$. Depending on the attribute type, different aggregation functions must be used. Dumas et al. [7] classify QoS attributes into additive, multiplicative, and attributes whose value is aggregated over the critical path. We consider attributes where the value is aggregated as sum (e.g. response time) or as product (e.g. reliability) over the tasks on the (previously known) critical path.

In QDSC, the goal is to find a binding for a workflow such that *(i)* certain minimum requirements on the quality of the composite workflow are respected, and *(ii)* a user-defined measure of optimality on the quality attributes of the workflow is optimized. Assuming that the attributes in A are ordered ($A = \{a_1, a_2, \ldots, a_n\}$), we can express the quality requirements on the composite workflow as vector $\overrightarrow{r} = (r_1, \ldots, r_n)$. A valid binding B must satisfy (1).

$$\forall i \in \{1, \ldots, n\} : QoS(W,B,a_i) \geq r_i \tag{1}$$

The ranking between different admissible bindings depends on the preferences of the user. Some users will prefer having a lower response time even if this does mean additional invocation costs, for other users it may be the inverse. Users specify their preferences via a vector of weights $\overrightarrow{w} = (w_1, \ldots, w_n)$ where the sum over all components is 1: $|\overrightarrow{w}| = 1$. We define the utility of a binding B:

$$Utility(W, B, \overrightarrow{w}) = \sum_{i \in \{1,\ldots,n\}} w_i QoS(W, B, a_i) \tag{2}$$

For a fixed set of available services, requirements and preferences, we define the relative quality of a binding B by comparison with the optimal binding B_{opt}:

$$relQuality(B, W, \overrightarrow{r}, \overrightarrow{w}) = \frac{Utility(W, B, \overrightarrow{w})}{Utility(W, B_{opt}, \overrightarrow{w})} \tag{3}$$

We assume that workflows are associated with an expected number of executions (during a specific time period) *nExec*. In our model, the number of executions determines the relative importance between workflows in the same set. In summary, a composition request CR is defined by the tuple $CR = (W, \overrightarrow{r}, \overrightarrow{w}, nExec)$. Clients submit sets of composition requests $crSet = \{CR_i\}$ and obtain a set of pairs $resultSet = \{(CR_i, B_i)\}$ with corresponding bindings for every request. The relative quality of a result set is the weighted average over the relative quality of all included bindings weighted by the number of executions:

$$relQuality(resultSet) = \frac{\sum_i relQuality(B_i, W_i, \overrightarrow{r}_i, \overrightarrow{w}_i) \cdot nExec_i}{\sum_i nExec_i} \tag{4}$$

4 Related Work

Among the most popular approaches for QDSC are integer linear programming and genetic algorithms. We will use these two approaches in different configurations for evaluating our dynamic selection algorithm. An **Integer Linear Program (ILP)** consists of a set of variables, a set of linear constraints and a linear objective function. After having translated the QDSC problem into this formalism, specific solver software such as CPLEX [2] can be used. Examples for this approach include the work by Zeng et al. [12] and Ardagna et al. [3]. Canfora et al. [5] introduced **Genetic Algorithms (GA)** for QDSC. Individuals of the population correspond to different bindings, their genes to the workflow tasks and the possible gene values to the available services. While GAs do not guarantee to find the optimal solution, they can be more efficient than ILP-based methods (which have exponential worst-case time complexity). By tuning parameters like the number of iterations, the probability of finding a close-to-optimal solution can be improved. Various other approaches have been applied to QDSC. Many of them offer specific parameters for trading result quality for lower running time (e.g. [6, 11]). Such parameters can be leveraged by our selection algorithm for reducing the composition effort for low-priority workflows.

5 Approach for Selecting Composition Algorithms

In this section we will present an algorithm that maps composition requests to composition methods. The goal is to guarantee an average quality for the result set while minimizing the composition cost. In Sect. 5.1, we describe a preliminary filtering for composition methods, in Sect. 5.2 the selection algorithm.

By $Methods$ we designate the set of composition methods. Every method refers to a specific algorithm with a specific parameter setting (e.g. genetic algorithm with population size 50 chromosomes and 100 generations). For selecting between different methods, we characterize them by the delivered average relative quality (see (3)) and invocation cost. However, the behavior of composition methods depends on the properties of the composition request (e.g. the running time of an ILP method correlates with the number of workflow tasks). We assume that requests can be classified such that the behavior does not vary too much for requests within the same class. *RequestClasses* designates the set of classes, $class(cr)$ the class of a request cr. We characterize methods for specific classes using the functions *Ecost* (expected cost) and *ErelQ* (expected relative quality)—data can be gained by experiments with representative request sets:

$$Ecost : Methods \times RequestClasses \longrightarrow \mathbb{N} \tag{5}$$

$$ErelQ : Methods \times RequestClasses \longrightarrow [0,1] \tag{6}$$

5.1 Initialization: Filtering Composition Methods

During initialization, each request class is assigned to a set of recommended composition methods. The result is the function

$$efficientMethods : RequestClasses \rightarrow \mathcal{P}(Methods) \tag{7}$$

Initially, all methods are considered efficient for all request classes. Then, two filtering steps are performed for each request class separately based upon the experimental data. First, composition methods have to be filtered out that risk to produce workflows of too low quality. We only consider average quality during our dynamic selection, therefore this step is important—having single bindings of very bad quality within the result set may dissatisfy clients even if the average quality is good. Further, methods can be filtered out for certain request classes if they are *dominated* by other methods, meaning that they have higher cost and deliver lower average quality. Filtering out dominated methods diminishes the search space for the dynamic selection and improves therefore the efficiency.

5.2 Mapping Composition Requests to Composition Methods

Every time that a new set of composition requests is submitted by the client, the requests in the set have to be mapped to composition algorithms based upon their relative importance and request class. This mapping has to be done efficiently since the mapping time adds as overhead to the total processing time. Our goal is to minimize the processing cost of the request set while the minimum requirements on the average quality must be met. We will show how our mapping problem can be reformulated as multi-choice 0-1 knapsack problem (MCKP) [9].

Algorithm 1 Select and execute composition methods for request set

```
function TreatRequestSet($crSet, efficientMethods, Ecost, ErelQ, tQ$)
    // Transform selection problem into multi-choice 0-1 knapsack
    $weightLimit \leftarrow 0$
    for all $cr = (W, \overrightarrow{r}, \overrightarrow{w}, nExec) \in crSet$ do
        $optM(cr) \leftarrow \text{argmax}_{m \in efficientMethods(class(cr))}(ErelQ(m, class(cr)))$
        $mckItems(cr) \leftarrow \emptyset$
        for all $m \in efficientMethods(class(cr)) \setminus \{optM(cr)\}$ do
            $costSavings \leftarrow Ecost(optM(cr), class(cr)) - Ecost(m, class(cr))$
            $qualityLoss \leftarrow ErelQ(optM(cr), class(cr)) - ErelQ(m, class(cr))$
            $newItem \leftarrow (m, qualityLoss \cdot nExec, costSavings)$
            $mckItems(cr) \leftarrow mckItems(cr) \cup \{newItem\}$
        end for
        $weightLimit \leftarrow weightLimit + nExec \cdot (ErelQ(optM(cr), class(cr)) - tQ)$
    end for
    $mckSelected \leftarrow$ approximateKnapsack($crSet, mckItems, weightLimit, \epsilon$)
    // Use approximated solution and call corresponding composition methods
    $resultSet \leftarrow \emptyset$
    for all $cr \in crSet$ do
        if $mckSelected(cr) = \perp$ then
            $binding \leftarrow$ Execute($optM(cr), cr$)
        else
            $(m, qualityLoss, costSavings) \leftarrow mckSelected(cr)$
            $binding \leftarrow$ Execute($m, cr$)
        end if
        $resultSet \leftarrow resultSet \cup \{(cr, binding)\}$
    end for
    return $resultSet$
end function
```

This problem is NP-hard but can be approximated efficiently using a *fully polynomial time approximation scheme (FPTAS).* Such an approximation scheme guarantees polynomial running time and a close-to-optimal solution. If the optimal utility value for a given problem instance is P_{opt}, then the approximation scheme finds a solution with utility value at least P such that $P_{opt} - P \leq \epsilon \cdot P_{opt}$ where ϵ can be chosen. The running time grows polynomial in $\frac{1}{\epsilon}$ and in the size of the problem. We use the MCKP FPTAS by Lawler [9] for our implementation.

Alg. 1 is executed every time a client submits a set of composition requests. It takes as input the submitted request set $crSet$, the set of recommended methods for every request class $efficientMethods$, the characteristics of the available methods $ErelQ$, $Ecost$, and the targeted relative quality of the result set tQ. In a first phase, the algorithm reformulates the problem of selecting optimal methods for every composition request as MCKP. The classes correspond to the different requests that have to be treated. Items within a specific class are associated with composition methods. Selecting an item for a class symbolizes the choice of the associated composition method for treating the request corresponding to that class. The item weight corresponds to the *quality loss* in comparison with the optimal method, weighted by the number of executions. The total weight limit is proportional to the total number of executions of all workflow templates in the request set. It integrates the distance between target quality $tQ \in [0, 1]$ and the expected quality of the best method for every request. We want to minimize the processing cost, a solution to the MCKP is optimal once it maximizes the aggregated profit. Therefore, item profits correspond to *cost savings* that can be realized by choosing the associated method instead of the optimal one.

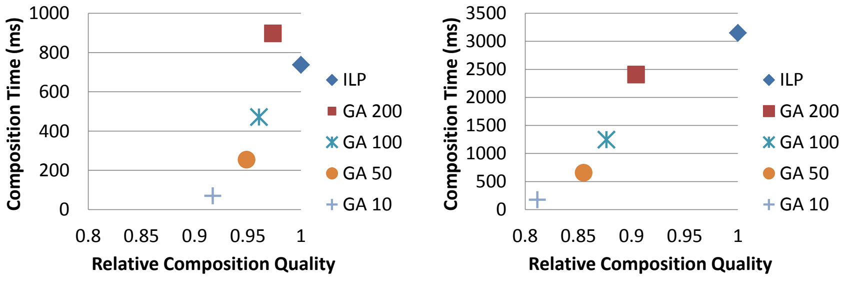
The item associated with the optimal method has weight and profit 0. This is equivalent to selecting no element in the class. Therefore, we do not integrate these items and interpret an empty selection for a class as selection of the optimal method for the corresponding request. The algorithm uses the auxiliary function approximateKnapsack which implements the FPTAS proposed by Lawler [9]. The functions takes as input the set of item classes, the sets of items for every class, the weight limit and the accuracy ϵ (Lawler's algorithm works with integer weights hence we round weights to percent). It returns a function that assigns classes to selected items or to $\perp$ if no item was selected. The algorithm uses the auxiliary function Execute(m, cr) which executes m on cr and returns the produced binding. The set of pairs between bindings and requests is returned.

Example 1. Let $crSet = \{cr1, cr2\}$ with $cl1 = class(cr1)$ and $cl2 = class(cr2)$, we have $nExec = 10$ for both requests. Assume that methods $m1$ and $m2$ are efficient for $cl1$ with $ErelQ(m1, cl1) = 0.9$, $ErelQ(m2, cl1) = 0.8$, $Ecost(m1, cl1) = 10$, and $Ecost(m2, cl1) = 5$. Only $m1$ is efficient for $cl2$ with $ErelQ(m1, cl2) = 0.95$ and $Ecost(m1, cl2) = 5$. Our algorithm generates item set $\{(m2, 1.0, 5)\}$ for knapsack class $cl1$ and $\emptyset$ for $cl2$. The weight limit is 2.5 for $tQ = 0.8$.

6 Experimental Evaluation

In this section, we experimentally evaluate our dynamic selection approach. In subsection 6.1, we benchmark two classic algorithms for QDSC—integer linear programming and genetic algorithms—in different configurations for different classes of composition queries. The experimental data we obtain in subsection 6.1 forms the input for our selection algorithm that we evaluate in subsection 6.2.

We implemented a test suite in Java that randomly generates composition requests including workflow templates and available services. We treat workflows with between 5 and 45 tasks. We considered 8 quality attributes for services: two additive attributes that depend on all tasks, two that depend only on critical tasks, two multiplicative attributes that depend on all tasks, and two that depend only on critical tasks. The QoS properties of services were chosen with uniform random distribution. We considered 50 functional categories and generated 100 services for every category. Workflow tasks were randomly assigned to functional categories. The probability that a task belongs to the critical path for one of the quality attributes that depend only on critical tasks was 50%. The quality weights were chosen randomly as well as the quality requirements which were chosen with uniform distribution between 0.01 and 0.5. The number of workflow



(a) Workflows with 5 to 15 tasks (b) Workflows with 31 to 45 tasks

Fig. 2. Characteristics of composition methods for different request classes

executions for every single request was chosen out of a Pareto distribution as motivated before. For implementing the ILP based algorithm, we used IBM ILOG CPLEX 12.1 [2] as solver. We set the thread count to 1 and used the default parameters otherwise. For the GA composition approach, we use the same Java libraries and settings as Canfora et al. [5]. However, we vary the number of generations between 10 and 200. The approximation algorithm for the MCKP was implemented in Java as well. All experiments were executed on a 2.53 GHz Intel Core Duo processor with 2.5 GB RAM running Windows 7.

6.1 Benchmarking and Filtering Composition Methods

In this section, we characterize different configurations of the two composition algorithms. We partitioned requests into 3 classes, based upon the number of workflow tasks ($5-15$, $16-30$, and $31-45$ tasks). Note that a more fine-grained partitioning could additionally consider different request properties like the strength of the quality requirements. For every class we generated 100 test cases (corresponding to a randomly generated registry and workflow request). We executed every method 10 times for every test case and take the arithmetic average execution times. Fig. 2 shows the characteristics of different composition methods within the cost-quality space. For determining the relative quality, we compared with the optimal solution produced by ILP. We benchmark GA with different numbers of generations (10, 50, 100, and 200).

We make the following observations. *i)* The only case of dominance between different methods occurs for small workflows: ILP dominates GA 200 since it delivers better quality at lower cost. *ii)* The running time of the GA-based methods is approximately proportional to the number of generations and the average number of tasks. The growth of composition time for the ILP approach is over-proportional such that "GA 200" is not dominated anymore for large workflows. *iii)* For the same number of generations, the relative quality of the genetic algorithms slightly decreases when the number of workflow tasks grows. *iv)* The standard deviation was always below 1% (of average value) for the relative quality while reaching up to 9% for the running time.

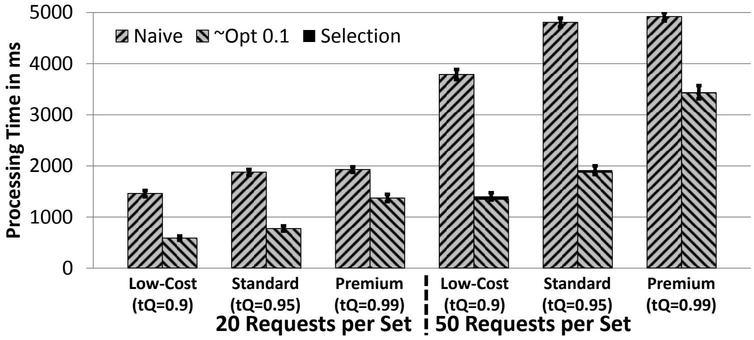
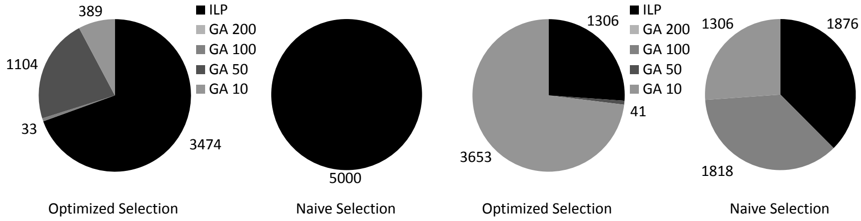


Fig. 3. Comparison of naive and optimized selection approaches



(a) Premium composition (tQ=0.99) (b) Low-Cost composition (tQ=0.9)

Fig. 4. No. selections for composition methods (100 times 50 requests per set)

6.2 Evaluating Selection Algorithms

We compare our near-optimal selection approach with $\epsilon = 0.1$ to a naive algorithm. Both variants work with the data from the previous subsection. The naive approach selects for a given request cr and target average quality tQ the composition method $m \in efficientMethods(class(cr))$ which has minimum expected cost among the methods that deliver the required target quality $ErelQ(m, class(cr)) \geq tQ$. Fig. 3 shows the results of our comparison with 5% confidence intervals. We generated and solved 100 request sets and report the arithmetic average times. We compare the two selection algorithms for request sets of different size (20 and 50 requests) and different quality requirements (from $tQ = 0.9$ to $tQ = 0.99$). Our criterion is the total processing time per request set. For the near-optimal selection strategy, we divide the time into time required to map requests to algorithms and time required for executing the selected algorithms. We observe the following. *i)* The processing time increases for higher number of requests and increasing quality requirements. *ii)* The time for the selection phase accounts only for between 0.2% and 4% of the total processing time for the near-optimal selection. *iii)* Our selection approach takes only 40% (37%) of the time of the naive approach for 20 requests per set (50 requests per set) and for $tQ = 0.9$, 40% (41%) for $tQ = 0.95$, and 71% (70%) for $tQ = 0.99$.

We verified that the relative quality of the result sets (rounded to percent) produced by our approach always met the specified bounds. Fig. 4 shows how many requests the different selection approaches assigned to the different composition methods. ILP is the dominant method for high target quality ($tQ = 0.99$) while GAs dominate for lower quality ($tQ = 0.9$). Our approach is able to select more low-cost composition methods which explains the higher efficiency.

7 Conclusion

In this paper, we classify existing composition algorithms in terms of running cost and expected quality. We dynamically assign different workflow templates to different composition algorithms based upon template structure and relative importance. Our experimental evaluation shows that our approach reduces composition cost significantly while introducing little overhead.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "SOSOA: Self-Organizing Service-Oriented Architectures" (SNF Sinergia Project No. CRSI22_127386/1).

References

1. Amazon elastic compute cloud, `http://aws.amazon.com/ec2/`
2. Ibm ilog cplex, `www.ibm.com/software/products/de/de/ibmilogcple/`
3. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Transactions on Software Engineering pp. 369–384 (2007)
4. Blake, M., Tan, W., Rosenberg, F.: Composition as a service [web-scale workflow]. Internet Computing 14(1), 78–82 (2010)
5. Canfora, G., Di Penta, M., Esposito, R., Villani, M.: An approach for QoS-aware service composition based on genetic algorithms. In: Conf. on Genetic and evolutionary computation. pp. 1069–1075. ACM (2005)
6. Comes, D., Baraki, H., Reichle, R., Zapf, M., Geihs, K.: Heuristic Approaches for QoS-Based Service Selection. In: Int. Conf. on Service-Oriented Computing. pp. 441–455. Springer (2010)
7. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate quality of service computation for composite services. In: Int. Conf. on Service-Oriented Computing. pp. 213–227. Springer (2010)
8. Koch, R.: Das 80-20-Prinzip. Campus-Verl. (1998)
9. Lawler, E.: Fast approximation algorithms for knapsack problems. In: 18th Annual Symposium on Foundations of Computer Science, 1977. pp. 206–213. IEEE (1977)
10. Rosenberg, F., Leitner, P., Michlmayr, A., Celikovic, P., Dustdar, S.: Towards composition as a service-a quality of service driven approach. In: Int. Conf. on Data Engineering. pp. 1733–1740. IEEE (2009)
11. Trummer, I., Faltings, B.: Optimizing the Tradeoff between Discovery, Composition, and Execution Cost in Service Composition. In: Int. Conf. on Web Services (2011)
12. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)