Efficient QoS-aware Service Composition

Mohammad Alrifai and Thomas Risse

L3S Research Center
Leibniz University of Hannover, Germany
{alrifai|risse}@L3S.de

Abstract. Web service composition requests are usually combined with end-to-end QoS requirements, which are specified in terms of non-functional properties (e.g. response time, throughput and price). The goal of QoS-aware service composition is to find the best combination of services such that their aggregated QoS values meet these end-to-end requirements. Local selection techniques are very efficient but fail short in handling global QoS constraints. Global optimization techniques, on the other hand, can handle global constraints, but their poor performance render them inappropriate for applications with dynamic and real-time requirements. In this paper we address this problem and propose a solution that combines global optimization with local selection techniques for achieving a better performance. The proposed solution consists of two steps: first we use mixed integer linear programming (MILP) to find the optimal decomposition of global QoS constraints into local constraints. Second, we use local search to find the best web services that satisfy these local constraints. Unlike existing MILP-based global planning solutions, the size of the MILP model in our case is much smaller and independent on the number of available services, yields faster computation and more scalability. Preliminary experiments have been conducted to evaluate the performance of the proposed solution.

1 Introduction

Industrial practice witnesses a growing interest in the ad-hoc service composition in the areas of supply chain management, accounting, finances, eScience as well as in multimedia applications. With the growing number of the services available within service infrastructures, the composition problem becomes a decision problem on the selection of component services from a set of alternative services that provide the same functionality but differ in quality parameters. In service oriented environments, where deviations from the QoS estimates occur and decisions upon replacing some services has to be taken at run-time (e.g. in multimedia applications), the efficiency of the applied selection mechanism becomes crucial.

Consider for example the personalized multimedia delivery scenario (from [1]) in Figure 1. A smartphone user requests the latest news from a service provider. Available multimedia content includes a news ticker and topical videos available in MPEG 2 only. The news provider has no adaptation capabilities, so additional services are required to serve the user's request: a transcoding service for the multimedia content to fit the target format, a compression service to adapt the content to the wireless link, a text translation service for the ticker, and also a merging service to integrate the ticker with the video

stream for the limited smartphone display. The user request can be associated with some end-to-end QoS requirements (like bandwidth, latency and price). The service composer has to ensure that the aggregated QoS values of the selected services match the user requirements at the start of the execution as well as during the execution. However, dynamic changes due to changes in the QoS requirements (e.g. the user switched to a network with lower bandwidth) or failure of some services (e.g. some of the selected services become unavailable) can occur at run-time. Therefore, a quick response to adaptation requests is important in such applications. Exponential time complexity for an infrastructure service like the composition service is only acceptable if the number of service candidates and constraints are very limited. Already in larger enterprises and even more in open service infrastructures with a few thousands of services the response time for a service composition request could already be out of the real-time requirements.

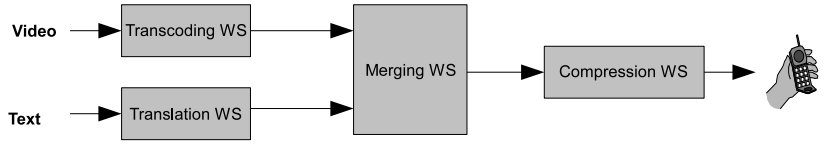

Fig. 1. Composition of Multimedia Web Services

Two general approaches exist for the QoS-aware service selection: local selection and global selection. In local selection, one service is selected from each class independently. Using a given utility function, each service candidate is assigned a utility value and the service with maximum utility value is selected. This approach is very efficient in terms of computation time as the time complexity of the local optimization approach is $O(l)$, where l is the number of service candidates in each class. This is specially useful for distributed environments where central QoS management is not desirable. However, local selection is not suitable for QoS-based service composition, with global end-to-end requirements (like maximum total price), since such global constraints cannot be verified locally.

On the other hand, the global selection [2–5] aims at solving the problem on the composite service level by considering all possible service combinations. The aggregated QoS values of each service combination is computed. This approach seeks the service combination, which maximizes the aggregated utility value, while guaranteeing global constraints. The global selection problem can be modeled as a *Multi-Choice Multidimensional Knapsack problem* (MMKP), which is known to be NP-hard in the strong sense [6]. Therefore it can be expected that an optimal solution may not be found in a reasonable amount of time [7].

Since the business requirements (such as response times, throughput or availability) are only approximate, we argue that finding a reasonable selection of services that covers the requirements "'approximately"' and avoids obvious violations of constraints at acceptable costs is more important than finding "'the optimal"' selection of services with a very high cost. The aim of our study is to find a compromise between optimality

and performance by combining local selection and global optimization to benefit from the advantages of both techniques.

The contribution of this paper can be stated as follows. We divide the global QoS optimization problem into two sub-problems that can be solved separately to improve the efficiency:

- The first sub-problem being the decomposition of global QoS constraints into local constraints, is modeled as a mixed integer linear program [8]. The size of the resulting program is independent on the number of service candidates and hence can be solved more efficiently than existing MILP-based solutions.
- The second sub-problem being the selection of component services is solved by means of local selection. We guide the local selection by means of some global parameters to ensure that the computation of the service utility is not biased by local characteristics.

The rest of the papers is organized as follows. In the next section we review some related work. Section 3 introduces the system model and gives a problem statement. Our approach for efficient QoS computation for web service composition is presented in Section 4. Preliminary empirical results for comparing the performance of our solution with the performance of global planning are presented in Section 5 Finally, Section 6 gives conclusions and an outlook on possible continuations of our work.

2 Related Work

The functional requirements for web service composition can be stated in a workflow language such as Business Process Execution Language (BPEL) [9]. In [10, 11] ontology-based representations for describing QoS properties and requests were proposed to support semantic and dynamic QoS-based discovery of web services. Quality of service management has been widely discussed in the area of middleware systems [12–16]. Most of these works focus on QoS specification and management. Recently, the QoS-based web service selection and composition in service-oriented applications has gained the attention of many researchers [2–5, 17, 18]. In [17] the authors propose an extensible QoS computation model that supports open and fair management of QoS data. The problem of QoS-based composition is not addressed by this work. The work of Zeng at al. [2, 3] focuses on dynamic and quality-driven selection of services. The authors use global planning to find the best service components for the composition. They use (mixed) linear programming techniques [8] to find the optimal selection of component services. Similar to this approach Ardagna et al. [4, 5] extends the linear programming model to include local constraints. Linear programming methods are very effective when the size of the problem is small. However, these methods suffer from poor scalability due to the exponential time complexity of the applied search algorithms [19, 7]. In [18] the authors propose heuristic algorithms that can be used to find a near-to-optimal solution more efficiently than exact solutions. The authors propose two models for the QoS-based service composition problem: 1) a combinatorial model and 2) a graph model. A heuristic algorithm is introduced for each model. The time

complexity of the heuristic algorithm for the combinatorial model (WS_HEU) is polynomial, whereas the complexity of the heuristic algorithm for the graph model (MCSP-K) is exponential. Despite the significant improvement of these algorithms compared to exact solutions, both algorithms do not scale with respect to an increasing number of web services and remain out of the real-time requirements. Any distributed implementation of these algorithms would rise a very high communication cost. The WS_HEU for example, is an improvement of the original heuristic algorithm named M-HEU [20]. The M-HEU algorithm starts with a pre-processing step for finding an initial feasible solution, i.e. a service combination that satisfies all constraints but not necessarily is the best solution. A post-processing step improves the total utility value of the solution with one upgrade followed by one or more downgrades of one of the selected component services. Applying this algorithm in a distributed setting where the QoS data of the different service classes is managed by distributed service brokers would rise very high communication cost among these brokers to find the best composition. In this paper, we propose a heuristic algorithm that solves the composition problem more efficiently and fits well to the distributed environment of web services.

3 System Model and Problem Statement

3.1 Abstract vs. Concrete Composite Services

In our model we assume that we have a universe of web services $\mathbb{S}$ which is defined as a union of *abstract service classes*. Each abstract service class $S_j \in \mathbb{S}$ (e.g. flight booking services) is used to describe a set of functionally-equivalent web services (e.g. Lufthansa and Qantas flight booking web services). In this paper we assume that information about service classes is managed by a set of service brokers as described in [17, 21]. Web services can join and leave service classes at any time by means of a subscription mechanism. We also distinguish between the following two concepts:

– An abstract composite service, which can be defined as an abstract representation of a composition request $CS_{abstract} = \{S_1, \ldots, S_n\}$. $CS_{abstract}$ refers to the required service classes (e.g. flight booking) without referring to any concrete web service (e.g. Qantas flight booking web Service).

– A concrete composite service, which can be defined as an instantiation of an abstract composite service. This can be obtained by bounding each abstract service class in $CS_{abstract}$ to a concrete web service s_j, such that $s_j \in S_j$.

3.2 QoS Vector

In our study we consider quantative non-functional properties of web services, which can be used to describe the quality of a service s. We use the vector $Q_s = \{q_1, q_2, \ldots, q_r\}$ to represent these properties. These can include generic QoS attributes like response time, availability, price, reputation etc, as well as domain-specific QoS attributes like bandwidth, video quality for multimedia web services. The values of these QoS attributes can be either collected from service providers directly (e.g. price), recorded

from previous execution monitoring (e.g. response time) or from user feedbacks (e.g. reputation) [17]. The set of QoS attributes can be divided into two subsets: positive and negative QoS attributes. The values of positive attributes need to be maximized (e.g. throughput and availability), whereas the values of negative attributes need to be minimized (e.g. price and response time). For the sake of simplicity, in this paper we consider only negative attributes (positive attributes can be easily transformed into negative attributes by multiplying their values by -1). We use the function $q_i(s)$ to determine the i-th quality parameter of service s. The QoS information of web services from class S are managed by the responsible service broker of this class.

3.3 QoS Computation of Composite Services

The QoS value of a composite service is decided by the QoS values of its component services as well as the composition model used (e.g. sequential, parallel, conditional and/or loops). In this paper, we focus on the service selection algorithm for QoS-based service composition, and its performance on the sequential composition model. Other models may be reduced or transformed to the sequential model. Techniques for handling multiple execution paths and unfolding loops from [2], can be used for this purpose.

The QoS vector for a composite service CS is defined as $Q_{CS} = \{q'_1(CS), \ldots, q'_r(CS)\}$ where $q'_i(CS)$ represents the estimated QoS values of a composite service CS and can be aggregated from the expected QoS values of its component services. Table 3.3 shows examples of some QoS aggregation functions.

Similar to [2, 17, 5, 18], we assume in our model that QoS aggregation functions either are linear or can be linearized to be represented by the summation relation. For example, QoS attributes that are typically aggregated as a product (e.g. availability) are transformed into a summation relation by applying a logarithm operation.

We extend our model to support the following aggregation function:

$$q'_k(CS) = \sum_{j=1}^{n} q_k(s_j) \tag{1}$$

Table 1. Examples of QoS aggregation functions for composite services

QoS Attribute	Aggregation Function
Response Time	$q'_{res}(CS) = \sum_{j=1}^{n} q_{res}(s_j)$
Price	$q'_{price}(CS) = \sum_{j=1}^{n} q_{price}(s_j)$
Availability	$q'_{av}(CS) = \prod_{j=1}^{n} q_{av}(s_j)$

3.4 Utility Function

In order to evaluate the multi-dimensional quality of a given web service composition a utility function is used. In this paper we use a Multiple Attribute Decision Making approach for the utility function: i.e. the *Simple Additive Weighting (SAW)* technique [22]. The utility computation involves scaling the values of QoS attributes to allow a uniform measurement of the multi-dimensional service qualities independent of their units and ranges. The scaling process is then followed by a weighting process for representing

user priorities and preferences. In the scaling process each QoS attribute value is transformed into a value between 0 and 1, by comparing it with the minimum and maximum possible aggregated value. These values can be easily estimated by aggregating the local minimum (or maximum) possible value of each service class in CS. For example, the maximum execution price of any concrete composite service can be computed by summing up the execution price of the most expensive service in each service class. Formally, we compute the minimum and maximum aggregated value of the k-th QoS attribute as follows:

$$Qmin'(k) = \sum_{j=1}^{n} Qmin(j,k) \text{ and } Qmax'(k) = \sum_{j=1}^{n} Qmax(j,k) \tag{2}$$

where $Qmin(j,k) = \min_{\forall s_{ji} \in S_j} q_k(s_{ji})$ is the minimum value (e.g. minimum price) and $Qmax = \max_{\forall s_{ji} \in S_j} q_k(s_{ji})$ is the maximum value (e.g. maximum price) that can be expected for service class S_j according to the available information about service candidates of this class.

Now the overall utility of a composite service is computed as

$$U'(CS) = \sum_{k=1}^{r} \frac{Qmax'(k) - q'_k(CS)}{Qmax'(k) - Qmin'(k)} \cdot w_k \tag{3}$$

with $w_k \in \mathbb{R}_0^+$ and $\sum_{k=1}^{r} w_k = 1$ being the weight of q'_k to represent user's priorities.

The utility function $U'(CS)$ is used to evaluate a given set of alternative service compositions. However, finding the best composition requires enumerating all possible combinations of service candidates. For a composition request with n service classes and l service candidate per class, there are l^n possible combinations to be examined. Performing exhaustive search can be very expensive in terms of computation time and, therefore, inappropriate for applications with many services and dynamic needs.

3.5 Problem Statement

The problem of finding the best service composition without enumerating all possible combinations is considered as an optimization problem, in which the overall utility value has to be maximized while satisfying all global constraints. Formally, the optimization problem we are addressing can be stated as follows:

For a given abstract composite service $CS_{abstract} = \{S_1, \ldots, S_n\}$ with a set of m global QoS constraints $C' = \{c'_1, \ldots, c'_m\}$, find an implementation $CS = \{s_{1b}, \ldots, s_{nb}\}$ by binding each S_j to a concrete service $s_{jb} \in S_j$ such that:

1. The aggregated QoS values satisfy: $q'_k(CS) \leq c'_k, \forall c'_k \in C'$, and
2. The overall utility $U'(CS)$ is maximized

4 Efficient QoS-aware Service Composition

The use of mixed integer linear programming [8] to solve the QoS-aware service composition problem has been recently proposed by several researchers [2–5]. The decision

(binary) variables in the model represent the service candidates. A service candidate s_{ij} is selected in the optimal composition if its corresponding variable x_{ij} is set to 1 in the solution of the program, and discarded otherwise. The program is formulated as follows (by re-writing (3) to include the decision variables):

maximize the overall utility value given by:

$$\sum_{k=1}^{r} \frac{Qmax'(k) - \sum_{i=1}^{n} \sum_{j=1}^{l} q_k(s_{ji}) * x_{ji}}{Qmax'(k) - Qmin'(k)} \cdot w_k$$

subject to the following global QoS constraints:

$$\sum_{i=1}^{n} \sum_{j=1}^{l} q_k(s_{ji}) * x_{ji} \leq c'_k, 1 \leq k \leq m$$

while satisfying the following allocation constraints on the decision variables:

$$\sum_{j=1}^{l} x_{ji} = 1, 1 \leq i \leq n$$

Because the number of variables in this model depends on the number of service candidates (number of variables = $n * l$), this MILP model may not be solved satisfactorily, except for small instances. To cope with this limitation, we divide the QoS-aware service composition problem into two sub-problems that can be solved more efficiently in two subsequent phases. In the first phase, we use mixed integer linear programming to find the best decomposition of global QoS constraints into local constraints on the component services. The size of the MILP model of this phase is much smaller than the size of the MILP model in [2–5] and can be, therefore, solved much faster. In the second phase, we use local search to find the best component services that satisfy the local constraints from the first phase. The two phases of our approach are described in the next subsections in more details.

4.1 Decomposition of Global QoS Constraints

To ensure that the results of local search comply with the global QoS constraints, we need to set up some local constraints on the QoS values of the individual services. For example, in a composition problem with n service classes, a global constraint such as: total response time ≤ 600 msec, i.e. $\sum_{i=1}^{n} q_{res}(s_i) \leq 600$, needs to be translated into n local constraints in the form of:

$$q_{res}(s_i) \leq R_i, 1 \leq i \leq n \qquad \text{, where} \quad \sum_{i=1}^{n} R_i = 600$$

A naive solution to this problem would be to divide the global constraint into n equal local constraints: $R_i \leq 600/n, 1 \leq i \leq n$. However, as different service classes can have different response times, a more sophisticated decomposition algorithm is required. The

decomposition algorithm need to ensure that the local constraints are not more restrictive than needed in order to avoid discarding any service candidates that might be part of a feasible solution. To solve this problem, we divide the quality range of each QoS attribute into a set of discrete quality values, which we call "*quality levels*". We then use mixed integer linear programming to find the best combination of these quality levels for using them as local constraints.

Quality Levels: In this paper, we use a simple method for constructing the quality levels. We divide the value ranges of each QoS attribute q_k of service class S_i into d levels: $q_{ik}^1, ..., q_{ik}^d$ as follows:

$$q_{ik}^z = \begin{cases} Qmin(i,k) & \text{if } z = 1 \\ q_{ik}^{z-1} + \frac{Qmax(i,k) - Qmin(i,k)}{d} & \text{if } 1 < z < d \\ Qmax(i,k) & \text{if } z = d \end{cases} \quad (4)$$

We then assign each quality level q_{ik}^z a value between 0 and 1, which indicates the probability p_{ik}^z that using this quality level as a local constraint would lead to finding a solution. The probability p_{ik}^z for the zth level of q_k at S_i is computed as follows:

$$p_{ik}^z = h/l \quad (5)$$

where h is the number of service candidates satisfying q_{ik}^z and l is the total number of service candidates at S_i.

MILP Formulation: The goal of our MILP model is to find the best decomposition of QoS constraints into local constraints. Therefore, we use a binary decision variable x_{ik}^z for each local quality level q_{ik}^z such that $x_{ik}^z = 1$ if q_{ik}^z is selected as a local constraint for the QoS attribute q_k at the service class S_i, and $x_{ik}^z = 0$ otherwise. Note that the total number of variables in the model equals to $n * m * d$, i.e. independent on the number of service candidates. By ensuring that the number of quality levels d is small enough such that $m * d \leq l$ we can ensure that the size of our MILP model is smaller than the size of the model used in [2–5].

To ensure that only one quality level is selected from the set of d levels of the QoS attribute q_k at the service class S_i, we add the following set of constraints to the model:

$$\sum_{z=1}^{d} x_{ik}^z = 1, \forall i, \forall k, 1 \leq i \leq n, 1 \leq k \leq m$$

The selection of the local constraints must ensure that global constraints are still satisfied (i.e. the first requirement in 3.5). Therefore, we add the following set of constraints to the model:

$$\sum_{i=1}^{n} \sum_{z=1}^{d} q_{ik}^z * x_{ik}^z \leq c'_k, \forall k, 1 \leq k \leq m$$

The objective function of our MILP model is to maximize the probability that the selected local constraints will lead to finding a feasible composition. Therefore, using (5) the objective function can be expressed as follows:

$$\text{maximize} \quad \bigcap_{i=1}^{n}\bigcap_{k=1}^{m} p_{ik}^{z} \quad \Rightarrow \quad \text{maximize} \quad \prod_{i=1}^{n}\prod_{k=1}^{m} p_{ik}^{z} \quad , 1 \leq z \leq d \qquad (6)$$

Using the logarithmic function to linearize (6) in order to be able to use it in the MILP model, we express the objective functions as follows:

$$\text{maximize} \quad \sum_{i=1}^{n}\sum_{k=1}^{m}\sum_{z=1}^{d} ln(p_{ik}^{z}) * x_{ik}^{z} \qquad (7)$$

By solving this model using MILP solver methods, we get a set of local quality levels that we use in the second phase for guiding local selection.

4.2 Local Selection

The local quality levels, which we obtain from the first phase, are used in the second phase as upper bounds for the QoS values of component services. We filter web services that violate these upper bounds and create a list of qualified services for each service class. The next step in this phase involves sorting the qualified web services by their utility values. We derive a local utility function $U'_{local}(s)$ from $U'(CS)$ that can be applied on the service component's level.

By applying (1) and (2) we get:

$$\begin{aligned} U'(CS) &= \sum_{i=1}^{r} \frac{\sum_{\forall s \in CS} Qmax(class(s), i) - \sum_{\forall s \in CS} q_i(s)}{Qmax'(i) - Qmin'(i)} \cdot w_i \\ &= \sum_{\forall s \in CS} \underbrace{(\sum_{i=1}^{r} \frac{Qmax(class(s), i) - q_i(s)}{Qmax'(i) - Qmin'(i)} \cdot w_i)}_{U'_{local}(s)} = \sum_{\forall s \in CS} U'_{local}(s) \end{aligned} \qquad (8)$$

The utility function $U'_{local}(s)$ can be computed for each service class S_j independently, provided that the global parameters $Qmin'$ and $Qmax'$ are specified. These parameters can be easily computed by beforehand by aggregating the local maximum and minimum values of each service class. Thus, by selecting the service candidate with the maximum U'_{local} value from each class, we can ensure that $U'(CS)$ is maximized (i.e. satisfying the second requirement in 3.5).

5 Experimental Evaluation

In this section we describe preliminary results of experimental evaluation of our proposed solution. We conducted our experiments on a HP ProLiant DL380 G3 machine

with 2 Intel Xeon 2.80GHz processors and 6 GB RAM. The machine is running under Linux (CentOS release 5) and Java 1.6.

In our evaluation we compare the performance of our solution with the performance of the MILP-based "pure" global planning solution [3, 5]. For solving the MILP problems in both approaches we use the open source (Mixed Integer Linear Programming) LpSolve system *lpsolve* version 5.5 [23].

For testing purposes we created 20 service classes with 100 service candidates for each class. The QoS dataset was randomly created by assigning each web service candidate 5 arbitrary values for 5 QoS attributes ranging between 0 and 100. All random values are normally distributed. We created several instances of the QoS composition problem by varying the number of service candidates per class l. The number of global constraints m in all problem instances was fixed to 5 constraints. In the current implementation of our approach we used the simple method for constructing quality levels as described earlier in 4.1. Using this method we divide each quality dimension into 5 quality levels.

Figure 2 shows a comparison of the performance between the "pure" global planning approach (labeled with "Global") and our approach (labeled with "Hybrid") that combines global optimization and local search. We measure the time required by each approach to find the best combination of services (i.e. with maximum utility value) that satisfies all the given global QoS constraints. The measured time does not include any data loading or initialization time as these times are independent of the applied approach. In our evaluation we consider only the pure computational time from the start of until the end of the search process. The results shown are averaged over a total of 100 different runs of the same experiment with the same parameters. In the graph shown on

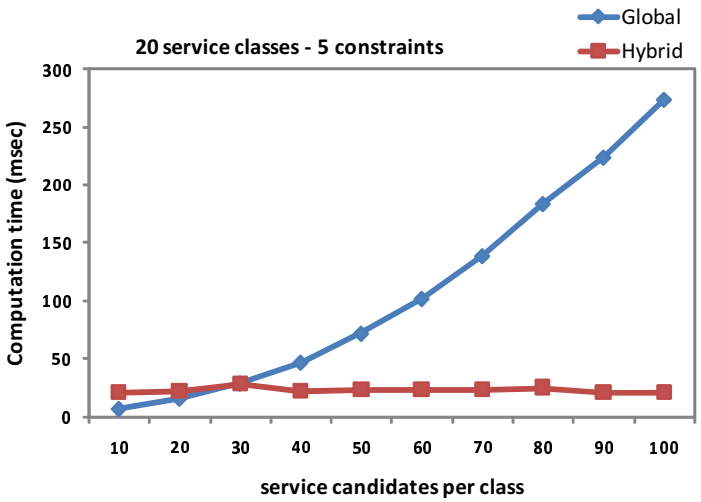

Fig. 2. Computational time with respect to the problem size

Figure 2 we measure the required computation time by each approach with respect to an increasing number of service candidates. The number of service classes n in this experiment is fixed to 20, while the number of service candidates l is varying between 10 and 100 service candidates per class.On the left-side graph we notice that for small-sized

problem instances (with $l \leq 25$) the global planning approach performs better than our solution. This is an expected behavior as we already discussed in Section 4.1. With the number of service candidates $l \leq m * d$ our solution does not gain any improvement in the performance as the size of the used MILP model remains greater than or equal to the original MILP model used by global planning approach. However, with an increasing number of candidates, the computation time of global planning increases dramatically, while the performance of our solution remains unaffected by the number of candidates, which makes our solution more scalable.

6 Conclusion and Current Work

This paper describes an efficient method for the QoS-based service composition. The problem is known to be NP-hard. We combine global optimization with local selection methods to benefit from the advantaged of both worlds. Our proposed method allows to dramatically reduce the (worst case) efforts compared to existing solutions. Preliminary imperial results show a very promising improvement in terms of computational time. This is specially useful for applications with dynamic changes and real-time requirements. Currently we are working on more extensive experiments with larger composition problems. In the work presented in this paper, we use a very simple method for constructing QoS quality levels. We are currently studying the impact of the applied method as well as the number of quality levels used on the performance and quality of the obtained results, especially when different datasets are used.

References

1. Wagner, M., Kellerer, W.: Web services selection for distributed composition of multimedia content. In: MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia, New York, NY, USA, ACM (2004) 104–107
2. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW. (2003) 411–421
3. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Trans. Software Eng **30**(5) (2004) 311–327
4. Ardagna, D., Pernici, B.: Global and local qos constraints guarantee in web service selection. icws **0** (2005) 805–806
5. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Trans. Software Eng. **33**(6) (2007) 369–384
6. Pisinger, D.: Algorithms for Knapsack Problems. PhD thesis, University of Copenhagen, Dept. of Computer Science (1995)
7. Parra-Hernandez, R., Dimopoulos, N.J.: A new heuristic for solving the multichoice multidimensional knapsack problem. IEEE Transactions on Systems, Man, and Cybernetics, Part A **35**(5) (2005) 708–717
8. Nemhauser, G.L., Wolsey, L.A.: Integer and combinatorial optimization. Wiley-Interscience, New York, NY, USA (1988)
9. OASIS: Web services business process execution language (April 2007) `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`.
10. Zhou, C., Chia, L.T., Lee, B.S.: Daml-qos ontology for web services. icws **0** (2004) 472

11. Bilgin, A.S., Singh, M.P.: A daml-based repository for qos-aware semantic web service selection. icws **0** (2004) 368
12. Aurrecoechea, C., Campbell, A.T., Hauw, L.: A survey of qos architectures. Multimedia Systems **6**(3) (1998) 138–151
13. Gillmann, M., Weikum, G., Wonner, W.: Workflow management with service quality guarantees. In: SIGMOD Conference. (2002) 228–239
14. Cui, Y., Nahrstedt, K.: Supporting qos for ubiquitous multimedia service delivery. In: ACM Multimedia. (2001) 461–462
15. Casati, F., Shan, M.C.: Dynamic and adaptive composition of e-services. Inf. Syst **26**(3) (2001) 143–163
16. Issa, H., Assi, C., Debbabi, M.: Qos-aware middleware for web services composition - a qualitative approach. In Bellavista, P., Chen, C.M., Corradi, A., Daneshmand, M., eds.: Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC 2006), 26-29 June 2006, Cagliari, Sardinia, Italy, "IEEE Computer Society (2006) 359–364
17. Liu, Y., Ngu, A.H.H., Zeng, L.: Qos computation and policing in dynamic web service selection. In: WWW. (2004) 66–73
18. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. TWEB **1**(1) (2007)
19. Maros, I.: Computational Techniques of the Simplex Method. Springer (2003)
20. Khan, S., Li, K.F., Manning, E.G., Akbar, M.M.: Solving the knapsack problem for adaptive multimedia systems. Stud. Inform. Univ. **2**(1) (2002) 157–178
21. Li, F., Yang, F., Shuang, K., Su, S.: Q-peer: A decentralized qos registry architecture for web services. In: ICSOC. (2007) 145–156
22. Yoon, K..P., Hwang, C.L.: Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences. Sage Publications (1995)
23. Michel Berkelaar, Kjell Eikland, P.N.: Open source (mixed-integer) linear programming system. Sourceforge `http://lpsolve.sourceforge.net/`.