A Scalable Approach for QoS-based Web Service Selection

Mohammad Alrifai[1], Thomas Risse[1], Peter Dolog[2], and Wolfgang Nejdl[1]

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

[2] Department of Computer Science
Aalborg University, Denmark
dolog@cs.aau.dk

Abstract. QoS-based service selection aims at finding the best component services that satisfy the end-to-end quality requirements. The problem can be modeled as a multi-dimension multi-choice 0-1 knapsack problem, which is known as NP-hard. Recently published solutions propose using linear programming techniques to solve the problem. However, the poor scalability of linear program solving methods restricts their applicability to small-size problems and renders them inappropriate for dynamic applications with run-time requirements. In this paper, we address this problem and propose a scalable QoS computation approach based on a heuristic algorithm, which decomposes the optimization problem into small sub-problems that can be solved more efficiently than the original problem. Experimental evaluations show that near-to-optimal solutions can be found using our algorithm much faster than using linear programming methods.

1 Introduction

Industrial practice witnesses a growing interest in the ad-hoc model for service composition in the areas of supply chain management, accounting, finances, eScience as well as in multimedia applications. With the growing number of available services 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.

Given an abstract representation of a composition request (e.g. in a workflow language like BPEL [1]), and given a list of functionally-equivalent web service candidates for each task in the composition request, the goal of service selection algorithms is to find one web service from each list such that the overall QoS is optimized and user's end-to-end QoS requirements are satisfied. This problem can be modeled as *Multi-Choice Multidimensional Knapsack problem* (MMKP), which is known to be NP-hard in the strong sense [2]. Therefore it can be expected that any exact solution to MMKP has an exponential cost. In the dynamic environment of web services, 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 service selection algorithm becomes crucial.

Due to the poor scalability of *(Mixed) Integer Linear programming* (MILP) methods [3], recently proposed solutions like [4, 5] fail short in addressing run-time requirements. In this paper we propose an efficient and scalable heuristic approach for the QoS-based service selection problem. The contribution of this paper can be stated as follows:

- We map the global QoS optimization problem into sub-problems that can be solved more efficiently by local QoS optimization.
- We show how the results of the problem decomposition can be applied in a distributed architecture that includes a service composer and a set of distributed service brokers.

Our heuristic approach to QoS-based service selection does not necessarily result in “the” optimal set of services. Nevertheless, since the business requirements (such as response times or throughput) are only approximate, we need to find a reasonable set of services that covers the requirements approximately at acceptable costs and avoids obvious violations of constraints. The experimental evaluation we present in this paper show that our approach outperforms all previous solutions in terms of computational complexity but still gives qualitative comparable results.

The rest of the paper is organized as follows. In the next section we discuss related solutions. Section 3 introduces the system model and gives a problem statement. Our approach for a scalable QoS computation for web service selection is presented in section 4. In section 5 we discuss the results from our experimental evaluation. Finally, section 6 gives conclusions and an outlook on possible continuations of our work.

2 Related Work

Recently, the QoS-based web service selection and composition in service-oriented applications has gained the attention of many researchers [4, 6, 5, 7]. In [6] 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. [4] 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 [3] to find the optimal selection of component services. Similar to this approach Ardagna et al. [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 [8]. In [7] the authors propose heuristic algorithms to find a near-to-optimal solution more efficiently than exact solutions. The time complexity of the heuristic algorithm (WS_HEU) is polynomial. Despite the significant improvement of this algorithm compared to exact solutions, it dose not scale with respect to an increasing number of web services and remain out of the real-time requirements.

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 [6, 9]. 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) [6]. 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 [4], 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 [4, 6, 5, 7], we assume in our model that QoS aggregation functions can be linearized and represented by the summation relation. For QoS attributes that are typically aggregated as a product (e.g. availability) we apply a logarithm operation to transform them into a summation relation. 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 [10]. 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:

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

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 bounding each S_j to a concrete service $s_{jb} \in S_j$ such that:

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

4 A Scalable QoS Computation

In this section we present a scalable solution to the QoS-bases web service composition problem. We decompose the global optimization problem into sub-problems that can be solved independently. For this purpose, we first map the global QoS computation on the composite service level $U'(CS)$ into local computations that can be performed on each service class independently (see Section 4.1). Second, we propose a simple algorithm for decomposing each global QoS constraint $c'_k \in C'$ into n local constraints that can be verified locally on the component services (see Section 4.2). Finally, we present a distributed service selection algorithm that leverages local search for achieving global QoS requirments (see Section 4.3)

4.1 Decomposition of Global QoS Computation

The use of (3) on the composite service level requires enumerating all possible combinations of the service candidates to find the optimal selection. This approach can be very inefficient for large scale problems or applications with run-time requirements. Therefore, we derive a modified utility function $U'_{local}(s)$ from $U'(CS)$ that can be applied on the component service level, without the need for evaluating all possible combinations.

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

$$U'(CS) = \sum_{k=1}^{r} \frac{\sum_{j=1}^{n} Qmin(j,k) - \sum_{j=1}^{n} q_k(s_j)}{Qmax'(k) - Qmin'(k)} \cdot w_k$$
$$= \sum_{j=1}^{n} \underbrace{(\sum_{k=1}^{r} \frac{Qmax(j,k) - q_k(s_j)}{Qmax'(k) - Qmin'(k)} \cdot w_k)}_{U'_{local}(s)} = \sum_{j=1}^{n} U'_{local}(s_j) \qquad (4)$$

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 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 first requirement in 3.5).

4.2 Decomposition of Global Constraints

To ensure that the outcome of the local QoS computation satisfies global QoS constraints (i.e. the second requirement in 3.5), we need to decompose each global constraint $c'_k \in C', 1 \leq k \leq m$ into n local constraints. We use local statistics about the quality values to estimate a reasonable decomposition of each global constraint c'_k as follows:

1. Initially set the local constraint value c_{jk} of each service class to the local maximum value of that class,

$$\forall c'_k \in C' : c_{kj} = Qmax(j,k), 1 \leq j \leq n \qquad (5)$$

2. Compute the difference between the global constraint values and the aggregated value of the local constraints,

$$\forall c'_k \in C' : d_k = \sum_{j=1}^{n} c_{kj} - c'_k \qquad (6)$$

3. Adjust the current set of local constraints based on the relative distance between the local maximum and minimum QoS value using the following formula:

$$\forall c'_k \in C' : c_{kj} = c_{kj} + d_k * \frac{Qmax(j,k) - Qmin(j,k)}{\sum_{x=1}^{n}(Qmax(x,k) - Qmin(x,k))}, 1 \leq j \leq n \qquad (7)$$

4.3 Distributed Optimization of the QoS Computation

We assume an architecture consisting of a *service composer* and a number of *service brokers* - either distributed or on a single machine. Each service broker is responsible for managing QoS information of a set of web service classes. A list of available web services is maintained by the service broker along with registered measurements of their non-functional properties, i.e. QoS attributes, like response time, throughput, price etc. The service composer instantiates a composite service CS in interaction with the service brokers.

The procedure of the distributed QoS-based service composition is depicted in figure **??**. The service composer requests statistical information for each service class from the responsible service brokers, namely, $Qmax(j,k)$ and $Qmin(j,k)$ and computes the global parameters: $Qmax'(k)$, $Qmin'(k)$, $1 \leq k \leq r$. Each global constraint $c'_k, 1 \leq k \leq m$ is decomposed as described in section 4.2 into a set of local constraints $c_{1k}, ..., c_{jk}$. The composer sends these local constraints along with the global parameters $Qmax'(k)$, $Qmin'(k)$, $1 \leq k \leq r$ to each service broker. Each service broker performs a local search and returns the best service candidate that satisfies the local constraints and has the maximum U'_{local} value. The service composer collects the results from the brokers and checks them for further optimization. Further optimization is possible if the total saving in the value of any of quality attributes is greater than zero. The total saving δ_k of the k-th QoS attribute is computed as:

$$\delta_k = \sum_{j=1}^{n} (c_{jk} - q_{jk}), 1 \leq k \leq m \tag{8}$$

The service composer uses the total saving in the quality value to relax the current local constraints as follows:

$$\forall c_{kj} \in C' : c_{kj} = q_{kj} + \delta_k * \frac{Qmax(j,k) - q_{jk}}{\sum_{x=1}^{n}(Qmax(x,k) - q_{xk})}, 1 \leq j \leq n \tag{9}$$

The relaxed local constraints are sent back to the service brokers for improving their local results. The procedure is repeated as long as a new solution is found. Otherwise, the procedure stops and the final composition is constructed from the currently select component services $CS = \{s_1, \ldots, s_n\}$.

Unlike the heuristic algorithm WS_HEU [7], the expected number of iterations in our algorithm is very low as we only consider upgrading the current solution by means of relaxing the constraints and no downgrading is required. By only relaxing the local constraints it is guaranteed that the new solution, if any exists, has a higher overall utility value than the current solution. Our algorithm is guaranteed to converge as after each round the service composer checks the new solution as well as the new total saving value δ_k against those of the previous round. The composer stops the optimization process as soon as there is no improvement in the utility value nor saving in the quality values.

5 Experimental Evaluation

To evaluate our proposed solution we conducted several experiments, which we describe in this section. 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.

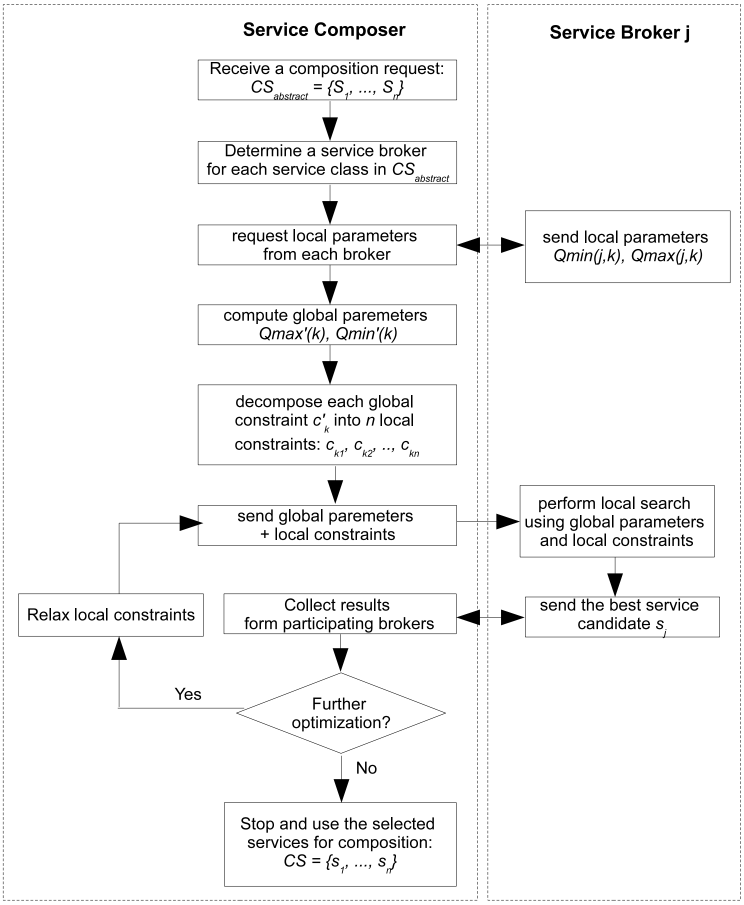
Service Composer
Service Broker j
Receive a composition request: $CS_{abstract} = \{S_1, ..., S_n\}$
Determine a service broker for each service class in $CS_{abstract}$
request local parameters from each broker
send local parameters $Qmin(j,k)$, $Qmax(j,k)$
compute global paremeters $Qmax'(k)$, $Qmin'(k)$
decompose each global constraint c'_k into n local constraints: $c_{k1}, c_{k2}, .., c_{kn}$
send global paremeters + local constraints
perform local search using global parameters and local constraints
Relax local constraints
Collect results form participating brokers
send the best service candidate s_j
Yes
Further optimization?
No
Stop and use the selected services for composition: $CS = \{s_1, ..., s_n\}$

Fig. 1. Service Composer - Service Broker interactions

In our evaluation we compare the performance and the quality of the results of our solution with those of the linear programming methods (LP) [4, 5] and the heuristic algorithm WS_HEU [7]. For LP we use the open source Linear Programming system *lpsolve* version 5.5 [11]. For WS_HEU we use our own implementation. We implemented it as fair as possible taking into account all possible optimizations to reduce the computation time as much as possible. We experimented with several instances of the QoS composition problem by varying the number of service classes n and the number of service candidates per class l. Each unique combination of these parameters represents one instance of the composition problem.

For the QoS data we use the QWS real dataset from [12]. This dataset includes measurements of 9 QoS attributes for 364 real web services. To evaluate the scalability of our solution, however, we need to run experiments with a much larger set of services. Therefore, we duplicated the QWS dataset several times, each time multiplying the quality values of web services by a uniformly distributed random value between 0.1 and 2.0. In this way we achieved a data set of about 100.000 services.

5.1 Performance Evaluation

We evaluate the performance of the three approaches, by measuring the required time for finding the solution (i.e. the best combination of concrete services) by each approach. Figure 2 shows a comparison of the performance of LP, WS_HEU and our solution, which we label with DIST_HEU. In this experiment we study the performance of the three approaches with respect to the size of the problem in terms of the number of service classes n and the number of service candidates per class l. In the graph shown to the left of figure 2 we fix the number of service class n to 20 and vary the number of service candidates l from 100 to 1000 per class. In the right part of the graph the number of service candidates l is fixed to 100 and the number of service classes n varied from 10 to 100. The number of global constraints m in both cases is fixed to 3. The results in both graphs show that DIST_HEU has a much better scalability than LP and WS_HEU in all problem instances (always less than 100 msec).

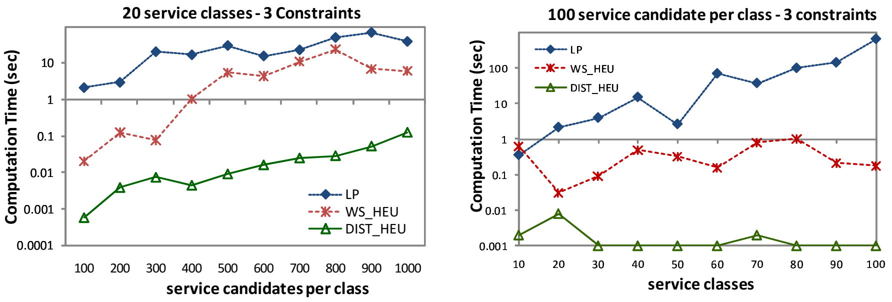

Fig. 2. Computational time with respect to the problem size (Logarithmic scale is used)

5.2 Optimality Evaluation

To evaluate the quality of the results of our approach, we measure the closeness of the returned results to the optimal results obtained by the LP method by calculating the optimality ration $R = \frac{U_{approx}}{U_{opt}}$. U_{approx} is the utility of the best composition returned by our approach according to (3) and U_{opt} is the utility of the composition returned by the LP method. The results shown in figure 3 indicate that DIST_HEU achieves good results with 98% optimality ratio in average. It can also be seen that result quality of our approach DIST_HEU is in average just 1% below the WS_HEU results. However, the cost of WS_HEU for this little improvement in terms of computation time is very high as we see from figure 2.

6 Conclusion and Future Work

This paper describes a scalable method for the QoS-based service selection. The problem is known to be NP-hard. Therefore heuristic solutions are commonly used, which achieve a close to optimal result at the cost of a polynomial complexity. Our proposed method allows to dramatically reduce the efforts compared to existing solutions by a

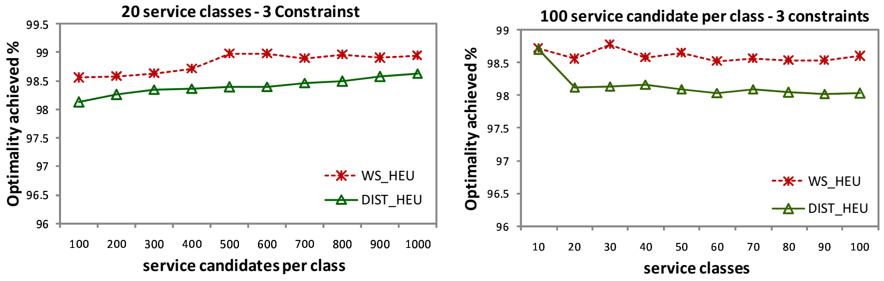

Fig. 3. Comparison of the achieved optimality with respect to the size of the problem

factor of 10 to 100 dependent on the complexity of the service environment. We decompose the global optimization problem into a number of sub-problems that can easily be solved by local search within each service class. In addition the decomposition allows distributing the processing to better utilize the available processing power while increasing the availability of service information within a service infrastructure. Overall our approach is based on a very limited set of architectural requirements. Therefore we believe that it can be easily integrated in a wide range service oriented infrastructures. We are currently working on extending our approach to support more complex composition models than the sequential model. Furthermore, we plan to develop a more sophisticated QoS constraint decomposition algorithm to improve the optimality of the obtained results.

References

1. OASIS: Web services business process execution language (April 2007) `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`.
2. Pisinger, D.: Algorithms for Knapsack Problems. PhD thesis, University of Copenhagen, Dept. of Computer Science (1995)
3. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley-Interscience, New York, NY, USA (1988)
4. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW. (2003) 411–421
5. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Trans. Software Eng. **33**(6) (2007) 369–384
6. Liu, Y., Ngu, A.H.H., Zeng, L.: Qos computation and policing in dynamic web service selection. In: WWW. (2004) 66–73
7. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web **1**(1) (2007) 6
8. Maros, I.: Computational Techniques of the Simplex Method. Springer (2003)
9. Li, F., Yang, F., Shuang, K., Su, S.: Q-peer: A decentralized qos registry architecture for web services. In: ICSOC. (2007) 145–156
10. Yoon, K..P., Hwang, C.L.: Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences. Sage Publications (1995)
11. Michel Berkelaar, Kjell Eikland, P.N.: Open source (mixed-integer) linear programming system. Sourceforge `http://lpsolve.sourceforge.net/`.
12. Al-Masri, E., Mahmoud, Q.H.: Investigating web services on the world wide web. In: WWW. (2008)