QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms

Michael C. Jaeger, Gero Mühl, and Sebastian Golze

Techn. Universität Berlin, Institute of Telecommunication Systems,
Sek. FR6-10, Franklinstrasse 28/29, D-10587 Berlin, Germany
{golze, gmuehl}@ivs.tu-berlin.de, mcj@cs.tu-berlin.de

Abstract. A composition arranges available services resulting in a defined flow of executions. Before the composition is carried out, a discovery service identifies candidate services. Then, a selection process chooses the optimal candidates. This paper discusses how the selection can consider different Quality-of-Service (QoS) categories as selection criteria to select the most suitable candidates for the composition. If more than one category is used for optimisation, a multi-dimensional optimisation problem arises which results in an exponential computation effort for computing an optimal solution. We explain the problem and point out similarities to other combinatorial problems – the knapsack problem and the resource constraint project scheduling problem (RCPSP). Based on this discussion, we describe possible heuristics for these problems and evaluate their efficiency when used for web service candidate selection.

1 Introduction

A Web service composition is a collection of single Web services that form a new, more complex service. The creation process of a composition can be divided into several phases. In the first phase, the control flow – the execution arrangement of individual tasks – is defined. The idea is that available Web services realise individual tasks needed for the composition. Different so called *flow languages* (e.g. BPEL4WS) are already available to describe such a flow. Based on the flow description, a discovery service identifies suitable task candidates. Then, a selection process chooses the optimal candidate for each task. After the assignment of available Web services to the tasks has been fixed and saved in a flow description, an execution engine uses the flow description to execute the composition. An execution engine can also provide the composition as a new *composed* Web service that is ready for invocation by third parties.

In the Web service domain, an open initiative hosted by the OASIS group proposes a specification for the service discovery called *Universal Description, Discovery and Integration*, in short UDDI [13]. The UDDI approach is part of the Web Services Architecture promoted by the W3C [2]. Another, more technology independent view on a service architecture is given by the ISO Reference Model for Open Distributed Processing (RM-ODP). In the RM-ODP, a so called *trader* facilitates the discovery and the selection process [7]. A trader matches requirements of service requesters to

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3760, pp. 646–661, 2005.

services that are offered by service providers. First, functional requirements are the relevant criteria for a matchmaking process in the service discovery which results in a set of candidates. In addition, non functional requirements represent preference criteria. A selection process can involve such criteria to identify the optimal service.

According to this separation proposed by the RM-ODP, this paper will explain how a trader *selects* Web services by different optimisation criteria for compositions comprised by more than one Web services. Thus, it is presumed that a preceding discovery process has identified a set of candidates that matches the functional requirements for the referring tasks. If a trader has to select an individual Web service from the set of candidates, a trading function identifies the most suitable service by comparing the discovered candidates. However, in a composition a trader should identify the optimal set of Web services which are about to be combined. Selecting for each tasks the best candidate in isolation will in most cases not lead to an optimal solution. This paper explains why this problem can be regarded as a combinatorial problem and discusses possible solutions. For our discussion, we consider different QoS criteria that represent non-functional characteristics that we can quantify using different metrics. First, we give an overview about QoS for Web services and explain how the QoS of individual services can be aggregated in compositions. After the problem description and the introduction of possible solutions, the related work is discussed. The paper ends with our conclusions and future plans in this field of research.

2 QoS in Web Service Compositions

In this paper, we use different QoS categories to define requirements, which serve as selection criteria for available services. Each QoS category has an increasing or a decreasing direction. An *increasing* direction means that a higher value indicates a better quality, for *decreasing* categories vice versa. A set of QoS categories has been already introduced by various authors for the use in workflows by Cardoso [3] or in the Web service domain by Zeng et al. [17], or Menasce [10]. From these categories we have chosen the following four to give a more concrete discussion and examples in the remainder of this paper. Please note that our methods and algorithms will work also for other QoS categories:

Maximal Execution Time (Decreasing). The execution time defines the amount of time to execute the service. Different definitions are possible, which may include different phases of the invocation of the Web service. In this paper it is presumed, that covering the values of individual services will result in the overall execution time of the composition. In other words, delays or interrupts of the control flow are ignored.

Cost (Decreasing). The cost defines generally the amount of resources needed to use a service. Like execution time this measure is decreasing meaning that a lower value is preferred.

Reputation (Increasing). The concept of a reputation is basically about a ranking given by users of the service. For example, the auction platform eBay allows clients to rank the behaviour of other clients [17]. The reputation is defined as the average of the individual ranks of users.

Availability (Increasing). The availability denotes the probability that the execution of the node performs successfully and is regarded as an increasing dimension.

For the description the QoS of Web services different proposals already exist. Tosic et al. have proposed the Web Service Offerings Language (WSOL) [12] which covers among other issues also the definition of QoS statements covering a Web service. WSOL represents an XML-based language and has the focus on specifying the non-functional aspects of Web service. WSOL directly builds upon a WSDL description. Considering the WSOL as well as other languages, we need to point out that the definition of the used QoS concepts must be given individually. For example, WSOL covers this issue with an external reference to a common definition of the particular QoS category.

2.1 QoS Aggregation

In order to select candidates for tasks of a composition that is based QoS of the individual services, a method is needed to calculate the resulting QoS of the whole composition. In a preceding paper we have introduced an aggregation method to calculate the QoS of the composition based on the QoS of the individual services [8]. The model identifies seven basic structural elements called *composition patterns*. These structural elements were derived from a set of workflow patterns by van der Aalst et al. [15]. Workflow patterns form a set of functional and structural requirements for workflow management systems.

Among different reasons, we have chosen to use the workflow patterns because van der Aalst has shown that the structural part of the workflow patterns also applies to commonly known flow languages for compositions [14]. Thus, we can assume that our elements cover these flow languages as well. Since the scope of this paper does not allow to explain which of the workflow patterns we have considered as a composition element, we would like to refer the reader to the according analysis in our preceding paper [8]. From this analysis, we identified the following composition patterns:

Sequence of service executions. A sequence can either prescribe a specific order in which the services have to be executed or the services can be executed in an arbitrary order. For the aggregation model, the order of the executions is not relevant.

Loop. The execution of a service or a composition of services is repeated for a certain amount of times.

XOR split followed by an XOR join. From a parallel arrangement only one task is started and the synchronising operation waits for this started task.

AND split followed by an AND join. From a parallel arrangement all tasks are started, and all tasks are required to finish for synchronisation.

AND split followed by a m-out-of-n join. From a parallel arrangement all n tasks are started, but $m < n$ tasks are required to finish for synchronisation.

OR split followed by OR join. From a parallel arrangement a subset of the available tasks are started, and all of the started tasks are required to finish for synchronisation.

OR split followed by a m-out-of-n join. From a parallel arrangement a subset of n tasks from the available are started, and $m < n$ tasks are required to finish for synchronisation.

Using these elements, we aggregate the QoS for each category and for each pattern element. Figure 1 shows the pattern-wise aggregation of a simple composition example. To aggregate the maximum execution time, an algorithm would start to determine the largest value in the parallel sub-arrangement. Then, the sum of the sequential arrangement including the aggregated value of the parallel arrangement is calculated. This approach enables us to view the aggregation in a pattern-perspective, i.e. not the whole composition is relevant at once, but rather the local pattern elements. Following this approach, we have defined aggregation rules for each QoS category and for each composition element. Of course, further, more specific patterns are possible. The intention of the patterns is to deliver a model that allows to define a QoS statement that covers the composition. Such a statement will always represent an approximation of the delivered QoS during run time. Our model represents also an approximation and is subject to the trade-off between complexity and feasibility. We have described the patterns and the rules for different QoS categories in detail and compared them to other approaches in our preceding papers [8] [9].

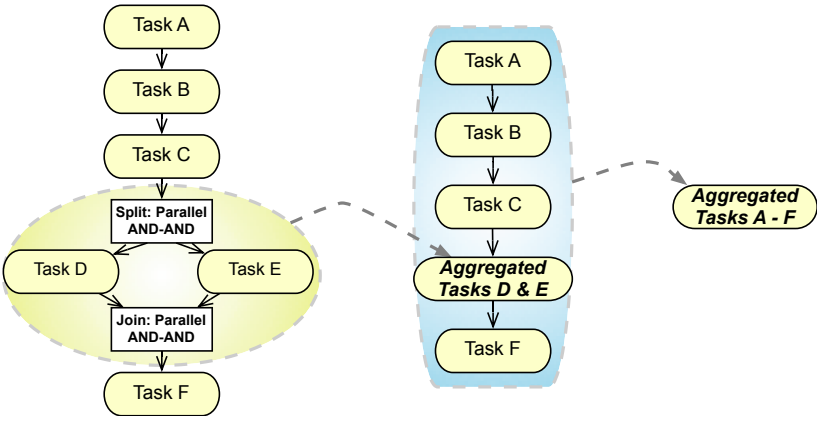

Fig. 1. Collapsing the Graph for Aggregation Example

2.2 The Selection of Services

The goal of the selection is to identify the best assignment of service candidates to the tasks of the composition. The selection can be performed by either considering or ignoring the arrangement of the tasks. When the structure is ignored, the selection can be formulated based on the following model:

Let the composition be a set of n tasks $\mathbb{T} = \{t_1, t_2, \ldots, t_n\}$. The output of the previously performed discovery process is a set of m candidates $\mathbb{S} = \{\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_m\}$.

Each of the p QoS categories is assigned a unique number $1, \ldots, p$. Then, each candidate can be expressed by a vector – a *QoS-vector* – with values $s_y, \quad y \in \{1, \ldots, p\}$ that represent the QoS categories that have been taken into account:

$$\overline{s}_x = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_p \end{pmatrix}$$

with $x \in \{1, \ldots, m\}$ denoting the number of the according candidate. The selection process assigns one candidate to each of the tasks. Optionally, a selection could also identify two or more candidates for each task to increase the dependability of the tasks by executing all these candidates. However, we ignore this specific aspect for our discussion.

To find the optimal assignment, an algorithm must evaluate different combinations from a global perspective, i.e. the assignment cannot take place by considering each task and its candidates separately. A simple example clarifies the problem: consider the parallel arrangement of the two tasks D and E shown in Figure 1. Let the optimisation goal be to identify the composition which results in the fastest execution while trying to keep the lowest price. Also, a faster service is usually more costly – in other words, cost and execution time form a trade-off couple. In the case that even the fastest candidate for task D executes longer than any of the candidates for task E, the optimal assignment for the task E is the cheapest candidate, regardless how long its execution would take.

Thus for parallel arrangements, all combinations of assignments must be tested to find the optimal assignment. If a composition contains a parallel arrangement at the top level, all combinations for assigning candidates to the included tasks must be tested. The resulting effort increases exponentially with the number of tasks if for each tasks more than one candidate could be chosen: if the number of candidates increases by one, then the number of combinations to evaluate is doubled. If the number of tasks increases by one, the number of combinations increases multiplied by the number of candidates found for this task. Thus, for large numbers of tasks and candidates a global approach for the selection is not feasible and efficient heuristics are desired.

2.3 The Selection Criteria

The selection of candidates must be performed by applying some criteria. For the selection of candidates, particular QoS categories can be the basis for defining an optimisation function or optimisation constraints:

- One or more QoS categories are relevant for optimisation. Thus, for each considered QoS category the referring value of the QoS vector is subject to an optimisation function. This function depends on the direction of the dimension. For categories with a decreasing dimension (such as execution time or cost) the function is about to minimise the aggregated value, for categories with an increasing dimension it is about to maximise the value. The optimisation criterion is about to find the optimal – minimal or maximal – resulting QoS value aggregated by function f:

$$\{min|max\}\big(f(\mathbb{S}_y)\big) \quad \mathbb{S}_y = \{s_{1_y}p_1, \ldots, s_{b_y}p_b\}$$

$$\text{with } p_x = \begin{cases} 1 \text{ if selected, and} \\ 0 \text{ otherwise.} \end{cases}$$

The index $y \in \{1, \ldots, q\}$ refers to the considered QoS category.

– One or more QoS categories are relevant for expressing a constraint on the composition. Depending on the direction of the considered QoS category with index y, the constraint denotes an upper or lower bound for the resulting aggregated value:

$$\{c_y > |c_y <\}\big(f(\mathbb{S}_y)\big) \quad \mathbb{S}_y = \{s_{1_y}q_1, \ldots, s_{m_y}q_m\}$$

$$\text{with } q_x = \begin{cases} 1 \text{ if selected, and} \\ 0 \text{ otherwise.} \end{cases}$$

3 Analogies to Knapsack and RCPSP

If exactly one QoS category is relevant for the optimisation, a selection algorithm must choose the candidate that offers the optimal referring value for each task. The effort for this operation is linear to the number of candidates and thus this specialisation of the selection problem can be regarded trivial. If more than one QoS category is relevant for optimisation or for expressing a constraint, the selection can be regarded as a combinatorial problem. This problem has similarities with the 0/1-Knapsack problem and to a specific kind of project scheduling problem (PSP). Our goal is to explain their differences to the selection problem and evaluate efficient (heuristic) solutions.

3.1 The 0/1-Knapsack Problem

The knapsack problem is about selecting a subset of available items for putting them into a knapsack. Each item has a specific weight, a specific value, and the knapsack has a limited weight capability. The problem is that the weight capability of the knapsack does not allow to take all items. Thus, a selection must be performed with the goal to identify the optimal subset which maximises the value while keeping the weight constraint. Transferred to the composition scenario, the knapsack problem could be formulated as follows:

- The composition represents the knapsack.
- Each candidate represents one item that can be put into the knapsack.
- A QoS constraint represents the limited weight capability of the knapsack.
- A QoS category which is subject to the optimisation represents the value of the item.
- The algorithm tries to find the optimal selection according to the optimisation function while keeping the constraint.
- Only ”complete” candidates can be selected, a candidate cannot be split to meet the constraint. The knapsack problem, which does not allow to split items, is also known as 0/1-knapsack problem.

However, some characteristics of the selection differ from the known knapsack problem. The selection algorithm must also find a solution that (1) optimises the value of a given QoS category, (2) keeps the constraint *and* (3) selects a candidate for each task. For a normal knapsack, the solution might result in as few items as possible to keep the constraint.

If more than one QoS category is relevant for optimisation, the relevant values of the QoS vector of a candidate could be aggregated to form a new measure representing the value of the candidate. Then, the *Simple Additive Weighting (SAW)* can be applied, which was introduced in the context of *Multiple Criteria Decision Making (MCDM)* [6]. By applying this procedure, we can normalise the individual values and assign a score to each QoS vector. As first step, each value s_{yx}, with $y \in \{1, \ldots, p\}$ indicating the QoS category and $x \in \{1, \ldots, m\}$ indicating a particular candidate is replaced by the normalised value n_{yx}:

$$n_{yx} = \begin{cases} \frac{max_y(s_{yx}) - s_{yx}}{max_y(s_{yx}) - min_y(s_{yx})} & \text{for decreasing categories} \\ \frac{s_{yx} - min_y(s_{yx})}{max_y(s_{yx}) - min_y(s_{yx})} & \text{for increasing categories} \end{cases}$$

The result of this replacement is that the value representing the best quality results in a value of 1 and the worst results in a value of 0. Other values will range between 0 and 1. Then, a score c_x can be applied to each candidate [6]:

$$c_x = \frac{1}{p} \sum_{y=1}^{p} w_y n_{yx}$$

The weight w_y is applied to the QoS categories by the user's preference. The sum of all weights must be equal to 1. The result of this procedure is a score for each QoS vector representing a Web service candidate of an aggregated statement. Considering the knapsack problem, the overall score represents the quotient of value and weight of an item, because the score combines increasing and decreasing categories.

3.2 The Project Scheduling Problem

If multiple QoS categories are subject to the optimisation or if they form a constraint, the selection problem is similar to a so called *Resource Constrained Project Scheduling Problem (RCPSP)*. A project scheduling problem occurs when resources (usually humans) must be distributed to jobs of a project. The most common optimisation goal of the basic RCPSP is to reduce the duration of the project while spending as few resources as possible. Two types of RCPSPs are distinguished: one is called *Single Mode RCPSP* and the other is known as *Multi-Mode RCPSP (MRCPSP)*. The single mode RCPSP only deals with fixed values for the duration and the cost of a task. In a MRCPSP, a job can be done by using different modes which vary in cost and duration. Thus, a MRCPSP is considered for our selection problem. In addition to its mode, a RCPSP can be classified by a couple of other characteristics. Based on an overview about RCPSPs by Yang et al. the selection problem seen as RCPSP has the following characteristics [16]:

Objective. Objectives are distinguished by being regular or irregular. *Regular* objectives do not interfere with the goal to minimise the duration of the project, while *irregular* objectives allow to follow another objective – for example to equalise the consumed resources among involved parties [16]. Applied to the selection problem, the RCPSP has a non-regular objective because depending on the considered QoS categories and the applied weight, a worse duration can be considered as a better solution if, for example, the cost is reduced accordingly. The objective can be defined as an optimisation function as given in section 2.3.

Precedence Relation. Two types of precedence relations are discussed in the literature. Either tasks can be started with a specified time window after a preceding task has finished or a succeeding task can start any time after the preceding task has finished. Regarding a composition of Web services, the common case is that a task is started immediately after the preceding task finished.

The constraint for the precedence in Web service compositions can be defined as follows: let α_{x+1} be the start time of a candidates $\overline{s}_{x+1} \in \mathbb{S}_i$ of task $i, i = 1, \ldots, n$, and ω_x the finish time of a candidate $\overline{s}_x \in \mathbb{S}_{i-1}$ for the preceding task $i-1$. Then, the precedence constraint is:

$$\alpha_{x+1} q_{x+1} \geq \omega_x q_x \quad \text{with } q_{x+1}, q_x \begin{cases} 1 & \text{if selected} \\ 0 & \text{otherwise} \end{cases}$$

Between two tasks a time delay might occur, because an execution environment cannot execute a task at exactly the same moment another task has finished. However, for defining the constraint this interval is irrelevant and thus ignored.

Preemption. If preemption is allowed, the execution of a task can be suspended in order to execute another task. This is useful if some resources are only available within a specific period of time and other tasks can be suspended then. Since the invocation of Web services in the context of a composition is usually an atomic operation, preemption is not considered to be possible.

Resource requirements per period. In the domain of project scheduling, using resources at different time periods can result in different costs. For example, performing a task at night results in higher payments for night shifts. For computer systems a similar idea might be applied: the execution of a service gets more expensive at peak hours. Since we are not aware of any example that applies this idea we ignore this aspect in the following.

Trade-offs. In the context of RCPSPs, the trade-off is characterised by two criteria, where an optimisation of the one means a change to the worse for the other. An algorithm must find an counterbalanced solution. A usual trade-off pair is formed by time and cost in which case the cost and the execution time should be kept as low as possible. For the selection problem possible trade-off couples can be formed by cost vs. one of the other three mentioned categories in the sense that a higher quality results in a higher cost. However, a trade-off couple could be also time vs. availability thinking about a provider, where services usually execute very quickly but may fail quite often.

4 Approaches for the Selection Problem

Building onto the analogies among the two combinatorial problems explained in the previous section, our approach is to apply heuristics for these problems that are known to be efficient to the selection problem. In this section, four approaches are explained and compared:

Greedy Selection. For a greedy selection, the score c_x represents the selection criteria. The algorithm starts with calculating the score for each candidate. Then, to each task the candidate with the highest value density is assigned. It should be noted that if this approach is used, it is not possible to consider a global constraint.

Discarding Subsets. This algorithm represents a backtracking approach. It starts by parsing a search tree which consists of nodes each representing a possible pair of candidate and task. Each level of the tree holds pairs of a particular task only, resulting in the tree having the same number of levels as tasks. Each possible combination of candidate assignments is represented by a particular path of the tree from the root to a leaf.

To lower efforts, the algorithm cut subregions, if (a) it can be determined that the constraint cannot be met anymore by following this particular subregion or if (b) we can guess that combinations represented by a particular subregion do not result in a better overall QoS than already found. If the algorithm finds an appropriate combination, it aggregates the overall QoS. The combination is stored if no combination has been identified so far resulting in a better QoS. The algorithm stops when all possible combinations have been evaluated. This approach will find a solution meeting the constraint, if it exists. However, it does not save any efforts in the worst case when worse branches cannot be identified at an early stage.

Since this approach normally identifies the optimal solution, it cannot be regarded a heuristic. We establish a cutting rule based on an estimation. Considering the execution time, the cutting rule is clear: if a complete combination has already been determined that shows a lower execution time as the partial combination processed at some moment, the algorithm cuts the subtree. Each additional candidate would worsen execution time. However, for categories where the aggregation calculates the arithmetic mean, a rule cannot determine whether the QoS gets worse or better. In our discussion, we consider the reputation as a QoS category which represents this case. Applied to the selection problem in our configuration, the discarding subsets algorithm guesses that the QoS gets worse and thus might ignore the optimal solution.

Bottom-Up Approximation. A large number of heuristics are already available for RCPSPs [16]. However, not every approach can be applied to the selection problem: RCPSPs and Web service compositions cover the execution order of tasks differently: for compositions the order is in most cases pre-defined in a flow description, while the tasks of a project are subject to precedence relations, which may allow to push a particular task for- or backwards in order to optimise the utilisation of resources. It turned out that several approaches for solving RCPSPs work

on a precedence model which does not allow the application to the selection problem. The resulting problem is that a solution space is created which considers the rearrangement of activities. The resulting bounding rules cannot be applied efficiently for the selection problem.

However, we have identified one heuristic covering a RCPSP introduced by Yang et al. which can be used for the selection [16]: The approach presumes that constraint and optimisation criteria form a trade-off couple, i.e. the quicker a task is performed the more it will cost. The heuristic applied to the selection would perform as follows:

1. The candidates are sorted by the QoS value s_{x_y}, with $x \in \{1, \ldots, m\}$ and $y \in \{1, \ldots, p\}$ denoting the QoS category for the constraint.
2. For each task, the candidate $\overline{s}_x$ with the best s_{x_y} is assigned. If a solution exists, it is found with this step.
3. As the next step, the algorithm replaces the firstly assigned candidates by the candidate with the next worse value s_{x_y}.
4. The new combination is tested for whether the constraint is still kept. If the constraint is still kept, the algorithm continues by looping back to step 3. The algorithm stops, if at one time for each task no additional candidate is found that lets the composition meeting the constraint and increases the overall QoS.

Pattern-wise Selection. In a preceding paper we have introduced another heuristic [4], which directly covers the example of tasks A and B explained in section 2.2. We apologise that we cannot go into much detail about the pattern-wise selection due to spatial limitations. Thus, we refer to the mentioned publication for more information about its motivation and how it works. In very brief words, the algorithm determines the best assignment considering each composition pattern in isolation. The algorithm takes advantage of already identified elements of composition patterns (cf. Section 2.1). It performs four steps:

1. The algorithm walks recursively into the structure and identifies pattern elements that do not contain any sub-patterns.
2. For all tasks within this element, all sets of candidate assignments are evaluated. The QoS is aggregated for each combination by using the pattern-based aggregation. The combination that delivers the best score using the SAW procedure is chosen.
3. If the optimal solution for a particular pattern is determined, the algorithm walks one level upwards to evaluate the assignment within the new pattern. The aggregated QoS of contained sub-patterns is taken as a fixed value.
4. The pattern wise optimisation and aggregation is performed until the whole composition is covered and one aggregated QoS is returned.

Since this algorithm operates on each pattern element, this approach cannot meet global constraints. Thus, the pattern-wise selection is only suitable for optimising the overall QoS.

4.1 Comparison

We have compared the four proposed and two additional selection methods using a simulation environment: the additional methods are $a)$ the *global* selection and the $b)$ *constraint optimised* selection. The global selection evaluates all possible combinations and determines the best QoS possible for the composition. In addition to the resulting best possible QoS, the algorithm shows the worst case computation effort. By the second method, the candidates are sorted by the QoS category relevant for the constraint. Then, for each task the candidate offering the best QoS constraint category is assigned. Thus, if a combination which respects the constraint exists, it is found using this approach. However, this algorithm does not optimise other QoS categories and thus results in a poor QoS for the overall composition. The software simulation performs the following steps:

1. Generation of an arbitrary test composition structure by randomly arranging the composition pattern elements which are introduced in Section 2.1.
2. Generation of candidate Web services each with random QoS values for execution time, cost, reputation, and availability.
3. Performing each of the selection methods on the same composition structure with the same set of candidates.

We have tested the algorithms with arbitrary compositions with an increasing number of tasks (from 4 to 12) but a fixed number of candidates for each task (5). The implementation of the software and the random generation of candidates and their QoS values involve a large number of issues, which cannot be discussed completely in this paper due to limited space. Some issues among them are:

- Each test case with a particular number of tasks has been repeated 50 times with each time a new random composition structures, new optimal QoS values for each task and new resulting QoS values of the candidates. The results shown are the arithmetic mean values of the 50 repetitions for one test case. For each task the same number of candidates is generated.
- The simulator generates random composition structures by choosing one of the seven elements with equal probability.
- For each task an optimal QoS is randomly set and the randomly generated QoS values of the candidates are within 0 and 100 percent worse compared to the optimal value. This ensures a realistic distribution of the QoS values among the candidates referring to one task.
- To form a trade-off couple between execution time and cost, the two are set as follows: the percentage a added to the optimal execution time is taken to calculate the percentage b added to the optimal cost with $a+b = 100$. Thus, the more optimal the execution time is, the worse will be the cost and vice versa.
- The constraint is determined to perform the constraint selection on the cost first. The aggregated cost for the composition is increased by 20% and then taken as the constraint that has to be met by the other selection methods.

The results of this simulation are shown in Figure 2 and 3. Please note that in both figures the discrete results are connected with interpolated lines for better visualisation only. Figure 2 shows the average resulting QoS for the composition performing

the selection methods for the generated test compositions relative to the global selection, which always finds the optimal solution. For example, a QoS ratio of $0,80$ means that this selection method has only gained 80% of the best QoS possible. The different aggregated QoS values resulting for a composition can be compared by applying the SAW approach to compute a normalised score (cf. section 3.1). As expected, the resulting QoS by using all other selection methods is worse than the global selection. The worst resulting QoS is delivered by the constraint selection, which does not optimise regarding the overall QoS.

The bottom-up method shows the next worse results. It results in a worse QoS than the greedy selection. However, the bottom-up method has met the given constraints while the greedy selection did not. Compared to the greedy selection, the pattern approach shows even better results. Among the methods which optimise the QoS while meeting a constraint, the discarding subsets methods shows the best resulting QoS. The results show that the average values seem to oscillate between a corridor. In fact, this corridor is quite small: for example for the difference between the highest and lowest average value of resulting QoS for the bottom-up method is lower than 0.02022 or less than 3% of the overall values. However, we need to admit that considering the wave-similar shape gives us the impression that future measurements should be repeated more often.

The discarding subsets method seems to increase its resulting QoS with a raising number of tasks whereas the greedy algorithms seem to get worse. This leads to the assumption that the discarding subsets algorithm copes better with larger/more complex structures. Vice versa, the declining performance of the greedy algorithm leads to the idea that this methods performs worse with more complex compositions.

Figure 3 shows the average execution times of the different selection methods for compositions with increasing number of tasks. Please note that the figure uses a logarithmic scale on its y-axis. The greedy selection and the bottom-up approximation show an almost linear increase of efforts with a larger number of tasks. Along with the simple constraint oriented selection, all three methods form a group of similar execution time behaviour: all calculations took about 1 millisecond or less covering compositions of from three to ten tasks. The bottom-up approach shows a break-out when testing for compositions with 4 and 6 tasks. Although the test computer has had a clean system installation, no connection to a network and the simulation application ran as the only user application at that time. Clearly, further activities must investigate this effect which might indicate an inefficiency of this algorithm depending on the number of tasks. But, since the measurements show that all results are still less than 1 millisecond, we regard the overall impact of this unexpected result as non-critical. The selection methods to determine the best QoS, discarding subsets and the pattern-wise selection form another group of similar execution time behaviour. All three methods scale exponentially, while the pattern-wise selection climbs slower with increasing number of tasks.

As explained in the previous section, it must be noted that the results from selecting the best QoS, the pattern-wise and the greedy selection do not necessarily meet the given constraint. The constraint limit for the cost is set to 20% worse than the optimal cost determined by the constraint-oriented selection. The three selection methods showed

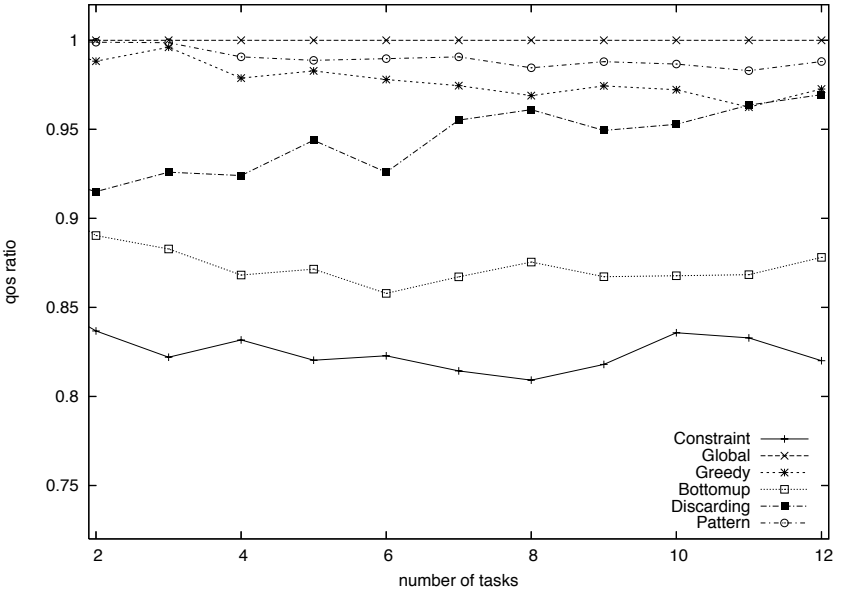

Fig. 2. Average Delivered QoS of Different Selection Methods Relative to Best QoS

together that the cost constraint is met incidentally by one third of the test cases with no noticeable relation to an increasing number of tasks. However, due to space limitations the results cannot be discussed in this paper.

5 Related Work

The work of Puschner and Schedel about calculating the execution time for software architectures represents our foundation for the aggregation of QoS in Web service compositions [11]. They have defined calculation rules for structural patterns as found in software executions. Since our composition model plays in the field of Web service compositions, we have adopted this principle and build our patterns onto the workflow patterns by van der Aalst et al. [15]. Cardoso has applied a similar approach in his work to calculate the QoS for workflows [3]. In his work he has identified different QoS criteria and defined calculation rules for these criteria in workflows based on the flow structures as found in the meteor workflow management system. Since our composition patterns are based on workflow patterns, we can also support structures like the two OR-split patterns along with the *m-out-of-n-join* constructs. As a consequence the composition patterns can be applied to more specific applications more easily.

Different authors have also discussed which QoS categories might be considered in Web service compositions [10] [17]. Their contribution has been taken up to determine the relevant categories for our work. If needed, the chosen categories can be extended without affecting the discussed approaches itself.

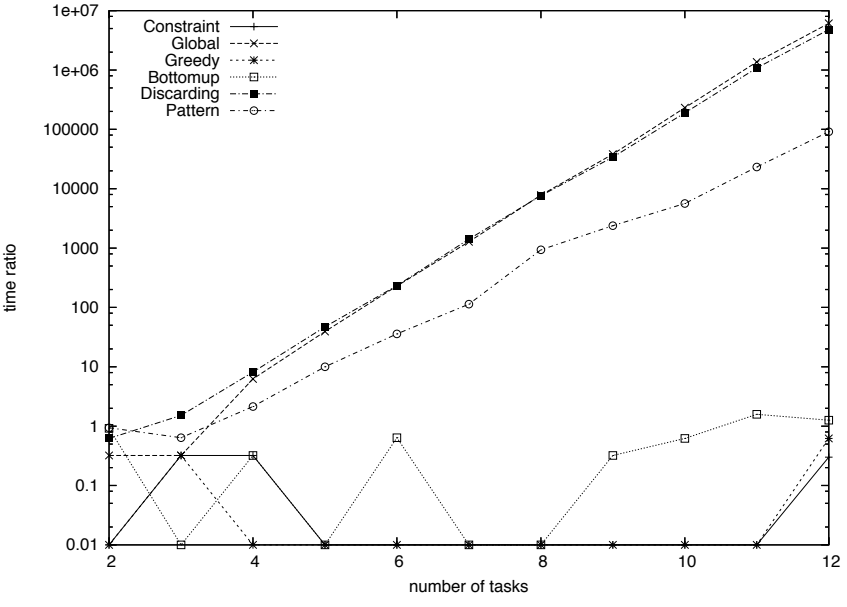

Fig. 3. Average Execution Times of Selection Methods

Using QoS statements as the main criteria for the selection of Web services is an already known idea, which has been introduced in fundamental papers at the time, when the Web services were emerging and the concept of Web service composition was presented [1] [5].

For the selection of candidates, Zeng et al. have identified two basic approaches: a local and a global selection [17]. To address the problem that a global selection has exponential effort, Zeng et al. have introduced an Integer Programming (IP) solution to compute the optimal assignment of services. Using the IP-approach reduces significantly the number of combinations by identifying constraints. According to their tests, the IP-based algorithm scales better with a growing number of candidates and tasks. We plan to test the IP-approach with our simulation tool to deliver a statement on how well it compares to pattern selection. Apart from the selection, Zeng et al. propose different aggregation mechanisms for their selection of QoS categories. We think that because of its uniform structure, the pattern-wise aggregation results in lower efforts for their implementation and the computation of the aggregation.

6 Conclusions

We have discussed different algorithms for performing the selection of candidates to optimise the overall QoS of a composition. Considering a computation algorithm, the used set of QoS categories can be extended or reduced depending on what is considered to be relevant for a specific application case. The intention of our work is to focus on the selection algorithms independent from the characteristics of different QoS categories.

For the optimisation *without* the need to meet a constraint, the results have shown, that a pattern-wise selection results in unfeasible efforts with a growing number of tasks. However, the pattern-wise selection performs still significantly quicker than the approach by discarding subsets. The pattern-wise selection also reaches almost the level of the best possible QoS. If a slight decrease of the overall QoS is tolerable (about 5% according to our results), the greedy selection delivers acceptable results with neglectable efforts. For the optimisation *with* the need to meet a global constraint our results have shown that the bottom-up approximation results in an overall QoS about 10% worse that for example the discarding subsets approach. However the computational effort of the bottom-up approximation remains feasible also with a growing number of candidates. The results show that for a selection in a time-critical scenario with a larger number of tasks, heuristics can be successfully applied with the penalty of a decrease on the overall QoS. Such an application scenario could be a reconfiguration of the composition during run-time when a Web service has become unavailable.

For future research in this direction possible test cases could operate on a fixed composition structure with an increasing number of candidates to evaluate the different algorithms with specific arrangements. Also the generation of the candidates and their QoS values could be improved in order to deliver more realistic example compositions. We also plan to evaluate other approaches such as genetic algorithms to solve the selection problem.

Acknowledgement

The authors would like to thank specifically the reviewer who did an excellent job by pointing out very good improvements and provided to us a very detailed list of suggestions.

References

1. Boualem Benatallah, Marlon Dumas, Marie-Christine Fauvet, and Fethi A. Rabhi. Towards Patterns of Web Services Composition. Technical Report UNSW-CSE-TR-0111, University of New South Wales, 2001.
2. David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. http://www.w3c.org/TR/ws-arch/, February 2004.
3. Jorge Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Department of Computer Science, University of Georgia, Athens, GA (USA), 2002.
4. Roy Gronmo and Michael C. Jaeger. Model-Driven Methodology for Building QoS-Optimised Web Service Compositions. In *Proceedings of the 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'05)*, pages 68–82, Athens, Greece, May 2005. Springer Press.
5. Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwan Su. E-Services: A Look Behind the Curtain. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'03)*, San Diego, USA, June 2003. ACM Press.

6. Ching-Lai Hwang and K. Paul Yoon, editors. *Multiple Attribute Decision Making: Methods and Applications*, volume 186 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, March 1981.
7. ISO/IEC. ITU.TS Recommendation X.950 — ISO/IEC 13235-1: Trading Function: Specification, August 1997.
8. Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. QoS Aggregation for Service Composition using Workflow Patterns. In *Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC'04)*, pages 149–159, Monterey, California, September 2004. IEEE Press.
9. Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. QoS Aggregation in Web Service Compositions. In *The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 181–185, Hong Kong, China, March 2005. IEEE Press.
10. Daniel A. Menasce. QoS Issues in Web Services. In *IEEE Internet Computing*, pages 72–75. IEEE Press, November-December 2002.
11. Peter Puschner and Anton Schedl. Computing Maximum Task Execution Times - A Graph-Based Approach. *Journal of Real-Time Systems*, 13(1):67–91, July 1997.
12. Vladimir Tosic, Kruti Patel, and Bernard Pagurek. WSOL – Web Service Offerings Language. In *Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web - WES (at CAiSE'02)*, volume 2512 of *Lecture Notes in Computer Science*, pages 57–67, Toronto, Canada, May 2002. Springer-Verlag.
13. UDDI Spec Technical Committee. UDDI Version 3.0.1. http://uddi.org/pubs/uddi-v3.0.1-20031014.pdf, 2003.
14. Wil M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *Jan/Feb 2003 Issue of IEEE Intelligent Systems*, pages 72–76, January 2003.
15. Wil M.P. van der Aalst and Arthur H.M. ter Hofstede and B. Kiepuszewski and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases 14(3)*, pages 5–51, 2003.
16. Bibo Yang, Joseph Geunes, and William J. O'Brien. Resource Constrained Project Scheduling; Past Work and New Directions. Technical Report Research Report 2001-6, Department of Industrial and Systems Engineering, University of Florida, 2001.
17. Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Transactions*, 30(5):311–327, May 2004.