Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, Ralf Steinmetz: Heuristics for QoS-aware Web Service Composition. In: 4th IEEE International Conference on Web Services (ICWS 2006), September 2006.

Heuristics for QoS-aware Web Service Composition

Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, Ralf Steinmetz
Dept. of Computer Science, Technische Universitaet Darmstadt, Germany
{berbner,spahn,repp,heckmann,steinmetz}@kom.tu-darmstadt.de

Abstract

This paper discusses the Quality of Service (QoS)-aware composition of Web Services. The work is based on the assumption that for each task in a workflow a set of alternative Web Services with similar functionality is available and that these Web Services have different QoS parameters and costs. This leads to the general optimization problem of how to select Web Services for each task so that the overall QoS and cost requirements of the composition are satisfied.

Current proposals use exact algorithms or complex heuristics (e.g. genetic algorithms) to solve this problem. An actual implementation of a workflow engine (like our WSQoSX architecture), however, has to be able to solve these optimization problems in real-time and under heavy load. Therefore, we present a heuristic that performs extremely well while providing excellent (almost optimal) solutions. Using simulations, we show that in most cases our heuristic is able to calculate solutions that come as close as 99% to the optimal solution while taking less than 2% of the time of a standard exact algorithm. Further, we also investigate how much and under which circumstances the solution obtained by our heuristic can be further improved by other heuristics.

1. Introduction

Web Services as technology based on open XML standards like SOAP, WSDL, and UDDI are widely used for integration purposes within enterprises. Beyond this, Web Services have the potential to be composed to cross-organizational workflows. Due to their loosely-coupled nature Web Services hosted by external providers can be integrated at runtime. This vision aims at dynamic ad-hoc collaborations between different business partners.

With the increasing number of Web Services with similar or identical functionality, the non-functional properties of a Web Service will become more and more important. Besides the costs for using Web Services, the Quality of Service (QoS) attributes (e.g. availability, response time, and throughput) are subsumed as non-functional attributes. Considering those non-functional properties is crucial for companies to meet the requirements of customers.

As a consequence, the QoS has to be explicitly managed at the designing phase of a Web Service composition as well as during its execution at runtime.

Focusing on the QoS-aware Web Service execution we have designed and implemented the proxy architecture WSQoSX (Web Services Quality of Service Architectural Extension) [2, 3] that supports late binding of Web Services at runtime as well as dedicated accounting and monitoring mechanisms (e.g. SLA violation detection).

In this paper, the focus is on QoS-aware Web service composition. We define QoS-aware Web Service composition as the selection of Web Services maximizing the QoS of the overall Web Service composition, taking into account preferences and constraints defined by the user. For this, a utility function maximizing the overall QoS subject to QoS constraints is introduced. This leads to an optimization problem that is NP-hard [5, 9]. Zeng et al. propose computing the optimal composition by linear integer programming (IP) [20]. However, their results reveal that this approach is hardly feasible in dynamic real-time scenarios with a large number of potential Web Services involved. This is exacerbated in a situation where the QoS-aware composition has to be replanned at runtime. In this case, the computation time of the composition becomes crucial. Replanning at runtime becomes necessary if Web Services selected at design time are not available anymore or SLA violations are detected. Obviously, customers are not willing to wait a few minutes when e.g. performing a financial transaction due to the replanning of the composition.

In this paper, we propose heuristics for solving the QoS-aware Web Service composition problem considering these timing constraints. We have designed, implemented and evaluated three heuristics: Heuristic H1_RELAX_IP consists of two steps: First, a MIP (Mixed Integer Programming) formulation of the composition problem is generated and its LP relaxation is solved. Second, we use a backtracking algorithm to create

a valid solution to the original, non-relaxed problem. Additionally, we study two further heuristics H2_SWAP and H3_SIM_ANNEAL, which are meta-strategies to improve the results of H1_RELAX_IP. We evaluate the performance of our heuristics compared with linear integer programming.

The rest of this paper is structured as follows: In Section 2 WSQoSX as foundation of our further research is introduced. The heuristics we have designed for computing QoS-aware Web Service compositions are discussed in Section 3. In Section 4 the focus is on the evaluation of our heuristics compared with linear integer programming. Related research is discussed in Section 5. This paper closes with a conclusion and a short outlook.

2. WSQoSX – A proxy architecture managing Web Service workflows

The execution of Web Service compositions needs mechanisms and architectural components that are beyond the traditional SoA approach (e.g. [8]). Thus, we have designed and implemented WSQoSX (Web Services Quality of Service Architectural Extension), a proxy-based architecture (Figure 1) that is able to

- manage Service Level Agreements (SLAs)
- detect SLA violations
- select a particular Web Service at runtime according to decision maker's preferences
- dynamically replace Web Services at runtime (e.g. due to a performance decrease of a specific Web Service)

For this, SLAs are modeled using IBM's WSLA (Web Service Level Agreement) [11]. Within a SLA the non-functional behavior of a Web Service (viz. costs and QoS attributes) is specified.

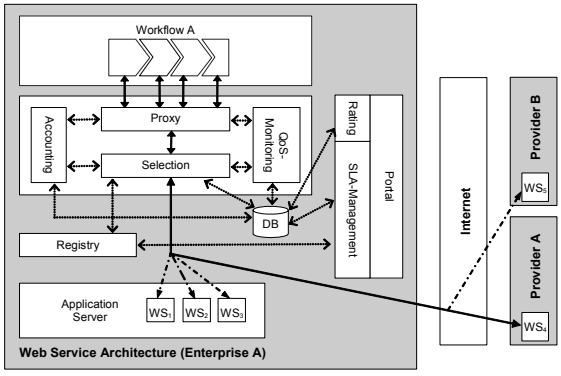

Figure 1. WSQoSX– architectural framework [3]

External Web Service providers have to register their Web Services and the corresponding SLAs at the WSQoSX *Web Portal* according to pre-defined categories (e.g. shipping, credit rating, and payment).

If a workflow managed by WSQoSX is started, the workflow engine does not invoke a Web Service directly. Web Service invocation is managed by a *Proxy Component* instead. This *Proxy Component* can determine which category (e.g. shipping) has been triggered for invocation and hands this information over to the *Selection Component*. The *Rating Component* calculates a score for each Web Services according to specific user preferences. Based on these calculations the *Selection Component* chooses and invokes the best suitable Web Service. The *Accounting Component* tracks detailed information about which Web Services have been invoked and their runtime behavior. This data is used by the *QoS Monitoring Component* to detect SLA violations during the execution of Web Services.

The management components of WSQoSX described above are implemented in Java. For further details about our work related to WSQoSX we refer to [2, 3].

3. Heuristics for QoS-aware composition

In this section, Web Service compositions are discussed and it is described how a QoS-aware Web Service composition can be modeled as an optimization problem. Furthermore, the heuristics we have designed for solving this optimization problem are described.

3.1. Web Service composition

Web Service composition aims at selecting and inter-connecting Web Services provided by different partners according to a business process [20]. Thus, Web Service compositions can be seen as workflows based on Web Services.

As depicted in Figure 2, there is a workflow model that consists of abstract tasks describing the required functionality (e.g. invoking a credit rating) of a specific workflow step. One of the main issues hereby is the selection of appropriate Web Services that form the execution plan for a Web Service composition. The functionality of each task can be provided by different candidate Web Services. Web Services that provide similar or identical functionality are grouped in the same category. Web Services within the same category may have different non-functional attributes.

Definition: A sequential Web Service composition consists of n tasks. Task i (i=1,...,n) will be executed before task i' (i'=1,...,n) if $i<i'$. The set of m_i different candidate Web Services that provide the required functionality for task i is called category i. A binary variable $x_{i,j}$ is introduced. $x_{i,j}$=1 means that Web Service j of category i is selected for being executed within the execution plan. To ensure that only one Web Service per task is selected, it is necessary that

$$\sum_{j=1}^{m_i} x_{i,j} = 1 \quad \forall i = 1,\ldots,n.$$

In this paper we assume sequential Web Service compositions.

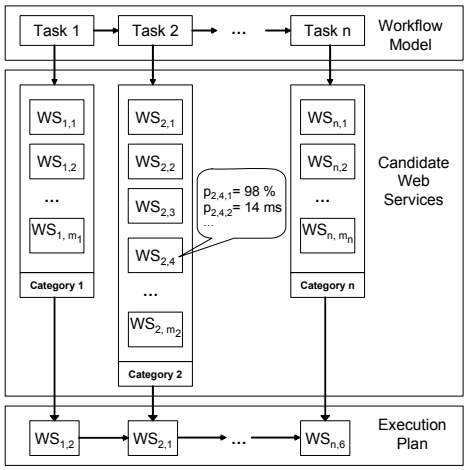

Figure 2. Web Service composition

3.2. QoS model

In the area of QoS-enabled Web Services a lot of QoS attributes (e.g. response time, availability, error rate, throughput, scalability, and reputation) have been addressed and evaluated (e.g. [6, 10, 13]).

To compute QoS attributes for the overall composition we need aggregation functions for each QoS parameter. Furthermore, the introduction of constraints is considered, e.g. restricting the overall response time to be less than 50s. An execution plan is only valid if it satisfies all constraints defined by the user. We define QoS-aware Web Service composition as the construction of an execution plan for a Web Service workflow so that the utility provided by the QoS attributes of the composition is maximized subject to constraints defined by the user.

$P_{i,j,k}$ stores the value of the QoS parameter with index k of Web Service j in category i. The aggregation of QoS parameters to an overall QoS attribute depends on the type of the QoS parameter k. An overall QoS attribute, like e.g. the overall response time aggregates additive whereas the overall availability aggregates multiplicative and the overall throughput is determined by the Min-operator.

In our model, three different kinds of parameters are considered:

– Additive parameters:

The overall QoS attribute x^+ of an additive non-functional parameter $p^+_{i,j}$ (e.g. response time) is calculated as $x^+ = \sum_{i=1}^{n}\sum_{j=1}^{m_i} p^+_{i,j} x_{i,j}$.

Constraints are defined as $x^+ \le c^+$ or $x^+ \ge c^+$.

– Multiplicative parameters:

The overall QoS attribute $x^\bullet$ of a multiplicative non-functional parameter $p^\bullet_{i,j}$ (e.g. availability) is calculated as $x^\bullet = \prod_{i=1}^{n}\sum_{j=1}^{m_i} p^\bullet_{i,j} x_{i,j}$. To be able to integrate constraints on $x^\bullet$ into our linear model, we linearize the term by applying the logarithm:

$$\ln(x^\bullet) = \ln\left(\prod_{i=1}^{n}\prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}}\right) = \sum_{i=1}^{n}\left(\ln\left(\prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}}\right)\right)$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{m_i}\ln\left(p_{i,j}^{\bullet\, x_{i,j}}\right) = \sum_{i=1}^{n}\sum_{j=1}^{m_i} x_{i,j}\ln\left(p^\bullet_{i,j}\right)$$

Using this, constraints can be defined as follows:

$$\ln(c^\bullet) \le \sum_{i=1}^{n}\sum_{j=1}^{m_i}\ln\left(p^\bullet_{i,j}\right)x_{i,j} \quad \text{or} \quad \ln(c^\bullet) \ge \sum_{i=1}^{n}\sum_{j=1}^{m_i}\ln\left(p^\bullet_{i,j}\right)x_{i,j}$$

However, with regard to the fact that the utility function cannot be expressed using $\ln(x^\bullet)$, we use an approximation to integrate $x^\bullet$ in the linear model: $x^\bullet = \prod_{i=1}^{n}\sum_{j=1}^{m_i} p^\bullet_{i,j} x_{i,j} \approx 1 - \sum_{i=1}^{n}\left(1 - \sum_{j=1}^{m_i} p^\bullet_{i,j} x_{i,j}\right)$

The approximation is very accurate with parameter values $p^\bullet_{i,j}$ close to 1, which is likely to happen in most real world scenarios (e.g. expressing availability).

– Attributes aggregated by the Min-operator:

The overall QoS attribute x^{min} of a non-functional parameter that is aggregated by the Min-operator (e.g. throughput) is calculated as follows:

$$x^{\min} = \underset{i=1}{\overset{n}{Min}}\left(\sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j}\right)$$

To integrate x^{min} in a linear model a constraint has to be introduced for each task i:

$$x^{\min} \le \sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j} \forall i = 1,\ldots,n$$

Due to the maximization of x^{min} during the optimization process, these constraints limit x^{min} to the smallest value p^{min} of any used Web Service, which is the desired minimum. Constraints are defined as $x^{min} \ge c^{min}$.

The objective function $F(\vec{x})$ expresses the overall utility of the Web Service composition with regard to the user's preferences as a weighted sum of the overall QoS attributes: $F(\vec{x}) = \sum_{l=1}^{k^+} w_l^+ x_l^+ + \sum_{l=1}^{k^\bullet} w_l^\bullet x_l^\bullet + \sum_{l=1}^{k^{\min}} w_l^{\min} x_l^{\min}$

Each of the k ($k=k^+ + k^\bullet + k^{min}$) QoS attributes is specifically weighted (by w_l^+, $w_l^\bullet$, and $w_l^{\min}$) to define

the importance of the improvement of one unit of the attribute relative to one unit of the other attributes.

3.3. A heuristic for a first feasible solution

The work presented in this paper aims at calculating an execution plan that maximizes the overall QoS taking into account the preferences and constraints defined by the user. This leads to an optimization problem based on the objective function and constraints discussed in the previous sub section. As the results of Zeng et al. [20] show, an approach based on linear integer programming is too time consuming for real time scenarios in e-business. This is also confirmed by the work of Canfora et al. [4] and Yu and Lin [19].

Thus, we concentrate on solving the optimization problem by the use of heuristics. We have designed a heuristic H1_RELAX_IP using a two step approach.

First, the LP relaxation of the MIP formulation of the composition problem is solved using a standard algorithm (e.g. simplex). This means that $x_{i,j}$ can take any real number between 0 and 1.

In the second step, a backtracking algorithm is used to construct a feasible solution based on the result of the relaxed integer program. The result of the relaxed integer program gives an indication, which particular Web Service should be considered in the optimal execution plan. For example, if $x_{i,g}$=0,25 and $x_{i,l}$=0,75 the probability of Web Service l of category i being part of the optimal execution plan is much higher than the one of Web Service g. Thus, the candidate Web Services within a specific category are ordered according to their likelihood to be part of the optimal solution (Figure 3). Furthermore, all Web Services having $x_{i,j}$=0 are ordered according to their potential benefit to the objective function[1]. Additionally, the categories are ordered according to the number of candidate Web Services having $x_{i,j}$>0. The backtracking algorithm starts with the category having the fewest Web Services with $x_{i,j}$>0. The fewer choices of Web Services having $x_{i,j}$>0 exist in a category, the higher the probability that an accurate decision is made. In Figure 3, for instance, Category 2 is selected before e.g. Category 3, since it offers fewer choices that have in addition a higher likelihood of being part of the optimal solution. Making decisions in that order causes potentially inaccurate decisions to be made at the end of the backtracking algorithm. This improves the performance of the backtracking algorithm because the earlier an inaccurate decision is made, the more expensive it is, to revise it.

[1] A comprehensive discussion of this issue can be found in [14].

Figure 3. Sorting Web Services

The pseudo code of the backtracking algorithm is depicted in Figure 4. Initially, the execution plan is empty. At the beginning, the first Web Service of the first category (according to the sort order) is placed in the appropriate position of the execution plan. In the example (given in Figure 3), this is the Web Service having index j=3 in Category 2. If no constraint is breached, the algorithm continues with the next category and its first Web Service in the same manner. If a violation occurs the current Web Service will be repeatedly replaced by the next Web Service in the category until no constraint is violated any more. If no such Web Service is available in the current category, the algorithm moves back one category and starts to replace the formerly selected Web Service of this category with the next candidate. If no constraint is breached, the algorithm again proceeds to the next category, or, if a constraint is still breached, it keeps on replacing the selected Web Service of the current category. After having selected a Web Service for all positions of the execution plan from the appropriate categories without violating any constraints, the algorithm terminates with a valid execution plan.

```
i=1;
Exec_Plan={0, 0, ..., 0};
end=false;
while (not end) {
        repeat {
                if (Exec_Plan[i]<mi) Exec_Plan[i]++;
        } until (Exec_Plan is valid or Exec_Plan[i]=mi);
        if (Exec_Plan is invalid) {
                Exec_Plan[i]=0;
                if (i>1) i--; else end=true;
        } else
                if (i<n) i++; else end=true;
}
```

Figure 4. Backtracking algorithm

3.4. Meta-heuristics for the improvement of feasible solutions

To analyze how far the results of H1_IP_RELAX can be improved with standard techniques, we have designed two meta-heuristics. H2_SWAP tries to find a solution with a higher QoS by randomly replacing Web Services of the execution plan calculated by H1_RELAX_IP. Thereby it only accepts valid execution plans raising the

value of the objective function. H3_SIM_ANNEAL is based on Simulated Annealing, which temporarily accepts worse solutions during the optimization process to be able to leave local optima and possibly find the global optimum.

4. Evaluation

In this section our heuristics are evaluated and compared to integer programming with regard to the computation time and the excellence in approximating the optimal solution. The experiments were run on a Pentium IV (3 GHz) system having 1024 MB of RAM. For performing the simulations a simulation engine and further simulation tools (e.g. a data generator) have been implemented.

4.1. Comparison of H1_RELAX_IP with integer programming

To study the performance of H1_RELAX_IP, we create sets of randomly generated test cases, each varying a parameter influencing the performance. We analyze the impact of *i.)* varying the process length (number of task items), *ii.)* varying the number of candidate Web Services and *iii.)* varying the strength of constraints. Each set of test cases is solved with H1_RELAX_IP and the MIP solver lp_solve[2]. The solver uses a Simplex algorithm and Branch&Bound to calculate the optimal solution. We compare the computation time and the value of the objective function of H1_RELAX_IP to the ones of the solver.

i.) Analysis of the numbers of tasks. To analyze the influence of the number of task items, we increase the number in seven steps from three to 21 and generate a sample of 35 test cases per number of task items. In each test case, there are 40 candidate Web services available per task item, with each Web Service having four non-functional parameters. Every overall QoS attribute is constrained.

As depicted in Table 1 and Figure 5, our heuristic significantly outperforms the integer programming with increasing numbers of tasks. Only with a very small number of task items the solver performs marginally better.

Table 1. Results of varying the number of tasks

Task items	Solver Avg: t_s [ms]	H1 RELAX IP Avg: t_h [ms]	Avg: $\frac{t_h}{t_s}$	Avg: $\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$
3	4.2864	5.2734	136.08%	99.96%
6	16.3458	8.4682	69.17%	99.89%
9	194.9850	13.9468	22.63%	99.72%
12	1,062.7156	20.3617	8.86%	99.44%
15	5,976.9378	26.2847	3.43%	99.38%
18	60,772.1917	35.5667	0.92%	99.23%
21	264,177.5668	43.3683	0.19%	98.83%

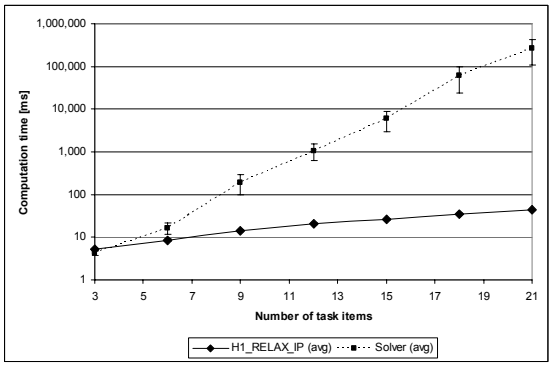

Figure 5. Comparison of the computation time

Increasing the number of task items, H1_RELAX_IP scales very well and shows a much better performance than the solver. In spite of the very good performance there is almost no loss with regard to the optimal solution calculated by the solver (Figure 6). In the case of a process composed of 21 tasks, H1_RELAX_IP reaches 98.83% of the objective function value of the optimal solution (approximation ratio), but only needs 0.19% of the computation time of the solver.

Since we primarily consider Web Services offering comprehensive business functionality that can be easily outsourced, the number of such coarse-grained Web Services forming a workflow can be assumed to be in the order of the magnitude of 5-20. Of course, such coarse-grained services might consist of a large number of small services.

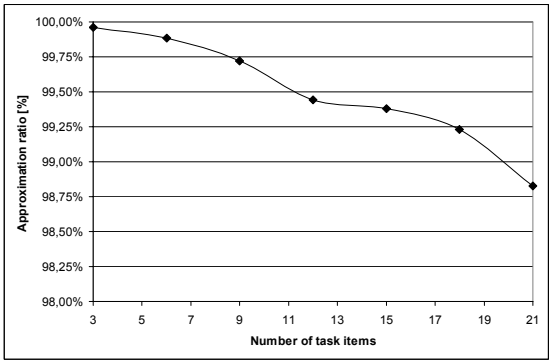

Figure 6. H1_RELAX_IP: Approximation ratio

ii.) Analysis of the numbers of candidate Web Services. In analogy to the previous analysis another set of test cases is generated, varying the number of candidate Web Services from 10 to 70. The length of the process is fixed to 15 task items. As depicted in Table 2 and Figure 7, H1_RELAX_IP outperforms the solver for every of the tested number of candidate Web Services.

[2] http://lpsolve.sourceforge.net/5.5/

Table 2. Results of varying the number of candidate Web Services

Web Service	Solver Avg: t_s [ms]	H1_RELAX_IP Avg: t_h [ms]	Avg: $\frac{t_h}{t_s}$	Avg: $\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$
10	617.4211	6.5910	8.48%	98.31%
20	4,528.6759	12.6866	2.04%	99.09%
30	5,122.3285	19.2043	2.69%	99.19%
40	5,723.9903	26.3751	2.71%	99.49%
50	12,302.9522	34.8580	3.51%	99.43%
60	13,789.3714	43.2929	1.84%	99.39%
70	25,840.1226	51.0882	1.32%	99.27%

As the percentage of computation time needed by H1_RELAX_IP compared to the solver even tends to decrease for large numbers of candidate Web Services, there is no decrease of the percentage of the reached objective function value.

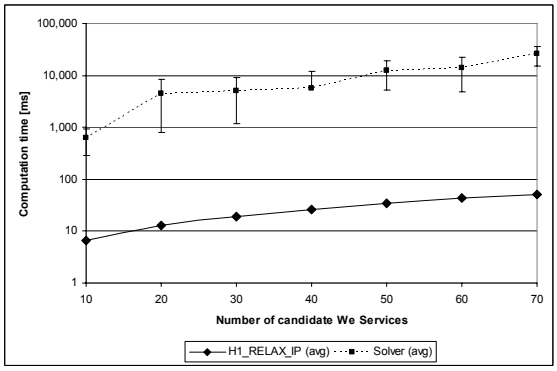

Figure 7. Comparison of the computation time

As shown in Figure 8, H1_RELAX_IP reaches about 99% of the objective function value of the optimal solution.

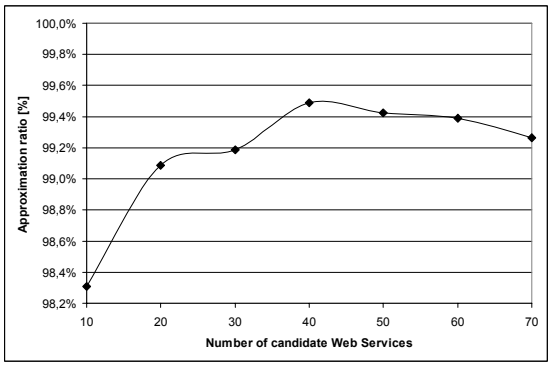

Figure 8. H1_RELAX_IP: Approximation ratio

iii.) Analysis of the strength of restrictions. The *strength of a restriction* is the value restricting a QoS attribute expressed relatively (as percentage) to the best possible value of the QoS attribute. A strength of 0% is equivalent to an unconstrained QoS attribute. A restriction with a strength of 100% can only be satisfied if the QoS attribute is aggregated of the best QoS values available in every category. A restriction with a strength of 50% is satisfied if the QoS attribute is aggregated of the average QoS value available in every category.

To analyze the influence of the *strength of restrictions*, samples of test cases with an increasing tightness of constraints on the overall QoS attributes are generated.

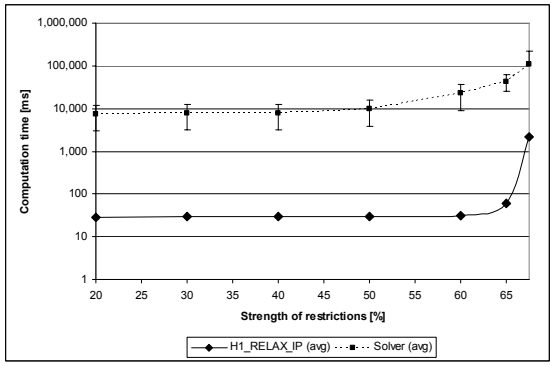

Figure 9. Comparison of computation time

We generate test cases with a process length of 15 task items, 40 candidate Web Services and each Web Service being described by four QoS parameters. The overall QoS attributes are being constrained by restrictions of a strength rising from 20% to 68%, which is near the border of insolubility for some test cases.

Table 3. Results of varying the strength of restrictions

Strength of restrictions	Solver Avg: t_s [ms]	H1_RELAX_IP Avg: t_h [ms]	Avg: $\frac{t_h}{t_s}$	Avg: $\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$
20%	7,585.1553	28.4762	1.84%	99.29%
30%	7,674.9336	29.2273	1.77%	99.29%
40%	7,888.8139	29.9284	1.74%	99.29%
50%	9,632.8835	30.1708	1.65%	99.16%
60%	22,996.1480	30.5376	0.76%	98.96%
65%	43,145.0710	59.1515	0.32%	98.92%
68%	108,692.6069	2,128.9318	1.05%	98.96%

As depicted in Table 3 and Figure 9, H1_RELAX_IP outperforms the solver for every *strength of restrictions* tested. In all cases H1_RELAX_IP needed less then 2% of the solver's computation time, but created a solution having about 99% of the objective function value of the optimal solution (Figure 10).

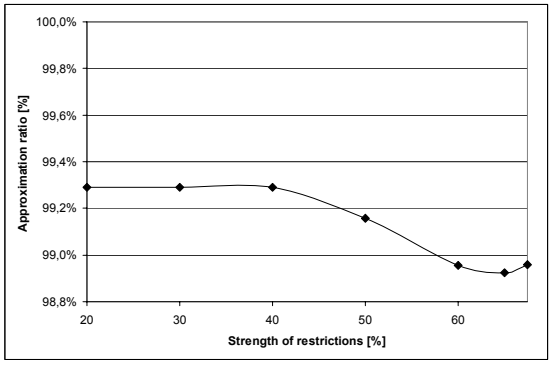

Figure 10. H1_RELAX_IP: Approximation ratio

Approaching the border of insolubility, the computation time dramatically increases (Figure 9), but nevertheless the generated solution of H1_RELAX_IP keeps its excellence on a high level (Figure 10).

4.2. Analysis of H2_SWAP and H3_SIM_ANNEAL

We apply H2_SWAP and H3_SIM_ANNEAL to study how far we can improve the solutions of H1_IP_RELAX with meta-heuristics. H2_SWAP and H3_SIM_ANNEAL are only applied if there might be a chance for further improvement. As a coarse estimate for the potential improvement, we use the deviation of the solution generated by H1_IP_RELAX to the solution of the relaxed integer program, which represents an upper bound for the optimal solution. If the solution of H1_IP_RELAX deviates more then 1% from the upper bound, H2_SWAP is applied. If the new solution still deviates more than 1% from the upper bound, H3_SIM_ANNEAL will be applied.

Analyzing the test cases discussed in Section 4.1., the revealed potential of improvement has to be seen as very marginal. The additional heuristics have been applied in 70.2% of the test cases, caused an increment of 287.48% to the computation time, but only improved the objective function value by 0.28%. As an example, Table 4 shows the increment of the computation time and the percentage of the objective function value generated by the heuristics for a varying *strength of restrictions*.

Table 4. Contribution of meta-heuristics

Strength of restrictions	Increase of computation time	Contribution to objective function value		
		H1	H2	H3
20%	196.85%	99.83%	0.17%	0.00%
30%	197.47%	99.90%	0.10%	0.00%
40%	208.94%	99.86%	0.14%	0.00%
50%	235.69%	99.81%	0.12%	0.06%
60%	404.28%	99.59%	0.35%	0.06%
65%	500.84%	99.61%	0.24%	0.15%
68%	627.97%	99.47%	0.39%	0.14%

5. Related Work

Zeng et al. [20] present comprehensive research about QoS modeling and QoS-aware composition. They propose a linear integer programming approach for calculating the optimal composition. However, as already mentioned above this approach is too time consuming in real-time e-business scenarios.

In the context of the METEOR-S project a lot of research related with QoS enabled Web Services has been done: Aggarwal et al. [1] present a *Constraint Driven Web Service Composition Tool* that enables the composition of Web Services considering their QoS attributes. Like Zeng et al. [20] a linear integer programming approach is proposed for solving the optimization problem. However, Aggarwal et al. do not present an evaluation of their approach. Cardoso et al. [7] present the *Stochastic Workflow Reduction (SWR)* algorithm to calculate QoS of complex Web Service workflows by decomposition into atomic tasks. Furthermore, the authors discuss different QoS attributes. In [6] Cardoso proposes a comprehensive ontology-based approach for modeling the functional and non-functional behavior of Web Service workflows.

Canfora et al. [4] discuss genetic algorithms as an approach for solving the Web Service composition problem. The results reveal that genetic algorithms show a better performance and scalability than linear integer programming with increasing numbers of candidate Web Services and tasks. In a further paper, the authors consider replanning at runtime as well [5].

Tosic et al. present the Web Service Offerings Language (WSOL) [16] and the Web Service Offerings Infrastructure (WSOI) [15] that support specification, monitoring, and manipulation of classes of service for Web Services.

Yu und Lin propose the QoS-capable Web Service Architecture (QCWS) [18], that is quite similar to our WSQoSX. In [17] they study algorithms and heuristics for a QoS-aware Web Service selection with only one QoS constraint. In [19] they extend their work to multiple QoS constraints. The composition problem is modeled as a multi-dimension multi-choice 0-1 knapsack problem (MMKP) as well as a multi-constraint optimal path (MCOP) algorithm. For both, heuristics are presented. However, the aggregation of parameters using the Min-operator is neglected. Furthermore, the evaluation lacks a metric describing the tightness of used constraints like our *strength of restrictions.*

In [12] Maximilien and Singh describe the Web Service Agent Framework (WSAF) to achieve service selection taking into account the preferences of service consumers as well as the trustworthiness of providers. However, this approach does not support the composition of an execution plan that complies with constraints defined by the user.

6. Conclusion and outlook

Web Services are becoming increasingly more important in realizing cross-organizational e-business scenarios. With a growing number of Web Services offering equal or similar functionality the non-functional attributes of a Web Service (e.g. costs and QoS attributes) are important selection criteria and crucial for enterprises to meet the requirements of sophisticated customers. Thus, the QoS-aware composition of Web Service workflows is an important issue.

In this paper, our work on WSQoSX is extended by proposing a heuristic based approach to solve the QoS-aware Web Service composition problem. For this, we present a heuristic H1_RELAX_IP that uses a backtracking algorithm on the results computed by a relaxed integer program. The evaluation of H1_RELAX_IP reveals that this heuristic is extremely fast and leads to results that are very close to the optimal solution. H1_IP_RELAX outperforms a linear integer programming based solution by a solver with regard to the computation time, especially with increasing number of candidate Web Services and process tasks.

Our further research aims at investigating effective replanning strategies during Web Service execution. Especially, the trade-off between the overhead due to the replanning and meeting a SLA despite of runtime failures will be studied.

7. Acknowledgements

The work on this paper is partly sponsored by the E-Finance Lab (http://www.efinancelab.de). A special thanks goes to Andreas U. Mauthe (InfoLab 21, Lancaster University) for the constructive discussions.

8. References

[1] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint Driven Web Service Composition in METEOR-S", *IEEE International Conference on Services Computing (SCC 2004)*, Shanghai, China, 2004.

[2] R. Berbner, T. Grollius, N. Repp, O. Heckmann, E. Ortner, and R. Steinmetz, "An approach for the Management of Service-oriented Architecture (SoA) based Application Systems", *Enterprise Modelling and Information Systems Architectures (EMISA 2005)*, Klagenfurt, Austria, 2005.

[3] R. Berbner, O. Heckmann, and R. Steinmetz, "An Architecture for a QoS driven composition of Web Service based Workflows", *Networking and Electronic Commerce Research Conference (NAEC 2005)*, Riva Del Garda, Italy, 2005.

[4] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms", *Genetic and Evolutionary Computation Conference (GECCO 2005)*, Washington DC, USA, 2005.

[5] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-Aware Replanning of Composite Web Services", *IEEE International Conference on Web Services (ICWS 2005)*, Orlando, FL, USA, 2005.

[6] J. Cardoso, *"Quality of Service and Semantic Composition of Workflows"*, Ph.D. Dissertation, Department of Computer Science, University of Georgia, Athens, GA, USA, 2002.

[7] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes", *Web Semantics: Science, Services and Agents on the World Wide Web*, volume 1, issue 3, 2004, pages 281-308.

[8] B. Esfandiari and V. Tosic, "Requirements for Web Service Composition Management", *11th HP OpenView University Association Workshop (HP-OVUA 2004)*, Paris, France, 2004.

[9] M. R. Garey and D. S. Johnson, *"Computers and Intractability: a Guide to the Theory of NP-Completeness"*, W.H. Freeman & Co., New York, NY, USA, 1979.

[10] D. Gouscos, M. Kalikakis, and P. Georgiadis, "An Approach to Modeling Web Service QoS and Provision Price", *4th International Conference on Web Information Systems Engineering Workshops (WISEW 2003)*, Rome, Italy, 2003.

[11] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, "*Web Service Level Agreement (WSLA) Language Specification*", IBM Corporation, 2003, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

[12] E. M. Maximilien and M. P. Singh, "Toward autonomic web services trust and selection", *2nd international conference on Service oriented computing*, New York, NY, USA, 2004.

[13] S. Ran, "A model for Web Services discovery with QoS", *ACM SIGecom Exchanges*, volume 4, issue 1, 2003, pages 1-10.

[14] M. Spahn, R. Berbner, O. Heckmann, and R. Steinmetz, "*Ein heuristisches Optimierungsverfahren zur dienstgütebasierten Komposition von Web-Service-Workflows*", Technical Report (in German) KOM 02-2006, Technische Universitaet Darmstadt, Darmstadt, Germany, 2006.

[15] V. Tosic, W. Ma, B. Pagurek, and B. Esfandiari, "Web Services Offerings Infrastructure (WSOI) - A Management Infrastructure for XML Web Services", *IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, Seoul, South Korea, 2004.

[16] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma, "Management Applications of the Web Service Offerings Language (WSOL)", *15th International Conference on Advanced Information Systems Engineering* Velden, Austria, 2003.

[17] T. Yu and K.-J. Lin, "The Design of QoS Broker Algorithms for QoS-Capable Web Services ", *International Conference on e-Technology, e-Commerce and e-Service (EEE 2004)* Taipei, Taiwan, 2004.

[18] T. Yu and K.-J. Lin, "A Broker-Based Framework for QoS-Aware Web Service Composition", *International Conference on e-Technology, e-Commerce, and e-Services (EEE 2005)*, Hong Kong, China, 2005.

[19] T. Yu and K.-J. Lin, "Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints", *3rd International Conference on Service-Oriented Computing (ICSOC 2005)*, Amsterdam, The Netherlands, 2005.

[20] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware Middleware for Web Service composition", *IEEE Transactions on Software Engineering*, volume 30, issue 5, 2004, pages 311-328.