[BSRHSt07] *Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, Ralf Steinmetz;* **Dynamic Replanning of Web Service Workflows;** IEEE International Conference on Digital Ecosystems and Technologies 2007 (IEEE DEST 2007) Cairns, February 2007, S.

# Dynamic Replanning of Web Service Workflows

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* — **The composition of Web Services to workflows is one of the major challenges in the area of service-oriented computing. To meet the business and user requirements, it is crucial to manage the Quality of Service (QoS) of Web Service workflows. In our approach, we calculate the execution plan of workflows on the QoS attributes ex ante based on predictions. However, due to the volatile nature of the Internet and the web servers, the runtime behavior of Web Services is likely to differ from the predictions. Therefore, we propose replanning as a mechanism to adapt the execution plan to the actual behavior of already executed services by a dynamic service selection at runtime, ensuring that the QoS and cost requirements will still be met. In this paper, we discuss replanning strategies, show how replanning leads to cost-savings in most cases, and evaluate the additional overhead caused by the adaptation of the execution plan at runtime.**

*Index Terms* — **SoA, Web Services, QoS, Replanning.**

## I. Introduction

Nowadays, the Service-oriented Architecture (SoA) paradigm is proposed as an upcoming architectural blueprint for enabling flexible workflows. Web Services as a technology adopting the SoA paradigm gain more and more importance in academic as well as in industrial environments. However, the dynamic and automated composition of Web Services to Web Service workflows is one of the major challenges in the area of service-oriented computing. Considering non-functional attributes of Web Services during the planning phase as well as during the execution phase of composed workflows is a crucial success factor for meeting the business requirements and preferences defined by the user. Besides costs the Quality of Service (QoS) attributes of a Web Service (e.g. availability, response time and throughput) are subsumed as non-functional attributes.

In our previous work, we have designed and prototypically implemented WSQoSX (Web Services Quality of Service Architectural Extension), a proxy architecture based on Web Service technology, that provides comprehensive QoS support for Web Service workflows, e.g. monitoring mechanisms for detecting SLA violations [2, 4]. As an extension of WSQoSX we have designed fast-performing heuristics for calculating an optimized execution plan of a workflow, considering user defined constraints and preferences regarding the non-functional behavior of a workflow. Since Web Services often do not behave as described in their Service Level Agreement (SLA), we identified replanning mechanisms to be crucial to ensure that an execution plan remains *feasible, valid,* and *optimal* subject to the user preferences during execution [3]. Within e-business scenarios, replanning mechanisms have to perform in real-time not to create further delay to the workflow execution. Thus, we propose a fast and efficient heuristic-based replanning mechanism. In this paper, we extend our research on replanning by a detailed evaluation of our algorithm proposed in [3]. For this, we analyze the trade-off between potential cost savings and the overhead due to the additional computation effort for calculating the adaptation of the execution plan at runtime.

The rest of this paper is structured as follows: In Section 2 Web Service workflows are discussed and replanning is introduced. The replanning algorithm is described and evaluated in detail in Section 3. In Section 4 related work in the area of replanning is discussed. The paper closes with a conclusion and a short outlook on our future research activities.

## II. Replanning of Web Service workflows

In this section we summarize the basic concepts of Web Service workflows and replanning as proposed in our previous work [3, 5].

### A. Web Service workflows

Web Service composition aims at selecting and interconnecting Web Services provided by different partners according to a business process [13]. Thus, Web Service compositions can be seen as workflows based on Web Services. In this paper, we assume sequential Web Service workflows. 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 $i$. For each Web Service a binary variable $x_{i,j}$ is introduced. $x_{i,j}$=1 means that Web Service $j$ of category $i$ is selected for execution within the execution plan. Web Services within the same category may differ in their $k$ non-functional parameters $p_{i,j,k}$ (e.g. cost or response time). QoS-aware Web Service composition can be defined as the assignment of specific Web Services to abstract workflow tasks in order to create an execution plan that is optimal subject to the preferences and constraints defined by the user. Following this definition, QoS-aware Web Service composition (i.e. calculating an optimal execution plan) leads to an optimization problem.

We propose a linear model to describe this optimization

problem. Table 1 shows how the overall QoS attributes of a workflow are calculated based on the QoS attributes of the involved Web Services by an aggregation function. The aggregation of QoS parameters to an overall QoS attribute depends on the type of the QoS parameter $k$. We distinguish three different operators, i.e. additive, multiplicative and min-operator. Table 2 shows how constraints restricting the overall QoS attributes are embedded in the model. For a more detailed explanation of our model we refer to [5]. To rate a concrete Web Service composition, an objective function $F(\vec{x})$ is used to calculate an overall utility value with regard to the user's preferences as a weighted sum of the composition's overall QoS attributes:

$$F(\vec{x}) = \sum_{l=1}^{k^+} w_l^+ x_l^+ + \sum_{l=1}^{k^\bullet} w_l^\bullet x_l^\bullet + \sum_{l=1}^{k^{min}} w_l^{\min} x_l^{\min} \qquad (1)$$

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.

As described in [6] and [12] the optimization problem specified above is NP-hard.

Table 1 Formulas for calculating overall QoS attributes

| | Parameter | Overall QoS attributes |
|---|---|---|
| Additive parameters | $p_{i,j}^+$ | $x^+ = \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^+ x_{i,j}$ |
| Multiplicative parameters | $p_{i,j}^\bullet$ | $x^\bullet = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^\bullet x_{i,j} \approx 1 - \sum_{i=1}^{n} \left(1 - \sum_{j=1}^{m_i} p_{i,j}^\bullet x_{i,j}\right)$ |
| Min-operator | $p_{i,j}^{\min}$ | $x^{\min} = \min_{i=1}^{n} \left( \sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j} \right)$ |

Table 2 Formulas for calculating overall constraints

| | Parameter | Constraints |
|---|---|---|
| Additive parameters | $p_{i,j}^+$ | $x^+ \le c^+$ or $x^+ \ge c^+$ |
| Multiplicative parameters | $p_{i,j}^\bullet$ | $\ln(c^\bullet) \le \sum_{i=1}^{n} \sum_{j=1}^{m_i} \ln(p_{i,j}^\bullet) x_{i,j}$ or $\ln(c^\bullet) \ge \sum_{i=1}^{n} \sum_{j=1}^{m_i} \ln(p_{i,j}^\bullet) x_{i,j}$ |
| Min-operator | $p_{i,j}^{\min}$ | $x^{\min} \le \sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j} \forall i = 1,\ldots,n$ |

*Replanning mechanism*

In this section, a replanning mechanism is discussed, which adapts an execution plan at run-time in a way that it remains *feasible*, *valid* and *optimal* subject to the preferences and constraints defined by the user [3].

If a Web Service is invoked, which is unavailable (e.g. due to server downtime), the workflow is not executable anymore. Thus, the execution plan has to be modified and the current Web Service has to be replaced by another (available) one in order to achieve that the execution plan of the workflow remains *feasible*.

If the server load of a provider is very high and the response time of a Web Service is much higher than expected, then a constraint, restricting the overall response time of the workflow, may be violated. In this case, the unexecuted part of the workflow has to be replanned and Web Services having a lower response time have to be used in order to ensure that the current execution plan of the workflow remains *valid* with regard to all constraints.

If all providers have a low or average server load, then the response time of the Web Services is likely to be much lower than guaranteed by their SLA. Each time a Web Service is executed, some response time is saved (compared to the expected behavior). If minimizing costs is the optimization criteria (assuming that Web Services having a higher response time are cheaper than faster ones), then the execution plan might not be optimal anymore. This is due to the fact that the execution plan could be modified in a way that cheaper Web Services with a higher response time are used, without violating a constraint restricting the overall response time. Thus, the unexecuted part of the workflow has to be replanned in order to ensure that the current workflow is *optimal* subject to the user's optimization preferences.

To ensure that an execution plan remains *feasible*, *valid* and *optimal*, we apply a replanning mechanism: After having executed a Web Service at position $i$ of an execution plan, the execution plan is divided into two parts as depicted in Fig. 1. The first part consists of all positions $i'$ ($i' \le i$), which already have been executed. The second part consists of all positions $i''$ ($i'' > i$), which still have to be executed. The $k$ overall attributes of the first part are calculated based on the according parameter values $p_{i,j,k}$ of the used Web Services and the appropriate aggregation functions ($x^+, x^\bullet, x^{\min}$) as discussed in the previous subsection.

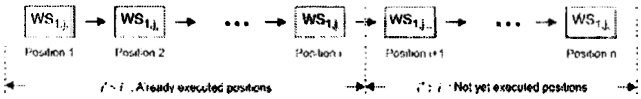



Fig. 1 Partitioning of an execution plan [3]

Since all Web Services of the first part have already been executed, the parameter values originally taken from a Web Service's SLA are replaced with the ones actually monitored during execution. The actual overall attributes are used to adjust the constraints restricting the overall attributes of the second part to the hitherto existing behavior of the workflow. Using the adjusted constraints, a new optimization problem for the unexecuted part of the execution plan is created. This leads to a new execution plan for the second part, which is valid with regard to all current constraints and is optimal with regard to the defined optimization preferences. The newly optimized execution plan is used for the further execution of the workflow and is updated using the replanning mechanism each time a Web Service has been executed.

## III. Heuristic Based Replanning

As mentioned before the calculation of the execution plan leads to an optimization problem that is NP-hard. Zeng et al. [13] propose calculating the execution plan with integer programming. However, the evaluation of their results reveals that this approach is hardly feasible in real-time e-business scenarios. This is exacerbated in the context of replanning. Therefore, high-performance heuristics are nec-

essary solving the underlying optimization problem.

In this section, the heuristic H1_RELAX_IP, which we use for replanning, is discussed. The focus of this chapter is on a detailed evaluation of the trade-off between cost-reduction and additional computation time due to replanning. As pointed out in [5], H1_RELAX_IP uses a two step approach.

First, the LP relaxation of the MIP (mixed integer problem) formulation of the composition problem introduced in the previous section is solved. The relaxation allows all decision variables $x_{i,j}$ to be any real number between 0 and 1. A problem of this kind can be solved very fast using a standard algorithm like e.g. simplex.


Fig. 2 Partitioning of an execution [3]

In the second step, a backtracking algorithm (Fig. 3) 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. Furthermore, all Web Services having $x_{i,j}$=0 are ordered according to their potential benefit to the objective function. 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 Fig. 2, Category 2 is selected before Category 3, since it offers fewer choices that have in addition a higher likelihood of being part of the optimal solution. This improves the performance of the backtracking algorithm because the earlier an inaccurate decision is made, the more expensive it is to revise it. For a further discussion of H1_RELAX_IP we refer to [5].

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

Fig. 3. Backtracking algorithm [5]

## IV. Evaluation

To analyze the effectiveness of the replanning strategy, three main sets of test cases have been generated, varying the length *n* of the business process (number of task items), the number *m* of candidate Web Services available per category and the strength *s* of the restriction constraining the overall response time.

The *strength of restriction s* is used to indicate the tightness of a constraint put on an overall attribute of the business process. The value of *s* is calculated as the value restricting a non-functional overall attribute (i.e. the overall response time) expressed relatively (as percentage) to the best possible value of the overall attribute. A strength of 0% is equivalent to an unconstrained overall attribute. A restriction with a strength of 100% can only be satisfied if the overall attribute is aggregated of the best values available in each category. An iteration-based evaluation has been performed, in which 35 different test cases are generated per iteration for a specific parameter value. The results of these test cases are aggregated to average values which define the result of an iteration.

### *A. Experiment setup*

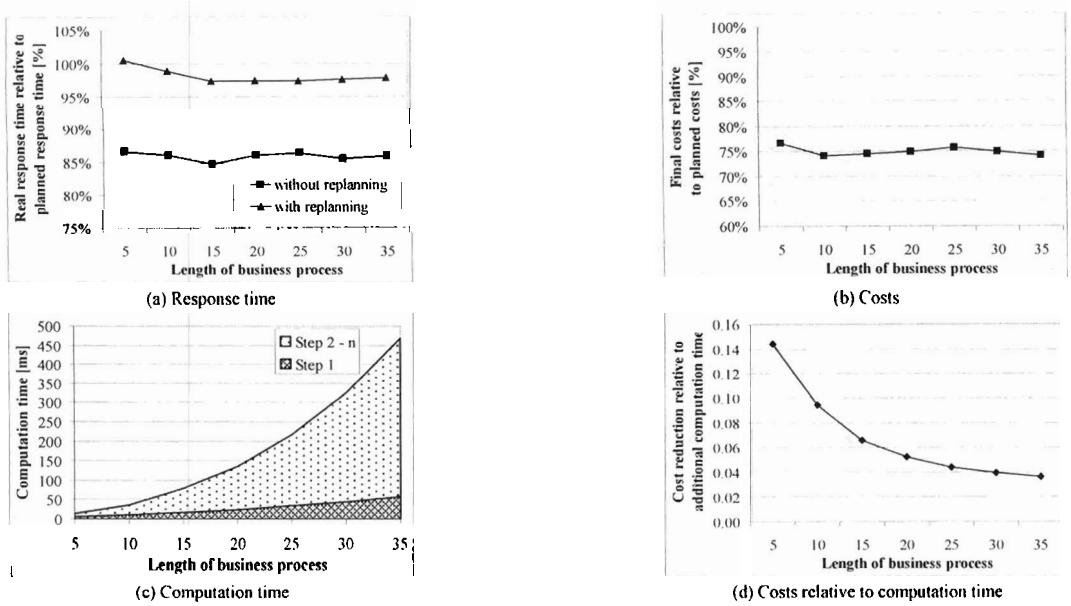
To evaluate the effects of replanning, a data generator has been developed, which generates two values for every non-functional parameter of a Web Service. The first value (plan value *p*) defines how a parameter has been set by the provider of a Web Service in its SLA. The second value (actual value *p'*) defines how the Web Service would really perform when being executed. The actual value *p'* is calculated as the realization of a normally distributed random variable $X_{p'}$, which has the plan value *p* as a border of a confidence interval to a certain confidence level *c* (e.g. 95%). To optionally restrict *p'* to a certain value range, an interval for the allowed value range can be defined ($[\underline{d},\overline{d}]$). If *p'* is not in the allowed range, the generation process is repeated until *p'* fits the range. To evaluate the effects of replanning, several test cases have been generated in which Web Services are described by the two non-functional properties response time and costs per call. The higher the response time, the lower are the costs per call (correlation coefficient ~ -0.958). The actual value of the costs per call is identical to the plan value. In all test cases the only optimization criteria is the minimization of the overall execution costs of the business process. The overall response time of the business process is restricted to a certain value by a constraint.

The experiments were run on a Pentium IV CPU with 3 GHz and 1 GB of RAM using a special business process simulation engine (BPSE) implemented in Java 1.5.

### *B. Varying the length of business processes*

In this set of test cases the length *n* of a business process is varied from 5 to 35 task items. The number of candidate Web Services available in each category is set to 40 and the *strength of the restriction* limiting the overall response time of the business process is set to 40%.

In the fist step of the business process execution an initial.




(a) Response time

(b) Costs

(c) Computation time

(d) Costs relative to computation time

Fig 4. Varying the length of the business process

execution plan is built based on the (predicted) plan values. Without using replanning the total response time after execution is about 85% of the originally planned response time (Fig. 4a) because most Web Services took less time to exe cute than predicted in their SLAs. With replanning the overall response time can be raised to about 97% of the originally planned response time by using slower and cheaper Web Services in the yet unexecuted part of the business process without violating the constraint on the overall response time. Due to the replanning, i.e. the use of slower and cheaper Web Services, the costs of the business process execution can be reduced to about 75% of the originally planned executions costs (Fig. 4b). The drawback of using replanning is that the heuristic has to be executed after each single Web Service execution to create a newly optimized execution plan, which results in an additional overhead of computation time (Step 2 until *n*, Fig. 4c) consumed by the BPSE. Fig. 4d shows the achieved cost reduction in percent per additional percent invested computation time in replanning. This value can be seen as a coarse measure of how effective an additional percent of computation time is used in average to reduce the execution costs. As the cost reduction is nearly constant for every length of business processes, but the additional computation time spent on replanning rises with an increasing length of business processes (Fig. 4c), the effectiveness of cost reduction decreases with an increasing length of the business process.

### *C. Varying the number of candidate Web Services*

Analogous to the previous experiment a set of test cases is generated with a varying number of candidate Web Services per task item (10 to 70 candidate Web Services). The length of the business process is set to 20 task items and the strength of the restriction limiting the overall response time to 40%.

Independently of the number of candidate Web Services, the execution of the initially created execution plan results in an overall response time of about 85% of the originally planned overall response time. Using replanning the overall response time can be raised to about 97% of the originally planned overall response time (Fig. 5a) and the execution costs can be reduced to about 76% to 71% of the originally planned execution costs (Fig. 5b). As a coarse trend it can be recognized that the higher the number of candidate Web Services per category, the higher the achieved cost savings. With an increasing number of candidate Web Services, the more choices exist from which the heuristic can choose from when creating a newly optimized execution plan for the unexecuted part of the business process. So the probability increases that saved time can be used for the placement of Web Service as cheap as possible resulting in a higher cost reduction. An increasing number of candidate Web Services per task item results in a higher computation time needed for replanning. But as the cost reduction rises, the achieved cost reduction in percent per additional percent computation time needed for replanning does only decrease slightly.

### *D. Varying the strength of a restriction*

In the last set of test cases we vary the *strength of restriction* (10% to 70%) to analyze the effect of different constraints limiting the overall response time of the business process. The length of the business process (20 task items) and the number of candidate Web Services (40 Web Services) remains unchanged.

Independently of the *strength of restriction*, the execution of the initially created execution plan results in an overall response time of about 86% of the originally planned overall response time, which is comparable to the previously analyzed sets of test cases. But in contrast to the previous sets, replanning is able to approximate the initially planned total response time better with an increasing *strength of restriction* and is even able to exceed it (Fig. 6a). However, assuming a high *strength of restriction* replanning can only use a few fast Web Services to create a valid execution

plan. Thus, cost reductions due to replanning decrease with an increasing *strength of restriction* (Fig. 6b). As depicted in Fig. 6c varying the strength of restriction has nearly no impact on the computation time needed for replanning, which is mainly determined by the number of candidate Web Services and the length of the business process, which are constant in this set of test cases. Because of the additional computation time being nearly constant and the cost reduction decreases with an increasing *strength of restriction*, the cost reductions relative to the additional computation time decreases (Fig. 6d).

### *E. Summary*

When using replanning, the *strength of restriction* is the main parameter influencing the possible increase in the real overall response time and the possible cost reduction. Having a *strength of restriction* of 40%, a real overall response time of about 97% of the initially planned overall response time and a cost reduction of about 25% can be achieved.

Parameters concerning the problem size, like the length of the business process or the number of candidate Web Services per task item do not influence these achievable values significantly – only an increase of the number of candidate Web Services is able to improve the achievable cost reduction slightly. Parameters concerning the problem size are the main parameters influencing the additional computation time needed for replanning. With regard to the achievable cost reduction the most important analysis results are: The lower the *strength of restriction*, the higher the achievable cost reduction. A further slight increase of the achievable cost reduction can be realized by increasing the number of available Web Services.

The length of a business process has nearly no impact on the achievable cost reduction, but as all parameters concerning the problem size, an increase of the length tremendously increases the additionally needed computation time for replanning. How effective a cost reduction can be achieved by the use of replanning (e.g. what average cost reduction in percent can be realized per percentage of additionally invested computation time) depends on the concrete combination of problem size and *strength of restriction*. The lower the *strength of restriction* and the smaller the problem size, the more effective replanning is with regard to the achieved cost reduction compared to the additionally invested computation time for replanning.

## V. Related Work

Within the METEOR-S project, Cardoso et al. [8, 9] present the Stochastic Workflow Reduction (SWR) algorithm to calculate the QoS of complex Web Service workflows by decomposition into atomic tasks. As further research in the context of the METEOR-S project Aggarwal et al. [1] present a Constraint Driven Web Service Composition Tool that enables the composition of Web Services considering their QoS attributes. As in Zeng et al. [13], a linear integer programming approach is proposed for solving the optimization problem. However, the authors do not present an evaluation of their approach.

Yu und Lin present the QoS-capable Web Service Architecture (QCWS) [11] that is quite similar to our WSQoSX. In [10], Yu und Lin study algorithms and heuristics for a QoS-aware Web Service selection with only one QoS constraint. In [12], the work is extended 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 and the evaluation lacks a metric describing the tightness of used constraints like our strength of restriction. Furthermore, using the proposed heuristics for replanning is not addressed.

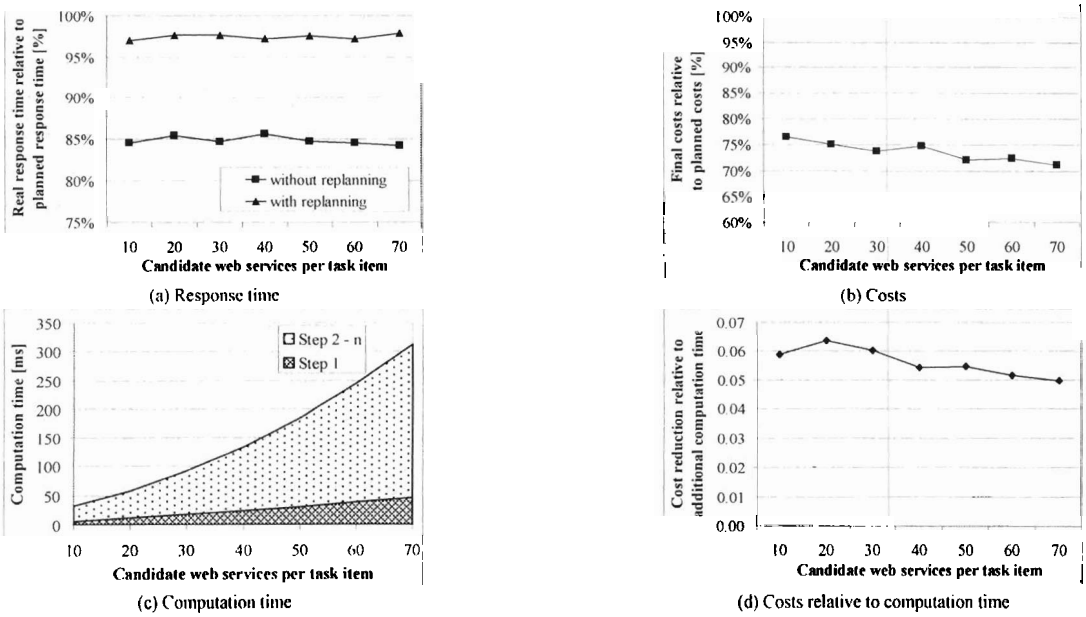
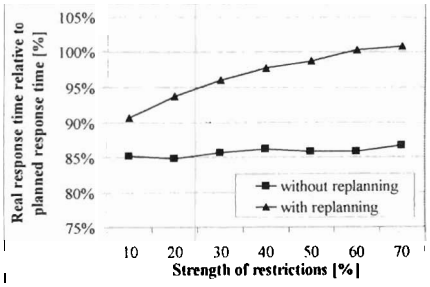
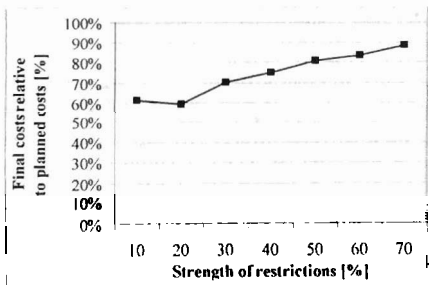


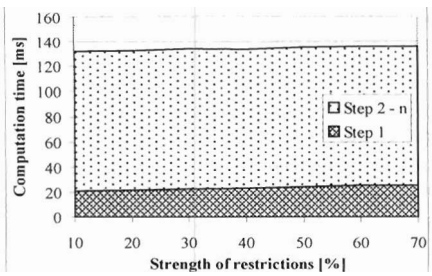

Fig 5. Varying the number of candidate Web Services



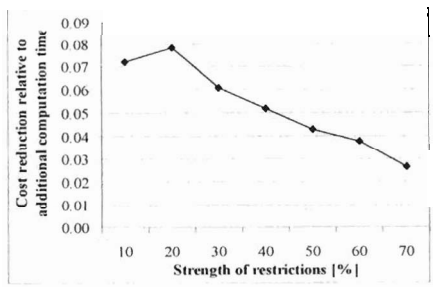

(a) Response time




(b) Costs




(c) Computation time




(d) Costs relative to computation time

Fig 6. Modifying the strength of restriction

Canfora et al. [6] apply genetic algorithms 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 [7]. Their understanding of replanning is very similar to ours.

## VI. Conclusion and Outlook

In this paper we extend our previous work on QoS-aware selection and execution of Web Service workflows. Due to the volatile nature of the Internet the runtime behavior of Web Services often differs from the one predicted during the planning phase. As a consequence, preferences and constraints defined by the user cannot be met anymore and SLAs concerning the whole business process may be violated. Thus, we have designed a replanning mechanism to adapt the unexecuted part of a workflow in a way that despite the deviation of the runtime behavior the user requirements will still be met. For this, we model replanning as an optimization problem, which can be efficiently solved by our heuristic H1_RELAX_IP. The focus of this paper is on the detailed evaluation of our replanning heuristic. As a result, we can show that our approach can achieve significant cost savings with reasonable computation effort.

As future work we will investigate different replanning strategies. For this, we will have a look at related work in the area of production planning as well as performance prediction.

## References


[1] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, Constraint Driven Web Service Composition in METEOR-S, in *Proceedings of the IEEE International Conference on Services Computing (SCC 2004)*, 2004, IEEE Computer Society Press, Shanghai, China, pp. 23-30.

[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, in *Proceedings of the Enterprise Modelling and Information Systems Architectures (EMISA 2005)*, 2005, Klagenfurt, Austria, pp. 208-221.

[3] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, An Approach for Replanning of Web Service Workflows, *Americas Conference on Information Systems (AMCIS 2006)*, 2006, Acapulco, Mexico.

[4] 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)*, 2005, Riva Del Garda, Italy.

[5] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, Heuristics for QoS-aware Web Service Composition, in *Proceedings of the 4th IEEE International Conference Web Services (ICWS 2006)*, 2006, Chicago, USA, pp. 72-82.

[6] G. Canfora, M.D. Penta, R. Esposito, and M.L. Villani, An approach for QoS-aware service composition based on genetic algorithms, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, 2005, Washington DC, USA, pp. 1069-1075.

[7] G. Canfora, M.D. Penta, R. Esposito, and M.L. Villani, QoS-Aware Replanning of Composite Web Services, in *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005)*, 2005, IEEE Computer Society, Orlando, USA, pp. 121-129.

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

[9] 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*, vol. 1, no. 3, 2004, 281-308.

[10] T. Yu and K.-J. Lin, The Design of QoS Broker Algorithms for QoS-Capable Web Services, in *Proceedings of the International Conference on e-Technology, e-Commerce and e-Service (EEE 2004)*, 2004, IEEE Computer Society, Taipei, Taiwan, pp. 17-24.

[11] T. Yu and K.-J. Lin, A Broker-Based Framework for QoS-Aware Web Service Composition, in *Proceedings of the International Conference on e-Technology, e-Commerce, and e-Services (EEE 2005)*, 2005, IEEE Computer Society, Hong Kong, China, pp. 22-29.

[12] T. Yu and K.-J. Lin, Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints, in *Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC 2005)*, 2005, Amsterdam, The Netherlands, pp. 130-143.

[13] 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*, 2004, vol. 30, no. 5, pp.311-328.