A Heuristic Approach to Optimal Service Selection in Service Oriented Architectures

Daniel A. Menascé
Department of Computer Science
George Mason University
Fairfax, VA, USA
menasce@gmu.edu

Emiliano Casalicchio
Dipartimento di Informatica Sistemi e Prod.
Università di Roma "Tor Vergata"
Roma, Italy
casalicchio@ing.uniroma2.it

Vinod Dubey
The Volgenau School of Information Technology and Engineering
George Mason University
Fairfax, VA, USA
vdubey@gmu.edu

ABSTRACT

Service Oriented Architectures (SOA) enable a multitude of service providers (SP) to provide loosely coupled and interoperable services at different Quality of Service (QoS) and cost levels. This paper considers business processes composed of activities that are supported by service providers. The structure of a business process may be expressed by languages such as BPEL and allows for constructs such as sequence, switch, while, flow, and pick. This paper considers the problem of finding the set of service providers that minimizes the total execution time of the business process subject to cost and execution time constraints. The problem is clearly NP-hard. However, the paper presents an optimized algorithm that finds the optimal solution without having to explore the entire solution space. This algorithm can be used to find the optimal solution in problems of moderate size. A heuristic solution is also presented and experimental studies that compare the optimal and heuristic solution show that the average execution time obtained with a heuristic allocation of providers to activities does not exceed 6% of that of the optimal solution.

Categories and Subject Descriptors

C.4 [**Modeling Techniques**]: Experimentation; D.2.11 [**Software Architectures**]: Patterns; D.4.8 [**Performance**]: Stochastic Analysis; G.1.6 [**Optimization**]

General Terms

Performance, Experimentation

Keywords

Service Oriented Architecture, service composition, QoS, optimization, heuristic

WOSP'08, June 24–26, 2008, Princeton, New Jersey, USA.

1. INTRODUCTION

There has been significant recent interest in Service Oriented Architectures (SOA) [4, 5, 8, 16, 20, 22]. The SOA model enables a multitude of service providers (SP) to provide loosely coupled and interoperable services at different Quality of Service (QoS) and cost levels in a number of service domains. This provides a unique opportunity for businesses to dynamically select services that better meet their business and QoS needs in a cost-effective manner. SOAs enable service composition and facilitate enterprises to reengineer their business processes (BP), commonly defined as *"...a structured set of activities designed to produce a specific output..."* [9, 12, 14, 23]. An important consideration in service composition is the selection of service providers in a way that meets the specific QoS requirements and cost constraints of the resulting business process.

The Business Process Execution Language (BPEL) [1] can be used to model business processes in a SOA. The execution of a business process is coordinated by a service broker (or broker for short). The business activities that compose the BP are implemented by services and executed by service providers, and invoked by service consumers using the Web Services communication paradigm. BPEL has emerged as a standards-based service orchestration technology that provides an XML-based grammar for describing the control logic required to orchestrate Web services participating in a process flow [1]. Business processes defined in BPEL are portable and can be executed in any BPEL-compliant process engine.

We consider a market of services in which SPs provide services at different QoS and cost levels. In this environment, it makes sense to investigate mechanisms to properly select a set of services that when composed together satisfy the business process QoS needs and cost constraints. This problem is referred in the literature as the QoS-aware Service Selection or Optimal Service Selection problem [2, 3, 6, 7, 18, 19, 25, 26].

There are two main problems to tackle in order to solve the optimal service selection problem. The first one is to provide a performance model of the business process that takes into account the composition of interacting services. The second one is to determine the service selection that satisfies the QoS and cost constraints.

To solve the first problem we provide a performance model that takes into account the business process structure, including cycles, parallel activities, and conditional branches.

For some performance metrics (e.g., cost, availability, reputation) the composition is a trivial linear combination of the performance measure of the composing services. On the contrary, for other metrics such as execution time, we have a non linear function of the performance level of the services being composed.

Several approaches can be used to solve the service selection problem: linear and nonlinear programming [2, 7, 25, 26], heuristics [3, 6], and predictive performance models [18]. We use a nonlinear programming formulation and propose an algorithm to find the optimal solution that avoids exploring the entire solution space and a heuristic algorithm to find a suboptimal solution.

Current proposals use exact algorithms or heuristics (e.g., [3] or genetic algorithms in [6]) to solve the QoS-aware (optimal) service selection problem for each request, whose exact solution has an exponential complexity. In [25], the authors define the problem as a multi-dimension multi-choice 0-1 knapsack one as well as a multi-constraint optimal path problem. A global planning approach to select an optimal execution plan by means of integer programming is used in [26]. In [2], the authors model the service composition as a mixed integer linear problem where both local and global constraints are taken into account. A linear programming formulation and flow-based approach is proposed in [7]. There, the authors consider not only sequential composition of services but also cycles and parallel activities. Algorithms for the selection of the most suitable candidate services to optimize the overall QoS of a composition are also discussed in [13]. A different approach, based on using a utility function as the QoS measure, is described in [18], where the authors propose a service selection mechanism based on a predictive analytical queuing network performance model. Other contributions to service selection and composition can be found in [26, 24].

The contributions of this paper are threefold. First, it provides a formulation of the optimization problem that is independent of the business process structure and of the characterization of the performance of the SPs. It is assumed that the SPs provide the CDF and pdf of their QoS metrics. These can be either obtained by the SPs themselves by analyzing historical data or by external agents that monitor the SPs at regular intervals and fit the data to a distribution. Second, it proposes an algorithm to find the optimal solution in a way that avoids an exhaustive search of the solution space. Finally, it proposes an algorithm to conduct a heuristic search of the solution space in order to find a sub-optimal solution that is very close to the optimal solution but is obtained by examining a drastically reduced number of selections. In fact, the experimental studies reported in this paper show that the heuristic solution comes very close to the optimal solution (less than 6% worse) after having examined a very small number of possible solutions. Both algorithms presented here (i.e., the optimal and suboptimal solution) work for any kind of complex BP structure that may include parallel activities, cycles, and conditional selection of activities.

The paper is organized as follows. Section 2 introduces the notation and formally defines the problem. The optimal solution approach is described in detail in Section 3 and the heuristics approach is described in Section 4. Experimental results are discussed in Section 5. Section 6 concludes the paper.

2. BACKGROUND AND NOTATION

We use the average execution time of the business process as its main QoS metric. As previously discussed, this metric is a nonlinear function of the execution times of individual business activities and depends on the BP structure and composition constructs used. The extension to other performance metric is straightforward.

We assume that the probability density function (pdf) and cumulative distribution function (CDF) of the execution time are known. We also assume that the execution cost of each business activity provided by the SPs is given.

Let,

- A business process B be composed of N business activities $a_i, i = 1, \cdots, N$.
- $R_{\max}$ be the maximum average execution time for B.
- $C_{\max}$ be the maximum cost for the execution of B.
- $S_i = \{s_{i_1}, ..., s_{i_k}\}$ be the set of service providers that can be used in the implementation of business activity a_i.
- $R_{i,j}$ be the execution time for business activity a_i when implemented by service provider $s_{i_j} \in S_i$. $R_{i,j}$ is a random variable with a probability density function $p_{i,j}$ and a cumulative distribution function $P_{i,j}$.
- $C_{i,j}$ be the execution cost of business activity a_i when it is implemented by service provider $s_{i_j} \in S_i$.
- $\mathcal{Z}$ be the set of all possible service provider selections of the business activities of B.
- $z \in \mathcal{Z}$ be a service selection of N service providers that support the execution of business process B.
- z^k be a subselection in which only the first k activities of a business process have service providers assigned to them.
- $R(z)$ and $C(z)$ be the average execution time and the cost for associated with service selection z, respectively.

The Optimal Service Selection problem is formulated as a nonlinear programming optimization problem where the objective is to find a service selection z that minimizes the average execution time subject to cost constraints:

$$\begin{aligned} & \min \ R(z) \\ \text{subject to} & \\ & R(z) \leq R_{max} \\ & C(z) \leq C_{max} \\ & z \in \mathcal{Z} \end{aligned}$$

As show in section 3.2, $R(z)$ can be a complex nonlinear function that can be obtained from well known results from order statistics.

2.1 Problem Solution

The Optimal Service Composition problem formulated above is solved here using two different approaches.

The first is an optimal solution approach (Optimal Service Selection) that avoids doing an exhaustive search of the solution space $\mathcal{Z}$ by dropping all selection sequences $z = s_{1_*}, \ldots, s_{k_*}, \ldots, s_{N_*}$ that have a non-feasible sub selection $z^k = s_{1_*}, \ldots, s_{k_*}$. A non-feasible selection is one that violates the execution time or cost constraint. It should be noted that this *reduced search* still finds the optimal selection without necessarily analyzing every possible selection. This approach may work well for small to moderate problems but it is still NP-hard in general.

The second approach (Heuristic Service Selection) adopts a heuristic solution that reduces the problem complexity.

The first required step for both the optimal reduced search and the heuristic is to be able to extract from the BPEL code that describes the business process, an expression for the global average execution time and another for the total execution cost. This expression needs to take into account the structure of the business process as well as the execution times and cost of the individual business activities.

3. OPTIMAL SERVICE SELECTION

BPEL offers different constructs to combine business activities into a business process. The business logic is a structured activity obtained by putting together elementary business activities (in the following, the term business process and business logic are used alternatively). Each business activity is essentially a synchronous or asynchronous invocation of a Web service operation. A structured activity includes sequential control, non-deterministic choice, and concurrency and synchronization of elementary activities. More specifically, the main structured activities in BPEL are:

- ordinary sequential control between activities: `<sequence>`, `<switch>`, and `<while>`;
- concurrency and synchronization between activities: `<flow>`;
- nondeterministic choice based on external events: `<pick>`.

Our goal is to compute the execution time R of a business process and its execution cost C. While the execution cost of a process is the sum of the execution costs of the activities of the business process (plus eventually some additional overhead), the execution time depends on how the business activities are structured. For example, if we have a sequence of business activities $a_1, \ldots, a_n$, and a service selection z, the execution time of the business process is $R(z) = \sum_{i=1,\ldots,n} R_{i,j}$ where s_{i_j} is the service provider assigned to activity a_i in z. The execution time of an activity a_i that is repeated n times and that is supported by service provider $s_{i,j}$ is simply $R(z) = n \times R_{i,j}$. In the case of deterministic or non deterministic choices, the computation of the total execution is easily computed as $R(z) = \sum_{i=1,\ldots,n} q_i \times R_{i,j}$ where q_i is the probability that activity a_i is invoked. Finally, the execution time of the parallel execution of n business activities is given by $R(z) = \max_{i=1,\ldots,n}\{R_{i_j}\}$.

Two problems need to be solved: the first is how to compute an expression for R, the execution time of a business process B, as a function of the individual execution times of the various activities in B taking into account the structure of B. Note that R is a random variable. The second problems is that of computing $E[R]$, the average execution time.

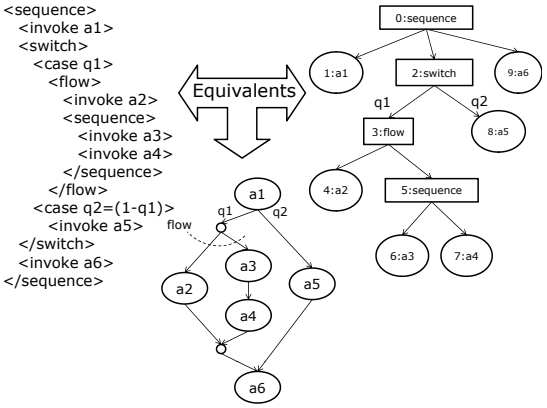

Figure 1: An example of a BPEL business process on the left, the corresponding BPTree on the right, and an execution graph on the middle.

.

3.1 Computation of R

A BPEL process can be represented in two equivalent way: as a directed execution graph or as an execution tree, called BPTree (see Fig. 1). It is possible to compute an expression for R and then for $E[R]$ by visiting the BPTree in post-order. We use an example to illustrate the approach. Consider the simple BPEL process shown in Fig. 1 that includes the `<sequence>`, `<switch>`, and `<flow>` constructs. For simplicity, we used a pseudo BPEL syntax. The corresponding execution tree is shown on the right of Fig. 1. We introduce two types of nodes for such a tree: construct nodes, represented as boxes, and activity nodes, represented as circles. Each node has associated with it an order number and a label. For example, node $(4 : a2)$ is the 4^{th} node and represents activity a_2 and node (2:`switch`) is the second node and represents the `switch`. The BPTree can be constructed from left to right, parsing the BPEL code from the beginning to the end, and associating the corresponding node to each BPEL tag. The leaves of a BPTree are always activity nodes. In the example, nodes are numbered according to the above rule.

An expression for the execution time of a business process can be automatically obtained from the BPTree using Algorithm 1, which visits the BPTree in post order, computing the response time for the different BPEL constructs according to the rules defined in lines 8 through 13, and described in the previous section. In the algorithm, children(i) denotes the set of nodes that are children of node i and label(i) denotes the label of node i. Applying the algorithm to the example of Fig. 1 we obtain:

$$R = R_1 + q_1 \times \max\{R_2, (R_3 + R_4)\} + q_2 \times R_5 + R_6 \quad (1)$$

3.2 Computation of $E[R]$

The computation of the average execution time $E[R]$ can be done using the property that the expected value of a lin-

Algorithm 1 Compute the execution time of a BPEL process

```
1: function Compute R(node i)
2: if i is a leaf node then
3:    return R_i;
4: else
5:    for all k ∈ children(i) do
6:       R_k =Compute R(k);
7:    end for
8:    if label(i) = sequence then
9:       return R_i = Σ_{k∈children(i)} R_k;
10:   else if label(i) = switch then
11:      return R_i = Σ_{k∈children(i)} q_i × R_k;
12:   else if label(i) = flow then
13:      return R_i = max_{k∈children(i)} R_k;
14:   end if
15: end if
```

ear combination of random variables is a linear combination of the expected values of these random variables. Thus,

$$E[q_1 \times R_1 + ... + q_n \times R_n] = \sum_{i=1}^{n} q_i \times E[R_i] \qquad (2)$$

The main challenge occurs when the business process has a `<flow>`, which yields a term in the expression of $E[R]$ that requires the computation of the expected value of a random variable defined as the maximum of several random variables. The expected value of a maximum of a set of independent random variables can be obtained using order statistics arguments as indicated below [10].

$$E[\max_{i=1}^{n} R_i] = \int_0^{\infty} x \left[\prod_{i=1}^{n} P_i(x)\right] \sum_{i=1}^{n} \frac{p_i(x)}{P_i(x)} dx \qquad (3)$$

where $R_1, \ldots, R_n$ are independent random variables, $p_i(x)$ is the pdf of R_i and $P_i(x)$ is the CDF of R_i.

Applying equations (2) and (3) to equation (1) we obtain:

$$\begin{aligned} E[R] \;=\; & E[R_1] + \\ & q_1 \times \int_0^{\infty} x \left([P_2(x) \times P_{3+4}(x)] \times \left[\frac{p_2(x)}{P_2(x)}\right] \right) dx + \\ & q_1 \times \int_0^{\infty} x \left([P_2(x) \times P_{3+4}(x)] \times \left[\frac{p_{3+4}(x)}{P_{3+4}(x)}\right] \right) dx + \\ & q_2 \times E[R_5] + \\ & E[R_6] \end{aligned}$$

where $p_{3+4}(x)$ and $P_{3+4}(x)$ are, respectively, the pdf and CDF of the random variable $R_3 + R_4$. The probability density function of the sum of two independent random variables, is the convolution of their individual density pdfs. Thus,

$$\begin{aligned} p_{3+4}(x) &= \int_{y=0}^{y=+\infty} p_3(y) p_4(x-y) dy \\ P_{3+4}(x) &= \int_{y=0}^{y=x} p_{3+4}(y) dy. \end{aligned}$$

Our implementation uses numerical integration methods to solve the integrals above.

3.3 JOSeS's algorithm

The optimal solution to the optimal service allocation problem can be obtained using Algorithm 1 and Eq. (3). A naive and inefficient way of doing it would be to generate all possible service selections $z \in \mathcal{Z}$ and evaluate $E[R(z)]$ and $C(z)$ for each z. After that, for example, we may order the solutions by increasing response time and decreasing cost, and choose the first allocation of the list (i.e., the fastest) such that $E[R(z)] \leq R_{max}$ and $C(z) \leq C_{max}$. Unfortunately, computing all possible solutions is an NP-hard problem. Indeed, $|\mathcal{Z}| = \prod_{i=1}^{N} |S_i| \approx O(m^N)$, where m is the average of $|S_i|$.

For example, consider the business process of Fig. 1 and suppose the service providers for activities $a_1, ..., a_6$ are as shown in Table 1. Then, the number of possible allocations is given by $\prod_{i=1}^{6} |S_i| = 36$, which shows that the size of the solution space is big even for such a small problem. If we consider a business process composed of one hundred activities and we have an average of 10 service providers for each activity, the solution space has cardinality of 10^{100}, which shows the impossibility to compute the optimal solution when N and $|S_i|$ grow.

The computational complexity of obtaining the optimal solution can be somewhat reduced as explained below. We propose an algorithm, that we call Jensen-based Optimal Service Selection (JOSeS), to accomplish this. This algorithm does not require one to generate the entire solution space $\mathcal{Z}$, but only a subset of the solution space where each point represents a feasible solution, i.e., one that satisfies the cost and execution time constraints.

The name of the algorithm is derived from our use of Jensen's inequality [21] to obtain a lower bound on the average of the maximum of independent random variables:

$$E[\max\{R_1, ..., R_n\}] \geq \max\{E[R_1], ..., E[R_n]\}. \qquad (4)$$

We use this inequality to compute a lower bound on $E[R]$ in the case of `flow`s of activities. This allows one to reduce the cost of computing the optimal solution by reducing the number of useless computations of the average execution time of a `flow` of activities. Instead of the expensive computation of $E[\max\{R_1, ..., R_n\}]$ (see Eq. (3)), we compute $\max\{E[R_1], ..., E[R_n]\}$ and check if $\max\{E[R_1], ..., E[R_n]\} \geq R_{\max}$. Only if the lower bound of the average execution time of an allocation z satisfies the time constraint it is worth computing $E[R(z)]$ through Eq. (3).

JOSeS's algorithm receives as input an expression for the execution time, such as Eq. (1), and analyzes partial selections of service providers. For each partial selection, the partial execution time and partial cost are compared against their respective constraints. If any of the constraints is violated, then that partial selection is dropped and all other selections that have the same partial selection as a subselection will not even be considered.

Before we present JOSeS's algorithm in detail, it is helpful to consider in the example below how its operation applies to Eq. (1) and the data in Table 1.

Business Activities	Service Providers	R(avg.)	C	R/C
a_1	s_{1_1}, s_{1_2}, s_{1_3}	1.5, 2, 3.5	3, 2, 1	0.5, 1, 3.5
a_2	s_{2_1}, s_{2_2}	1, 1.5	1, 0.5	1, 3
a_3	s_{3_1}	2	1.5	1.33
a_4	s_{4_1}, s_{4_2}	0.7, 1.2	1, 0.5	0.7, 2.4
a_5	s_{5_1}, s_{5_2}, s_{5_3}	0.8, 1.2, 1.5	2, 1, 0.5	0.4, 1.2, 3
a_6	s_{6_1}	2.3	1	2.3

Table 1: Sets of service providers for the execution of business activities $a_1, ..., a_6$.

$$
\begin{aligned}
E[R] &\geq E[R_1] \\
E[R] &\geq E[R_1] + q_1 \times E[R_2] \\
E[R] &\geq E[R_1] + q_1 \times \max\{E[R_2], E[R_3]\} \\
E[R] &\geq E[R_1] + q_1 \times \max\{E[R_2], E[R_3] + E[R_4]\} \\
E[R] &\geq E[R_1] + q_1 \times \max\{E[R_2], E[R_3] + E[R_4]\} + \\
&\quad q_2 E[R_5] \\
E[R] &\geq E[R_1] + q_1 \times \max\{E[R_2], E[R_3] + E[R_4]\} + \\
&\quad q_2 E[R_5] + E[R_6].
\end{aligned}
$$

The algorithm considers sub-expressions and sub-selections to each sub expression. Jensen's inequality is used to reduce the number of computations of the expected value of the maximum of a number of random variables. For example suppose that we are considering a sub selection $z^4 = \{s_{1_1}, s_{2_1}, s_{3_1}, s_{4_1}\}$ and that $q_1 = 0.3$, $R_{\max} = 15$ and $C_{\max} = 4$. Consider also the fourth inequality (see above). Then the execution time is bounded below by $1.5 + 0.3 \times \max\{1, 2+0.7\} = 2.31$ and the cost of the partial allocation is $3 + 0.3 \times (1 + 1.5 + 1) = 4.05$. This allocation violates the cost constraint and therefore all service selections that have $z^4 = \{s_{1_1}, s_{2_1}, s_{3_1}, s_{4_1}\}$ as a subsequence need not be considered.

The steps of JOSeS's algorithm are detailed in what follows (Algorithm 2). Let us define the ordered list $l_k = \{s_{k_1}, ..., s_{k_{n_k}}\}$ of service providers implementing activity a_k. `next` (k) is a function that returns the next, not yet evaluated, service provider in l_k, or returns *null* if all the SPs in l_k were already evaluated. `reset (k)` is a function that sets all SPs in all lists l_j $(j = k, \ldots, N)$ as "not-visited" and resets the pointer to the lists so that when `next`(j) is invoked it returns the first element in the list l_j $(j = k, \ldots, N)$. We use k $(1, \ldots, N)$ as an activity counter. In Algorithm 2, z stands for the current allocation of service providers being examined and s stands for a generic service provider. The notation $z||s$ indicates that service provider s is concatenated to the right of partial allocation z. The notation $z\diamond$ indicates that the last service provider of allocation z should be removed. For example, if $z = afg$ and $s = h$ then $z||s$ results in $z = afgh$. If $z = afgh$, then $z\diamond$ becomes afg. The notation $\mathcal{L}(E[R(z)])$ stands for the lower bound on the average execution time obtained through Jensen's inequality for the execution time of allocation z.

If we apply the JOSeS's algorithm to our example, with constraints $R_{\max} = 8$ and $C_{\max} = 4$, we have to evaluate the value of $E[R(z)]$ only on 19 of the total 36 possible cases. The optimal value obtained is $E[R(z)] = 7.97$ and $C(R(z)) = 4$ for the allocation $z = s_{1_3}, s_{2_2}, s_{3_1}, s_{4_1}, s_{5_2}, s_{6_1}$.

4. HEURISTIC SERVICE SELECTION

The goal of the proposed heuristic solution is to reduce the cost of finding the optimal solution, providing a sub-optimal selection as close as possible to the optimal.

Algorithm 2 Compute the Optimal Solution

```
1: function AdvanceList (k) returns (s)
2: s ← next (k);
3: if s = NULL then
4:   if k > 1 then
5:     reset (k);
6:     k ← k − 1;
7:     z ← z◇
8:     AdvanceList (k);
9:   else
10:    return s
11:  end if
12: else
13:  return s;
14: end if
15: end function
16:
17: function OptimalSolution()
18: reset (1); k ← 1; /* initialize activity pointers */
19: s ← AdvanceList (k); z ← s; /* initialize solution */
20: z_opt ← any allocation in 𝒵;
21: while s ≠ NULL do
22:   if k < N then
23:     if (𝓛(E[R(z)] ≤ R_max) and (C(z) ≤ C_max) then
24:       k ← k + 1
25:     else
26:       z ← z◇
27:     end if
28:   else
29:     if (E[R(z)] ≤ R_max) and (C(z) ≤ C_max) then
30:       if E[R(z)] < E[R(z_opt)] then
31:         z_opt ← z
32:       end if
33:     end if
34:     z ← z◇
35:   end if
36:   AdvanceList (k);
37:   z ← z||s
38: end while
39: return z_opt
40: end function
```

Figure 2 shows the solution space and the feasible solution space of our problem. The solution space is the area delimited by the dashed lines, which indicate the lower and upper bounds for cost and execution time of the business process. The lower bound C_C for the execution cost is the cost obtained by selecting the cheapest service providers for each activity. Similarly, the upper bound R_S for the execution time can be obtained by selecting the slowest service provider for each business activity. On the contrary, the upper bound C_E for the execution cost is obtained by selecting the most expensive service provider for each activity, and the lower bound R_F for the execution time is obtained by selecting the fastest service provider for each activity. The feasible solution space is represented by the dotted area and is the portion of the solution space delimited by the lower bound for execution time and execution cost and by the time and cost constraints (bold lines)

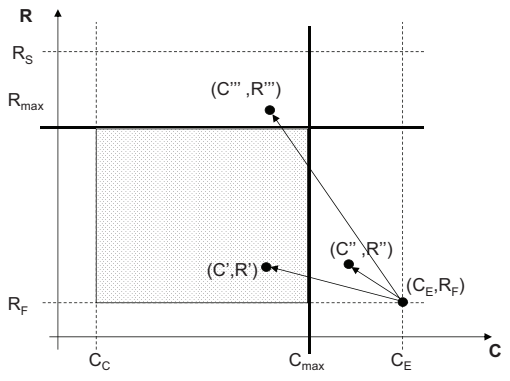

Figure 2: A conceptual representation of the solution space and of the feasible solution space

The proposed heuristic is based on the following idea (see Fig. 2). We start from the service selection z_0 characterized by the lowest execution time, i.e., the point (C_E, R_F). Assume that this point is outside the feasible solution space and that the cost constraint is violated. To find a feasible solution as close as possible to the optimum, we have to chose a selection z' that moves the problem solution inside the dotted area, say the point (C', R'). To choose the solution z' we replace the service provider that provides the highest reduction in the execution cost C with the lowest increase in the execution time R. To determine such provider, we need to compute the ratio

$$\Delta_{i,j,j'} = p_i \times \frac{C_{i,j} - C_{i,j'}}{R_{i,j'} - R_{i,j}} \quad j^{'} > j \tag{5}$$

for each activity a_i $(i = 1, \cdots, N)$. In Eq. (5), j represents the service provider allocated to activity a_i, j' represents an alternate service provider for a_i, and p_i is the probability that activity a_i is executed in the business process. This probability is a function of the structure of the business process and its branching probabilities. For example, the probability that activity a_4 is executed in the business process of Fig. 1 is q_1. We then select the activity for which there is an alternate provider that maximizes the value of the ratio for all such ratios. More precisely,

$$(k, m) = \text{argmax}_{i=1,\cdots,N; j' \neq j} \{\Delta_{i,j,j'}\}. \tag{6}$$

According to Eq. (6), the service provider m when replacing service provider j in activity k yields the maximum value for the ratios Δ.

We then evaluate the execution cost and execution time for the new service selection z'. If the constraints are satisfied we have a suboptimal solution. Otherwise, there are two possibilities: if the cost constraint is still violated and the time constraint is not yet violated, we are in point such as (C'', R'') and we have to repeat the above mentioned process, i.e., the selection of a new service provider that maximizes the ratio Δ among all activities. If the execution time constraint is violated but the cost constraint is satisfied, we are at point (C''', R''') and we cannot accept such solution as we would continue to violate the exection time constraint at any further attempt of cost reduction. Then, we select the service provider that has the second best ratio Δ. The process is repeated until a feasible solution is found.

Algorithm 3 shows the detailed steps of the heuristic. Let $\mathcal{L}_\Delta$ be the set of $\Delta_{i,j,j'}$ for all activities a_i and service providers s_{i_j} and $s_{i_{j'}} \in S_i$ where $j^{'} > j$. Let $z \ominus_k s$ stand for the operation of removing from solution z provider s for activity k. Similarly, let $z \oplus_k s$ denote the addition to solution z of provider s to activity k.

Algorithm 3 Heuristic Solution

```
1: function ComputeDeltaSet (z) returns (DeltaSet)
2: DeltaSet = {Δ_{i,j,j'}, ∀a_i, ∀s_{i_j} and s_{i_j'} ∈ S_i, and j' > j};
3: return DeltaSet
4: end function
5:
6: function heuristic()
7: Find z such that E[R(z)] = R_F;
8: if (E[R(z)] ≤ R_max) and (C(z) ≤ C_max) then
9:    return z
10: end if
11: L_Δ = ComputeDeltaSet (z)
12: while L_Δ ≠ ∅ do
13:    Δ_{k,j,j'} = max{L_Δ};
14:    s ← s_{k,j};
15:    z ← z ⊖_k s;
16:    z ← z ⊕_k s_{k,j'};
17:    if E[R(z)] ≤ R_max then
18:       if C(z) ≤ C_max then
19:          return z;
20:       else
21:          L_Δ = ComputeDeltaSet (z)
22:       end if
23:    else
24:       z ← z ⊖_k s_{k,j'};
25:       z ← z ⊕_k s;
26:       L_Δ ← L_Δ − {Δ_{k,j,j'}}
27:    end if
28: end while
29: return infeasible solution
30: end function
```

Now we use the proposed heuristic to compute the suboptimal solution for the business process in Fig. 1 using the service providers listed in Table 1. Figure 3 shows the solution space, the admissible region and the values for R_F, C_E, R_S, C_C, $R_{\max}$ and $C_{\max}$. The steps in this example are given below for $R_{\max} = 6$, $C_{\max} = 6$, and $q_1 = q_2 = 0.5$:

- *First assignment: (6.75, 5.648).* $z_F = s_{1_1} s_{2_1} s_{3_1}\ s_{4_1} s_{5_1}\ s_{6_1}$, which is the service selection with the lowest execution time $E[R(z_F)] = R_F = 5.648$ and $C(z_F) = 6.75$. This solution violates the cost constraint.

- *Step I: (5.75, 6.148).* $\mathcal{L}_\Delta = \{\Delta_{112}, \Delta_{113}, \Delta_{212}, \Delta_{412}, \Delta_{512}, \Delta_{513}\} = \{2, 1.0, 0.5, 0.5, 1.25, 1.07\}$. Then, max $\{\mathcal{L}_\Delta\} = \Delta_{112} = 2$ and the new assignment is $z_1 = s_{1_2} s_{2_1} s_{3_1} s_{4_1} s_{5_1} s_{6_1}$, with a corresponding average execution time and cost equal to $E[R(z_1)] = R(z_F)+0.5 = 6.148$ and $C(z_1) = C(z_F) - 1 = 5.75$. This assignment has to be discarded because it violates the execution time constraint.

- *Step II: (6.25, 5.848).* Removing Δ_{112} from $\mathcal{L}_\Delta$ (line 26 of Algorithm 3) we obtain $\mathcal{L}'_\Delta = \{\Delta_{113}, \Delta_{212}, \Delta_{412}, \Delta_{512}, \Delta_{513}\}$, $\max\{\mathcal{L}'_\Delta\}$ is $\Delta_{512} = 1.25$. Computing again the average execution time and execution cost for assignment $z_2 = s_{1_1} s_{2_1} s_{3_1} s_{4_1} s_{5_2} s_{6_1}$, we obtain the new values for $E[R(z_2)] = R(z_F) + 0.5 \times 0.4 = 5.848$ and $C(z_2) = C(z_F) - 1 \times 0.4 = 6.25$.

- *Step III: (5.25, 6.348).* Because the previous solution still violates the cost constraint, we need to compute again $\mathcal{L}_\Delta$ (line 21), which becomes $\mathcal{L}''_\Delta = \{\Delta_{112}, \Delta_{113}, \Delta_{212}, \Delta_{412}, \Delta_{523}\} = \{2, 1.0, 1.0, 0.5, 0.833\}$ and $\max\{l''_\Delta\} = \Delta_{112} = 2$. The new solution is $z_3 = s_{1_2} s_{2_1} s_{3_1} s_{4_1} s_{5_2}\ s_{6_1}$ does not violate the cost constraint but has an execution time of $E[R(z_3)] = 6.348$, which still violates the execution time constraint.

- *Step IV: (4.25, 7.848).* We remove $\Delta_{max} = \Delta_{112}$ obtaining $\mathcal{L}'''_\Delta = \{\Delta_{113}, \Delta_{212}, \Delta_{412}, \Delta_{523}\} = \{1, 0.5, 0.5, 0.83\}$. Δ_{max} is Δ_{113} and the new solution is $z_4 = s_{1_3} s_{2_1}\ s_{3_1} s_{4_1} s_{5_2} s_{6_1}$ with an execution time of $E[R(z_3)] = 7.848$, which violates the constraint.

- *Step V: (6, 5.998).* We need one more step, and we remove Δ_{113} from the $\mathcal{L}_\Delta$ obtaining $\mathcal{L}''''_\Delta = \{\Delta_{212}, \Delta_{412}, \Delta_{523}\} = \{0.5, 0.5, 0.833\}$. Δ_{max} is now $\Delta_{523} = 0.833$ and the corresponding solution $z_4 = s_{1_1} s_{2_1}\ s_{3_1} s_{4_1} s_{5_3} s_{6_1}$ is acceptable: $E[R(z_4)] = R(z_2) + 0.5 \times 0.3 = 5.998$ and $C(z_4) = C(z_2) - 0.5 \times 0.5 = 6$.

Figure 3 shows the evolution of the heuristic in the solution space and the optimal solution that is $(C_{\text{opt}}, R_{\text{opt}}) = (6, 5.998)$. The solution z is evaluated six times (5 steps plus the initial selection for z_F).

5. EXPERIMENTS

We implemented the heuristic (Algorithm 3) and the optimal algorithm (Algorithm 2) to conduct experiments aimed at evaluating the efficiency of the former. The business process used has the structure shown in Fig. 1. We assume that the execution time of each service provider s is exponentially distributed with an average execution time equal to $E[R_s]$ specified in Fig. 4 for each service provider. The cost of obtaining an average execution time $E[R_s]$ from service provider s is assumed to be equal to $1/E[R_s]$. In other words, the cost decreases with the inverse of the average service time offered by a service provider.

For each experiment we obtained for the heuristic and optimal algorithms: a) the average execution time of the business process, b) the cost of the business process, and c) the number of solutions evaluated. We varied the number of SPs per activity as follows: 2, 3, 5, 7, and 9. We also used different values for the execution and cost constraints.

The results of the experiments are shown in Table 2, which is divided into sections according to the number of SPs available per activity. For the cases where there are $n < 9$ SPs per activity we use the first n SPs for each activity shown in Fig. 4. We also varied the cost constraint and for the case of 9 SPs we varied both the execution time and cost constraints.

We can draw the following observations from Table 2 :

- For all experiments, the execution time of the business process obtained with the heuristic algorithm is at best the same as that obtained with the optimal solution (runs 1, 8, 12, 16, and 20) and at worse 5.8% higher (runs 6, 10, 14, and 18).

- The cost of the heuristic solution varies from the same cost obtained with the optimal algorithm (runs 1, 4, 8, 112, 16, and 20) to 95.3% lower (run 2).

- While the optimal algorithm evaluates a large number of solutions (which is still far lower than would be required by an exhaustive search), the heuristic algorithm did not require more than 18 solutions to be evaluated (runs 21 and 23). Consider for example run no. 21. An exhaustive search would require $9^6 = 531,441$ solutions to be evaluated. The optimal algorithm requires 369,927 solutions to be evaluated and the heuristic only 18.

6. CONCLUDING REMARKS

The SOA model brings several new benefits to software design and architecture by enabling re-use and sharing of components through dynamic discovery. Service composition enables complex applications to be put together in a variety of ways. Each possible selection of services brings different levels of QoS and cost. Thus, there is a need to devise fast and efficient mechanisms that can be used for dynamic service selection among a set of service providers.

This paper presented such an efficient mechanism that, in the experiments reported and all experiments carried out and not-reported due to lack of space, comes very close to the optimal solution (less than 6% worse) after having examined a very small number of possible solutions. All 23 experiments reported in the paper use a BP with the same topology but with a varying number of SPs per activity and different execution time and cost constraints. As part of the ongoing work, we are applying the algorithms to a much larger set of BPs that are randomly generated with different topologies.

We are also currently working on an interesting extension of the work reported here. It has to do with the situation in which a service provider may offer several related but different services, as is often the case of Web Services. For example, an airline reservation SP may offer the following services: reserve flight, check seat availability, and book flight. Consider a travel planning business process that includes the following activities: check seat availability, reserve flight, and purchase ticket. If a given service provider is selected for the reserve flight activity, then the same service provider should be used for the book ticket activity. Thus, the current work needs to be extended to allow for service selection

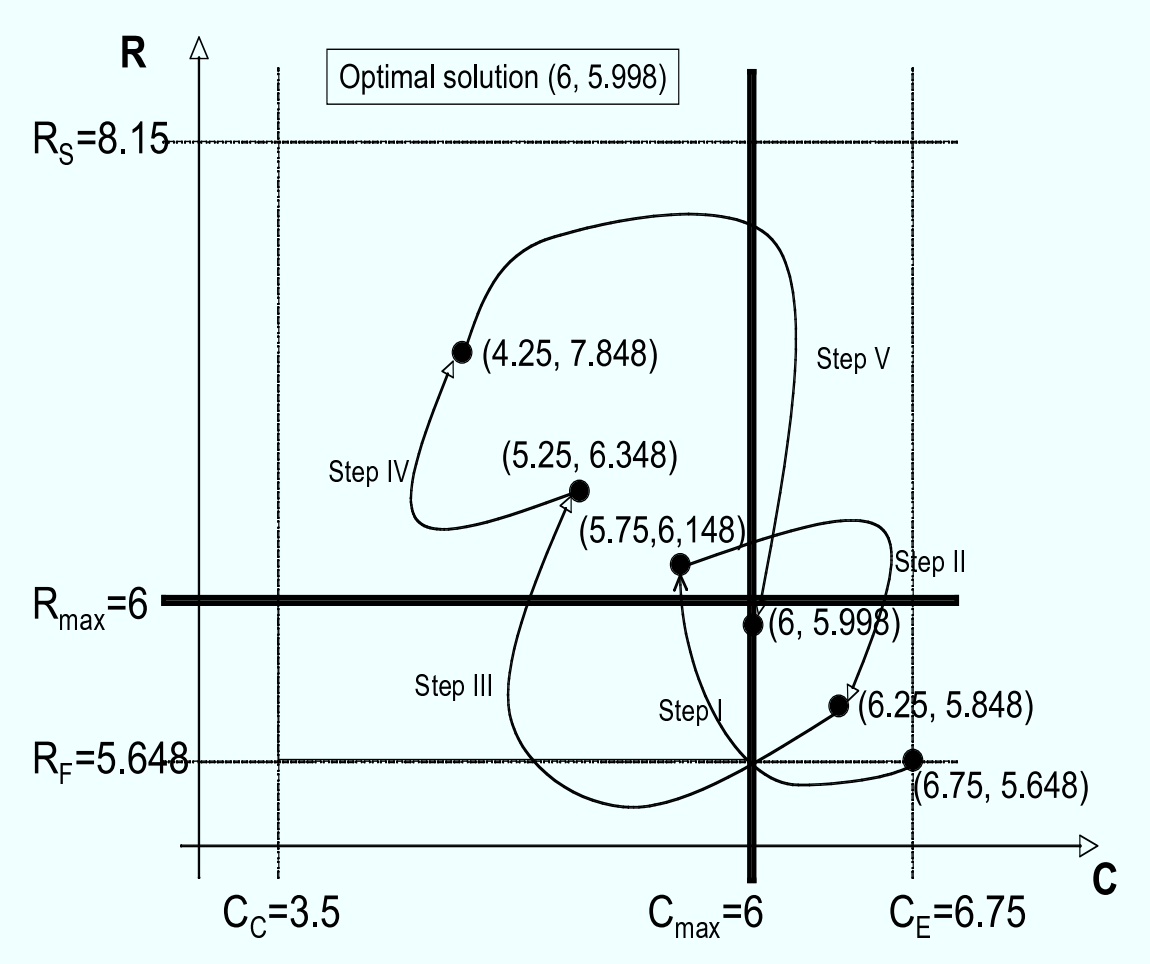

Figure 3: An example of the evolution of the heuristic solution.

Activity 1 SPs		
s11	1.5	0.67
s12	2.0	0.50
s13	2.5	0.40
s14	3.0	0.33
s15	3.5	0.29
s16	4.0	0.25
s17	4.5	0.22
s18	5.0	0.20
s19	5.5	0.18

Activity 2 SPs		
s21	1.0	1.00
s22	1.5	0.67
s23	2.0	0.50
s24	2.5	0.40
s25	3.0	0.33
s26	3.5	0.29
s27	4.0	0.25
s28	4.5	0.22
s29	5.0	0.20

Activity 3 SPs		
s31	2.0	0.50
s32	2.5	0.40
s33	3.0	0.33
s34	3.5	0.29
s35	4.0	0.25
s36	4.5	0.22
s37	5.0	0.20
s38	5.5	0.18
s39	6.0	0.17

Activity 4 SPs		
s41	0.7	1.43
s42	1.2	0.83
s43	1.7	0.59
s44	2.2	0.45
s45	2.7	0.37
s46	3.2	0.31
s47	3.7	0.27
s48	4.2	0.24
s49	4.7	0.21

Activity 5 SPs		
s51	0.8	1.25
s52	1.2	0.83
s53	1.5	0.67
s54	1.8	0.56
s55	2.1	0.48
s56	2.4	0.42
s57	2.7	0.37
s58	3.1	0.32
s59	3.4	0.29

Activity 6 SPs		
s61	2.3	0.43
s62	2.7	0.37
s63	3.1	0.32
s64	3.5	0.29
s65	3.9	0.26
s66	4.3	0.23
s67	4.7	0.21
s68	5.1	0.20
s69	5.5	0.18

Figure 4: Input data for the experiments.

Run No.	SPs per activity	$(R_{\max}, C_{\max})$	Execution Time			Cost			No. Solutions Examined	
			Optimal	Heuristic	H/O	Optimal	Heuristic	H/O	Optimal	Heuristic
1	2	(8.0, 2.3)	7.079	7.079	1.000	2.287	2.287	1.000	32	6
2	2	(8.0, 2.5)	6.406	6.679	1.043	2.468	2.352	0.953	50	5
3	2	(8.0, 3.0)	5.848	5.876	1.005	2.982	2.893	0.970	64	2
4	2	(8.0, 3.5)	5.648	5.648	1.000	3.190	3.191	1.000	64	1
5	3	(8.0, 2.0)	7.570	7.670	1.013	1.998	1.962	0.982	366	9
6	3	(8.0, 2.5)	6.313	6.679	1.058	2.435	2.351	0.966	696	5
7	3	(8.0, 3.0)	5.848	5.876	1.005	2.982	2.893	0.970	729	2
8	3	(8.0, 3.5)	5.648	5.648	1.000	3.191	3.191	1.000	729	1
9	5	(8.0, 2.0)	7.388	7.701	1.042	1.995	1.990	0.997	13570	9
10	5	(8.0, 2.5)	6.313	6.679	1.058	2.435	2.351	0.966	15560	5
11	5	(8.0, 3.0)	5.848	5.876	1.005	2.982	2.893	0.970	15625	2
12	5	(8.0, 3.5)	5.648	5.648	1.000	3.191	3.191	1.000	15625	1
13	7	(8.0, 2.0)	7.388	7.701	1.042	1.995	1.990	0.997	92939	9
14	7	(8.0, 2.5)	6.313	6.679	1.058	2.435	2.351	0.966	97846	5
15	7	(8.0, 3.0)	5.848	5.876	1.005	2.982	2.893	0.970	97951	2
16	7	(8.0, 3.5)	5.648	5.648	1.000	3.191	3.191	1.000	97951	1
17	9	(8.0, 2.0)	7.388	7.701	1.042	1.995	1.990	0.997	269001	9
18	9	(8.0, 2.5)	6.313	6.679	1.058	2.435	2.351	0.966	277776	5
19	9	(8.0, 3.0)	5.848	5.876	1.005	2.982	2.893	0.970	277929	2
20	9	(8.0, 3.5)	5.648	5.648	1.000	3.191	3.191	1.000	277929	1
21	9	(10.0, 1.5)	9.645	9.982	1.035	1.497	1.492	0.996	369927	18
22	9	(10.0, 2.0)	7.388	7.701	1.042	1.995	1.990	0.997	466218	9
23	9	(12.0, 1.5)	9.645	9.982	1.035	1.497	1.492	0.996	424899	18

Table 2: Summary of Experimental Results.

constraints to be expressed within the business process description so that the constraints can be taken into account by the service selection algorithm.

7. REFERENCES

[1] "Web Service - Business Process Execution Language (WS BPEL)," *Version 2.0 - OASIS Committee Draft,* 17th May, 2006.

[2] D. Ardagna and B. Pernici, "Global and Local QoS Guarantee in Web Service Selection," *Proc. of Business Process Management Workshops,* pp. 32–46, 2005.

[3] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for QoS-aware Web Service Composition," *Proc. Int'l Conf. on Web Services,* Sept. 2006.

[4] M.B. Blake, "Decomposing Composition: Service Oriented Software Engineers," *IEEE Software,* Nov. 2007, pp. 68-77.

[5] J. Bosch, "Service Orientation in the Enterprise," *IEEE Computer,* Nov. 2007, pp. 51-56.

[6] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An Approach for QoS-aware Service Composition Based on Genetic Algorithms," *Proc. Genetic and Computation Conf.,* June 2005.

[7] V. Cardellini, E. Casalicchio, V. Grassi, and L. P. Francesco, "Flow-based service selection for web service composition supporting multiple qos classes," *ICWS 2007. IEEE Intl. Conf. Web Services,* pp. 743–750, July 9-13,2007.

[8] F. Curbera, "Component Contracts in Service Oriented Architectures," *IEEE Computer,* Nov. 2007, pp. 74-80.

[9] T. Davenport, "Process Innovation: Reengineering work through information technology," *Harvard Business School Press, Boston,* 1993.

[10] H. David, "Order Statistics," *John Wiley & Sons,* 1981.

[11] C.K. Fung, P.C.K. Hung, G. Wang, R.C. Linger, and G.H. Walton, "A Study of Service Composition with QoS Management," *Proc. 2005 IEEE Intl. Conf. Web Services (ICWS05),* 2005.

[12] M. Hammer and J. Champy, "Reengineering the Corporation: A Manifesto for Business Revolution," *Harper Business,* 1993.

[13] M. Jaeger, G. Muhl, and S. Golze, "Qos-aware composition ofweb services: A look at selection algorithm," *Proc. 2005 IEEE Intl. Conf. Web Services (ICWS05),* 2005.

[14] H. J. E. Johansson, "Business Process Reengineering: BreakPoint Strategies for Market Dominance," *John Wiley & Sons,* 1993.

[15] P. Lalanda and C. Marin, "A Domain-Configurable Development Environment for Service-Oriented Applications," *IEEE Software,* Nov. 2007, pp. 31-38.

[16] M. Kaiser, "Towards the Realization of Policy-Oriented Enterprise Management," *IEEE Computer,* Nov. 2007, pp. 51-56.

[17] T. Margaria, "Service Is in the Eyes of the Beholder," *IEEE Computer,* Nov. 2007, pp. 33-37.

[18] D.A. Menascé and V. Dubey, "Utility-based QoS brokering in service oriented architectures," *IEEE*

2007 Intl. Conf. Web Services (ICWS 2007), Application Services and Industry Track, Salt Lake City, Utah, July 9-13, 2007, pp. 422–430.

[19] D.A. Menascé, H. Ruan, and H. Gomma, "QoS management in service oriented architectures," *Performance Evaluation Journal,* , 64(7-8):646–663, August 2007.

[20] O. Nano and A. Zisman, "Realizing Service-Centric Software Systems," *IEEE Software,* Nov. 2007, pp. 28-30.

[21] R. Nelson, "Probability, Stochastic Processes, and Queueing Theory," *Springer-Verlag,* 1995.

[22] M.P. Papazoglou et al, "Service-Oriented Computing: State of the Art and Research Challanges," *IEEE Computer,* Nov. 2007, pp. 38-45.

[23] M. Papazoglou, "Web Services: Principles and Technology," *Prentice Hall,* 2008.

[24] M.A. Serhani, R. Dssouli, A Hafid, and H. Sahraoui, "A QoS Broker based Architecture for Efficient Web Service Selection," *Proc. 2005 IEEE International Conference on Web Services (ICWS05),* 2005.

[25] T. Yu and K. J. Lin, "Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints," *Proc. of 3rd Int'l Conf. on Service Oriented Computing,* pp. 130–143, Dec. 2005.

[26] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Softw. Eng.,* 30(5):311–327, Aug. 2004.