Scalable Service Selection for Web Service Composition Supporting Differentiated QoS Classes *

Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti
Università di Roma "Tor Vergata"
Dipartimento di Informatica, Sistemi e Produzione
{cardellini, casalicchio}@ing.uniroma2.it, {vgrassi, lopresti}@info.uniroma2.it

Università di Roma "Tor Vergata"
Dipartimento di Informatica, Sistemi e Produzione
Technical Report RR-07.59

February 8, 2007

Abstract

A composite Web service can be constructed and deployed by combining independently developed component services, each one may be offered by different providers with different non-functional Quality of Service (QoS) attributes. Therefore, a selection process is needed to identify which constituent services are to be used to construct a composite service that best meets the QoS requirements of its users.

In this paper, we consider a service broker that offers a composite service characterized by differentiated QoS classes which imply diverse monetary prices. These QoS classes are settled on the basis of some Service Level Agreements (SLAs) that the broker negotiate with both the requestors and the service providers. Differently from most of the current approaches, which optimize independently the end-to-end QoS of single requests and often require the solution of an NP-hard problem for each request, we optimize the end-to-end aggregated QoS of all incoming flows of requests by means of a simple linear programming problem, which can be efficiently solved. As a result, the proposed approach is scalable and lends itself to an efficient implementation.

1 Introduction

The service-oriented paradigm encourages the implementation of new applications through the composition of independently developed Web services. In this scenario, it is possible that different providers offer equivalent services corresponding to the same functional description (we refer to the former as *concrete services* and the latter as *abstract service*). Quality of Service (QoS) attributes provide a differentiation among the competing services [9], allowing a prospective user to choose the services which best suit to his/her QoS requirements. To formally define the QoS level required from the selected provider, the latter and the user may engage in a negotiation process, which culminates in the creation of a Service Level Agreement (SLA).

The management of QoS-based SLAs has become a very active area of research and standardization, including among its most interesting and promising challenges the QoS-aware service description, composition, and selection (e.g., [7, 8, 10, 14]).

In this paper, we tackle the issue of the QoS-aware selection of the concrete services from a larger set of candidates, that plays an important role in the provisioning and management of a composite service by a service-oriented architecture. To address the service selection issue, we propose the formulation of a Linear Programming optimization problem, considering various QoS attributes, such as response time, cost, and availability of the composite service. The solution provided by the optimization problem is used in the context of a broker-based architecture for the provisioning of a composite service, which is advertised with differentiated service classes. The broker negotiates SLAs with both the requestors and the service providers. These SLAs specify both the values of the QoS attributes and the average flow of requests that can be generated in a given time interval.

Our approach differs from previous works which have tackled the service selection as an optimization problem [2, 3, 4, 13, 14] in that our optimization is performed on a *per-flow* rather than *per-request* basis. Current proposals use exact algorithms or heuristics (e.g., [3] or genetic algorithms in [4]) to solve the QoS-aware service selection problem for each request, whose exact solution has an exponential complexity. Yu and Lin [13] defines the problem as a multi-dimension multi-choice 0-1 knapsack one as well as a multi-constraint optimal path problem. Zeng et al. [14] present a global planning approach to select an optimal execution plan by means of integer programming. Ardagna and Pernici [2] model the service composition as a mixed integer linear problem where both local and global constraints are taken into account.

However, an actual implementation of a broker-based architecture should be able to solve the optimization problem in real-time and under high volumes of service requests. Current solutions based on the per-request service selection approach could be too complex for run-time broker decisions and may suffer from scalability problems because of the computational overhead for solving the optimization problem for each single requests (even more times per request [14]). On the contrary, in our approach the solution of the optimization problem (*i.e.*, a given selection of concrete services) holds for all the requests in a flow, and needs to be recalculated only when some significant event occurs (*e.g.*, a change in the QoS values of the selected concrete services). Moreover, in our proposal the broker solves the optimization problem taking into account simultaneously the flows of requests generated by multiple requestors, with possibly different QoS constraints agreed in the SLAs. This is not possible in the per-request optimization approaches which, since they do not account for the presence of other simultaneous requests, can result in sub-optimal and possible instable solutions.

Our approach is able to give only a statistical guarantee to each request that its QoS constraints will be actually met. Specifically, our guarantee takes the form of bounds on the expected values of the QoS attributes. Hence, our approach is suitable for scenarios where "soft" rather than "hard" QoS goals must be satisfied, because violations of guarantees are tolerated and may be compensated with penalties. Actually, most proposals, which do not support a dynamic service re-binding executed during the process execution (*e.g.*, [5]), are not able to guarantee the strict enforcement of the agreed QoS levels, because they also use some statical measurement in the problem formulation (*e.g.*, the mean number of times a loop in the

workflow is executed).

We have introduced the per-flow service selection approach in [6]. However, this paper presents novel and significant contributions, which include the problem formulation as a Linear Programming problem that can be efficiently solved via standard techniques and the management of the concurrent execution pattern in the workflow of the composite service.

The rest of the paper is organized as follows. Section 2 describes the broker architecture and introduces the model of the composite service and the notation used in the problem formulation. Section 3 presents the QoS model for the composite Web service and discusses how to compute the global QoS attributes of the composite service. Section 4 discusses the formulation of the optimization problems and Section 5 presents some examples of solution of the optimization problems. Finally, Section 6 concludes the paper.

2 Broker Architecture

In this section, we outline the broker architecture. We focus on the broker components and introduce the composite service model and the notation we will later use to formulate the optimal selection of the concrete services.

The *service broker* [11, 12] acts as an intermediary between service requestors and providers, performing a role of service provider towards the requestors and being in turn a requestor to the providers of the concrete services. We consider an independently operated broker, which is maintained by a third party; it is a Web service itself and advertises the offered composite service in a public registry [11].

As a first step, the broker defines the business process for the composite service it wants to offer, and discovers concrete services which offer the required functionalities and are therefore candidates for the selection. In this paper, we do not focus on the discovery and selection process for creating a pool of candidate services. For each candidate service in this pool, the broker negotiates a SLA with its provider, establishing the values of the QoS attributes provided by each concrete service in correspondence with a mean volume of requests generated by the broker for that service. Then, the broker may negotiate a SLA with each requestor, establishing the offered QoS level of the composite service (*i.e.*, its Service Level Objectives or SLO) in correspondence with a mean volume of requests generated by the requestor to the broker. In our architecture, we assume that the broker manages different, but fixed, QoS levels for the operated composite service. Within this framework, one of the main broker tasks is to determine a service selection that fulfills the SLAs it negotiates with its requestors, given the SLAs it has negotiated with the providers. The selection criteria correspond to the optimization of a given utility goal of the broker.

In this paper, we mainly focus on the service selection task. However, in the following we briefly discuss the main components of the overall broker architecture, assuming that the broker acts as a full intermediary, which really provides the composite service to the requestors (the alternative approach relies on a heavy client, which owns an execution engine to execute the process, e.g. [11, 12]). To this end, we consider that the logic of the composite service is expressed through the Business Process Execution Language for Web Services (WS-BPEL or BPEL for short) [1], which has emerged as the de-facto standard language for the orchestration of Web services and the description of abstract business processes.

As illustrated in Figure 1, the broker consists of the following components: the *Composition Manager*, the *SLA Negotiation Manager*, the *Admission Control Manager*, the *BPEL Engine*, the *Selection Manager*, the *Optimization Engine*, the *Execution Path Analyzer*, and the *SLA Monitor*.

The main functions of the Composition Manager are the service composition (*i.e.*, the specification of the business process in BPEL) and the discovery of the candidate concrete services.

The SLA Negotiation Manager is responsible for establishing the SLAs with both the requestors and the service providers.

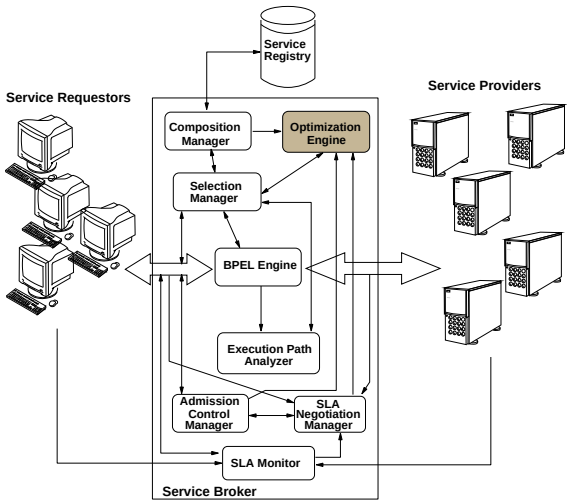

Figure 1: Broker architecture.

The role of the Control Admission Manager is to determine whether a new requestor can be accepted for the required classes of service, without violating the SLAs of already accepted requestors.

The BPEL Engine is the software platform which allows the execution of the business process described in BPEL. Once the request has been admitted and classified as pertaining to a flow with an established SLA, the BPEL Engine acts as the broker front-end to the service requestors for the service provisioning; when the requestor invokes the process, the BPEL Engine creates a new instance of the process itself.

The Selection Manager is invoked by the processes under execution and is responsible for binding each request to the concrete services that meet the contracted QoS level by assigning to each invoked service a real endpoint. The endpoints list is obtained at run time from the solution of the optimization problem provided by the Optimization Engine. Moreover, the Selection Manager may trigger a new solution of the optimization problem, when some relevant environmental change is detected.

The Optimization Engine determines the selection of the concrete services by solving the optimization problem. In this paper, we mainly focus on methodologies underlying the implementation of this part of the broker architecture.

Finally, the Execution Path Analyzer and the SLA Monitor are responsible, respectively, for collecting information about the composite service usage, about the performance perceived by the requestors and offered by the service providers and about the mean volume of requests generated by the requestors. This information is used by the Selection Manager to find out whether a new solution of the optimization problem is required. In a dynamic environment, service providers can change their services at any time in order to remain competitive. This implies that there is no guarantee that the QoS obtained at run time for a particular concrete service is still a valid value. However, we do not focus on this problem here, but rather assume that the broker can use some SLA monitoring service that ensures a high level of objectivity [8]. Furthermore, the information obtained by the SLA Monitor may be used to carry out a SLA enforcement [9], which is out of the scope of this paper.

2.1 Composite Service

We assume that the composite service structure is defined using BPEL [1]. In this paper, we actually refer to a significant subset of the whole BPEL definition, focusing on its structured style of modeling (rather than on its graph-based one, thus omitting to consider the use of control links). Specifically, in the definition of the workflow describing the composite service, besides the primitive `invoke` activity, which specifies the

synchronous or asynchronous invocation of a Web service, we consider all the different kinds of structured activities: `sequence`, `switch`, `while`, `pick`, and `flow`, whose meaning is summarized in Table 1.

Activity	Meaning
`sequence`	Sequential execution of activities
`switch`	Conditional execution of activities
`while`	Repeated execution of activities in a loop
`pick`	Conditional execution of activities based on external event/alarm
`flow`	Concurrent execution of activities

Table 1: Structured activities in BPEL.

```
...
<sequence>
   <while condition="condition=trigger" ...>
      <flow ...>
        <sequence>
            <invoke ...
              operation="FlightTicketBooking" .../>
            <invoke ...
              operation="HotelBooking" .../>
         </sequence>
         <invoke ...
           operation="AttractionSearch" .../>
      </flow>
   </while>
   <invoke ...
     operation="DrivingTimeCalculation" .../>
   <switch ....>
      <case condition="carRental=OK">
        <invoke ...
          operation="CarRental" .../>
      </case>
      <otherwise>
        <invoke ...
          operation="BikeRental" .../>
       </otherwise>
    </switch>
</sequence>
```

Figure 2: The Travel Planner BPEL code.

As a running example throughout the paper we use the Travel Planner composite service, whose BPEL representation is listed in Figure 2, while Figure 3 illustrates the BPEL code as a workflow, showing the different abstract services that form the composite service. With the exception of the `pick` construct, this example encompasses all the structured activities listed above.

The business process for the composite service defines a set of abstract services $\mathcal{V}$. We denote by I_i the set of all concrete services that can be used to implement the abstract service $i \in \mathcal{V}$ and by $i.j \in I_i$ the j-th concrete service for i. Figure 4 shows the Travel Planner workflow with the concrete services that can implement each abstract service. We assume that there are two concrete services for each abstract one that

is, $|I_i| = 2$ for each $i \in \mathcal{V}$. For the sake of simplicity, in the figure, we have renamed the activities with numbers: 1 for the FlightTicketBooking activity, 2 for the HotelBooking activity, etc.

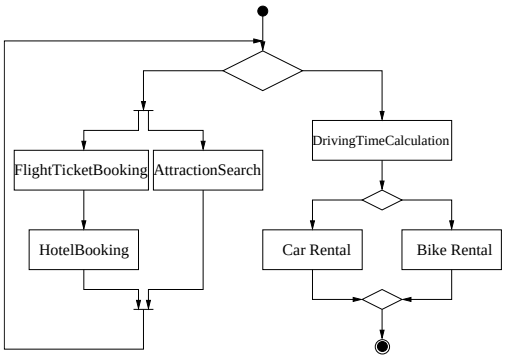

Figure 3: The Travel Planner workflow.

As the broker generally acts on behalf of a significant amount of requestors, it is able to identify recurrent requests for typical compositions of services, as well as usage patterns of these compositions. We assume that this knowledge is embodied in the workflow, which therefore models not only the structure of a given set of abstract services, but also their usage pattern. Therefore, we consider an annotated version of the workflow, where each branching point is annotated with suitable probabilities, that might differ for the different service classes (in the following, we will refer to them as *execution probabilities*). These probability values can be initialized by the workflow designer and are then periodically updated considering the information obtained by the Execution Path Analyzer component, which monitors the workflow executions. Figure 4 also shows the Travel Planner workflow with the annotated execution probabilities, which are denoted with p.

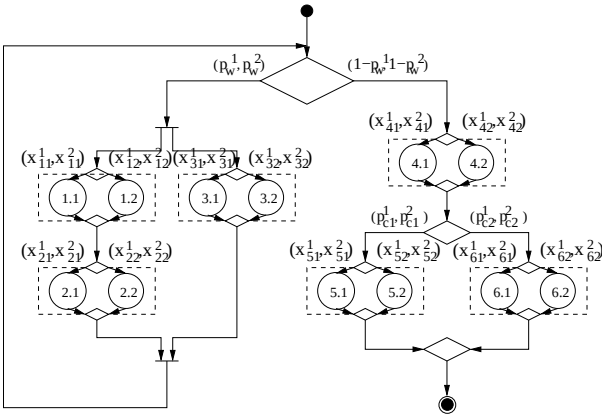

Figure 4: The annotated Travel Planner workflow.

2.2 SLA Negotiation

The broker is involved in the SLA negotiation with two counterparts: on one side the requestors of the composite service, on the other side the providers of the concrete services. Let us first discuss the SLA settled with the latter. The QoS of each concrete service can be characterized according to various attributes of interest, such as the response time, the cost, the reputation, the availability, and the throughput [7, 14].

The values of these QoS attributes are advertised by the service providers as part of the SLA. Without loss of generality, in this paper we will consider the following QoS attributes for each concrete service $i.j$:

- the response time r_{ij}, which is the interval of time elapsed from the invocation to the completion of the concrete service $i.j$;
- the cost c_{ij} which represents the price charged for each invocation of the concrete service $i.j$;
- the log of the availability, a_{ij}, *i.e.*, the logarithm of the probability that the concrete service $i.j$ is available when invoked.

As in [14], we consider the *logarithm* of the availability, rather than the availability itself, in order to obtain linear expressions when composing the availability of different services.

For a given concrete service $i.j$, the SLA established by the broker with the service provider defines the service cost (measured in money per invocation), the service availability, and the expected response time (measured in time unit), provided the volume of requests generated by the broker does not exceed the negotiated average load. Therefore, the SLA for the concrete service $i.j$ is represented by the template $< r_{ij}, c_{ij}, a_{ij}, L_{ij} >$, where L_{ij} is the agreement on average load.

We denote by K the set of QoS classes offered by the broker. In the SLAs created with the requestors, the broker characterizes the QoS of the composite service in terms of bounds on the expected response time, cost, and availability for each QoS class $k \in K$ (*i.e.*, $R^k_{\max}, C^k_{\max}, A^k_{\min}$). Each requestor has to negotiate for each QoS class the volume of requests it will generate in that class (denoted by $\Delta\gamma^k$). The SLA established by the broker with the requestor for the QoS class k, $k \in K$, has therefore a template $< R^k_{\max}, C^k_{\max}, A^k_{\min}, \Delta\gamma^k >$.

2.3 Admission Control

The Admission Control Manager determines whether a new requestor can be accepted for the required class of service, without violating the SLAs of already accepted requestors. Let γ be the aggregate arrival rate of already accepted requestors (*i.e.*, $\gamma = (\gamma^1, ..., \gamma^m)$, where $m = |K|$) and denote by $\Delta\gamma$ the arrival rate requested by the new user for all the service classes (*i.e.*, $\Delta\gamma = (\Delta\gamma^1, ..., \Delta\gamma^m)$). The Admission Control Manager determines whether the new requestor can be accepted by invoking the Optimization Engine and asking for a new resolution of the optimization problem with $\gamma + \Delta\gamma$ as aggregate arrival rate. We have two possible cases. If a feasible solution to the optimization problem exists, it means that the additional requests can be satisfied - at the requested QoS - without violating the QoS of already accepted users. The new requestor can be thus accepted and the SLA finalized for the requested rate and QoS class. If, instead, a feasible solution does not exists, the broker can: 1) turn down the new requestor; 2) renegotiate the SLA with the requestor, *e.g.*, by proposing another service class; 3) renegotiate the SLAs with the service providers by redefining response time, costs, availability and/or the amount of requests per unit of time the concrete services accept.

2.4 Optimal Service Selection

In this section, we present a general formulation of the optimization problem solved by the Optimization Engine component. Each class k is characterized by a QoS vector $< R^k_{\max}, C^k_{\max}, A^k_{\min}, \gamma^k >$ which defines the minimum/ maximum guaranteed values for the different QoS attributes.

The goal of the Selection Manager is to select, for each QoS class, the concrete service $i.j$ that must be used to fulfill a request for the abstract service i. We model this selection by associating with each abstract service i a vector $\boldsymbol{x}_i = (\boldsymbol{x}^1_i, ..., \boldsymbol{x}^m_i)$, where $\boldsymbol{x}^k_i = [x^k_{ij}]$, and $i.j \in I_i$. Each entry x^k_{ij} of $\boldsymbol{x}^k_i$ denotes the probability that the class-k request will be bound to the concrete service $i.j$. With this model, we assume

that, in general, the Selection Manager can probabilistically bind to different concrete services the requests (belonging to a same QoS class k) for an abstract service i. The deterministic selection of a single concrete service corresponds to the case $x_{ij}^k = 1$ for a given $i.j \in I_i$. Figure 4 also shows the Travel Planner example with the decision vector $\boldsymbol{x}$.

We formulate the optimization problem solved by the Optimization Engine in the general form:

$$\text{Max } F(\boldsymbol{x}) \tag{1}$$

$$\begin{aligned} \text{subject to } & Q^{\alpha}(\boldsymbol{x}) \leq Q^{\alpha}_{\max} \\ & Q^{\beta}(\boldsymbol{x}) \geq Q^{\beta}_{\min} \\ & S(\boldsymbol{x}) \leq L \\ & \boldsymbol{x} \in A \end{aligned} \tag{2}$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$ is the decision vector (being $n = |\mathcal{V}|$), $F(\boldsymbol{x})$ is a suitable objective function, $Q^{\alpha}(\boldsymbol{x})$ and $Q^{\beta}(\boldsymbol{x})$ are, respectively those QoS attributes whose SLA values is settled as a maximum and a minimum, $S(\boldsymbol{x})$ are the constraints on the offered load determined by the SLAs with the service providers, and $\boldsymbol{x} \in A$ is a set of functional constraints (*e.g.*, this latter set includes the constraint $\sum_{j \in I_i} x_{ij}^k = 1$).

To carry out the service selection, the Selection Manager uses the solution of this optimization problem as follows. Given the request service class (denoted by k), the Selection Manager considers only the elements of the solution vector $\boldsymbol{x}$ that pertains to class k. If for a given abstract service i there is more than one $x_{ij}^k \neq 0$, the Selection Manager selects randomly, using the probability values x_{ij}^k, one concrete service to which it binds the request.

The service selection provided by the solution of the optimization problem is valid until some event occurs that requires a new solution. This happens in the following cases: *a*) some execution probabilities changes: their value is periodically recomputed by the Execution Path Analyzer; *b*) the service composition changes, because either an abstract service or a concrete service is added or removed; *c*) the SLA Monitor identifies some significant change in the negotiated SLA parameters.

3 Web Service QoS Model

In this section we present the QoS model for the composed Web service and how to compute the composite service QoS attributes. We also derive closed form expressions for the different attributes which we will use in the optimization problem formulation.

For each class offered by the broker, the overall QoS attributes, namely,

- the expected response time R^k, which is the time needed to fulfill a class-k request for the composite service;
- the expected execution cost C^k, which is the price to be paid to fulfill a class-k request;
- the expected availability A^k, which is expected value of the logarithm of the probability that the composed service is available for a class-k request

depends on: 1) the actual concrete services selected to perform the different activities; and, 2) how they are orchestrated to carry out the composed service. As shown below, the computation of R^k, C^k and A^k can be carried out by recursively computing the QoS of the structured BPEL activities from the QoS of the component activities (see, *e.g.* [7] where the same approach is applied to workflows).

3.1 Process Activities Tree

The algorithm to compute the QoS attributes can be easily described if we represent the BPEL process by a labeled tree $T = (V, E)$ which concisely captures the nesting relationship among the BPEL process activities. In T, nodes are the activities in the BPEL code, and edges reflect the nesting relationship among the activities. More precisely, we associate a node $i \in V$, with label $t(i) =$x, with each structured activity x$\in$ {sequence, switch, pick, while, flow}. Similarly, we associate a node $i \in V$ with label $t(i) =$a with each primitive invoke a activity. We recall that in our composite service model a primitive activity corresponds to a service invocation. For the sake of simplicity, in the following, we will interchangeably speak of activity i and node i. For each non root node $i \in V$, its parent node $f(i)$ is the structured activity within which activity i occurs. Primitive activities are thus associated with leaf nodes, while structured activities are associated with internal nodes. For each node $i \in V$, we will denote by $d(i)$, the (possibly empty) set of its children. We will also write $i \prec l$ if node i is a descendant of node l. Finally, we label each edge $(f(l), l) \in E$, with the expected number of times the activity l is invoked within $f(l)$. Since execution probabilities may differ for the different QoS class, the label $\ell((f(l), l))$ is a m element vector $\ell((f(l), l)) = (\ell^k((f(l), l)))_{k \in K}$. Figure 5 summarizes how each BPEL activity is represented in T. Figure 6 shows the process activity tree for the Travel Planner example.

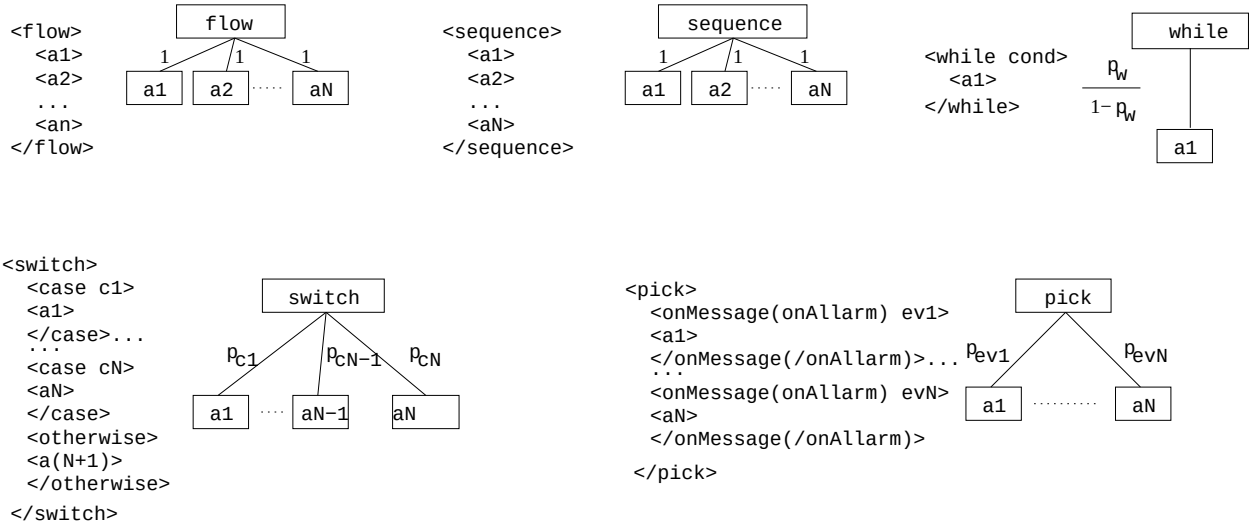

Figure 5: BPEL structured activities tree representation.

3.2 QoS Computation

Given the definitions above, we can now provide the algorithm to compute the composite service QoS attributes. We will then derive closed form expressions for the different attributes.

First, let Z_i^k, denote the QoS attribute of the abstract service $i \in \mathcal{V}$, $Z \in \{R, C, A\}$. We have

$$Z_i^k = \sum_{j \in I_i} x_i^k z_{ij} \tag{3}$$

where z_{ij}, $z \in \{r, c, a\}$ is the corresponding QoS attribute of the concrete services which can implement i.

For each structured activity, the expressions for the aggregate QoS, given the QoS of the composing activities QoS are straightforward and are summarized in Table 2 (where p_i^k denotes the probability of executing activity i in the pick, switch and while statement for class k-requests). Observe that we

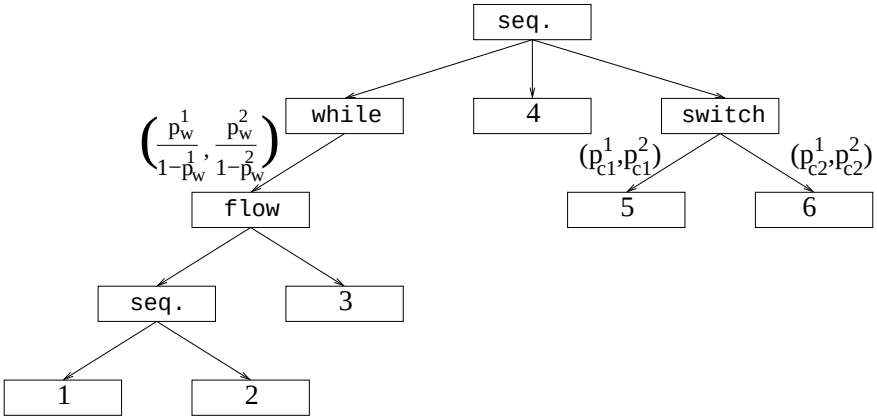

Figure 6: BPEL process tree for the Travel Planner example. For the sake of clarity, only the labels which differ from $(1, 1)$ are shown.

Aggregate Function			
`sequence`	`pick, switch`	`while`	`flow`
$\sum Z_i^k$	$\sum p_i^k Z_i^k$	$p_i^k/(1-p_i^k)Z_i^k$	$\max Z_i^k$, $Z = R$, $\sum Z_i^k$, $Z \in \{C, A\}$

Table 2: QoS attributes aggregation function for BPEL structured activities.

have similar expressions for all QoS attributes but for the response time of a `flow` activity which is given the by largest response time of the parallel activities appearing in it.

The composed service QoS Z^k, $Z \in \{R, C, A\}$ is obtained by means of the recursive procedure specified in Algorithm 1, with arguments the root node $\bar{\imath}$ and the class k of interest. Computation is carried out by visiting in post-order the process activity tree. For each leaf node $i \in \mathcal{V}$ the procedure simply returns the QoS attribute Z_i^k (line 3). For each internal node the procedure first recursively compute the QoS attribute of its child nodes (lines 5-6) and then returns the aggregate QoS of the corresponding BPEL structured activity (lines 8-12). The computations in line 9 and 11 correspond to those listed in Table 2 for the different structured activities.

Algorithm 1 *Compute Composite QoS Attributes*

1: **function** Compute Composite QoS(node i, class k)
2: **if** i is a leaf node **then**
3: **return** Z_i^k
4: **else**
5: **for all** $i' \in d(i)$ **do**
6: $Z_{i'}^k$ =Compute Composite QoS(i',k);
7: **end for**
8: **if** $t(i) =$ `flow` AND $Z = R$ **then**
9: **return** $Z_i^k = \max_{i' \in d(i)} Z_{i'}^k$;
10: **else**
11: **return** $Z_i^k = \sum_{i' \in d(i)} \ell(i, i') Z_{i'}^k$
12: **end if**
13: **end if**

We now derive closed form expressions for the QoS attributes of the composite service we will use in the optimization problem formulation.

Cost and Availability Let us consider first cost and availability. For these attributes, computation is carried out by recursively composing linear functions (line 11 of Algorithm 1). From the algorithm, it is easy to realize that the procedure yields

$$Z^k = \sum_{i \in \mathcal{V}} \left(\prod_{j \succeq i} \ell(f(j), j) \right) Z_i^k \tag{4}$$

$$= \sum_{i \in \mathcal{V}} V_i^k Z_i^k. \tag{5}$$

where we let $V_i^k = \prod_{l \succeq i} \ell^k(f(l), l)$, $i \in V$. Observe that V_i^k has a simple interpretation. To this end, note that for a given node $i \in V$, V_i^k is the product of all labels $\ell^k(.)$ along the path from i to the root node $\bar{\imath}$, *i.e.*, the product of the expected number of times activity i is executed within $f(i)$ times the expected number of times activity $f(i)$ is executed within $f(f(i))$ and so on till the root node. Thus, it is not difficult to realize that V_i^k is just the expected number of times activity i is invoked by the composed web service for a class k user.

By replacing the proper expressions, we obtain from (5) the explicit form for cost and availability of the composite service:

$$C^k(\boldsymbol{x}) = \sum_{i \in \mathcal{V}} V_i^k C_i^k(\boldsymbol{x}) = \sum_{i \in \mathcal{V}} V_i^k \sum_{j \in I_i} x_{ij}^k c_{ij} \tag{6}$$

$$A^k(\boldsymbol{x}) = \sum_{i \in \mathcal{V}} V_i^k A_i^k(\boldsymbol{x}) = \sum_{i \in \mathcal{V}} V_i^k \sum_{j \in I_i} x_{ij}^k a_{ij}. \tag{7}$$

Response Time. For the response time, instead, we obtain a closed form expression only as long as the composed service does not include `flow` structured activities. In such a case, by using the same arguments above, we immediately have:

$$R^k(\boldsymbol{x}) = \sum_{i \in \mathcal{V}} V_i^k \sum_{j \in I_i} x_{ij}^k r_{ij}. \tag{8}$$

In the general case, instead, the situation is more complex because the response time of a `flow` activity is given by the largest response time among all component activities; therefore, we cannot obtain a closed form expression as in (8).

To deal with this case, we derive a set of equations for the response time, whose number is linear in the number of `flow` activities in the process. As shown in the next section, these equations directly translate in a set of inequalities which we will use in the derivation of the optimization problem.

To this end, we first introduce the notion of *direct descendant* among nodes in the BPEL process tree. We say that a node $l \in T$ is a direct descendant of $l' \in T$, denoted by $l \preceq_{dd} l'$, if $l \prec l'$ and for any other node $l'' \in T$, $l \prec l'' \prec l'$ implies $t(l'') \neq$ `flow`, *i.e.*, if there is no node labeled `flow` in the path from l to l'.

Let $\mathcal{F} \subset V$ denote the set of nodes corresponding to `flow` activities. We have the following result for the response time for a general BPEL process. The proof is postponed to the Appendix.

Theorem 1 *For an activity $l \in V$, and QoS class $k \in K$, the response time R_l^k is:*

$$R_l^k = \begin{cases} \max_{l' \in d(l)} R_{l'}^k & l \in \mathcal{F} \\ \sum_{i \in \mathcal{V}, i \prec_{dd} l} \frac{V_i^k}{V_l^k} \sum_{j \in I_i} x_{ij}^k r_{ij} + & \\ + \sum_{h \in \mathcal{F}, h \prec_{dd} l} \frac{V_h^k}{V_l^k} R_h^k & l \notin \mathcal{F} \end{cases} \tag{9}$$

Moreover, the overall expected response time $R^k(\mathbf{x})$ is given by the following expressions:

$$R^k(\boldsymbol{x}) = \begin{cases} \max_{l' \in d(\bar{\imath})} R_{l'}^k & \bar{\imath} \in \mathcal{F} \\ \sum_{i \in \mathcal{V}, i \prec_{dd} \bar{\imath}} V_i^k \sum_{j \in I_i} x_{ij}^k r_{ij} + & \\ + \sum_{h \in \mathcal{F}, h \prec_{dd} \bar{\imath}} V_h^k R_h^k & \bar{\imath} \notin \mathcal{F} \end{cases} \tag{10}$$

Theorem 1 provides the response time R_l^k of each activity $l \in V$ and the composite service response time $R^k(\boldsymbol{x})$, for each $k \in K$. The second of the expressions for $R^k(\boldsymbol{x})$ comprises two terms. The first term is the expected overall response time of the services which do not appear within a `flow` structured activity. The second term is the sum of the response times R_l^k of the *outer* `flow` activities, *i.e.*, `flow` activity which are non nested within other `flow` activities.

4 Optimization Problem

In this section we present the broker optimization problem. The Optimization Engine goal is to determine the variables x_{ij}^k, $i \in \mathcal{V}$, $k \in K$, $j \in I_i$ as to maximize a suitable objective function. We assume that the broker wants, in general, to optimize multiple QoS attributes (which can be either mutually independent or possibly conflicting), rather than just a single one, *i.e.*, the response time. Therefore, in general the optimal service selection takes the form of a multi-objective optimization. Here, we tackle the multi-objective problem by transforming it into a single objective problem. Specifically, we consider as objective function $F(\boldsymbol{x})$ an aggregate QoS measure given by a weighted sum of the (normalized) QoS attributes. More precisely, let

$$\begin{aligned} R(\boldsymbol{x}) &= \frac{1}{\sum_{k \in K} \gamma^k} \sum_{k \in K} \gamma^k R^k(\boldsymbol{x}) \\ C(\boldsymbol{x}) &= \frac{1}{\sum_{k \in K} \gamma^k} \sum_{k \in K} \gamma^k C^k(\boldsymbol{x}) \\ A(\boldsymbol{x}) &= \frac{1}{\sum_{k \in K} \gamma^k} \sum_{k \in K} \gamma^k A^k(\boldsymbol{x}) \end{aligned}$$

denote the expected overall response time, cost and availability, respectively. We define the objective function as follows

$$\begin{aligned} F(\boldsymbol{x}) = w_r \frac{R_{\max} - R(\boldsymbol{x})}{R_{\max} - R_{\min}} \; + \; w_c \frac{C_{\max} - C(\boldsymbol{x})}{C_{\max} - C_{\min}} + \\ + w_a \frac{A(\boldsymbol{x}) - A_{\min}}{A_{\max} - A_{\min}} \end{aligned} \tag{11}$$

where $w_r,\ w_c,\ w_a \geq 0$, $w_r + w_c + w_a = 1$, are weights for the different QoS attributes. $R_{\max}$ ($R_{\min}$), $C_{\max}$ ($C_{\min}$) and $A_{\max}$ ($A_{\min}$) denote, respectively, the maximum (minimum) value for the overall response time, cost and the (log of) availability (for sake of presentation we postpone to the Appendix how to compute these quantities). After normalization, each term takes value in the interval $[0, 1]$. Note that in (11) the response

time $R(\boldsymbol{x})$ and cost $C(\boldsymbol{x})$, which we wish to minimize, appear with negative sign, while the availability $A(\boldsymbol{x})$, which we wish to maximize, appears with positive sign.

The Optimization Engine task consists in finding the variables x_{ij}^k, $i \in \mathcal{V}$, $k \in K$, $j \in I_i$ which solves the following optimization problem.

$$\textbf{max } F(\boldsymbol{x})$$

$$\textbf{subject to :}$$

$$R^k(\boldsymbol{x}) \leq R^k_{\max} \quad k \in K \tag{12}$$

$$R^k_{l'} \leq R^k_l \quad l' \in d(l), \quad l \in \mathcal{F}, k \in K \tag{13}$$

$$R^k_{l'} = \sum_{i \in \mathcal{V}, i \prec_{ddl} l} \frac{V^k_i}{V^k_l} \sum_{j \in I_i} x^k_{ij} r_{ij} + \sum_{h \in \mathcal{F}, h \prec_{ddl} l} \frac{V^k_h}{V^k_l} R^k_h \quad l \notin \mathcal{F}, k \in K \tag{14}$$

$$C^k(\boldsymbol{x}) \leq C^k_{\max} \quad k \in K \tag{15}$$

$$A^k(\boldsymbol{x}) \geq A^k_{\min} \quad k \in K \tag{16}$$

$$\sum_{k \in K} x^k_{ij} V^k_i \gamma^k \leq L_{ij} \quad i \in \mathcal{V}, j \in I_i \tag{17}$$

$$\sum_{j \in I_i} x^k_{ij} = 1 \quad i \in \mathcal{V}, k \in K \tag{18}$$

$$x^k_{ij} \geq 0 \quad i \in \mathcal{V}, j \in I_i, k \in K \tag{19}$$

Equations (12)-(16) are the QoS constraints on response time, cost and availability, where $R^k_{\max}$, $C^k_{\max}$, and $A^k_{\min}$ are respectively the maximum response time, the maximum cost and the minimum (log of the) service availability that characterize the QoS class k. The constraints(13)-(14), which can be easily derived from (9), provide the expressions for the response times. Inequalities (13), in particular, allow us to express the relationship among the response times R^k_l of a `flow` activity and that of its component activities $R^k_{l'}$. Equations (17) are the broker-providers SLA constraints and ensure the broker does not exceed the SLA with the service providers. Finally, Equations (18)-(19) are the functional constraints.

We conclude by observing that the proposed Optimization Engine problem is a Linear Programming problem which can be efficiently solved via standard techniques. The solution thus lends itself to both on-line and off-line operations.

5 Numerical Results

In this section, we illustrate the behavior of the proposed selection scheme. We consider the Travel Planner service of Figure 3. The composite Web service offers two QoS classes: the *gold* service and the *silver* service, denoted by the superscripts, 1 and 2, respectively. Table 3 summarizes the two classes QoS attributes. Users in the gold class accept to pay a higher cost to get better response time and availability, while users in the silver class accept worse performance to pay a lower cost.

QoS Class	$R^k_{\max}$	$C^k_{\max}$	$A^k_{\min}$
gold	12	20	log(0.95)
silver	20	12	log(0.9)

Table 3: Composite service class attributes.

We assume that for each abstract service there are two concrete services which implement it, *i.e.*, $n_i = 2$, $i \in \mathcal{V}$ (the resulting workflow is displayed in Figure 4). The concrete services differ in terms of response time, cost and availability. Table 4 summarizes the parameters of the concrete services. They have been chosen so that for each abstract service $i \in \mathcal{V}$, concrete service $i.1$ represents the *better* service, which at a higher cost ensures lower response time and higher availability with respect to service $i.2$ which costs less but has higher response time and lower availability. For all services, we assume $L_{ij} = 10$. Finally, we consider the following values for the execution probabilities, $(p^1_w, p^2_w) = (0.6, 0.6)$, $(p^1_{c1}, p^2_{c1}) = (0.7, 0.5)$ and $(p^1_{c2}, p^2_{c2}) = (0.3, 0.5)$.

Service	r_{ij}	c_{ij}	a_{ij}	Service	r_{ij}	c_{ij}	a_{ij}
1.1	2	6	log(0.999)	4.1	0.5	0.5	log(0.999)
1.2	4	3	log(0.99)	4.2	1	0.3	log(0.99)
2.1	2	4	log(0.999)	5.1	2	1	log(0.999)
2.2	4	2	log(0.99)	5.2	2.2	0.7	log(0.99)
3.1	1	2	log(0.999)	6.1	1.8	0.5	log(0.999)
3.2	3	1	log(0.99)	6.2	2	0.2	log(0.99)

Table 4: Concrete services QoS attributes.

We assume that the arrival rates for the two QoS classes are $(\gamma^1, \gamma^2) = (4, 7)$ and consider the selection strategy for two different objective functions: 1) the Optimization Engine minimizes the average response time ($w_r = 1$); and, 2) it minimizes the mean cost ($w_c = 1$). Figure 7 shows the solution of the optimization problem in the two scenarios; the values within the graphs are those of the variables x^k_{ij} when different from 1.

In the first scenario, the goal is to minimize the average response time. The broker treats quite differently the requests of the two QoS classes. For the *gold* service requests, the broker always selects the faster concrete services $i.1$ (see the upper left workflow in Figure 7. This allows to achieve the lowest possible response time (8.44) at a cost (19.35) which is within *gold* user upper limit C^1_{max}. For the *silver* requests, instead, the solution adopted for the *gold* users cannot be applied as it is too expensive (well above $C^2_{\max} = 12$). For this class, instead, a fraction of requests is assigned to the cheaper services $i.2$ to satisfy the cost constraints. It is worth observing that the abstract service 3 is handled differently as all requests are assigned to 3.2. This is easily explained by observing that no matter how concrete services are chosen, abstract service 3 response time is lower than the sum of the response time of the 1 and 2, $R^2_3 < R^2_1 + R^2_2$; hence, there is no benefit in assigning requests to service 3.1 which would only increase cost without any gain in the response time. Finally, we observe that, as a byproduct of the concrete services attributes, *gold* users also enjoy better service availability with an availability of 98% versus an availability of about 95% for *silver* users.

In the second scenario, the goal is to minimize the average cost. Intuitively, this should be achieved by using as much as possible the concrete services $i.2$, $i \in \mathcal{V}$ since they cost less (as long the other QoS requirements are satisfied). To show that this is indeed the case, we list in Table 5 the concrete service utilization, defined as $\sum_{k \in K} x^k_{ij} V^k_i \gamma^k / L_{ij}$, which shows that the cheaper web services are fully utilized (except service 6.2 which, nevertheless, is assigned all requests for abstract service 6). Requests which cannot be assigned to these services are assigned to the more expensive services $i.1$. Besides ensuring full

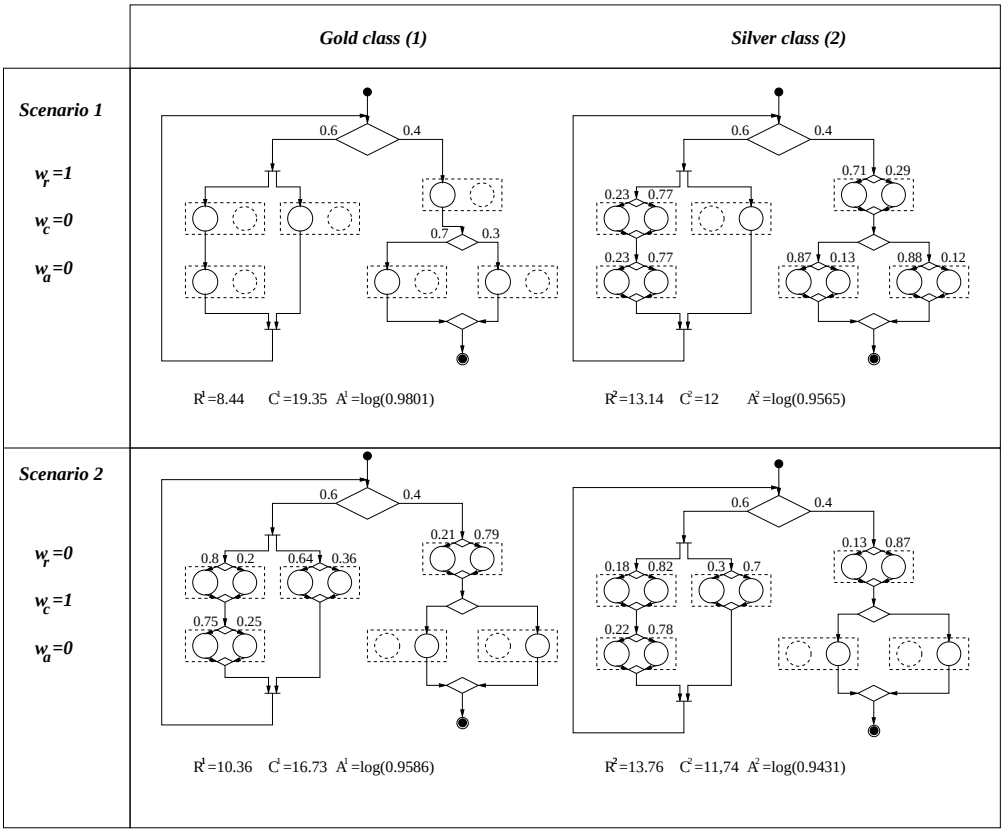

Figure 7: Solution of the optimization problem.

utilization of the cheaper services, users requests are also assigned as to satisfy the other QoS attributes: *gold* users are mainly assigned to the more expensive services because of the more stringent response time constraints, while the opposite is true for the *silver* requests.

Service	Util. (%)	Service	Util. (%)
1.1	65%	4.1	10%
1.2	100%	4.2	100%
2.1	65%	5.1	0%
2.2	100%	5.2	63%
3.1	65%	6.1	0%
3.2	100%	6.2	47%

Table 5: Scenario 2: concrete services utilization.

6 Conclusions

In this paper, we have addressed the problem of selecting concrete services in a composite service offered to the requestors by a broker which supports differentiated QoS service classes. Most of the existing approaches to the service selection problem deal with SLAs concerning a single service request. This could cause scalability problems in the case of a sustained flow of requests, also because of the exponential complexity of the proposed solution methodologies. Differently from these approaches, we consider SLAs

encompassing the overall flow of requests that can be generated by a requestor in some negotiated time interval. Our method is based on the definition of a process activities tree to determine the global QoS attributes of the whole composition and on the formulation of a constrained optimization problem, which determines a probabilistic selection of the concrete services. This optimization problem can be efficiently solved via standard techniques for linear programming; this makes its solution suitable not only to off-line operations but also to on-line ones. Therefore, our approach allows to efficiently manage the service selection in a real operating broker-based architecture, where the broker efficiency and scalability in replying to the requestors are important factors.

Our problem formulation can be easily modified to take into account other QoS attributes in the selection process. Moreover, it can be easily extended to determine a new resource provisioning that can be contracted with the service providers in order to satisfy a new flow of requests or a change in an already existing SLA.

This paper is part of an ongoing research; future work concerns the issue of dynamically binding the concrete services during the process execution, the implementation of the broker-based architecture, and the analysis and modeling of a multi-broker scenario, where the brokers cooperate or compete in the use of the same concrete services.

A Proof of Theorem 1

The proof is by induction. For a leaf node $l \in \mathcal{V}$, the second of (9) reduces to $R_l^k = \sum_{j \in I_i} x_{ij}^k r_{ij}$ which is the expression for the response time of activity l. For a non leaf node $l \in V$, assume (9) holds for all l child nodes, *i.e.*, for all $l' \in d(l)$. We distinguish two cases. If $l \in \mathcal{F}$, $R_l^k = \max_{l' \in d(l)} R_{l'}^k$. If $l \notin \mathcal{F}$, we have:

$$R_l^k = \sum_{l' \in d(l)} \ell(l,l') R_{l'}^k \tag{20}$$

$$= \sum_{l' \in d(l), l' \notin \mathcal{F}} \ell(l,l') r_{l'}^k + \sum_{l' \in d(l), l' \in \mathcal{F}} \ell(l,l') R_{l'}^k \tag{21}$$

$$= \sum_{l' \in d(l), l' \notin \mathcal{F}} \ell(l,l') \sum_{i \in V, i \preceq_{dd} l'} \frac{V_i^k}{V_{l'}^k} \sum_{j \in I_i} x_{ij}^k r_{ij} + \\ + \sum_{l' \in d(l), l' \notin \mathcal{F}} \ell(l,l') \sum_{h \in \mathcal{F}, h \preceq_{dd} l'} \frac{V_h^k}{V_{l'}^k} R_h^k + \sum_{l' \in d(l), l' \in \mathcal{F}} \ell(l,l') R_{l'}^k \tag{22}$$

$$= \sum_{l' \in d(l), l' \notin \mathcal{F}} \frac{V_{l'}^k}{V_l^k} \sum_{i \in V, i \preceq_{dd} l'} \frac{V_i^k}{V_{l'}^k} \sum_{j \in I_i} x_{ij}^k r_{ij} + \\ + \sum_{l' \in d(l), l' \notin \mathcal{F}} \frac{V_h^k}{V_{l'}^k} \sum_{h \in \mathcal{F}, h \preceq_{dd} l'} \frac{V_h^k}{V_{l'}^k} R_h^k + \sum_{l' \in d(l), l' \in \mathcal{F}} \frac{V_l^k}{V_{l'}^k} R_{l'}^k \tag{23}$$

$$= \sum_{i \in V, i \preceq_{dd} l} \frac{V_i^k}{V_l^k} \sum_{j \in I_i} x_{ij}^k r_{ij} + \sum_{h \in \mathcal{F}, h \preceq_{dd} l} \frac{V_l^k}{V_l^k} R_h^k. \tag{24}$$

(22) is obtained by replacing the expression for $R_{l'}^k$ for all the child nodes $l' \in d(l)$, $l' \notin \mathcal{F}$. (23) is then derived by substituting $\ell(l,l')$ with $V_l^k / V_{l'}^k$ (observe that $V_{l'}^k = \prod_{h \succeq l'} \ell(f(h),h) = \prod_{h \succ l'} \ell(f(h),h) \cdot \ell(f(l),l) = V_l^k \cdot \ell(f(l),l)$). Finally, (24) is obtained: 1) by simplifying V_l^k in the ratios; 2) and, by rearranging the terms in the sums. This completes the proof.

B Computation of maximum and minimum values of the QoS attributes

The maximum and minimum values of the QoS attributes $R_{\max}$ ($R_{\min}$), $C_{\max}$ ($C_{\min}$), and $A_{\max}$ ($A_{\min}$)in the objective function (11) are determined as follows. $R_{\max}$, $C_{\max}$, and $A_{\min}$ are simply expressed respectively in terms of $R^k_{\max}$, $C^k_{\max}$, and $A^k_{\min}$. For example, the maximum response time $R_{\max}$ is given by:

$$R_{\max} = \frac{1}{\sum_{k \in K} \gamma^k} \sum_{k \in K} \gamma^k R^k_{\max}$$

Similar expressions hold for $C_{\max}$ and $A_{\min}$. The values for $R_{\min}$, $C_{\min}$, and $A_{\max}$ are determined by solving a modified optimization problem in which the objective function is the QoS attribute of interest, subject to the constraints (17)-(19). For example, the minimum cost $C_{\min}$ is given by the solution of the following constrained optimization problem:

$$\begin{aligned}
&\textbf{min } C(\boldsymbol{x}) \\
&\textbf{subject to :} \\
&\sum_{k \in K} x^k_{ij} V^k_i \gamma^k \leq L_{ij} \quad i \in \mathcal{V}, j \in I_i \\
&\sum_{j \in I_i} x^k_{ij} = 1 \quad i \in \mathcal{V}, k \in K \\
&x^k_{ij} \geq 0 \quad i \in \mathcal{V}, j \in I_i, k \in K
\end{aligned}$$

Similar optimization problems have to be solved to obtain the values for $R_{\min}$ and $A_{\max}$. In the latter case, the objective function $A(\boldsymbol{x})$ has to be maximized.

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services Version 1.1, May 2003. `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/`.

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

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

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

[5] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. QoS-Aware Replanning of Composite Web Services. In *Proc. of Int'l Conf. on Web Services*, July 2005.

[6] V. Cardellini, E. Casalicchio, V. Grassi, and R. Mirandola. A Framework for Optimal Service Selection in Broker-based Architectures with Multiple QoS Classes. In *Proc. of 2006 IEEE Service Computing Workshops*, Sept. 2006.

[7] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. J. Kochut. Modeling Quality of Service for Workflows and Web Service Processes. *Web Semantics J.: Science, Services and Agents on the World Wide Web*, 1(3):281–308, 2004.

[8] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. Web Services on Demand: WSLA-driven Automated Management. *IBM Systems J.*, 43(1):136–158, 2004.

[9] A. Dan, H. Ludwig, and G. Pacifici. *Web Service Differentiation with Service Level Agreements*. White Paper, IBM Corporation, Mar. 2003.

[10] N. Milanovic and M. Malek. Current Solutions for Web Service Composition. *IEEE Internet Computing*, 8(6):51–59, Nov./Dec. 2004.

[11] M. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui. A QoS Broker Based Architecture for Efficient Web Services Selection. In *Proc. of 2005 Int'l Conf. on Web Services*, pages 113–120, July 2005.

[12] T. Yu and K. J. Lin. A Broker-Based Framework for QoS-Aware Web Service Composition. In *Proc. of 2005 IEEE Int'l Conf. on e-Technology, e-Commerce and e-Service*, Mar. 2005.

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

[14] 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.