Flow-Based Service Selection for Web Service Composition Supporting Multiple QoS Classes

Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti
University of Roma "Tor Vergata"
{cardellini, casalicchio}@ing.uniroma2.it, {vgrassi, lopresti}@info.uniroma2.it

Abstract

In the service oriented paradigm applications are created as a composition of independently developed Web services. Since the same service may be offered by different providers with different non-functional Quality of Service (QoS) attributes, a selection process is needed to identify the constituent services for a given composite service that best meet the users QoS requirements.

In this paper, we consider a broker that offers a composite service with multiple QoS classes to several users each generating a flow of requests over time. We propose a service selection scheme which optimizes the end-to-end aggregated QoS of all incoming flows of requests by means of a simple linear programming problem which scales as the number of users, request volumes and/or services grows. This approach differs from most of the current proposals which may not scale well since: a) requests, even from the same user, are handled independently from one another; and b) the selection process often requires the solution of an NP-hard problem.

1. Introduction

The service oriented paradigm encourages the implementation of new applications through the composition of independently developed and deployed Web services. Different providers may 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, allowing a prospective user to choose the services which best suit 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 definition of a Service Level Agreement (SLA).

Web service composition has received increasing attention by the research community in the past few years (*e.g.*, [7, 8, 9, 11, 13] to cite a few). An important role for the provisioning and management of a composite service is played by the QoS-aware selection of the concrete services from a larger set of candidates. Several works [2, 3, 4, 10, 12, 13] have tackled this issue, proposing exact algorithms or heuristics (*e.g.*, [3] or genetic algorithms in [4]) to optimally determine the appropriate concrete services for each individual component invocation or over the whole composite request. Yu and Lin [12] formulate the problem as a multi-dimension multi-choice 0-1 knapsack as well as a multi-constraint optimal path problem. Zeng et al. [13] present a global planning approach to select an optimal execution plan by means of integer programming. In [2, 10] the service selection is tackled through mixed integer programming. All these proposals perform the optimization on a *per-request* basis that is, they solve the optimization problem for each single request (or even more times per request [13]); moreover, the optimization problems have exponential complexity and can be efficiently solved only via heuristics. Therefore, it appears that the proposed request-based approaches might not be suitable for on-line broker operations where potentially high volumes of service requests must be handled.

To overcome these limitations, we propose a different approach, whereby service selection is carried out per groups of requests rather than per-request. More precisely, we consider flows of requests, where each flow is a sequence of homogenous requests originating by the same user/organization over time, all requiring the same QoS level. We formulate the service selection problem as a Linear Programming optimization problem, which takes into account various non-functional QoS global attributes of the composite service, such as response time, cost, and availability. The solution provided by the optimization problem is used by a service broker, which advertises and offers the composite service with a range of service classes which imply different monetary prices. Differently from the request-based approach to optimal service selection, in our proposal 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 recomputed only when there is a relevant change in the set of SLAs, or in the set of offered QoS classes, or in the set of request flows. Moreover, the broker solves the optimization problem taking into account the coexistence of competing flows of requests generated by multiple requestors, with possibly different QoS constraints agreed in the SLAs. This is not possible in the request-based approach, which could result in sub-optimal and possible instable solutions as it does not take into account the presence of other simultaneous requests.

Our flow-based approach is able to give to each flow of requests a statistical guarantee that its QoS constraints will be actually met. Specifically, our guarantee takes the form of bounds on the expected values of the QoS attributes agreed on for each class. These guarantees are similar to those provided by the request-based approaches and only differ in that they hold on a per-flow rather than a per-request basis. We have introduced the flow-based service selection approach in [6]. However, this paper presents novel and significant contributions, which include the formulation as a Linear Programming problem 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. Section 3 presents the QoS model for the composite service and discusses how to compute its global QoS attributes. Section 4 discusses the formulation of the optimization problem and Section 5 presents some examples of solution of the optimization problem. Finally, Section 6 concludes the paper.

2. Broker architecture

In this section, we outline the broker architecture with its components. We also introduce the composite service model and the notation later used to formulate the optimal selection of the concrete services.

The *service broker* 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. The broker is independently operated and maintained by a third party; it is a Web service itself and advertises the offered composite service in a public registry.

As a first step, the broker defines the business process for the composite service and discovers those concrete services which offer the proper functionalities and are therefore candidates for the selection (we do not focus on the discovery and selection process of a pool of candidates *e.g.*, [9, 11]). For each candidate service, 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 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, we briefly discuss the main components of the broker architecture, assuming that the broker acts as a full intermediary, which really provides the composite service to the requestors. 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*.

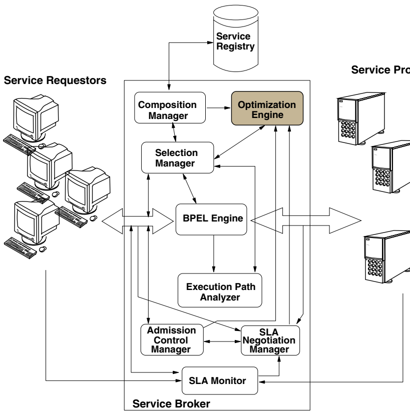

Figure 1. Broker architecture

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. 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 to execute 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 business process, the BPEL Engine creates a new instance of the process itself.

At the beginning of each process instance, the Selection Manager binds each request to the concrete services that meet the contracted QoS level by assigning a real endpoint to each invoked service. The endpoints list is obtained at run time from the solution of the optimization problem provided by the Optimization Engine. The Selection Manager may also trigger a new solution of the optimization problem when some relevant 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 the methodologies underlying the implementation of this broker component.

Finally, the Execution Path Analyzer collects information about the composite service usage, while the SLA Monitor verifies whether the performance perceived by the requestors and offered by the providers complies with the SLAs. These information are used by the Selection Manager to find out whether a new solution of the optimization problem is required. Moreover, the information obtained by the SLA Monitor can be used to take proper actions when a SLA violation is detected (*e.g.*, SLA enforcing [8]).

2.1. Composite service model

We assume that the composite service structure is defined using BPEL [1]. In this paper, we 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, 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

As a running example, throughout the paper we use the Travel Planner (TP) composite service, whose BPEL code is sketched in Figure 2, while Figure 3 illustrates the corresponding workflow with its abstract services. With the exception of the `pick` construct, this example encompasses all the structured activities listed above.

```
...
<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 (TP) BPEL code

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 TP workflow with the concrete services. 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 readability, in the figure, we have renamed the activities with numbers: 1 for the FlightTicketBooking activity, 2 for the HotelBooking activity, etc.

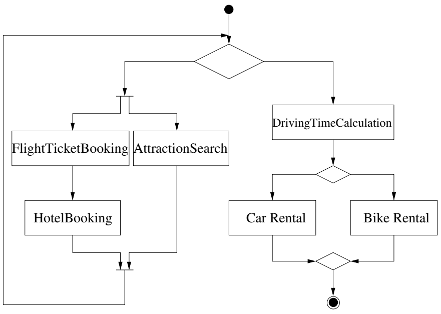

Figure 3. The TP workflow

As the broker acts on behalf of a significant amount of requestors, it is able to identify usage patterns of the composite service. To embody this knowledge in the workflow, we consider an annotated version of the workflow, where each abstract service is annotated with the average number of times it is invoked by each service class. These values can be initialized by the workflow designer and are then periodically updated by the Execution Path Analyzer component, which monitors the workflow executions. Figure 4 shows the annotated TP workflow, where V denotes the number of invocations for each abstract service.

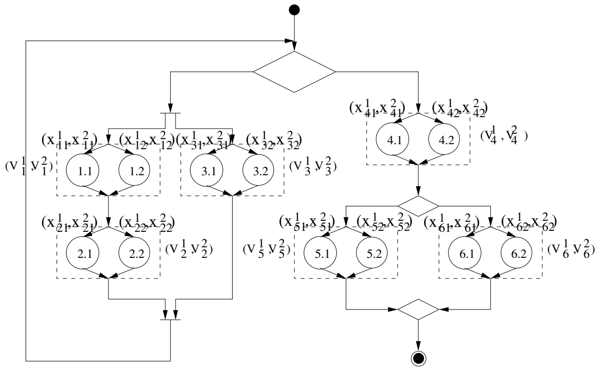

Figure 4. The annotated TP workflow

2.2. SLA model

The broker is involved in the SLA negotiation with two counterparts: the requestors of the composite service and 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, 13]. 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 logarithm of the availability, a_{ij}, *i.e.*, the logarithm of the probability that the concrete service $i.j$ is available when invoked.

As in [13], we consider the *logarithm* of the availability, rather than the availability itself, to obtain linear expressions when composing the availability of different services.

For each concrete service $i.j$, the SLA established by the broker with its 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. The SLA for the concrete service $i.j$ is therefore represented by the tuple $\langle r_{ij}, c_{ij}, a_{ij}, L_{ij} \rangle$, where L_{ij} is the agreed 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 γ^k_u). The SLA established by the broker with the requestor u for the QoS class k is therefore a tuple $\langle R^k_{\max}, C^k_{\max}, A^k_{\min}, \gamma^k_u \rangle$.

2.3. Service selection model

The goal of the Selection Manager is to determine, 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 $\boldsymbol{x}_i = (\boldsymbol{x}^1_i, ..., \boldsymbol{x}^{|K|}_i)$, 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 $i.j$. Figure 4 also shows the probabilities x^k_{ij} for the TP example. With this model, we assume that 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 $x^k_{ij} = 1$ for a given $i.j \in I_i$.

The Selection Manager determines the values of the x^k_{ij} by invoking the Optimization Engine. The optimization problem takes the following general form (the explicit form of the problem will be detailed in Section 4):

$$\begin{aligned} \textbf{max}\ & F(\boldsymbol{x}) \\ \textbf{subject to:}\ & Q^\alpha(\boldsymbol{x}) \leq Q^\alpha_{\max} \\ & Q^\beta(\boldsymbol{x}) \geq Q^\beta_{\min} \\ & S(\boldsymbol{x}) \leq L \\ & \boldsymbol{x} \in A \end{aligned} \tag{1}$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_{|\mathcal{V}|})$ is the decision vector, $F(\boldsymbol{x})$ is a suitable objective function, $Q^\alpha(\boldsymbol{x})$ and $Q^\beta(\boldsymbol{x})$ are, respectively, those QoS attributes whose SLA values are 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 $\sum_{j \in I_i} x^k_{ij} = 1$).

To bind the requests to the concrete services, the Selection Manager uses the solution of this optimization problem as follows. Given a class-k request, the Selection Manager considers only the elements of the solution vector $\boldsymbol{x}$ that pertain to class k. If, for each abstract service i, there is more than one $x^k_{ij} \neq 0$, the Selection Manager selects randomly the concrete service using the x^k_{ij} values.

A new solution of the optimization problem may be triggered when: *a*) the Execution Path Analyzer identifies some change in the average number of visits to the abstract services; *b*) the service composition changes, because either an abstract service or a concrete service is added or removed; *c*) the SLA Monitor detects some violation in the negotiated SLA parameters; *d*) a new requestor, which does not have yet a SLA with the broker, asks for the composite service.

3. Web service QoS model

In this section, we present the QoS model for the composite Web service and how to compute its QoS attributes. For each class $k \in K$ 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 the logarithm of the probability that the composite service is available for a class-k request

depend on: 1) the actual concrete service $i.j$ selected to perform each activity $i \in V$; and, 2) how the services are orchestrated. To compute these quantities, let $Z_i^k(\boldsymbol{x})$ denote the QoS attribute of the abstract service $i \in \mathcal{V}$, $Z \in \{R, C, A\}$. We have $Z_i^k(\boldsymbol{x}) = \sum_{j \in I_i} x_{ij}^k z_{ij}^k$ where z_{ij}^k, $z \in \{r, c, a\}$ is the corresponding QoS attribute offered by the concrete service $i.j$ which can implement i.

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

Cost and Availability. The cost and (logarithm of the) availability QoS metrics are additive [7]. Therefore, for their expected value we readily obtain

$$\begin{aligned} C^k(\boldsymbol{x}) &= \sum_{i \in \mathcal{V}} V_i^k C_i^k(\boldsymbol{x}) &&= \sum_{i \in \mathcal{V}} V_i^k \sum_{j \in I_i} x_{ij}^k c_{ij} \\ A^k(\boldsymbol{x}) &= \sum_{i \in \mathcal{V}} V_i^k A_i^k(\boldsymbol{x}) &&= \sum_{i \in \mathcal{V}} V_i^k \sum_{j \in I_i} x_{ij}^k a_{ij}. \end{aligned}$$

where V_i^k is the expected number of times service i is invoked for a class k request.

Response Time. Differently from cost and availability, the response time metric is additive as long as the composite service does not include `flow` structured activities. In such cases, we readily have:

$$R^k(\boldsymbol{x}) = \sum_{i \in \mathcal{V}} V_i^k \sum_{j \in I_i} x_{ij}^k r_{ij}. \qquad (2)$$

In the general case, instead, we need to account for the fact that the response time of a `flow` activity is given by the largest response time among its component activities. Hence, in the general case, the response time is not additive and (2) does not hold.

In this case, we can still derive an expression for the response time $R^k(\boldsymbol{x})$ by recursively computing the response time of the constituent workflow activities. To this end, it is convenient to represent the BPEL process by means of a 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. 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. We will write $i \prec l$ if node i is a descendant of node l. Figure 5 shows the process activity tree for the TP example. We will 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 $l'' \neq$ `flow`, *i.e.*, if there is no node labeled `flow` in the path from l to l'.

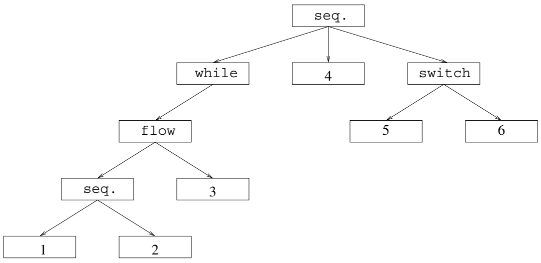

Figure 5. Activity tree for the TP example

We can now state the following theorem, which provides the expressions for the response time of each activity in the BPEL code. Let $\mathcal{F} \subset V$ denote the set of nodes corresponding to `flow` activities and let $\bar{\imath}$ denote the root node.

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

$$R_l^k(\boldsymbol{x}) = \begin{cases} \max_{l' \in d(l)} R_{l'}^k(\boldsymbol{x}) & l \in \mathcal{F} \\ \sum_{i \in \mathcal{V}, i \prec_{dd} l} \frac{V_i^k}{V_l^k} \sum_{j \in I_i} x_{ij}^k r_{ij} + & \\ + \sum_{h \in \mathcal{F}, h \prec_{dd} l} \frac{V_h^k}{V_l^k} R_h^k(\boldsymbol{x}) & l \notin \mathcal{F} \end{cases} \qquad (3)$$

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

$$R^k(\boldsymbol{x}) = \begin{cases} \max_{l' \in d(\bar{\imath})} R^k_{l'}(\boldsymbol{x}) & \bar{\imath} \in \mathcal{F} \\ \sum_{i \in \mathcal{V}, i \prec_{dd} \bar{\imath}} V^k_i \sum_{j \in I_i} x^k_{ij} r_{ij} + \\ + \sum_{h \in \mathcal{F}, h \prec_{dd} \bar{\imath}} V^k_h R^k_h(\boldsymbol{x}) & \bar{\imath} \notin \mathcal{F} \end{cases} \tag{4}$$

Theorem 1 provides the response time $R^k_l(\boldsymbol{x})$ 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^k_l of the *outer* `flow` activities, *i.e.*, `flow` activities which are not nested within other `flow` activities. These expressions can be easily derived by visiting the tree in postorder and properly aggregating the response time of the child nodes $d(i)$ to derive the response time of the parent node i. The theorem proof can be found in [5].

4. Optimization problem

In this section we detail the instance of the general optimization problem outlined in Section 2.3. The Optimization Engine goal is to determine the variables x^k_{ij}, $i \in \mathcal{V}$, $k \in K$, $j \in I_i$ which 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 $Z(\boldsymbol{x}) = \frac{1}{\sum_{k \in K} \gamma^k} \sum_{k \in K} \gamma^k Z^k(\boldsymbol{x})$, where $Z \in \{R, C, A\}$ is the expected overall response time, cost and availability, respectively, and $\gamma^k = \sum_u \gamma^k_u$ is the aggregate flow of class-k requests. We define the objective function as follows:

$$F(\boldsymbol{x}) = w_r \frac{R_{\max} - R(\boldsymbol{x})}{R_{\max} - R_{\min}} + w_c \frac{C_{\max} - C(\boldsymbol{x})}{C_{\max} - C_{\min}} + w_a \frac{A(\boldsymbol{x}) - A_{\min}}{A_{\max} - A_{\min}} \tag{5}$$

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 (logarithm of) availability. We will describe how to determine these values shortly.

The Optimization Engine task consists in finding the variables x^k_{ij}, $i \in \mathcal{V}$, $k \in K$, $j \in I_i$, which solve the following optimization problem:

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

$$\textbf{subject to: } R^k(\boldsymbol{x}) \leq R^k_{\max} \quad k \in K \tag{6}$$

$$R^k_{l'}(\boldsymbol{x}) \leq R^k_l(\boldsymbol{x}) \quad l' \in d(l), l \in \mathcal{F}, k \in K \tag{7}$$

$$\begin{aligned} R^k_l(\boldsymbol{x}) = & \sum_{i \in \mathcal{V}, i \prec_{dd} l} \frac{V^k_i}{V^k_l} \sum_{j \in I_i} x^k_{ij} r_{ij} + \\ + & \sum_{h \in \mathcal{F}, h \prec_{dd} l} \frac{V^k_h}{V^k_l} R^k_h(\boldsymbol{x}), l \notin \mathcal{F}, k \in K \end{aligned} \tag{8}$$

$$C^k(\boldsymbol{x}) \leq C^k_{\max} \quad k \in K \tag{9}$$

$$A^k(\boldsymbol{x}) \geq A^k_{\min} \quad k \in K \tag{10}$$

$$\sum_{k \in K} x^k_{ij} V^k_i \gamma^k \leq L_{ij} \quad i \in \mathcal{V}, j \in I_i \tag{11}$$

$$x^k_{ij} \geq 0, j \in I_i, \sum_{j \in I_i} x^k_{ij} = 1 \quad i \in \mathcal{V}, k \in K \tag{12}$$

Equations (6)-(10) are the QoS constraints for each service class 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 (logarithm of the) availability that characterize the QoS class k. The constraints (7)-(8), which can be easily derived from (3), provide the expressions for the response times. Inequalities (7), in particular, allow us to express the relationship among the response time R^k_l of a `flow` activity and that of its component activities $R^k_{l'}$. Equations (11) are the broker-providers SLA constraints and ensure the broker does not exceed the SLA with the service providers. Finally, equations (12) are the functional constraints.

The maximum and minimum values of the QoS attributes in the objective function (5) 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 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 (11)-(12).

We observe 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 through the Travel Planner service of Figure 3. The composite service offers two QoS classes, *gold* and *silver*, denoted by the superscripts 1 and 2, respectively.

Table 2 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 2. Composite service class attributes

We assume that for each abstract service there are two concrete services which implement it, *i.e.*, $|I_i| = 2$, $i \in \mathcal{V}$ (the resulting workflow is displayed in Figure 4). The concrete services differ in terms of response time, cost, and availability. Table 3 summarizes the services parameters. 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 the number of service invocations: $(V^1_1, V^2_1) = (V^1_2, V^2_2) = (V^1_3, V^2_3) = (1.5, 1.5)$, $(V^1_4, V^2_4) = (1, 1)$, $(V^1_5, V^2_5) = (0.7, 0.5)$, and $(V^1_6, V^2_6) = (0.3, 0.5)$.

Serv.	r_{ij}	c_{ij}	a_{ij}	Serv.	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 3. 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 6 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 6). 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, the response time of abstract service 3 is lower than the sum of the response time of 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.

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 4. Scenario 2: services utilization

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 4 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 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 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.

6. Conclusions

This paper deals with service selection in composite Web services offered by a broker which supports multiple QoS classes. Differently from most of the existing proposals for service selection, we consider SLAs encompassing the overall flow of requests and formulate a constrained optimization problem which can be efficiently solved via standard techniques for linear programming. Our approach is applicable to 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. Moreover, it can be

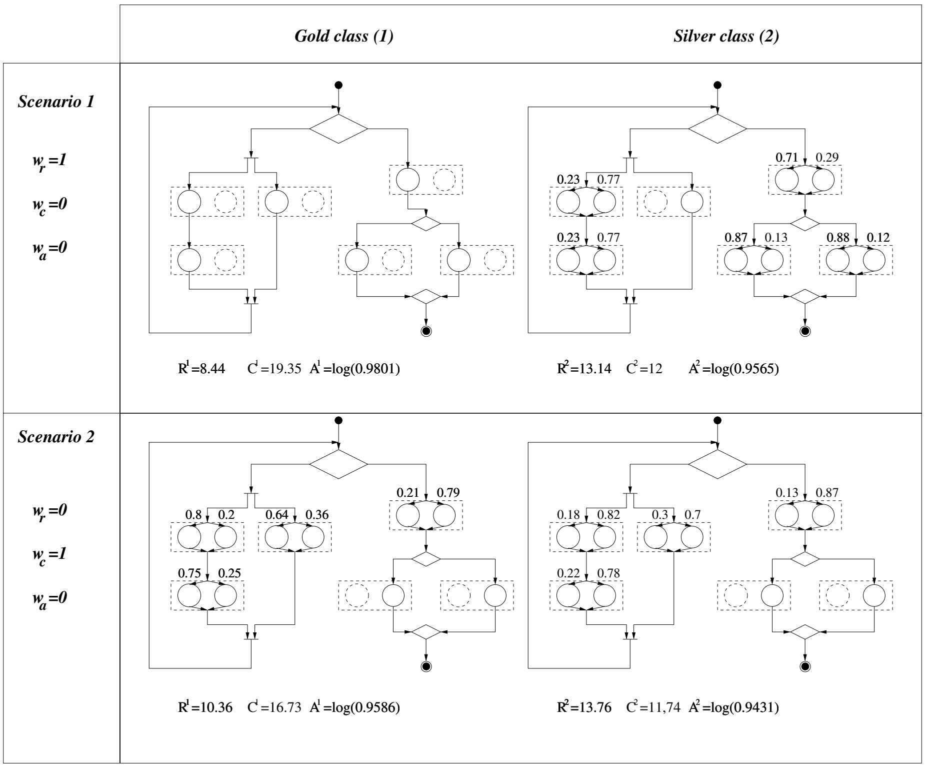

Figure 6. Solution of the optimization problem

easily extended to determine a new resource provisioning that the broker can negotiate with the providers to satisfy a new flow of requests or a change in an existing SLA.

The model proposed in this paper provides a statistical guarantee on the expected QoS attributes. Our future work will address the support for more general types of statistical guarantees (*e.g.*, upper bound on the 99-percentile of the response time), the dynamic re-binding of the concrete services during the process execution, and the sharing of services among multiple brokers.

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.

[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, Sept. 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 GECCO 2005*, June 2005.

[5] V. Cardellini, E. Casalicchio, V. Grassi, and F. Lo Presti. Scalable Service Selection for Web Service Composition Supporting Differentiated QoS Classes. Technical Report DISP RR-07.59, Univ. Roma Tor Vergata, Feb. 2007. `http://www.ce.uniroma2.it/publications/RR-07.59.pdf`.

[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. *J. Web Semantics*, 1(3), 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] E. M. Maximilien and M. P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Comp.*, 8(5):84–93, Sept./Oct. 2004.

[10] Y. Qu, C. Lin, Y. Wang, and Z. Shan. QoS-aware Composite Service Selection in Grids. In *Proc. of Int'l Conf. on Grid and Cooperative Computing*, pages 458–465, Oct. 2006.

[11] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *J. Web Semantics*, 1(1), 2003.

[12] T. Yu and K. J. Lin. Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In *Proc. of ICSOC 2005*, pages 130–143, Dec. 2005.

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