Adaptive Service Composition in Flexible Processes

Danilo Ardagna and Barbara Pernici

Abstract—In advanced service oriented systems, complex applications, described as abstract business processes, can be executed by invoking a number of available Web services. End users can specify different preferences and constraints and service selection can be performed dynamically identifying the best set of services available at runtime. In this paper, we introduce a new modeling approach to the Web service selection problem that is particularly effective for large processes and when QoS constraints are severe. In the model, the Web service selection problem is formalized as a mixed integer linear programming problem, loops peeling is adopted in the optimization, and constraints posed by stateful Web services are considered. Moreover, negotiation techniques are exploited to identify a feasible solution of the problem, if one does not exist. Experimental results compare our method with other solutions proposed in the literature and demonstrate the effectiveness of our approach toward the identification of an optimal solution to the QoS constrained Web service selection problem.

Index Terms—Web services, quality of service, service composition, integer programming.

1 Introduction

In service oriented environments, complex applications can be described as processes invoking services selected at runtime. In this scenario, applications are defined as flexible processes composed of abstract Web services. Web services are selected from a set of functionally equivalent services, that is, services which implement the same functionality but differ for nonfunctional characteristics, i.e., Quality of Service (QoS) parameters. The goal is to select the best set of services available at runtime, taking into consideration process constraints, but also end-user preferences and the execution context.

The Web service selection problem has been studied for business processes and e-science. Dynamic Web service selection for composed Web services focused in particular on context aware business processes. Context awareness may be needed both when considering Web service personalization, where a generic process is personalized choosing services according to user preferences [10], and in mobile composed services, to provide ubiquitous services where selection and execution depend on the available services and their QoS [29]. An interesting application area of service selection optimization is e-science. Complex processes, defined as workflows in this research context, based on grid technology are being developed, reaching the dimension of thousands of tasks in "in silico" experiments [20]. Each task is performed selecting and invoking a service.

Web service selection results in an optimization problem that has been studied both in the research areas of service oriented computing for business processes and of grid environments. The literature has provided *two generations* of solutions.

First generation solutions implemented *local* approaches [26], [35], [3], which select Web services one at the time by associating the running abstract activity to the best candidate service which supports its execution. Local approches can guarantee only local QoS constraints, i.e., candidate Web services are selected according to a desired characteristic, e.g., the price of *a single Web service invocation* is lower than a given threshold.

Second generation solutions proposed *global* approaches [35], [7], [13], [21]. The set of services that satisfy the process constraints and user preferences for the whole application are identified before executing the process. In this way, QoS constraints can predicate at a global level, i.e., constraints posing restrictions over the *whole composed service execution* can be introduced. In order to guarantee the fullfillment of global QoS constraints, second generation optimization techniques consider the worst case execution scenario for the composed service. For cyclic processes, loops are unfolded, i.e., unrolled according to their maximum number of iterations [35], [7]. These approaches could be very conservative and constitutes the main limitation of second generation techniques.

Furtheremore, global approaches introduce an increased complexity with respect to local solutions. The main issue for the fulfillment of global constraints is Web service performance variability. Indeed, the QoS of a Web Service may evolve relatively frequently, either because of internal changes or because of workload fluctuations [35], [12], [36]. If a business process has a long duration, the set of services identified by the optimization may change their QoS properties during the process execution or some services can become unavailable or others may emerge. In order to guarantee global constraints Web service selection and execution are interleaved: Optimization is performed when

• *The authors are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Via Ponzio 34/5, 20133 Milano, Italy. E-mail: {ardagna, pernici}@elet.polimi.it.*

Manuscript received 29 June 2006; revised 20 Aug. 2006; accepted 15 Mar. 2007; published online 3 Apr. 2007.
Recommended for acceptance by S. Donatelli.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0019-0106.
Digital Object Identifier no. 10.1109/TSE.2007.1011.

the business process is instantiated and its execution is started, and is iterated during the process execution performing *reoptimization* at runtime.

To reduce optimization/reoptimization complexity, a number of solution have been proposed that guarantee global constraints only for the critical path [35] (i.e., the path which corresponds to the highest execution time), or reduce loops to a single task [7], satisfying global contraints only statistitically, by applying the reduction formula proposed in [8].

Another drawback of second generation solutions is that, if the end-user introduces severe QoS constraints for the composed service execution, i.e., limited resources which set the problem close to unfeasibility conditions (e.g., limited budget or stringent execution time limit), no solutions can be identified and the composed service execution fails [7].

While first and second generation approaches have been applied, e.g., [6], [35], the need for further research toward more advanced optimization techniques, in particular for cyclic processes [20], [3], is advocated. In addition, none of the previous approaches considers in the optimization the case of processes composed by *stateful Web services*, where more than one task must be performed by the same Web service.

The goal of this paper is to set the basis to overcome the limits of the previous approaches to Web services selection. The aim is to discover the optimum mapping between each abstract Web service of a flexible process and a Web service that implements the abstract description, such that the overall QoS perceived by the user is maximized under severe QoS constraints. Severe constraints are very relevant whenever processes have to be performed with stringently limited resources. We introduce a new modeling approach to the service selection problem, based on the following main contributions: 1) loops peeling is adopted in the optimization, which significantly improves the solutions based on loops unfolding, 2) negotiation is exploited if a feasible solution cannot be identified, to bargain QoS parameters with service providers offering services, reducing process invocation failures, and 3) a new class of global constraints, which allows the execution of stateful Web service components, is introduced. Furthermore, we extend other literature approaches identifying the optimal solution of the Web service selection problem instead of identifying suboptima as in [7], [13], [21]. As will be discussed in the remainder of the paper, our joint optimization and negotiation approach is effective in particular for large processes, when QoS constraints are severe, and it reduces the reoptimization overhead.

Our approach is implemented within the MAIS (Multichannel Adaptive Information Systems) architecture, a platform which supports the execution of flexible processes in multichannel adaptive systems [29].

The remainder of the paper is organized as follows: An overview of the MAIS architecture is reported in Section 2. Section 3 introduces the composed service model and specification and the set of quality dimensions considered in the optimization problem. The composition approach, including optimization and reoptimization problem formulation and negotiation, is discussed in Section 4. Experimental results in Section 5 demonstrate the effectiveness of our solutions. Section 6 discusses other literature approaches. Conclusions are drawn in Section 7.

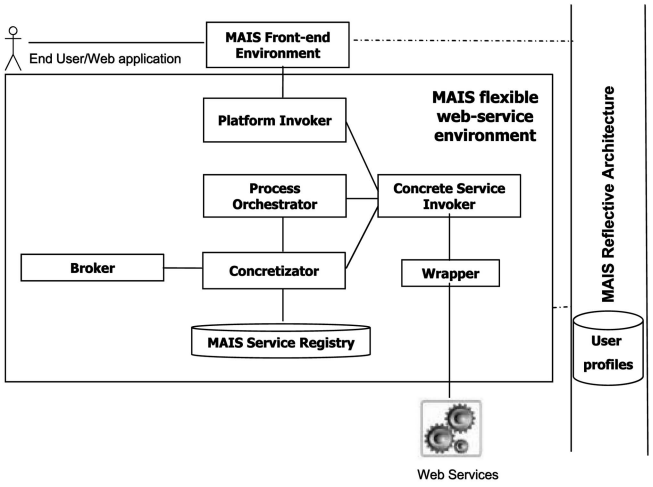


Fig. 1. MAIS architecture.

2 The MAIS Architecture

The adaptive service composition presented in this paper is based on the approach to flexible Web services introduced in the MAIS project[1] [29]. In the MAIS framework, Web service invocation is based on the dynamic selection of concrete services at runtime. The user or front-end application which invokes a Web service may specify only its abstract interface requirements and quality of service constraints. As shown in Fig. 1, the flexible Web service environment provides a *Concrete Service Invoker* service, which invokes the concrete service corresponding to user's requirements, chosen from a service registry by a *Concretizator* module. The MAIS service registry, an extension of a UDDI registry, also provides, for each concrete service, information on its abstract interface and quality of service characteristics. The Concretizator selects the best concrete services to be invoked for each task of the composed service according to the optimization criteria discussed in the remainder of this paper. Furthermore, a *Broker* service allows negotiation of QoS parameters. The selection is performed by using both contextual information and user preferences, as specified in the service request. Context information is provided through the *MAIS Reflective Architecture* and includes both the user's context of invocation, including the user profile, information about the invocation channel, and the user's device. Flexible services allow the invocation of different services in different user's contexts, such as, for instance, a high resolution image information service when bandwidth is not constrained or a textual information service when a low bandwidth wireless connection is available and interaction is on a small portable device. We assume that the invocation context can vary over time, so at different points in time different services can be invoked due to modifications of the context.

In this paper, we focus on techniques to provide flexible composed Web services, expressed as processes composed

1. Multichannel Adaptive Information Systems project Web site: http://www.mais-project.it.

of abstract Web services. We assume that the structure of such flexible composed Web services is fixed and local and global QoS constraints can be specified.

3 Composed Service Model

This section introduces the specification, execution and QoS model of a MAIS composed Web service. First, in Section 3.1, we discuss the specification of a composed Web service. Then, in Section 3.2, the execution model is presented. The set of quality dimensions considered in the optimization problem formulation are presented in Section 3.3.

3.1 Process Specification

A Web service is modeled as a software component that implements a set of operations. A composed service is specified at an abstract level as a high-level business process. We assume that a composed service is characterized by a single initial task and a single end task and that task composition follows a block structure so that, in particular, only structured loops can be specified, i.e., loops with only one entry and exit point. In the remainder of the paper, we represent composed services by UML activity diagrams, where activities represent tasks to be executed by Web services. The operational language we use for process implementation is BPEL [18].

In the following, we refer to component abstract Web services operations to be executed in the process with the term *task* (t_i), while Web services selected to be executed are called *concrete Web services* (ws_j). The notation adopted in the paper is summarized in the Appendix. To support adaptive concretization, a set of *semantic annotations* are associated to the process specification to specify either intrinsic characteristics of the process, or requirements by the user of the composed service:

- *Probability of execution of conditional branches*. For every switch s, the probability of execution $\{p_1^s, p_2^s, \ldots, p_{NB^s}^s\}$ of conditional branches is specified ($\sum_{h=1}^{NB^s} p_h^s = 1$, NB^s indicates the number of disjoint branch conditions of s).
- *Loop constraints*. The expected maximum number of iteration NI^l is defined for every loop l; the probability distribution $\{p_0^l, p_1^l, \ldots, p_{NI^l}^l\}$ of the loop number of iterations is specified ($\sum_{h=0}^{NI^l} p_h^l = 1$, p_0^l indicates the probability that the loop is not executed, p_1^l indicates the probability that the loop is executed once, and so on).
- *Global and local constraints on quality dimensions.* Global constraints specify requirements at the process level, while local constraints define the quality of Web services to be invoked for a given task in the process. We assume that quality constraints may be defined on a set of N predefined quality dimensions q_n.
- *Web service dependency constraints*. Impose that a given set of tasks in the process are executed by the same Web service. This type of constraint allows considering both stateless and stateful Web services in composed services.

The probability of execution of conditional branches and the distribution of loops number of iterations can be evaluated from past executions by inspecting system logs or can be specified by the composed service designer [7], [35]. We assume that for every loop l, an upper bound NI^l for the loop number of iterations is determined. Otherwise, if an upper bound does not exist, the process cannot be optimized since infinite resources might be needed for its execution and global constraints cannot be guaranteed [35]. Local constraints can be specified by the composed service designer. Vice versa, the user can specify only global constraints since we assume the user has no knowledge of the structure of the composed process. For the same reason, Web service dependency constraints are specified by the composed service designer. User preferences may be either specified explicitly by the user requesting the service, or can be implicit in the user profile and, therefore, the same for all service requests, or not specified at all. In the last case, all quality dimensions are considered at the same level of preference giving each dimension a weight $1/N$.

3.2 Process Execution

When a composed service is invoked, the BPEL specification is analyzed and the set of candidate Web services for executing its component tasks is retrieved from the MAIS registry. Services are selected from the registry by considering the signature of the operation to be performed and according to the specified local quality constraints for the tasks in the process. In the following, Web services will be indexed by j and we will indicate with $\mathcal{WS}_i$ the set of indexes of Web services ws_j candidate for the execution of task t_i, with $\mathcal{OP}_j$ the set of indexes of operations implemented by Web service ws_j, and with $ws_{j,o}$ the invocation of operation $o \in \mathcal{OP}_j$ of Web service ws_j. Let I be the number of tasks of the composed service specification and J the number of candidate Web services retrieved from the MAIS registry.

The goal of the concretization process illustrated in Section 4.2 is to determine the optimum execution plan EPL^* of the composed process, i.e., the set of ordered couples $\{(t_i, ws_{j,o})\}$, indicating that task t_i is executed by invoking $ws_{j,o}$ for all tasks in the process, such that the overall QoS perceived by the user for the application instance execution is maximized, while (local) global and dependency constraints are guaranteed.

During execution, quality constraints may be violated due to a number of reasons: First, quality values considered in the optimization are the ones advertised by Service Providers and are subject to variability (due, for instance, to performance changes as a consequence of workload fluctuations). Furthermore, global constraints could be violated as a consequence of a failure in an operation invocation. At runtime, process execution and optimization are interleaved. Criteria for executing the reoptimization step are defined and discussed in Section 4.4.

3.3 The Quality Model

Several quality criteria can be associated with Web services execution. The MAIS service registry includes about 150 relevant quality dimensions. For each dimension, a definition, a metric, and a measuring system are proposed. A comprehensive discussion of quality dimensions can be found in [29, Appendix A]. In the present paper, we assume that quality values are real numbers that vary in a bounded range with a minimum and a maximum value. Note that, if the same operation is accessible from the same Web service and the same provider, but with different quality characteristics, then multiple copies of the same operation will

be stored in the registry, each copy being characterized by its quality profile.

In the following, quality dimensions will be analyzed according to their *aggregation pattern*, the *negotiability* property, and will be classified as *positive* and *negative* qualities. The quality *aggregation pattern* defines how the value of a given quality dimension for a composed service can be determined starting from the value of quality of component services. The following typologies of quality aggregation functions are considered: weighted sum, product, min, or max of the corresponding quality dimension of component services. Quality dimensions are defined as *negotiable* when they can be contracted between the Broker and the provider. The Broker can express preferences or define constraints on these dimensions, which, in the service selection phase, are written in a service level agreement contract. Negotiable quality dimensions are used in the service selection phase in order to identify a feasible solution of the concretization problem if it does not exist. A quality dimension can be also classified as *positive* and *negative* criteria. A quality attribute is positive (negative) if the higher the value the higher (the lower) the quality. In the process optimization, in order to guarantee constraints, the minimum (maximum) values advertised by Service Providers are considered for positive (negative) quality dimensions.

In this paper, examples are based on a subset of quality dimensions, which have been the basis for QoS consideration also in other approaches [11], [35], [27] and which are representative for every dimension of analysis discussed above. The approach proposed for the classical dimensions of the literature could be easily generalized to other dimensions of the MAIS framework. The following subset of quality dimensions is considered:

- *Execution time* $e_{j,0}$. The expected delay between the time instant when a request is sent ($ws_{j,o}$ is invoked) and the time when the result is obtained. Execution time is measured in seconds.
- *Availability* $a_{j,0}$. The probability that the service operation $ws_{j,o}$ is accessible. Availability is a number in the range [0, 1].
- *Price* $p_{j,0}$. The fee that a service requester has to pay to the Service Provider for the service invocation $ws_{j,o}$. Price is measured in dollars ($).
- *Reputation* $r_{j,0}$. A measure of the service invocation $ws_{j,o}$ trustworthiness. It is defined as the ratio between the number of service invocations which comply the negotiated QoS over the total number of service invocations. Reputation is a number in the range [0, 1].
- *Data quality* $d_{j,0}$. The ability of a data collection to meet user requirements , defined as the proximity of a value v returned by $ws_{j,o}$ to a value v' considered as correct. The measure of data quality is considered here as a real number in the range [0, 1], where 1 represents the most desirable score.

The selected quality dimensions represent all types of aggregation patterns: The execution time of a composed service is given by the sum of execution time of invoked services. Availability is given by the product of availabilities provided by component services. The reputation is the average reputation of selected services. As in [29], the aggregate value of data quality is given by the minimum value of data quality of invoked services. With respect to negotiability, we assume that price, execution time, and data quality are negotiable, while availability and reputation are not. Finally, availability, reputation, and data quality are examples of positive criteria, price and execution time are negative criteria.

Fig. 2a shows an example of a composed service specification, with corresponding semantic annotations and constraints. Concrete services information are specified as shown in Fig. 2b.

4 Interleaving Web Service Selection and Execution

In our approach, Web service selection and optimization and Web service execution are interleaved. Optimization is computed when the composed service execution starts; reoptimization is performed periodically at runtime with the time interval varying according to environment changes and user behavior. Furthermore, negotiation gives another degree of freedom for the solution of optimization and reoptimization problems. If a feasible solution cannot be identified, QoS parameters are bargained with Service Providers, which implement negotiation protocols.

Section 4.1 introduces some basic concepts that will be used in the remainder of the paper. The mathematical formulation of the optimization problem is provided in Section 4.2. The negotiation approach is presented in Section 4.3. Reoptimization is discussed in Section 4.4.

4.1 Process Graph Transformations and Preliminary Definitions

If the BPEL specification includes some loops (*while* construct), then they are peeled [4] prior to start the optimization process. Loop Peeling is a form of loop unrolling in which loop iterations are represented as a sequence of branches (see Fig. 3a). Each branch condition evaluates if the loop l has to continue with the next iteration (according to the probability distribution $\{p_h^l\}$) or it has to exit. As discussed in Section 3.1 and in [35], if the process includes a single entry and exit task, then, after loops peeling, a composed Web service can be modeled as a Directed Acyclic Graph (DAG).

We adopt the following definitions that we introduced in [3]:

Execution Path. A set of tasks $\{t_1, t_2, \ldots, t_I\}$ such that t_1 is the initial task, t_I is the final task, and no t_{i_1}, t_{i_2} belong to alternative branches. Execution paths will be denoted by ep_k. As shown in the example reported in Fig. 3b, an execution path can include parallel sequences. A probability of execution $freq_k$ is associated with every execution path and can be evaluated as the product of the probability of execution of the branch conditions included in the execution path. In the example above, ep_1, where the loop l is not executed, has probability $freq_1 = p_0^l$; ep_2, where l is executed once, has probability $freq_2 = \left(1 - p_0^l\right) \cdot \frac{p_1^l}{1-p_0^l} = p_1^l$; ep_3, where l is executed twice, has probability $freq_3 = \left(1 - p_0^l\right) \cdot \left(1 - \frac{p_1^l}{1-p_0^l}\right) \cdot \frac{p_2^l}{1-p_0^l-p_1^l} = p_2^l$ and so on. Note that the set of execution paths of an activity diagram identifies all the possible execution scenarios of the composed service. The optimization problem will consider all of the possible

process.bpel

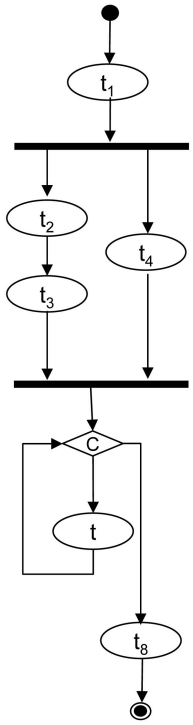
```
<process>
  <sequence>
     <invoke t1 />
     <flow>
       <sequence>
           <invoke t2 />
           <invoke t3 />
       </sequence>
       <invoke t4 />
     </flow>
     <while name="loop1" cond=C>
           <invoke t />
     </while>
     <invoke t8 />
  </sequence>
<process>
```

constraints.xml

```
<UserProfile name=user1Channel1>
    <exeTimeWeight value=0.7 />
    <reputationWeight value=0.3 />
    <priceConstraint value=5 />
</UserProfile>
<UserProfile name=user1Channel2>
    <exeTimeWeight value=0.5 />
    <reputationWeight value=0.5 />
    <priceConstraint value=4 />
</UserProfile>
```

annotation.xml

```
<annotation>
  <while name="loop1" >
    <probability number="0" value=p0 />
    <probability number="1" value=p1 />
    <probability number="2" value=p2 />
    <probability number="3" value=p3 />
  </while>
</annotation>
```



concreteServices.xml

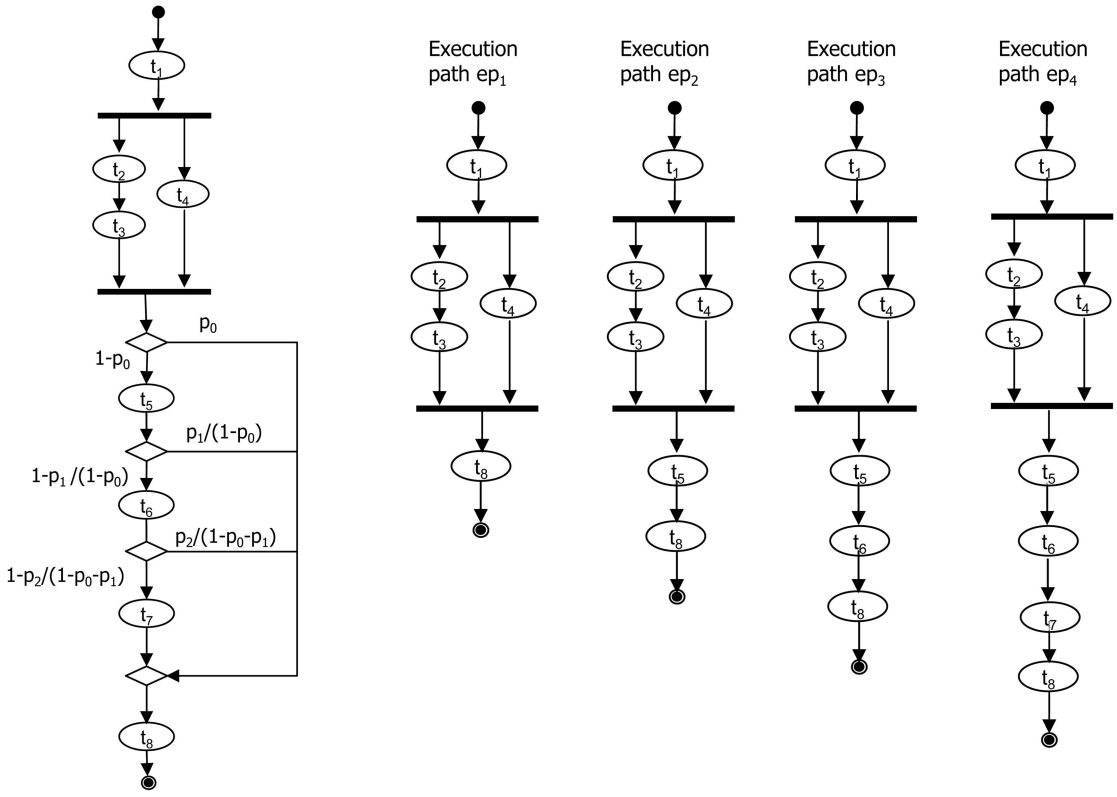
```
<?xml version="1.0" ?>

<webServices>
   <webService name="ws_1">
       <operation name="ws_1_1">
           <maxExeTime value=6.76 />
           <minExeTime value=5.767389 />
           <maxPrice value=1.12 />
           <minPrice value=0.19 />
           <maxReputation value=0.89 />
           <minReputation value=0.82 />
           <maxAvailability value=0.99 />
           <minAvailability value=0.95 />
           <maxDataQuality value=0.98 />
           <minDataQuality value=0.95 />
       </operation>
   .
   .

   </webService>
   .
   .
   .
</webServices>
```

(a) (b)

Fig. 2. Process specification and annotations.



(a) (b)

Fig. 3. Loop peeling and process execution paths.

TABLE 1
QoS Aggregation Function

Quality Dimension	Aggregation function
Price	$price_k(EPL) = \sum_{\substack{t_i \in ep_k, \\ (t_i, ws_{j,o}) \in EPL}} p_{j,o}$
Reputation	$rep_k(EPL) = \frac{1}{\lvert\mathcal{A}_k\rvert} \sum_{\substack{t_i \in ep_k, \\ (t_i, ws_{j,o}) \in EPL}} r_{j,o}$
Execution Time	$exeTime_k(EPL) = \max_{sp_m^k \in ep_k} \sum_{\substack{t_i \in sp_m^k, \\ (t_i, ws_{j,o}) \in EPL}} e_{j,o}$
Availability	$avail_k(EPL) = \prod_{\substack{t_i \in ep_k, \\ (t_i, ws_{j,o}) \in EPL}} a_{j,o}$
Data quality	$dq_k(EPL) = \min_{\substack{t_i \in ep_k, \\ (t_i, ws_{j,o}) \in EPL}} d_{j,o}$

execution scenarios according to their probability of execution. In the following, $\mathcal{A}_k$ will denote the set of indexes of tasks included in the execution path ep_k.

Subpath. A subpath of an execution path ep_k is a sequence of tasks $[t_1, t_2, , t_l]$, $t_i \in ep_k \; \forall i$, from the initial to the end task, that does not contain any parallel sequence. Subpaths will be indexed by m and denoted by sp_m^k. In our example, ep_1 has two subpaths $sp_1^1 = [t_1, t_2, t_3, t_8]$ and $sp_2^1 = [t_1, t_4, t_8]$.

For a generic execution plan EPL, the quality dimensions can be formally evaluated under the hypothesis that the composed service is executed along an execution path ep_k using the aggregation patterns discussed in Section 3.3, as illustrated in Table 1.

4.2 Optimization Problem Formulation

In this section, we formulate the Web Service Concretization (WSC) problem as a mixed integer linear programming (MILP) problem. The decision variables of our model are the following:

$z_{i,j}$:= equals 1 if the task t_i is executed by Web service ws_j, $j \in \mathcal{WS}_i$, 0 otherwise.

$y_{i,j,o}$:= equals 1 if the task t_i is executed by $ws_{j,o}$, i.e., by invoking operation $o \in \mathcal{OP}_j$ of Web Service ws_j with $j \in \mathcal{WS}_i$, 0 otherwise.

Note that the variable $\mathcal{Y} = [y_{i,j,o}]$ is the characteristic vector of a generic execution plan EPL; in the following, execution plans will be represented by their characteristic vectors.

The goal of the WSC problem is to maximize the aggregated value of QoS, considering all of the possible execution scenarios, i.e., all of the execution paths arising from the composed service specification, and their probability of execution $freq_k$.

The aggregated value of QoS is obtained by applying the Simple Additive Weighting (SAW) technique [22], one of the most widely used techniques to obtain a *score* from a list of dimensions. Let us denote with $score_k(\mathcal{Y})$ the aggregated value of QoS of the execution plan $\mathcal{Y}$ along the execution path ep_k. Since the quality dimensions q_n have different units of measure, the SAW method first normalizes the raw values for each quality dimension. Each quality dimension q_n is also associated with a weight w_n in the process execution request (see Section 3.1). Let us denote with q_n^k the aggregated value of the quality dimension q_n evaluated along the execution path ep_k. The score of an execution plan (i.e., its overall value of QoS) is calculated as a weighted sum of the normalized values of quality dimension.

In the normalization phase, positive and negative criteria are scaled in different ways, as defined in (1) and (2), respectively:

$$v_n^k(\mathcal{Y}) = \begin{cases} \frac{q_n^k(\mathcal{Y}) - \min q_n^k}{\max q_n^k - \min q_n^k} & \text{if } \max q_n^k \neq \min q_n^k \\ 1 & \text{if } \max q_n^k = \min q_n^k, \end{cases} \quad (1)$$

$$v_n^k(\mathcal{Y}) = \begin{cases} \frac{\max q_n^k - q_n^k(\mathcal{Y})}{\max q_n^k - \min q_n^k} & \text{if } \max \mathrm{q_n^k} \neq \min \mathrm{q_n^k} \\ 1 & \text{if } \max \mathrm{q_n^k} = \min \mathrm{q_n^k}, \end{cases} \quad (2)$$

where $\min q_n^k = \min_{\forall \mathcal{Y}} q_n^k(\mathcal{Y})$ and $\max q_n^k = \max_{\forall \mathcal{Y}} q_n^k(\mathcal{Y})$ indicates the minimum and maximum values, respectively, for the nth quality dimension along the execution path ep_k. Note that, if $\max q_n^k = \min q_n^k$, then, along the execution path ep_k, every execution plan is characterized by the same value for the quality dimension n, the quality dimension is not a differential characteristic for the execution path ep_k, and $v_n^k(\mathcal{Y})$ is set to 1. As discussed in [35], the assessment of the maximum and minimum value of each quality dimensions can be done without considering all possible execution plans, since, for example, the execution plan of maximum price can be obtained by assigning to each task the candidate service of maximum price. The normalization phase complexity is linear in the number of tasks of the composed process.

The overall score along the execution path ep_k associated with an execution plan $\mathcal{Y}$ is evaluated as

$$score_k(\mathcal{Y}) = \sum_{n=1}^{N} w_n v_n^k(\mathcal{Y}). \quad (3)$$

Let $\mathcal{B}$ the set of indexes of tasks included in any loop of the composed service specification. Let us indicate with K the number of different execution paths arising from the composed service specification. Let E be the execution time global constraint for the composed service execution; while A, R, and DQ indicate the availability, the reputation, and the data quality global constraints, respectively. Let B be the price global constraint, which will be considered as the budget for the composed service execution. In order to consider the optimization overhead, let $OptTime$ be an estimate of the time required to optimize the composed

service and let $E' = E - OptTime$. The optimum execution plan EPL^* can be determined by solving the following problem:

P1. $\max \sum_{k=1}^{K} freq_k \cdot score_k(\mathcal{Y})$.

$$\sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} y_{i,j,o} = 1; \quad \forall i \quad (4)$$

$$y_{i,j,o} \leq z_{i,j}; \quad \forall i; \forall j \in \mathcal{WS}_i; \forall o \in \mathcal{OP}_j \quad (5)$$

$$\sum_{j \in \mathcal{WS}_i} z_{i,j} = 1; \quad \forall i \quad (6)$$

$$\sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} e_{j,o} y_{i,j,o} = exeT_i; \quad \forall i \quad (7)$$

$$x_{i_2} - (exeT_{i_1} + x_{i_1}) \geq 0; \quad \forall t_{i_1} \rightarrow t_{i_2} \quad (8)$$

$$\sum_{i \in sp_m^l} exeT_i \leq exeTime_k; \quad \forall sp_m^k \in ep_k \quad (9)$$

$$avail_k = \prod_{i \in \mathcal{A}_k} \prod_{j \in \mathcal{WS}_i} \prod_{o \in \mathcal{OP}_j} a_{j,o}^{y_{i,j,o}} \quad \forall k \quad (10)$$

$$price_k = \sum_{i \in \mathcal{A}_k} \sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} p_{j,o} y_{i,j,o} \quad \forall k \quad (11)$$

$$rep_k = \frac{1}{|\mathcal{A}_k|} \sum_{i \in \mathcal{A}_k} \sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} r_{j,o} y_{i,j,o} \quad \forall k \quad (12)$$

$$\sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} d_{j,o} y_{i,j,o} \geq dq_k; \quad \forall k; \forall i \in \mathcal{A}_k \quad (13)$$

$$\sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} r_{j,o} y_{i,j,o} \geq R; \quad \forall i \in \mathcal{B} \quad (14)$$

$$exeTime_k \leq E'; \quad \forall k \quad (15)$$

$$avail_k \geq A; \quad \forall k \quad (16)$$

$$price_k \leq B; \quad \forall k \quad (17)$$

$$rep_k \geq R; \quad \forall k \quad (18)$$

$$dq_k \geq DQ; \quad \forall k \quad (19)$$

$$y_{i,j,o}, z_{i,j} \in \{0,1\}; \quad \forall i; \forall j \in \mathcal{WS}_i; \forall o \in \mathcal{OP}_j$$

$$exeT_i, x_i \in \mathbb{R}^+ \quad \forall i;$$

$$exeTime_k, avail_k, price_k, rep_k, dq_k \in \mathbb{R}^+; \quad \forall k.$$

Constraints family (4) guarantees that every task is associated to exactly one Web service operation invocation (i.e., for every i, only one variable $y_{i,j,o}$ is set to 1). In the same way, (6) entails that every task is associated to exactly one Web service. Constraint family (5) relates $y_{i,j,o}$ and $z_{i,j}$ variables; indeed if the task t_i is executed by invoking operation $ws_{i,j}$, i.e., $y_{i,j,o} = 1$, then $z_{i,j}$ is raised to one. Constraints family (7) expresses the duration of every task in term of the duration of the selected service (note that for (4), only one operation invocation is selected and, hence, the duration of a task is given by the selected Web service operation execution time). Constraints family (8) represents precedence constraints for subsequent tasks in the activity diagram. The variable x_i indicates task t_i's starting time instant. Let us denote with $t_{i_1} \rightarrow t_{i_2}$, $i_1, i_2 \in I$, that task t_{i_2} is a direct successor of task t_{i_1}. Constraints family (8) entails that the execution of task t_{i_2} can start only after task t_{i_1} termination. Constraints family (9) evaluates the duration of every execution path as the maximum execution time over the set of subpaths of the execution path (see Table 1). Indeed, the maximum value v_{max} of a set V is defined as the value in the set ($v_{max} \in V$) such that $v \leq v_{max}$, $\forall v \in V$, and the first term of (9) represents the duration of the subpath sp_m^k. Constraints families (10) through (13) express execution path ep_k availability, price, reputation, and data quality of an execution path according to the rules of Table 1. Constraints families (15) through (19) are the global constraints to be fulfilled. Finally, (14) guarantees the fulfillment of the reputation constraint for composed services which include some loops. The rationale can be explained by considering the example reported in Fig. 2. Let us assume that the expected maximum number of iteration of the loop is an even number and the reputation of service invocations outside the loop is equal to R. Without (14), the task t could be executed by alternatively invoking ws_{j_1,o_1} with reputation $r_1 = R + \Delta$ and ws_{j_2,o_2} with reputation $r_2 = R - \Delta$, with $\Delta > 0$. If, at runtime, the number of iteration of the loop is an odd number, the reputation constraint is violated. This drawback can be avoided by introducing in a conservative way (14), for tasks in a loop of the composed service specification.

Problem P1 can include Web service selection constraints and can also formally encompass local constraints. Web service selection constraints can be formulated as follows. If two tasks t_{i_1} and t_{i_2}, $i_1, i_2 \in I$, must be executed by the same Web service, the following constraint families are introduced:

$$z_{i_1,j} = z_{i_2,j}, \ \forall j \in \mathcal{WS}_{i_1} \cap \mathcal{WS}_{i_2};$$
$$z_{i_1,j} = 0, \ \forall j \in \mathcal{WS}_{i_1} \setminus \mathcal{WS}_{i_2};$$
$$z_{i_2,j} = 0, \ \forall j \in \mathcal{WS}_{i_2} \setminus \mathcal{WS}_{i_1}.$$

Local constraints can predicate on properties of a single task and can be formally included in the model as follows. For example, if the designer requires that the price for task t_{i_1} has to be less or equal than a given value $\overline{p}$, then the constraint

$$\sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} p_{j,o} y_{i_1,j,o} \leq \overline{p}$$

is introduced. Local constraints are enforced when Web services are selected from the MAIS service registry (see Section 3.2). Indeed, if a Web service does not satisfy local constraints, then it can be filtered from the list of candidate Web services, reducing the number of variables of the model.

The Problem P1 has integer variables and a nonlinear objective function and constraints (the availability term in the objective function is nonlinear as (10) and (16)). Availability constraints families can be linearized by applying the logarithm function. Equation (16) then becomes

$$\sum_{i \in \mathcal{A}_k} \sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} \ln(a_{j,o}) y_{i,j,o} \geq \ln(A) \quad \forall k. \quad (20)$$

The objective function is linearized in the same way. Every term

$$\frac{avail_k(\mathcal{Y}) - \min avail_k}{\max avail_k - \min avail_k}$$

is replaced by

$$\frac{\ln(avail_k(\mathcal{Y})) - \min \ln(avail_k)}{\max \ln(avail_k) - \min \ln(avail_k)} = \frac{\sum_{i \in \mathcal{A}_k} \sum_{j \in \mathcal{WS}_i} \sum_{o \in \mathcal{OP}_j} \ln(a_{j,o}) y_{i,j,o} - \min \ln(avail_k)}{\max \ln(avail_k) - \min \ln(avail_k)}.$$

In this way, the WSC problem can be reduced to an MILP model.

In [3], we have shown that a WSC problem for a process with a block structure is equivalent to a Multiple choice Multiple dimension Knapsack Problem (MMKP), which is NP-hard. Every instance of the MMKP can be formulated as a WSC; hence, the WSC problem is NP-hard.

The optimum execution plan EPL^* assigns the optimum candidate service operation invocation $ws_{j,o}$ to each task t_i. After the evaluation of EPL^*, for every task t_i, the set of candidate services that guarantee the fulfillment of WSC constraints are ranked according to the value provided to the objective function of Problem P1. The goal is to obtain a ranking of services which can be invoked as substitute services in case of a failure. In this way, the delay associated with the invocation of a failed service invocation is minimized, since the substitute service is invoked without waiting for the result of the reoptimization.

4.3 Negotiating QoS Parameters

If a feasible solution for the WSC problem does not exist, negotiation is performed in order to determine new quality values for Web service invocations. For example, if the data quality global constraints cannot be fulfilled, the Broker can ask service providers to improve the data quality of service invocations (e.g., by performing data cleaning procedures), which will be provided at a higher price in turn. If the Broker and a provider find an agreement on the new price and quality parameters for a given operation invocation, a new candidate operation invocation is considered in the optimization/reoptimization process, i.e., successful negotiations enlarge the solution domain of the optimization problem.

If the WSC problem is infeasible, first we iteratively solve a relaxation of the problem in order to identify the largest set of global constraints specified by the user which could be fulfilled. Subsequently, the quality parameters of the operation invocations which lead to constraints violation are negotiated. The next section introduces negotiation concepts. Our negotiation algorithm is presented in Section 4.3.2.

4.3.1 Negotiation in the SOA

Our negotiation approach is based on the service oriented negotiation algorithms described in [19] and [15]. The negotiation process is multiparty (each provider included in a partially feasible plan is involved), multiattribute (at least two attributes, the price and a quality dimension, are negotiated), and single encounter (each Broker-provider negotiation can be considered as an independent bilateral bargaining problem). Thus, the whole negotiation process is implemented as a set of parallel bilateral bargaining sessions between the negotiation Broker and each provider.

In the negotiation process, the Broker b and each provider p have the role of *agents*. Each agent x ($x \in \{b, p\}$) delimits each quality attribute $q_{n,j,o}$ of a service invocation $ws_{j,o}$ in a range $[q^x_{n,j,o,min}, q^x_{n,j,o,MAX}]$. Let us denote with $\overline{\mathcal{N}} \subseteq [1, N]$ the set of indexes of the negotiated quality attributes. Each agent has a *utility* function $U^x_n : [q^x_{n,j,o,min}, q^x_{n,j,o,MAX}] \rightarrow [0, 1]$ that gives the evaluation the agent x assigns to a quality attribute $q_{n,j,o}$. As in the SAW technique, the relative importance that an agent assigns to each quality attribute under negotiation is modeled as a weight u^x_n, ($\sum_{n \in \mathcal{N}} u^x_n = 1$). The overall evaluation for the *contract* $\mathbf{q}_{j,o} = \{q_{n,j,o} | n \in \overline{\mathcal{N}}\}$ of a service invocation $ws_{j,o}$ is given by $U^x(\mathbf{q}_{j,o}) = \sum_{n \in \overline{\mathcal{N}}} u^x_n U^x_n(q_{n,j,o})$.

The negotiation process between two agents consists of an alternate succession of offers and counteroffers $\mathbf{q}_{j,o}(t)$. This continues until an offer is accepted by the other side or one of the agents terminates (e.g., because a time deadline expires). An agent x accepts an offer $\mathbf{q}_{j,o}(t)$ if the value $U^x(\mathbf{q}_{j,o}(t))$ is greater than the value of the counteroffer $U^x(\mathbf{q}_{j,o}(t+1))$ the agent is ready to send in the next iteration. In order to prepare a counter offer, an agent uses a set of tactics to generate new values for each negotiated quality attribute. We have implemented time dependent tactics and the offer for the quality dimension $q_{n,j,o}(r)$ at the iteration r is evaluated as

$$q_{n,j,o}(r) = q^x_{n,j,o,min} + \alpha^x_n(r) \cdot (q^x_{n,j,o,MAX} - q^x_{n,j,o,min}). \quad (21)$$

Here and in the following, we limit the presentation to negative quality criteria since, in [3], we have shown that every positive quality metric can be replaced by a corresponding negative one and vice versa. The function $0 \leq \alpha^x_n(r) \leq 1$ depends on the iteration r and is such that the value $\alpha^x_n(0)$ equals the initial bid $q^x_{n,j,o}(0)$ of the agent x and $\alpha^x_n(r^x_{max}) = 1$, where r^x_{max} is the deadline for the agent, i.e., the maximum number of iterations in the negotiation. We adopt polynomial time functions since they concede faster in the beginning of the negotiation and this increases the probability of success of the negotiation process, i.e., we set

$$\alpha^x_n(r) = \left(\frac{\min(r, r^x_{max})}{r^x_{max}}\right)^{\frac{1}{\beta^x_n}}$$

and $\beta^x_n \in \mathcal{R}$. The β^x_n parameter determines the concavity of the $\alpha^x_n(r)$ function and the behavior of the tactic. The tactic is conceder if $\beta^x_n > 1$ and is boulware if $\beta^x_n < 1$, while $\alpha^x_n(r)$ is linear for $\beta^x_n = 1$ (see [19] for more details). For each quality attribute, we assume linear utility functions, i.e., $U^x_n(q_n) = \frac{q^x_{n,j,o,MAX} - q^x_n}{q^x_{n,j,o,MAX} - q^x_{n,j,o,min}}$. In the MAIS framework, the negotiation is automatic and is executed by the Broker according to service providers negotiation parameters that are stored in the MAIS registry when concrete services are deployed. Negotiation parameters are specified by an extension of WS-Policy (see [15] for further details). The negotiation process complexity is proportional to the number of negotiated quality attributes and to the agents deadlines, i.e., the negotiation complexity is $O(|\overline{\mathcal{N}}| \cdot r^x_{max})$.

4.3.2 Identifying a Feasible Solution

If a feasible solution for an optimization/reoptimization problem does not exist, the procedure illustrated in Algorithm 1 is executed.

Algorithm 1: Negotiation Procedure.

1 $iteration \leftarrow 0$
2 $STOP \leftarrow false$;

```
3  while iteration < iteration_MAX and not(STOP) do
4      Identify an execution plan EPL which satisfies the
       maximum number of constraints;
5      Start a negotiation process with every SP providing
       services leading to global constraints violation;
6      if negotiation is successful with at least one provider
       then
7          if P1 is feasible then
8              STOP ← true
9          end
10         iteration ← iteration + 1;
11     else
12         STOP ← true;
13     end
14 end
15 if P1 is feasible then
16     return SUCCESS;
17 else
18     return Concretization Process Failure;
19 end
```

First, an execution plan EPL that satisfies the maximum number of constraints is identified (Step 4). In Step 5, a negotiation process is performed with every SP providing services that contribute to the violation of at least one global constraint. If the negotiation is successful with at least one SP (Step 6), then the procedure is repeated until a feasible solution is found or a maximum number of iterations is reached. Otherwise (Step 18), the overall concretization process fails, the composed service execution terminates, and an error notifying that the specified set of constraints are too restrictive and cannot be fulfilled is sent to the user.

Identifying the maximum number of constraints that can be fulfilled in the WSC is a NP-hard problem, even if the linear programming relaxation of the WSC is considered (the problem in the Operation Research literature is known as the max feasible linear subsystem [2]). Anyway, we can expect that, in a real environment, the number of global constraints is limited and the maximum number of constraints that can be fulfilled can be determined by an exhaustive search. In the following, we focus only on global constraints, but the approach can be easily extended to include local ones. The identification of a feasible solution can introduce a significant overhead that depends on the number of violated constraints, on the parameter $iteration_{MAX}$, and on negotiation deadlines r^x_{max}. Algorithm 1 overhead ($NegTime$) can be determined experimentally and, in order to prevent execution time global constraints violations, the execution time global constraint value $E'' = E' - NegTime$ is adopted in (15).

Let us denote with $\overline{q}_n$ the global constraint value specified for the nth quality dimension, with $\mathcal{N}'$ the set of quality dimensions included as global constraints by the user, let $\mathcal{N}_1 \subseteq \mathcal{N}'$ be the set of nonnegotiable quality dimensions, and let us assume that a budget constraint is specified in $\mathcal{N}'$ (otherwise, the negotiation process and the problem solution are trivial). First, the following relaxation of P1, which includes only constraints for nonnegotiable qualities, is considered:

TABLE 2
Negotiation Ranges

	Broker	Provider
Price Range	$[p_{j,o}, p_{j,o} + \sum_{n\in\mathcal{N}'\setminus\mathcal{N}_{feas}} \frac{1}{\lvert\mathcal{SP}_n\rvert} EB_n]$	$[p_{j,o}, p_{j,o}(\mathbf{q}_{j,o,MAX})]$
n-th QoS dim. of invoc. ws$_{j,o}$	$[q_{n,j,o}, q_{n,j,o} - \frac{q_{n,j,o}}{q_n(\mathcal{Y}_{feas})}\overline{q}_n]$	$[q_{n,j,o}, q^p_{n,j,o,MAX}]$

P2.

$$\max \sum_{k=1}^{K} freq_k \cdot score_k(\mathcal{Y})$$
$$q_n(\mathcal{Y}) \leq \overline{q}_n \quad \forall n \in \mathcal{N}_1.$$

If P2 is infeasible, then negotiation cannot be effective to find a solution and an error message is sent to the user. Vice versa, if a feasible solution of P2 exists, then all of the possible combinations of constraints $\mathcal{N}'$ are considered incrementally in order to identify the largest set of constraints $\mathcal{N}_{feas}$ such that $\mathcal{N}_{feas}$ includes the price constraint and can be guaranteed. Let us indicate with $\mathcal{Y}_{feas}$ the optimum solution of P1 limited to the set of constraints $\mathcal{N}_{feas}$ and with $EB = B - price(\mathcal{Y}_{feas}) = B - \sum_k freq_k \cdot price_k(\mathcal{Y}_{feas})$ the extra-budget. In the negotiation process,

1. the extra-budget EB is split among the violated constraints $\mathcal{N}' \setminus \mathcal{N}_{feas}$ according to their percentage of violation $PV_n = \frac{q_n(\mathcal{Y}_{feas}) - \overline{q}_n}{\overline{q}_n}$,
2. the extra-budget
$$EB_n = \frac{PV_n}{\sum_{n\in\mathcal{N}'\setminus\mathcal{N}_{feas}} PV_n}$$
assigned to a constraint $n \in \mathcal{N}' \setminus \mathcal{N}_{feas}$ in Step 1 will be evenly shared among the set of service providers $\mathcal{SP}_n$ involved in the violation, and
3. the Broker will ask the service providers for an improvement proportional to the QoS percentage violation.

Table 2 reports the negotiation ranges adopted by the Broker and the service providers. For service providers, the upper bound of each quality dimension range is related to the providers' ability to improve the negotiated quality dimension, while $q_{n,j,o}$ indicates the quality values advertised for services selected in $\mathcal{Y}_{feas}$, i.e., the services which minimize the number of constraints violations. As in other approaches proposed in the literature [3], [36], we assume that the price of a service invocation depends linearly on execution time and reputation, quadratically on data quality and exponentially on availability. Hence, in the negotiation process, the price of offers is evaluated as

$$p_{j,o} = \gamma_i e_{j,o} r_{j,o} d^2_{j,o} e^{\delta_i a_{j,o}}, \tag{22}$$

where γ_i and δ_i are constants that depend on the particular abstract service operation (i.e., the higher is the complexity of the operation that a task implements and the higher is the price).

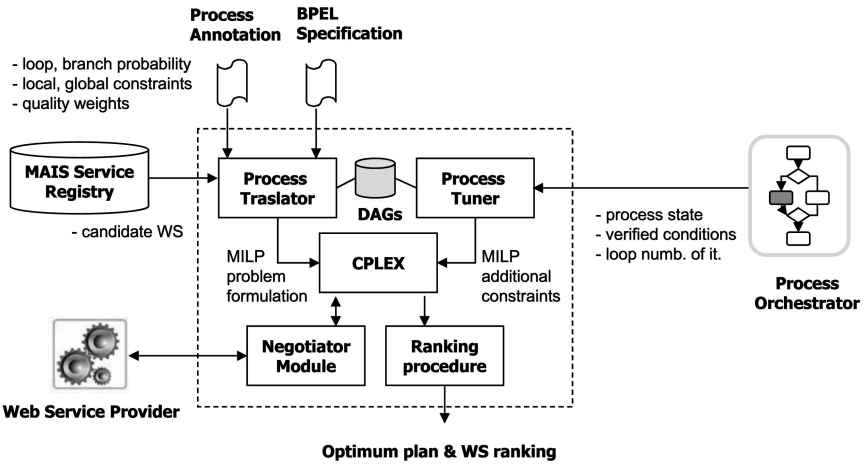


Fig. 4. Concretizator module architecture.

4.4 Runtime Reoptimization

At runtime, the optimum plan identified by initially solving the WSC problem can be updated in order to take into account the variability of Web services and of user behavior. A reoptimization step should be performed if a service invocation fails but, from a theoretical point of view, could be performed after the execution of each task since new Web services with better characteristics could become available. On the other hand, reoptimization introduces a system overhead, since it is time consuming and the MAIS service registry has to be accessed in order to retrieve the set of candidate Web services and their corresponding quality values.

As in other approaches [7], [24], the basic idea is to monitor the QoS of service invocations and reestimate the quality values expected for the composed service. Whenever the new estimate indicates a large deviation from the initial value obtained by EPL^*, services that remain to be executed must be replanned in order to avoid SLA violations.

Reoptimization is triggered in the following cases:

- The current QoS value q_n differs from the corresponding prediction $\tilde{q}_n$ by more than a given threshold Δ_n, i.e., $|q_n - \tilde{q}_n| > \Delta_n$. Note that, if the quality attribute is positive (negative), then, if $q_n \gg \tilde{q}_n$, the execution plan could be suboptimal (could lead to a global constraint violation) while, vice versa, if $q_n \ll \tilde{q}_n$, the execution plan could lead to a global constraint violation (could be suboptimal). In this way, the reoptimization faces the problem of variability of Web services performance.
- If a Web service invocation fails, a substitute service is invoked as discussed in Section 4.2. The invocation of a substitute service is suboptimal with respect to the previous execution plan. The reoptimization is triggered if $|q_n - \tilde{q}_n| > \Delta_n$.
- Different sets of weights $\{w_n\}$ can be specified for a user in different contexts (see Fig. 2a). Reoptimization is triggered by a user's context switch.
- Optimization is performed statistically, i.e., by evaluating branch conditions and loops number of iteration probability distribution. Reoptimization is triggered after the evaluation of branch conditions (excluding branches introduced by loops peeling) since the knowledge of the branch to be executed could lead to a better execution plan. Reoptimization should be performed also when a loop execution ends. If the number of iterations is overestimated, then the execution plan is suboptimal. Vice versa, if the maximum number of iterations is underestimated, global constraints could be violated.
- Reoptimization is also periodically performed with a time period T_p that varies in a time window $[T_{min}, T_{max}]$. In this way, the execution plan can be updated in highly variable execution contexts. In order to adapt the time period to environment changes, T_p is updated as follows, similarly to the strategy adopted in the TCP protocol [14]. If, after a periodic reoptimization, the new execution plan is equal to the previous one, we argue the environment has not changed significantly in the last period, and T_p is increased and is set to $\min\{2 \cdot T_p, T_{max}\}$. Vice versa, if the two execution plans are different, T_p is reduced and is set to $\max\{T_p/2, T_{min}\}$ in order to detect environment changes.

In the reoptimization problem, the quality parameters for tasks already executed are set according to the values monitored for Web service operation invocations provided by the MAIS reflective architecture.

4.5 Concretizator Module Implementation

The adaptive concretizator module has been implemented in Java and is deployed in the MAIS framework as a Web service (Fig. 4). The input are the BPEL abstract specification, constraints, and annotations. The optimization is performed by the following steps:

1. When a composed service is deployed in the MAIS framework, it is translated into a DAG internal representation. Loops are peeled, execution paths are extracted from the DAG, and a depth-first algorithm identifies the set of subpaths of every execution path.
2. The set of candidate Web services (with their quality values) is retrieved from the MAIS service registry.

The MILP model formulation is computed, and the optimization problem is solved by running *CPLEX*, a state of the art integer linear programming solver based on the branch and cut technique [33]. Given enough time (see the discussion in Section 5), the solver identifies the optimum solution of the WSC problem, i.e., for a given composed Web Service, the optimum execution plan is identified.

3. Finally, the *Ranking procedure* evaluates the Web services ranking as discussed in Section 4.2.

The number of execution paths arising from the BPEL specification depends on the process structure and is given by the product of branch conditions and loops number of iterations, i.e., $K = \prod_s NB^s \prod_l NI^l$. The depth-first algorithm is executed for every execution path and has a complexity proportional to the number of nodes and edges of the DAG representation of the execution path which are $O(I)$. Hence, the overall complexity of Step 1 is $O(I \cdot \prod_s NB^s \cdot \prod_l NI^l)$.

The complexity of Step 2 is given by the solution of the MILP model, which, in the worst case, is exponential in the number of binary decision variables which are

$$O(I \cdot \max_i |\mathcal{WS}_i| \cdot \max_j |\mathcal{OP}_j|).$$

Anyway, as will be discussed in the next section, *CPLEX* is very efficient in the generation of cutting planes [33] and the problem solution can be computed quickly for realistic problems of reasonable size. Finally the complexity of the *Ranking procedure* is $O(I \cdot \max_i |\mathcal{WS}_i| \cdot \max_j |\mathcal{OP}_j|)$.

Reoptimization is implemented by the *Process Tuner* module, which obtains as input the state of the running process and the composed Web service execution trace by the *Process Orchestrator*. The Process Tuner module modifies the internal DAG representation associated with the composed process instance and generates additional constraints, which assign concrete Web services to tasks already executed; the solution of the reoptimization problem is obtained by invoking *CPLEX*. In the worst case, optimization and reoptimization have the same complexity. In general, the reoptimization process is faster than the optimization, since the DAG is simplified by using the result computed in Step 1, additional constraints can be added, and the number of binary decision variables can be reduced [3], [35], [7]. Finally, the complexity of Algorithm 1 is $O(iteration_{MAX} \cdot 2^{|\mathcal{N}'|})$.

5 Experimental Results

Our WSC model and algorithms have been tested on a wide set of randomly generated processes instances. Experiments have been performed to compare the solutions obtained by applying the peeling technique with respect to loop unfolding and to evaluate the effectiveness of the negotiation approach.

Quality of services values of candidate Web services have been randomly generated according to the values reported in the literature. Maximum and minimum availability values were randomly generated assuming a uniform distribution in the interval 0.95 and 0.99999. Reputation was determined in the same way, considering the range [0.8, 0.99]. As in [11], we assume that the execution time and data quality have a Gaussian distribution and the min max interval (stored in the MAIS registry) includes the value of the quality dimension with probability 0.999 (i.e., the min max values equal $\mu \pm 3\sigma$, where μ and σ indicate the mean and the standard deviation of the quality dimension). The price of each service invocation was determined according to the QoS level experienced by the user by applying (22), where the constant factors γ_i and δ_i were randomly generated assuming a uniform distribution in the range [0.1, 1]. Finally, the set of weights w_i was randomly generated and weights were adjusted to sum 1.

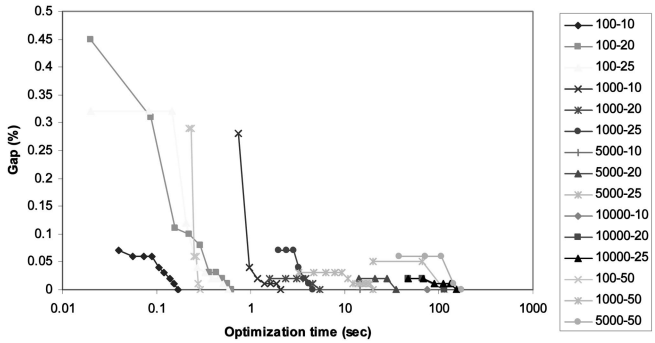


Fig. 5. Gap to the optimum solution versus optimization time.

Analyses have been performed on a 2 GHz Intel Pentium-D Workstation with 2 GB of RAM. Problems with up to 100 execution paths, 10,000 tasks, 50 candidate Web services operations per task, and 5 global constraints have been considered. For problems of maximum size, the execution time required to deploy a composed service and evaluate its DAG representation was about one minute, while CPLEX execution time was about three minutes. Even if the branch and cut technique implemented by CPLEX has a worst case exponential time complexity, CPLEX is very efficient in determining the problem optimum solution. Fig. 5 reports the gap between the best solution found by CPLEX within a limited time interval and the final global optimum. The gap is less than 1 percent in all cases; a feasible solution is found in less than a second for small/medium size problem instances (i.e., up to 1,000 tasks), while it is about one minute for large processes with 5,000-10,000 tasks. Emmerich et al. [18] reports that 10,000 tasks is the maximum number of tasks that could be orchestrated by current BPEL engine implementations in modern grid environments, so our approach is valid for the maximum process sizes that are currently considered in research. Finally, in the worst case, the execution time required by Algorithm 1 to obtain a feasible solution using negotiation was about one minute.

Every experiment has been conducted by considering the execution plan determined initially when the composed service execution starts (static global plan) and the variable QoS values obtained at runtime. Runtime QoS values have been evaluated through simulation. As in [35], the same test case was analyzed by varying the variance of concrete services execution time. For every test case, the standard deviation of concrete services execution time has been varied between 0.1μ and 0.3μ with step 0.1μ. For a fixed value of the standard deviation, the comparison is performed by running 10 simulations for every test case.

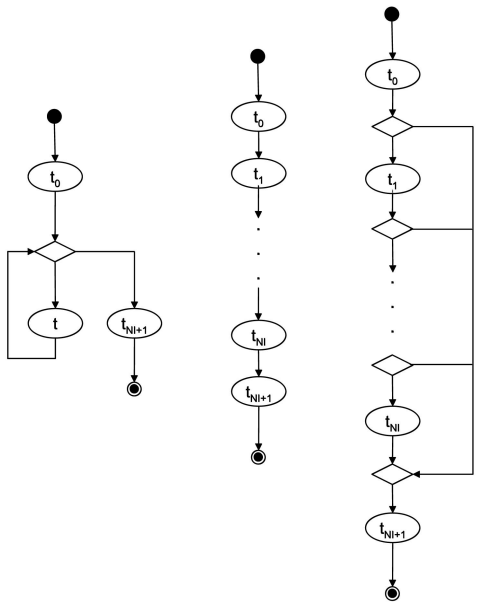


Fig. 6. Loop test case.

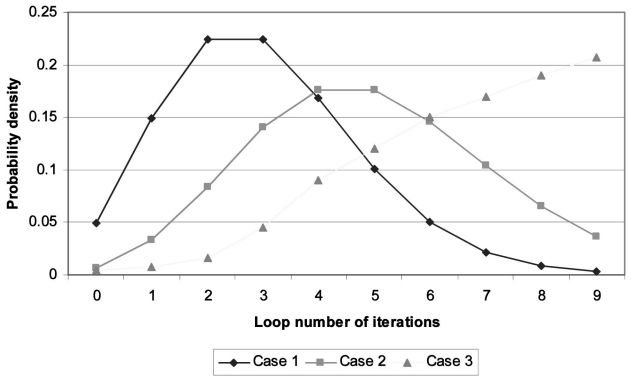


Fig. 7. Loop probability distribution

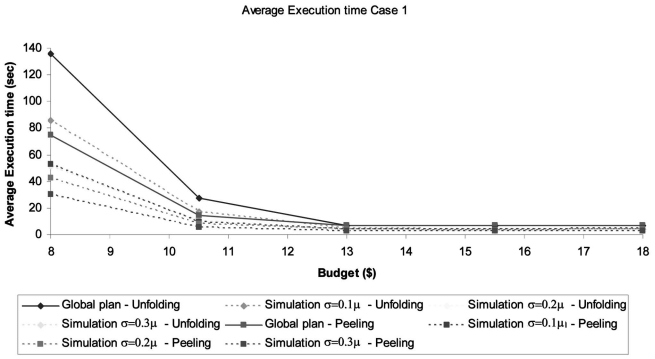


Fig. 8. Execution time versus budget dependency.

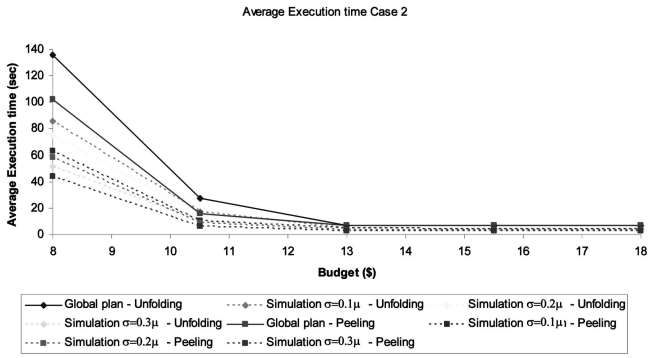


Fig. 9. Execution time versus budget dependency.

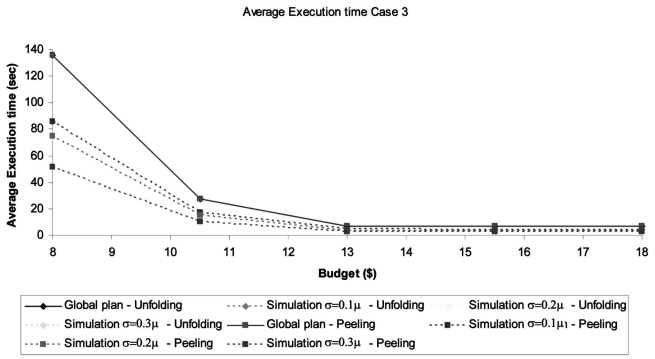


Fig. 10. Execution time versus budget dependency.

In order to evaluate the effectiveness of loop peeling, we have considered a process which invokes a task t_0, then invokes a task t several times in a loop, and, finally, invokes a task t_{NI+1} (see Fig. 6). The number of candidate Web services for the task t has been varied between 10 and 30 with step 10, while the number of candidate Web services for task t_0 and t_{NI+1} has been varied between 100 and 300 with Step 100. The high number of candidate Web services of the two tasks external to the loop allow analyzing loop peeling advantage with respect to loop unfolding in a general way. Indeed, a set of Web services in a composed process specification can be represented by a single task applying the reduction formula proposed in [8], [7]. Task t_0 and t_{NI+1} represent the "average" behavior of a complex service before and after the loop execution.

We have considered several discrete probability distributions for the loop number of iterations (uniform, geometric, and Poisson). The global constraints of the process have been determined by considering the quality value obtained by the local approach algorithm proposed in [35] and are reduced progressively (negative quality criteria) until no feasible solution exists.

Results vary depending on the global constraints value. When the global constraints are not stringent, loop peeling and loop unfolding give the same results. Vice versa, when the global constraints are more severe (the process has to be executed with limited resources which set the problem close to unfeasibility conditions), then the loop peeling gives better results. In order to show the peeling behavior, we report a representative test case result, obtained by considering three different probability distributions for the loop number of iterations (see Fig. 7). The problem of minimization of the execution time with a budget global constraint has been considered. The budget was initially set equal to \$18 and was further reduced with step \$2.5 down to \$8. With a budget lower than \$8, the problem becomes unfeasible. The plots in Figs. 8, 9, and 10 to report the average composed process execution time obtained by applying the unfolding and peeling techniques as a function of the budget constraint. When the budget constraint is loose or the probability to run the maximum number of iterations is high (*Case 3* distribution; see Fig. 7), the unfolding and peeling approaches determine the same solution (the plots in Fig. 10 do overlap). Vice versa, when

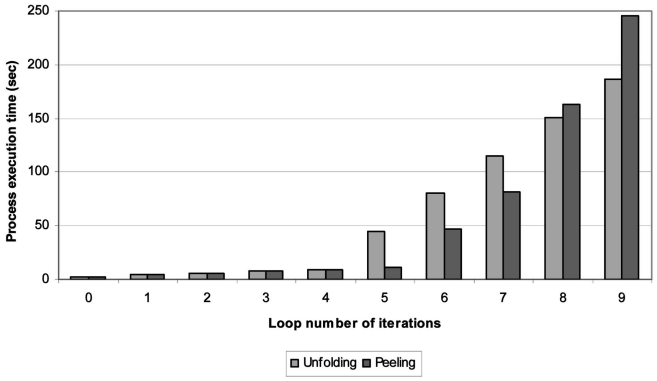


Fig. 11. Process execution time.

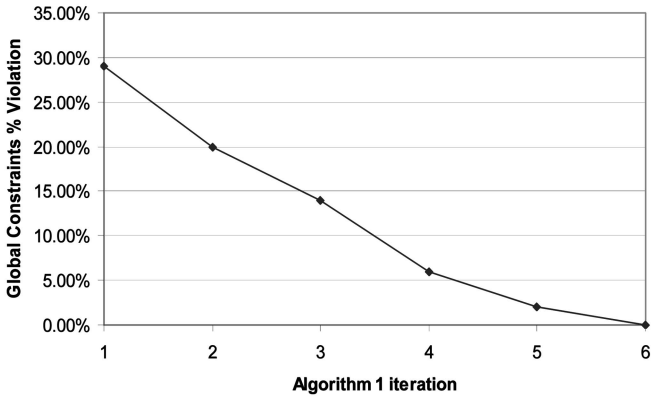


Fig. 12. Constraints violation in Algorithm 1.

the probability to run the maximum number of iterations is low (first and second distributions in Fig. 7), and the budget constraint is reduced, the peeling approach improves the average execution time up to 45 percent. This could be explained considering that, when the budget constraint is reduced (see Fig. 11), then the peeling approach selects Web service invocations which give better performance and have a high price in the first iterations of the loop. In order to fulfill the budget constraint, in the last iterations, Web services with high execution time and low price are selected instead. In the first part of the loop, the peeling approach is "aggressive" and tries to optimize the objective function while always guaranteeing the global constraints.

The reoptimization, hence, is triggered only if the loop's current number of iteration is greater than its expected value. Analyses have shown that this behavior is quite independent of loops' probability distributions. Hence, loop peeling allows reducing the reoptimization effort with respect to loop unfolding. We obtained almost the same results considering the static execution plan determined when the composed process execution starts and the average value of QoS obtained by running the composed service throught simulation. Peeling improvement decreases as the standard deviation of Web services execution time increases. Note that, in that case, the process reoptimization is performed also more frequently.

The higher is the gap between the maximum and the average loop number of iterations, and the more severe are the global constraints, the better are the results which can be obtained by the peeling approach with respect to loop unfolding.

In order to verify negotiation effectiveness, we have considered linear tactics both for the Broker and service providers ($\beta_n^x = 1$) and we set $r_{max}^x = 10$. Results depend on the value of global constraints and extra budget; anyway, if the Broker and service provider price and quality intervals do overlap, then the Broker-provider negotiation converges, on average, in five steps and Algorithm 1 identifies a feasible solution for the problem in five or six steps.

As an example, the plot reported in Fig. 12 shows the trend of the percentage constraint violation PV_1 and the negotiation procedure number of steps as a function of Algorithm 1 number of iterations for a representative example where four providers are involved in a global constraint violation. The negotiation is effective and the percentage constraint violations decreases at each iteration. A feasible solution of the problem is finally identified.

6 Related Work

Recently, dynamic Web service composition has attracted great interest in the research community. Literature approaches can be classified into two main categories: composition by planning and business process optimization [31]. The former approach, proposed by the Semantic Web and AI communities, investigates the problem of synthesizing a complex behavior from an explicit goal and a set of candidate services which contribute to a partial solution of the complex problem. In the latter case [28], [35], complex applications are specified as BPEL processes and the best set of services is dynamically selected at runtime by solving an optimization problem.

The Semantic Web and AI approach is very flexible since a composed service process is built automatically or semiautomatically from a high level specification of the required functionality. The work in [23], [24] proposes a framework which interleaves planning and execution of complex applications whose functional goal and QoS requirements are specified by assertions through XSRL and XSAL languages. The planning is performed by model checking as in [30]. In a similar way, in [16], a complex application is built from a high level workflow specification which is synthesized by applying contingency plans. Planning is very flexible but usually is computation intensive and, from the QoS point of view, only suboptimal solutions can be identified [24]. The work in [1] proposes a trade-off between planning and optimization approaches. In a first semiautomatic *logical composition* step, the goal is translated into a workflow-based specification that introduces abstract tasks. A second *physical composition* step maps abstract tasks to concrete services and is supervisioned by the composed service designer.

Business process optimization approaches allow the specification of complex applications as BPEL processes composed by *abstract services* which act as *place holders* of Web service components invoked at runtime. In that case, the best set of services, selected by solving an optimization problem, is invoked at runtime by implementing a *dynamic/late binding* mechanism. *Two generations of solutions* have been proposed in the literature (see Section 1). First generation solutions consider only *local constraints*; the

service composition is very simple and can be performed at runtime by a greedy approach which selects the best candidate service suitable for the execution. The work presented in [25] introduces an agent-based framework where agents can migrate to invoke Web services locally. Anyway, network traffic and execution time are the only quality dimensions considered and constraints can be specified only locally.

Second generation solutions support *global constraints*. In [5] the complexity of some variants of the Web service composition problem is analyzed, while an overview of heuristic techniques, which hence identify only suboptimal solutions, can be found in [21]. In [34], the Web service composition problem is modeled as a multiple choice, multiple dimension knapsack problem and as a graph constrained optimum path problem. Ad hoc efficient techniques are proposed to identify suboptimal solutions of the WSC problem, but composed processes which include only a single execution path are considered.

Our approach starts from the work presented by Zeng et al. [35]. The authors separately optimize each execution path and obtain the execution plan by composing separate solutions according to the frequency of execution. Their approach has several limitations. First, in the optimization of a single execution path, the fulfillment of availability and response time constraints is guaranteed only for the critical path (i.e., the path which corresponds to the highest execution time). Furthermore, the execution plan is obtained as the merge of separate execution plans. If a task belongs to multiple execution paths, then the task is executed by the service identified for the most frequently executed execution path. In a previous work [3], we have shown that Zeng et al.'s work cannot always guarantee the fulfillment of global constraints since the WSC problem is not separable. In this paper, we extend our previous work by introducing loop peeling, negotiation, and Web services dependencies constraints allowing the execution of stateful Web services.

Some recent proposals face the WSC problem by implementing genetic algorithms [7], [13]. In Canfora et al.'s work [7], the reduction formulas presented in [8] are adopted and the reoptimization is considered, but only sub-optimal solutions are identified since tasks specified in cycles are always assigned to the same Web service. Furthermore, by applying reduction formulas, the execution plan guarantees global constraints only statistically. At runtime, if low probability paths are taken (see [7]), then the execution plan could become infeasible and the reoptimization must be triggered. In [13], the multiobjective evolutionary approach NSGA-II (Nondominated Sorting Genetic Algorithm; see [17]) is implemented, which identifies a set of Pareto optimal solutions without introducing a ranking among different quality dimensions. Every identified solution is characterized by the fact that no other plans exist such that a quality dimension is improved without worsening the other ones. Genetic algorithms are more flexible than our mixed integer linear approach since they allow considering also nonlinear composition rules for composed Web services but are less computationally efficient. In current implementations [7], [13], some execution time is wasted by also generating nonfeasible solutions and sometimes, no solution can be identified even when the problem is feasible, in cases in which the global constraints are stringent.

The work in [9] proposes the application of process mining techniques to solve the WSC problem. The approach does not require the definition of several QoS parameters and can consider time of the day or day of the year dependency of the QoS values. Anyway, QoS constraints can be guaranteed only statistically.

An overview of negotiation techniques applied to service oriented environments is presented in [19], where average system utility and probability of negotiation convergence are analyzed for different negotiation parameters and tactics. In [32], a negotiation framework based on an extension of WS-Policy and assertion verification is presented. The goal of our negotiation approach is bargaining QoS parameters in order to identify a feasible solution of the optimization/reoptimization problem and reduce the number of processes invocation failures.

The problem of execution of large BPEL processes has been studied in the area of workflows and grid for e-science [18]. While BPEL finds a wide adoption in this area in the last couple of years, it still faces scalability, mainly due to hardware limitations. The experimentation discussed by [18] with a workflow containing 84,000 tasks could not be executed with a single workflow. Possible solutions are creating a hierarchy of processes or distributing the workflow over a number of engines. For e-science workflows, either local optimization or second generation linear programming techniques based on lp-solve have been proposed (e.g., the Vienna Grid [6]), while additional research work is needed for cyclic processes [20], since in the literature, at most cycle unfolding is considered and the variability in loops number of iteration is not analyzed. Moreover, negotiation is either not performed or performed for each of the available tasks. In our paper, we propose an optimization model that is applicable to the size of the abovementioned processes to obtain an optimal solution that significantly improves previous approaches, as discussed in Section 5, and strategies for negotiating QoS only in critical cases where no solution can be found, thus significantly limiting negotiation overhead.

7 Conclusions

In this paper, we have presented an optimization approach for the composition of Web services using dynamic service selection which allows specifying constraints on quality requirements for the user both at local and global level, and to fulfill constraints at runtime through adaptive reoptimization under variable QoS characteristics of Web services. Peeling techniques have been implemented for the optimization of loop iterations and negotiation techniques are exploited in order to identify a feasible solution of the problem. With respect to other literature approaches, we guarantee the fulfillment of global constraints under more stringent conditions [35] and we identify the global optimal solution instead of local optima or suboptima [7]. Future work will consider the optimization of execution of multiple process instances. This is a very critical issue since, in the current implementation, if a very large number of requestors are assigned to the same "best" service, critical load conditions could be reached and the quality of service degrades. Nonlinear aggregation function for quality

TABLE 3
Notation Summary

Symbol	Description
i	Task index
t_i	i-th task
j	Web service index
ws_j	j-th Web service
s	Switch
p_h^s	Probability of execution of the h-branch condition of switch s
NB^s	Number of disjoint branch conditions of switch s
l	Loop
p_h^l	Probability of execution of the h-iteration of loop l
NI^l	Expected maximum number of iteration of loop l
N	Number of quality dimension of interest
n	Quality dimension index
q_n	n-th quality dimension
w_n	Weight associated to the n-th quality dimension by the end user
$\mathcal{WS}_i$	Set of indexes of Web services candidate for the execution of task t_i
$\mathcal{OP}_j$	Set of indexes of operations implemented by the WS j
$ws_{j,o}$	Invocation of operation $o \in \mathcal{OP}_j$ of Web service j
I	Number of tasks of the composite service specification
J	Number of candidate Web services
$e_{j,o}$	$ws_{j,o}$ execution time
$a_{j,o}$	$ws_{j,o}$ availability
$p_{j,o}$	Price for $ws_{j,o}$ execution
$r_{j,o}$	$ws_{j,o}$ reputation
$d_{j,o}$	$ws_{j,o}$ data quality
k	Execution path index
ep_k	k-th execution path
$\mathcal{A}_k$	Set of indexes of tasks included in the execution path ep_k
$freq_k$	Frequency of execution for the execution path ep_l
m	Sub path index
sp_m^k	m-th sub path of the execution path ep_k
EPL	Execution plan
q_n^k	Value of the n-th quality dimension evaluated along execution path ep_k
v_n^k	Normalized value of the n-th quality dimension evaluated along execution path ep_k
K	Number of execution path arising from the composite service specification
$\mathcal{B}$	Set of indexes of tasks included in loops of the composite service specification
E	Execution time global constraint for the composed service execution
A	Availability global constraint for the composed service execution
B	Budget global constraint for the composed service execution
R	Reputation global constraint for the composed service execution
DQ	Data quality global constraint for the composed service execution
$q_{n,j,o}$	n-th quality dimension value for service invocation $ws_{j,o}$
$[q^x_{n,j,o,min}, q^x_{n,j,o,MAX}]$	Interval for the quality value q_n of the service invocation $ws_{j,o}$ for agent x
U_n^x	Agent x utility function for quality attribute q_n
u_n^x	Agent x weight for quality attribute q_n
r	Negotiation iteration step
$\alpha_n^x(r)$	Agent x conceding function for quality attribute q_n
β_n^x	Agent x concession parameter for quality attribute q_n

dimensions will be also considered and a hybrid local search optimization approach that interleaves the solution of linear integer programming problems with nonlinear problems will be developed. Furthermore, the effectiveness of other negotiation techniques and the boulware behavior of service providers will be analyzed. Finally, the problem of selection of Web services with variable periodical QoS profiles requires further investigation for its interest in business processes due to variable load conditions.

APPENDIX
Notation Summary

See Table 3.

ACKNOWLEDGMENTS

The work reported in this paper has been partially supported by the MIUR MAIS and Tekne FIRB Projects, and by DISCoRSO FAR Project. Thanks are expressed to Professor Marco Trubian for many fruitful discussions on optimization issues and to Dr. Gianpaolo Agosta and Dr. Marco Comuzzi for peeling and negotiation technique considerations.

REFERENCES

[1] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava, "A Service Creation Environment Based on End to End Composition of Web Services," *Proc. Int'l World Wide Web Conf. (WWW '05)*, pp. 128-137, 2005.

[2] E. Amaldi and V. Kann, "The Complexity and Approximability of Finding Maximum Feasible Subsystems of Linear Relations," *Theoretical Computer Sciences*, vol. 147, pp. 181-210, 1995.

[3] D. Ardagna and B. Pernici, "Global and Local QoS Guarantee in Web Service Selection," *Proc. Business Process Managment Workshop (BPM '05)*, pp. 32-46, 2005. Extended version in *Int'l J. Business Performance Managment*, vol. 1, no. 4, pp. 233-243, 2005.

[4] D.F. Bacon, S.L. Graham, and O.J. Sharp, "Compiler Transformations for High-Performance Computing," *ACM Computing Surveys*, vol. 26, no. 4, pp. 345-420, 1994.

[5] P.A. Bonatti and P. Festa, "On Optimal Service Selection," *Proc. Int'l World Wide Web Conf. (WWW '05)*, pp. 530-538, 2005.

[6] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt, "Support for Time-Critical Gridworkflow Applications," *Proc. Int'l Conf. e-Science and Grid Computing (e-Science '05)*, pp. 108-115, 2005.

[7] G. Canfora, M. Penta, R. Esposito, and M.L. Villani, "QoS-Aware Replanning of Composite Web Services," *Proc. Int'l Conf. Web Services (ICWS '05)*, 2005.

[8] J. Cardoso, "Quality of Service and Semantic Composition of Workflows," PhD thesis, Univ. of Georgia, 2002.

[9] F. Casati, M. Castellanos, U. Dayal, and M.C. Shan, "Probabilistic, Context-Sensitive, and Goal-Oriented Service Selection," *Proc. Int'l Conf. Service Oriented Computing (ICSOC '04)*, pp. 316-321, 2004.

[10] S. Ceri, F. Daniel, M. Matera, and F. Facca, "Model-Driven Development of Context-Aware Web Applications," *ACM Trans. Internet Technology*, vol. 7, no. 2, 2007.

[11] S. Chandrasekaran, J.A. Miller, G. Silver, I.B. Arpinar, and A.P. Sheth, "Performance Analysis and Simulation of Composite Web Services," *Electronic Market: The Int'l J. Electronic Commerce and Business Media*, vol. 13, no. 2, 2003.

[12] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, and R.P. Doyle, "Managing Energy and Server Resources in Hosting Centers," *Proc. Symp. Operating Systems and Principles (SOSP '01)*, pp. 103-116, 2001.

[13] D.B. Claro, P. Albers, and J.K. Hao, "Selecting Web Services for Optimal Composition," *Proc. Int'l Conf. Web Services (ICWS '05) Workshop Proc.*, 2005.

[14] D. Comer, *Internetworking with TCP/IP Volume 1: Principles Protocols, and Architecture*, fifth ed. Prentice Hall, 2006.

[15] M. Comuzzi and B. Pernici, "An Architecture for Flexible Web Service QoS Negotiation," *Proc. Int'l Enterprise Distributed Object Conf. (EDOC '05)*, pp. 70-82, 2005.

[16] L.A.G. Dacosta, P.F. Pires, and M. Mattoso, "Automatic Composition of Web Services with Contingency Plans," *Proc. Int'l Conf. on Web Services (ICWS '04) Workshop Proc.*, 2004.

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, 2002.

[18] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S.L. Price, "Grid Service Orchestration Using the Business Process Execution Language (BPEL)," *J. Grid Computing*, vol. 3, pp. 283-304, 2006.

[19] P. Faratin, C. Sierra, and N.R. Jennings, "Negotiation Decision Functions for Autonomous Agents," *Int'l J. Robotics and Autonomous Systems,* vol. 24, nos. 3-4, 1998.
[20] G.C. Fox and D. Gannon, "Workflow in Grid Systems," *Concurrency and Computation: Practice and Experience,* vol. 18, no. 10, pp. 1009-1019, 2006.
[21] M.C. Jaeger, G. Muhl, and S. Golze, "QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms," *Proc. Int'l Conf. Cooperative Information Systems,* 2005.
[22] C.L. Hwang and K. Yoon, *Multiple Criteria Decision Making, Lecture Notes in Economics and Mathematical Systems.* Springer-Verlag, 1981.
[23] A. Lazovik, M. Aiello, and M. Papazoglou, "Associating Assertions with Business Proesses and Monitoring Their Execution," *Proc. Int'l Conf. Service Oriented Computing (ICSOC '04),* pp. 94-104, 2004.
[24] A. Lazovik, M. Aiello, and M. Papazoglou, "Planning and Monitoring the Execution of Web Service Requests," *J. Digital Libraries,* pp. 1-31, 2005.
[25] Z. Maamar, Q.Z. Sheng, and B. Benatallah, "Interleaving Web Services Composition and Execution Using Software Agents and Delegation," *Proc. Web Services and Agent-Based Eng. (WSABE '03),* 2003.
[26] D. Menascé, "QoS Issues In Web Services," *IEEE Internet Computing,* vol. 6, no. 6, pp. 72-75, 2002.
[27] M. Ouzzani and A. Bouguettaya, *IEEE Internet Computing,* vol. 37, no. 3, pp. 34-44, 2004.
[28] A.A. Patil, S.A. Oundhakar, A.P. Sheth, and K. Verma, "METEOR-S Web Service Annotation Framework, *Proc. World Wide Web Conf. (WWW '04),* pp. 553-562, 2004.
[29] *Mobile Information Systems—Infrastructure and Design for Flexibility and Adaptivity,* B. Pernici, ed. Springer, 2006.
[30] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso, "Automated Composition of Web Services by Planning at the Knowledge Level," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI '05),* pp. 1252-1259, 2005.
[31] B. Srivastava and J. Koehler, "Web Service Composition—Current Solutions and Open Problems, *Proc. Int'l Conf. Automated Planning and Scheduling (ICAPS '03),* 2003.
[32] E. Wohlstadter, S. Tai, T.A. Mikalsen, I. Rouvellou, and P.T. Devanbu, "QoS: Middleware to Sweeten Quality-Of-Service Policy Interactions," *Proc. Int'l Conf. Software Eng. (ICSE '04),* pp. 189-199, 2004.
[33] L. Wolsey, *Integer Programming.* John Wiley and Sons, 1998.
[34] T. Yu and K.J. Lin, "Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints," *Proc. Int'l Conf. Service-Oriented Computing (ICSOC '05),* 2005.
[35] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnamam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.,* vol. 30, no. 5, May 2004.
[36] L. Zhang and D. Ardagna, "SLA-Based Profit Optimization in Autonomic Computing Systems, *Proc. Int'l Conf. Service Oriented Computing (ICSOC '04),* pp. 173-182, 2004.

Danilo Ardagna received the PhD degree in computer engineering in 2004 from Politecnico di Milano, from which he graduated in December 2000. Now, he is an assistant professor of Information Systems at the Department of Electronics and Information. His research interests include Web services composition, autonomic computing, and computer system costs minimization.

Barbara Pernici is a full professor of computer engineering at Politecnico di Milano. Her research interests include cooperative information systems, service management, workflow management systems, information systems modeling and design. She is an editor of the *ACM Journal of Data and Information Quality*, the *Requirements Engineering Journal*, and the *International Journal of Cooperative Information Systems*. She was chief scientist of the Italian FIRB MAIS (Multichannel Adaptive Information Systems) project from 2002 to 2006 and participated in many European projects, among which were WS-Diamond, WIDE, F3, EQUATOR, and ITHACA. She has been the chair of Working Group 8.1 (Information Systems Design) of the International Federation for Information Processing (IFIP) for the period 2004-2006. She is the second vice-chair of IFIP Technical Committee 8 (TC8, Information Systems) for 2007-2009.