Performance Prediction of Web Service Workflows*

Moreno Marzolla[1] and Raffaela Mirandola[2]

[1] INFN Sezione di Padova, via Marzolo 8, 35131 Padova, Italy
moreno.marzolla@pd.infn.it
[2] Dip. di Elettronica e Informazione, Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy
mirandola@elet.polimi.it

Abstract. Web Services play an important role in the Service-oriented Architecture paradigm, as they allow services to be selected on-the-fly to build applications out of existing components. In this scenario, the Business Process Execution Language notation can be used as an *orchestration language* which allows the user to describe interactions with Web Services in a standard way. The performance of a BPEL workflow is a very important factor for deciding which components must be selected, or to choose whether a given sequence of interactions can provide the requested quality of service. Due to its very dynamic nature, workflow performance evaluation can not be accomplished using traditional, heavy-weight techniques. In this paper we present a multi-view approach for the performance prediction of service-based applications encompassing both users and service provider(s) perspectives. As a first step towards the realization of this integrated framework we present an efficient approach for performance assessment of Web Service workflows described using the BPEL notation. Starting from annotated BPEL and WSDL specifications, we derive performance bounds on response time and throughput. In such a way users are able to assess the efficiency of a BPEL workflow, while service provider(s) can perform sizing studies or estimate performance gains of alternative upgrades to existing systems. To bring this approach to fruition we developed a prototype tool called `bpel2qnbound`, using which we analyze a simple case study.

1 Introduction

The Service-oriented Architecture (SOA) paradigm foresees the creation of business applications from independently developed services. In this vision, providers offer similar competing services corresponding to a functional description of a service; these offerings can differ significantly in some Quality of Service (QoS) attributes like performance [1]. On the other side, prospective users of services

* Work partially supported by EU FP7 STREP Project "PLASTIC" (IST 026955), and by Italian MUR-FIRB project "ART DECO".

S. Overhage et al. (Eds.): QoSA 2007, LNCS 4880, pp. 127–144, 2007.

dynamically choose the best offerings for their purposes. Using the SOA paradigm to build applications, services can be dynamically selected and integrated at run-time, so enabling system properties like flexibility, adaptiveness, and reusability.

In this context, the key point is to build applications through the composition of available services. The application can be specified as a process in Business Process Execution Language (BPEL) language in which the composed Web Services (WSs) are specified at an abstract level. The interfaces of individual services are specified using Web Service Description Language (WSDL), the W3C standard to model WSs interfaces, with documented quality properties. Specifically, the agreed performance attributes and levels can be specified by means of appropriate notations that augment the service specifications [2].

Applications built on services face different challenges: on one hand they should ensure that users experience the required performance, and on the other hand they have to maximize the resource utilization, so that provider incomes are maximize. Besides, due to the high dynamism of the applications, the quality assessment should be performed both at development and at run time. In fact the quality of the application depend not only on the selected services but also on the underlying support systems and on the network resources.

All these aspects pose a mix of new and old problems whose solution give rise to a multi-view approach employing different techniques to performance analysis/prediction. Specifically we envisage a new approach called Multi-views Approach for Performance analysis of web Services (MAPS) that encompasses users and providers viewpoints. One of the goal of MAPS is to validate the provided performance of an application keeping the aspects strictly pertaining to the observed system separated from the aspects that depend on the underlying platform.

To this end we distinguish two different levels: the first one, called MAPS-U(sers), is concerned with the description of the application level behavior, described as a BPEL workflow. The second level, called MAPS-P(roviders), describes the physical resources where the provided services are deployed. Those levels are combined, and we show how to derive performance bounds based on the well-known operational laws of Queueing Network (QN) analysis [3]. The bounds can be used to analyse bottlenecks at the system specification level, without requiring the explicit derivation of the performance model. This makes the approach well suited for efficiently answering many performance-related questions without the need for providing too many details.

The advantage of the proposed approach is that performance bounds can be obtained with little computational effort, allowing the client to quickly answer a set of common performance-related questions arising during the application development cycle. If necessary, more accurate bounds can be derived by simply applying different techniques (e.g., the one described in [4]). Our approach can be applied: (1) at design time, to select services based on their expected performance, or to estimate the expected overall system performance; (2) at run time to reconfigure the system, e.g., to deal with changes of users requirements

or with modification of the underlying environment. On the other hand, a more complex, detailed and precise approach can be applied off-line at the resource level, to update the performance information published with the services.

To implement part of the proposed approach, we developed a prototype tool called `bpel2qnbound`, using which it is possible to parse of the annotated BPEL (and associated WSDL) specifications and obtain as outputs the performance bounds, as will be shown in more details in the following sections.

This paper is organized as follows. Section 2 briefly surveys related work. In Section 3 we present the proposed multi-view approach, while in Section 4 we give the details of the approach for early performance assessment of workflows described using a combination of annotated BPEL and WSDL descriptions. In Section 5 we show how the proposed approach can be applied to a case study. Finally, conclusions and future work are described in Section 6.

2 Related Work

Recently, QoS issues in WSs selection and composition have obtained great interest in the Web Service research community. Different approaches have been followed so far, spanning the use of QoS ontologies [5], the definition of ad-hoc methods in QoS-aware framework [6,7], and the application of optimization algorithms [8,9,10].

One of the first works in this area is proposed in [1] where a framework for composed services modeling and QoS evaluation is presented. A composite service is modeled as a directed weighted graph where each node corresponds to a WS and edge weights represent the transition probabilities of two subsequent tasks. The author shows how to evaluate quality of service of a composed service from basic services characteristics and graph topology.

Some recent proposals face the problem of composition of WSs by implementing genetic algorithms [10]. In Canfora et al. [10] the reduction formulas presented in [11] are adopted, and the problem is also periodically re-optimized in order to take into account WS performance variability. However, only sub-optimal solutions are identified since WSs specified inside execution loops are always assigned to the same Web service implementation.

Proposals of QoS-aware frameworks can be found in [6,12,7]. Yu and Lin [7] present a broker-based framework for the dynamic integration of Web services with end-to-end QoS constraints. The main functions of the proposed QoS broker include: service tracking, dynamic service composition model, dynamic service selection, and dynamic service adaptation. WebQ [6] is a QoS-based WS framework where the service selection is based on the parallel execution and monitoring of the candidate target services. Serhani et al. [12] propose a broker-based architecture which adopts QoS verification and certification in the service selection process. Zeng et al. [8] present a global planning approach to select an optimal execution plan by means of integer programming. Yu and Lin [13] discuss selection algorithms for multiple QoS attributes defining the problem as a multi-dimension multi-choice 0-1 knapsack problem as well as a multi-constraint

optimal path problem. Ardagna and Pernici [9] model the service composition as a mixed integer linear problem where both local and global constraints are taken into account. Their approach is formulated as an optimization problem handling the whole application instead of each execution path separately. Claro et al. [14] propose the use of multi-objective optimization techniques to find a set of optimal Pareto solutions from which a requestor can choose.

The works closest to ours concern methods to derive performance related measures of workflow processes [15,16]. Cardoso [15] proposes two different metrics to evaluate the control-flow complexity of BPEL web processes before their actual implementation. In [16] a mathematical model based on operations research techniques is proposed to estimate the influence of the execution of orchestrated processes on utilization and throughput of the system. Being based on QN analysis, our approach has the advantage that the bounding technique used in this paper is one of the many possible solution algorithms of the performance model. It if hence possible to compute more precise results by simply applying more sophisticated QN solution techniques [17].

3 Overview of the Proposed Approach

In this section we illustrate the main steps of our methodology for the performance evaluation of WS applications, while the description of the realized tool is deferred to section 5.

We generically consider WS-based applications, built up from software services glued together by means of some integration mechanism. In this context, the services provide the application-specific functionalities (and are considered as black boxes), and the glue defines the workflow that integrates these functionalities to deliver the functionalities required from the application.

We envisage a two layers approach to derive performance indices of the WS application. At the service providers level we can analyze the set of resources devoted to provide a service (including, if necessary, network resources) by means of a QN. The QN analysis results are then used as performance annotation characterizing the service. The users, at the upper level, use this information without concerning with the characteristics of the underlying platform.

In Fig. 1 we show a UML Activity Diagram with the main steps of MAPS. Boxes show who is responsible for executing each action.

Users side The Users-side of our approach starts from the application workflow specifications and derive performance bounds on the application response time and throughput, as follows:

Identify application requirements. At this step the user describes the application (s)he intends to realize and details its functional and non-functional requirements.

Discover and compose services. The user sees only the services with their performance annotations and builds its application based on the service

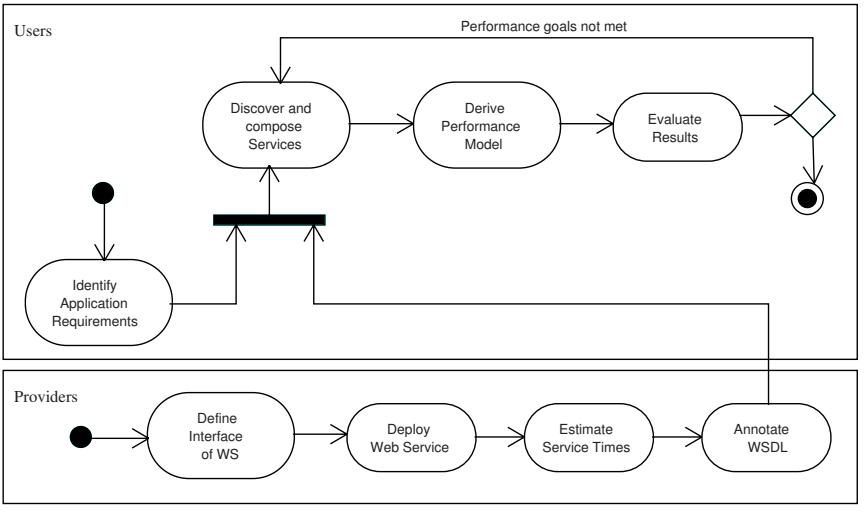


Fig. 1. Activity diagram for the proposed methodology

description using tools like BPEL and performing service discovery and composition using methods like [18]; the details of this step are beyond the scope of the paper.

Derive Performance model. The annotated BPEL specifications are used to implicitly derive a QN model. The activities in BPEL describe the sequence of requests which customers perform on service centers. The QN model is used to compute performance bounds on throughput and response time. Such information correspond to bounds on throughput and response times of the resources on the software system under evaluation. A detailed description of this step is given in Section 4.

Evaluate Results. The user exploits the computed performance bounds to choose among the services available those that better fulfill the performance requirements; performance results can also be used to answer "what-if" questions about the system. Based on the analysis results, the client can reach a more informed decision about the system design. If the performance requirements are fulfilled, (s)he can proceed with the acquisition of the preselected WSs; otherwise (s)he has to iterate the process by repeating the steps described, or lastly admit the unfeasibility of the performance requirements with the acquisition of publicly offered services.

Providers-side. At this side, the service providers must deploy the set of WS on suitable physical resources, exposing a well-defined WSDL interface annotated with QoS-oriented information. Service providers will execute the following steps:

Define interface of WS. At this step, the service providers must define the interface of the services they offer. In the WS scenario this is done by defining

an appropriate WSDL which clearly describes the interfaces, including the operations provided, the input parameters and the type of the returned results.

Deploy WS. At this step, the service providers must define the physical deployment of the services they offer. This means that the services must be implemented and installed on the appropriate hardware resources. Further resources (disk space, additional CPUs, dedicated network connections and so on) must be allocated, depending on the nature of the functionalities provided by the WS.

Estimate Service times. At this step, the service providers must estimate, for each individual operation they provide through their WSs, the average service time on each of their resource. Thus, the service providers not only have to compute the mean response time of each operation, but also break down this response time to identify the fractions of it spent on each resource. Service times can be estimated by the providers by using synthetic workloads and monitoring the response times of the physical resources. It is possible to constantly improve the estimate of the response times by continuously profiling the resource usage under real workloads: that is, providers monitor the resources utilization as client applications access them, and dynamically adjust the advertised average service times as better estimates are computed.

Annotate WSDL. The information collected during the previous step are inserted as performance annotations in the WSDL of the services. Those information, combined with the structure of the BPEL workflows which are executed on the system, are used to compute the performance bounds as will be shown in Section 4.

4 Performance Modeling of BPEL Workflows

In this section we present the main contribution of this paper. Specifically, we provide an algorithm for efficient computation of performance bounds for WSs driven by BPEL workflows. We compute optimistic and pessimistic bounds for system throughput and response time, where the "system" is the set of all physical resources (CPUs, disks, network connections) where all the WSs referenced by a BPEL are deployed.

Bounding techniques are interesting for several reasons [19]. First, they quantify the critical effect of bottleneck devices on system performance; it is also easy to analyze the performance improvements which are gained by replacing the bottlenecks. Moreover, bounds can be computed quickly and efficiently: they can be used during early planning stages to answer many performance-related questions, and eliminate inadequate design alternatives at early design phases, and at run-time assisting reconfiguration operations.

BPEL allows users to describe interactions with WSs; each interaction (request, response, one-way remote method invocation) is described by an appropriate BPEL action; moreover, elements are provided to model loops and branches.

The WS-BPEL version 2.0 [20] specifies, among others, the following types of activity:

⟨`receive`⟩ The executing process waits for a specific incoming message to be received;
⟨`reply`⟩ Sends a message in reply to a message which was received through a ⟨`receive`⟩ tag;
⟨`invoke`⟩ Invokes a one-way or request-response operation on a partner;
⟨`wait`⟩ Waits for a given time period, or until a certain time has passed;
⟨`sequence`⟩ Denotes a set of activities which should be executed sequentially; for each activity it is possible to specify additional dependencies, that is, other activities which must complete before executing the current one.
⟨`if`⟩ Selects one activity from a set of choices;
⟨`while`⟩ Repeats an activity until a certain predicate is no longer true;
⟨`repeatUntil`⟩ Repeats an activity until a condition becomes true;
⟨`forEach`⟩ This activity repeats its child activity for a number N of times; the child activity instances can be executed sequentially, or in parallel;
⟨`pick`⟩ The process blocks until a certain message is received, or a timeout goes off. When one of these events occurs, the associated activity is executed and the pick completes;
⟨`flow`⟩ Denotes a set of concurrent activities;
⟨`switch`⟩ Allows the process to choose exactly one branch of an activity with multiple choices.

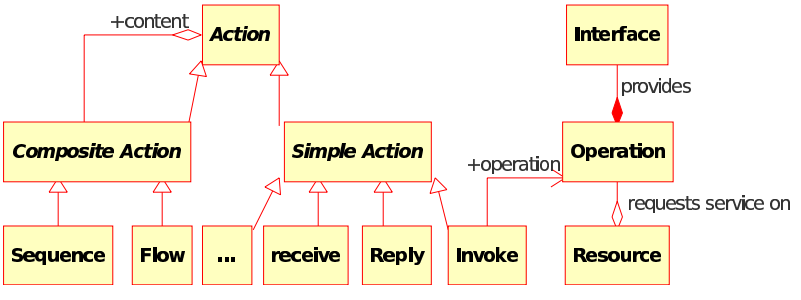


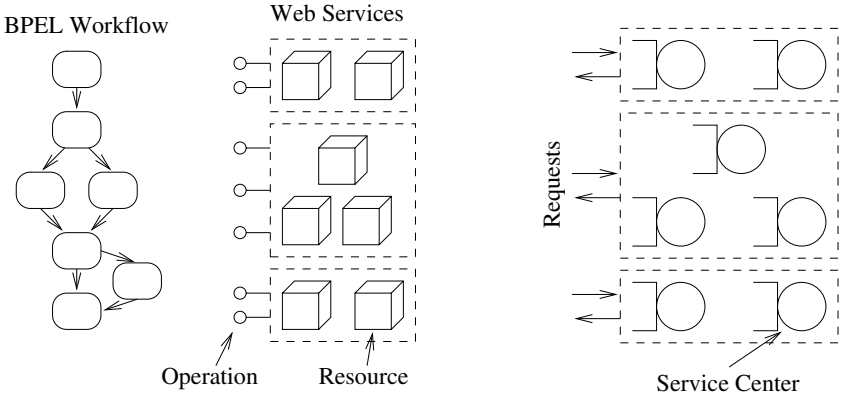
Fig. 2. Class diagram of a portion of the BPEL performance metamodel

We illustrate in Fig. 2 a portion of the BPEL metamodel as a UML class diagram. There are two kinds of BPEL actions: *composite* actions (such as `Sequence` and `Flow`), which act as containers, and *simple* actions (such as `Invoke` and `Reply`) which represent atomic actions. In particular, the `Invoke` action is used to perform a two-way (request-response) WS operation, which is described in an appropriate WSDL. Each WS operation requests service on a set of resources. For example, a WS operation may require CPU time, disk I/O operations, or in general use other (physical) resources on the executing host.

From a performance point of view, a BPEL workflow applies a *workload* on the resources. The workload may be *open* (if there is an infinite stream of instances of the BPEL which are executed at a given rate λ), or *closed* (if there is a finite

population of N BPEL instances, each spending Z time units outside the system before being executed again).

We see in Fig. 3 the mapping between the BPEL model and the QN performance model. Resources in the BPEL model correspond to service centers, and BPEL actions represent requests which arrive to the service centers.



(a) BPEL model (b) QN model

Fig. 3. BPEL model and QN performance model

We consider the system made of all resources where WSs are deployed. Our approach computes bounds on the system response time and throughput, from which bounds on individual resources utilization, throughput and response times can be obtained. We now briefly recall the main results related to QN bound analysis, which will be used later in this section; more details on bound analysis can be found in [3,19].

Let us consider a BPEL specification $\mathbf{A} = \{A_1, A_2, \ldots A_N\}$, with actions $A_1, A_2, \ldots A_N$. Let X be the system throughput (i.e., the rate of completion of BPEL $\mathbf{A}$). Let $\mathbf{R} = \{R_1, R_2, \ldots R_K\}$ be the set of all the K resources available in the system, that is, all the resources used by WS operations. Then, according to the *utilization law*, the utilization of resource $R_i \in \mathbf{R}$ can be expressed as:

$$U[R_i] = X[R_i] S_i = X D[R_i] \tag{1}$$

where $U[R_i]$ is the utilization of device R_i, X is the whole system throughput, $D[R_i]$, $X[R_i]$ and S_i are resource R_i service demand, throughput and mean service time, respectively. The utilization law states that the utilization of resource R_i is proportional to its service demand. Thus, the device R_{max} with the highest resource demand (and hence utilization) is the *bottleneck device*. Note that bottleneck identification should be one of the first steps in any performance study; any system upgrade which does not remove the bottleneck(s) will have no impact on the system performance at high loads.

The first step in the bound derivation process is to compute the service demand $D[R]$ for each resource $R \in \mathbf{R}$. In this paper we analyze the scenario in which a single kind of BPEL workflow is executed in the system; from the QN model perspective, this results in considering a QN performance model with a single customer class. Extension of the proposed approach to the scenario of multiple kinds of workflows being executed on the system would result in the computation of bounds for multiclass QN models. The same approach described in [21] applies to BPEL workflows as it did for UML Activity diagrams. However, bounding techniques are seldom used in multiclass QN analysis: the reason is that bounds are mainly used to study bottleneck centers for which single class models suffice.

Let $\mathsf{Service}[A, R]$ be the average service time on resource R for a single invocation of action A; let $\mathsf{Visits}[A]$ be the *visit count* to action A. The visit count is defined as the ratio of the number of visits to action A and the number of completions of the whole BPEL $\mathbf{A}$. The total *service demand* $D[R]$ for resource R is given by:

$$D[R] = \sum_{A \in \mathbf{A}} \mathsf{Visits}[A] \times \mathsf{Service}[A, R] \tag{2}$$

Let us denote with $D = \sum_{i=1}^{K} D[R_i]$ the sum of all service demands; let D_{max} and D_{ave} denote the maximum and average of the service demands at the centers of the model, respectively.

Different kind of bounds can be computed, depending whether the BPEL represents a closed or open workload. Let us consider the two cases separately.

Bounds for Open Workloads. Let λ be the rate at which the BPEL $\mathbf{A}$ is executed; let $X(\lambda)$ and $R(\lambda)$ respectively denote the system throughput and response time with respect to parameter λ. Then the following equations hold [19]:

$$X(\lambda) \leq 1/D_{max} \tag{3}$$

$$\frac{D}{1 - \lambda D_{ave}} \leq R(\lambda) \leq \frac{D}{1 - \lambda D_{max}} \tag{4}$$

Bounds for Closed Workloads. Let N the total number of instances of BPEL $\mathbf{A}$ which are executed; let Z be the time spent by each BPEL instance outside the system before being executed again. If we denote with $X(N)$ and $R(N)$ the system throughput and response time as a function of the request population N, then the following equations hold [19]:

$$\frac{N}{D + Z + \frac{(N-1)D_{max}}{1+Z/(ND)}} \leq X(N) \leq \min\left(\frac{1}{D_{max}}, \frac{N}{D + Z + \frac{(N-1)D_{ave}}{1+Z/D}}\right) \tag{5}$$

$$\max\left(ND_{max} - Z, D + \frac{(N-1)D_{ave}}{1 + Z/D}\right) \leq R(N) \leq D + \frac{(N-1)D_{max}}{1 + Z/(ND)} \tag{6}$$

Note that Equations 3–6 provide bounds for the whole system throughput and response time. These quantities are very important for customers executing BPEL workflows on a system. From the system provider point of view, individual resource utilization $U[R_i]$ are an equally important parameter. Note that according to the utilization law (Eq. 1), bounds on $U[R_i]$ can be directly derived from bounds on X, by multiplying the latter by $D[R_i]$ (we will show shortly how to compute $D[R_i]$).

The modeler can specify whether the BPEL represents an open or closed workload by putting a suitable annotation in the BPEL specification. For example, a closed workload can be modeled with this code fragment (note that the `workload` element is in a different namespace with respect to the other standard BPEL elements):

```
<bpws:process>
  <perf:workload type="closed" thinktime=Z />
  ...
</bpws:process>
```

Similarly, an open workload can be modeled as follows:

```
<bpws:process>
  <perf:workload type="open" arrivalrate=λ />
  ...
</bpws:process>
```

In order to be able to compute the bounds, we need to know the service demand $D[R_i]$ for each resource (see Eq. 2). The service demand can be computed if we know the visit count $\mathsf{Visits}[A]$ for each action $A \in \mathbf{A}$, and the average service time on resource R for each execution of action A. Let us address these two issues separately.

Definition of the Service Time

In order to compute the mean service time $\mathsf{Service}[A_i, R]$ from Eq. 2, the system modeler is requested to annotate each method of the the WSs with which the workflow interacts with their average service time. This can be done using suitable XML elements in the WSDL specifications of the WS. Performance-oriented extensions of WSDL have been proposed in the literature (see [22,2] for one of these proposals). As we are only interested in representing service time for WSDL operations, we adopt here a stripped down notation for the sake of simplicity. Of course, our performance modeling approach is completely independent from the notation actually used to enrich BPEL and WSDL specifications with performance-oriented information.

Consider the following (simplified) WSDL describing an interface for an electronic flight booking application. Only the `checkAvailability` and `bookFlight` operations are shown; we also omit all details related to the input and output data types of such methods.

```
<wsdl:definitions name="BookingApp">
  <wsdl:portType name="BookingAppType">
    <wsdl:operation name="checkAvailability">
      ...
      <perf:PAdemand resource="disk" value="5"/>  <!-- (1) -->
      <perf:PAdemand resource="CPU" value="1"/>   <!-- (2) -->
    </wsdl:operation>
    <wsdl:operation name="bookFlight">
      ...
      <perf:PAdemand resource="disk" value="2"/>
      <perf:PAdemand resource="CPU" value="15"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definition>
```

In this code we added the child elements ⟨`perf:PAdemand`⟩ of ⟨`wsdl:operation`⟩ which are used to specify the mean service time of the operation on various resources; the `PAdemand` name has been chosen for similarity with the notation adopted in the UML Profile for Schedulability, Performance and Time Specification (UML-SPT profile [23]). The line labelled (1) denotes a mean service time of 5 on a resource named "disk", while line (2) denotes a service demand of 1 on a resource named "CPU". Both service times are related to the *checkAvailability* operation.

We use the `perf` prefix to denote the namespace where performance-oriented annotations are defined; this is to distinguish the new elements from the standard WSDL ones. The value of the `resource` attribute is a string denoting the name of a resource; the name is used for identification purposes only. The `value` attribute is a real number denoting the service time required by the specific resource. The WSDL should be annotated by the service provider, who of course is in the position of knowing how the operations are implemented, and can measure or estimate their service time.

Now we show how to compute the value of $\mathsf{Service}\,[A, R]$, for each $A \in \mathbf{A}$, $R \in \mathbf{R}$. If A is not an ⟨`invoke`⟩ action, then $\mathsf{Service}\,[A, R] = 0$ for every $R \in \mathbf{R}$. If A is an ⟨`invoke`⟩ action, defined as:

$$A \equiv \boxed{\begin{array}{l}\texttt{<invoke operation=}Op\texttt{>}\\ \quad \cdots \\ \texttt{</invoke>}\end{array}}$$

and the operation Op is defined and annotated in a WSDL as follows:

$$Op \equiv \boxed{\begin{array}{l}\texttt{<wsdl:operation name=}Op\texttt{>}\\ \quad \cdots \\ \quad \texttt{<perf:PAdemand resource=}R_1\ \texttt{value=}v_1\texttt{/>}\\ \quad \texttt{<perf:PAdemand resource=}R_2\ \texttt{value=}v_2\texttt{/>}\\ \quad \vdots \\ \quad \texttt{<perf:PAdemand resource=}R_K\ \texttt{value=}v_K\texttt{/>}\\ \texttt{</wsdl:operation>}\end{array}}$$

then, for each $i \in \{1, \ldots K\}$ we let $\mathsf{Service}[A, R_i] = v_i$. We assume $\mathsf{Service}[A, R_i] = 0$ if the ⟨`perf:PAdemand`⟩ tag is omitted for resource R_i.

Computation of Visit Counts
We recall that the visit count $\mathsf{Visits}[A_i]$ represents the ratio of the number of visits to BPEL activity A_i versus the number of completions of the whole BPEL **A**. Visit counts can be computed by solving a system of linear equations; the equations are derived by structural analysis of the BPEL activity, as follows. Given a BPEL fragment A_i, we denote with $\mathcal{E}[A_i]$ a set of linear equations. The function is defined by structural induction on A_i, according to the rules shown in Table 1.

5 Case Study

In this section we illustrate how the technique described in Sec. 4 can be applied to a case study. As a motivating example, we consider a set of WSs which can be used to execute jobs in a computational Grid [24]; in fact, the Basic Execution Service (BES) [25] and Open Grid Services Architecture Data Access and Integration (OGSA-DAI) [26] are WS interfaces for job submission and data transfer respectively, which are being standardized in an effort to allow interoperability between Grid components provided by different projects.

Let us consider a system where the following WSs are available: a *Storage Element*, which is responsible for storing (possibly large amount of) data; a *Computing Element*, which is a WS which can accept and execute computational jobs; an *Analysis Element*, which is a service for analyzing the output data produced by running some application on the Computing Element. The annotated WSDL interfaces of these services are reported in a simplified form in Appendix A.2. Each operation is annotated with the (estimated) average service time on the resources they require.

We consider the BPEL sketched in Appendix A.1. The BPEL represents a closed workload, where each workflow spends 120 time units outside the system (*think time*) before being executed again. The workflow executes the following sequence of actions:

- The user authenticates on the system.
- The executable application and the files it needs to operate (called *Input SandBox*) are transferred in parallel with the input data that must be processed by the application.
- The executable is started a number of times, possibly with different parameters (this latter detail is not shown in the BPEL); this is done by iterating the `JobStart` operation inside a ⟨`while`⟩ statement until a certain condition (not shown in the BPEL) is false. The probability of the condition being true is set to be 0.7.
- The output produced by the executable (called the *Output SandBox*) is transferred to another WS to be analyzed.
- The output data are finally analyzed.

Table 1. Computation of the visit count

$A_i \equiv$ <sequence|flow> A_{j_1} $\vdots$ A_{j_k} </sequence|flow>	$\mathcal{E}[A_i] = \begin{cases} \mathsf{Visits}[A_{j_1}] = \mathsf{Visits}[A_i] \\ \vdots \\ \mathsf{Visits}[A_{j_k}] = \mathsf{Visits}[A_i] \\ \mathcal{E}[A_{j_1}] \\ \vdots \\ \mathcal{E}[A_{j_k}] \end{cases}$
$A_i \equiv$ <if> <condition perf:prob=p_{j_1}> C_1 </condition> A_{j_1} <elseif> <condition perf:prob=p_{j_2}> C_2 </condition> A_{j_2} </elseif> $\vdots$ <else>A_{j_k}</else> </if>	$\mathcal{E}[A_i] = \begin{cases} \mathsf{Visits}[A_{j_1}] = p_{j_1}\mathsf{Visits}[A_i] \\ \vdots \\ \mathsf{Visits}[A_{j_{k-1}}] = p_{j_{k-1}}\mathsf{Visits}[A_i] \\ \mathsf{Visits}[A_{j_k}] = \left(1 - \sum_{t=1}^{k-1} p_{j_t}\right) \times \mathsf{Visits}[A_i] \\ \mathcal{E}[A_{j_1}] \\ \vdots \\ \mathcal{E}[A_{j_k}] \end{cases}$
$A_i \equiv$ <pick> <onMessage perf:prob=p_{j_1} ...> A_{j_1} </onMessage> $\vdots$ <onMessage perf:prob=p_{j_k} ...> A_{j_k} </onMessage> </pick>	$\mathcal{E}[A_i] = \begin{cases} \mathsf{Visits}[A_{j_1}] = p_{j_1}\mathsf{Visits}[A_i] \\ \vdots \\ \mathsf{Visits}[A_{j_k}] = p_{j_k}\mathsf{Visits}[A_i] \\ \mathcal{E}[A_{j_1}] \\ \vdots \\ \mathcal{E}[A_{j_k}] \end{cases}$
$A_i \equiv$ <repeatUntil> A_j <condition perf:prob=p> C </condition> </repeatUntil>	$\mathcal{E}[A_i] = \begin{cases} \mathsf{Visits}[A_j] = \frac{1}{p}\mathsf{Visits}[A_i] \\ \mathcal{E}[A_j] \end{cases}$
$A_i \equiv$ <while> <condition perf:prob=p> C </condition> A_j </while>	$\mathcal{E}[A_i] = \begin{cases} \mathsf{Visits}[A_j] = \frac{p}{1-p}\mathsf{Visits}[A_i] \\ \mathcal{E}[A_j] \end{cases}$
$A_i \equiv$ <forEach> <startCounterValue> S </startCounterValue> <finalCounterValue> T </finalCounterValue> <scope>A_j</scope> </forEach>	$\mathcal{E}[A_i] = \begin{cases} \mathsf{Visits}[A_j] = (T - S + 1)\mathsf{Visits}[A_i] \\ \mathcal{E}[A_j] \end{cases}$

We developed a command-line tool written in C++ which is able to parse the annotated BPEL (and associated WSDL) specifications and outputs the

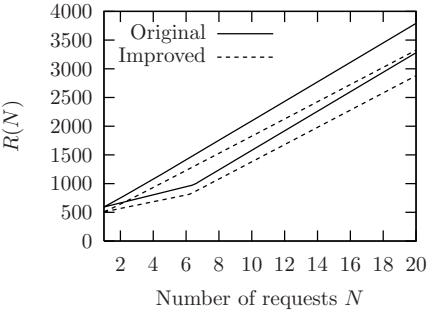
Table 2. Service demands for the case study

Resource	Visit count	Service Demand
CE:CPU	5.33	22.33
CE:Disk	2.0	150.00
DA:CPU	1.0	100.00
DA:Disk	1.0	30.00
DF:CPU	1.0	1.00
DF:Disk	1.0	120.00
Network	3.0	170.00

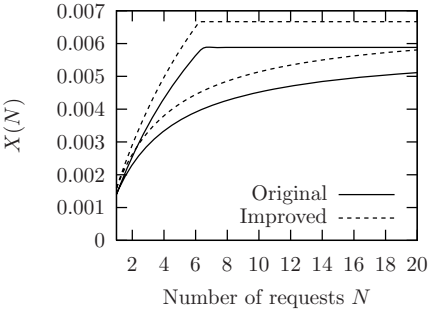
performance bounds computed using Eq. 3–6. The tool, called `bpel2qnbound`, builds a BPEL performance model based on the metamodel shown in Fig. 2. The tool, then, computes the visit counts and the service demands using the approach described in Section 4, and produces the appropriate bound equations as output.

By using the `bpel2qnbound` tool, we get the visit counts and service demands shown in Table 2. As can be seen, the bottleneck device is the network. This means that system performance, at heavy load, will not improve unless that bottleneck is removed.

Given that the BPEL of the case study represents a closed workload, the bounds from Eq. 5 and 6 apply. Fig. 4 shows the upper and lower bounds for the Response Time and Throughput, as a function of the request population size N.



(a) Response Time Bounds (lower is better)



(b) Throughput Bounds (higher is better)

Fig. 4. Performance Bounds for the Case Study, original system vs. an improved one where the bottleneck is removed

One of the most important questions a system modeler may ask is: "What should be done in order to improve the system throughput/response time?". This question is easily answered by looking at the service demands on Table 2: better performance can be obtained if the system bottleneck (the network, in this case) is removed. According to Eq. 2, the service demand decreases if either (1) fewer visits are made

to the bottleneck device, or (2) the average service time is shortened. Solution (1) implies that the BPEL workflow (and possibly the system as well) is restructured so that fewer network communications are required. On the other hand, solution (2) requires moving to a faster network. If we consider an improved system where the service demand on the network is exactly half of that on the original system, we obtain the new bounds shown with the dashed lines in Fig. 4. The service demand on the network for the improved system is 85, so that the bottleneck device becomes the disk on the Computing Element. From the figure, the improved system offers definitely better performance (both throughput and response time) than the original one, as the request population size N increases.

6 Conclusions

In this paper we described a multi-view approach for the performance prediction of service-based applications encompassing both users and service provider(s) perspectives. As a first step towards the realization of this integrated framework we described an algorithm for efficient computation of performance bounds for BPEL workflows. Our approach applies QN analysis techniques directly on the BPEL specification of the workflow and the WSDL associated with the Web Services it references. We showed how to compute bounds for the system throughput and response time using QN bounds. The approach can be fully automated: we developed a prototype tool, called `bpel2qnbound`, which automatically derives the appropriate bounds from the annotated workflow specification. Our technique does not require the derivation (and solution) of the underlying QN model; nevertheless, bounding techniques are useful to identify and quantify the effect of system bottlenecks, and to perform quick analysis and discard inappropriate (performance-wise) alternatives at an early stage of a study. The results of the `bpel2bound` tool can be interpreted both from a customer perspective, to select among the available WSs those providing the best performance, and also from a system provider perspective, to identify bottlenecks and estimate the performance gains obtained by upgrading different parts of the system.

The research described in this paper can be extended in several directions. A technique very similar to that described in [27] can be applied to explicitly derive a multiclass QN model from annotated BPEL specifications; while the resulting model would be more difficult to analyze, it would provide more accurate performance measures. The long term goal is to integrate different performance analysis techniques for BPEL into a single tool, where the system modeler can choose the most appropriate type of analysis depending on speed/accuracy tradeoff.

References

1. Menasce, D.A.: QoS Issues in Web Services. IEEE Internet Computing 6(6), 72–75 (2002)
2. D'Ambrogio, A.: A model-driven WSDL extension for describing the qos of web services. In: 2006 IEEE International Conference on Web Services (ICWS 2006), Chicago, Illinois, pp. 789–796. IEEE Computer Society Press, Los Alamitos (2006)

3. Denning, P.J., Buzen, J.P.: The operational analysis of queueing network models. ACM Comput. Surv. 10(3), 225–261 (1978)
4. Balbo, G., Serazzi, G.: Asymptotic analysis of multiclass closed queueing networks: Multiple bottlenecks. Performance Evaluation 30, 115–152 (1997)
5. Maximilien, E.M., Singh, M.P.: A Framework and Ontology for Dynamic Web Services Selection. IEEE Internet Computing 8(5), 84–93 (2004)
6. Patel, C., Supekar, K., Lee, Y.: A QoS Oriented Framework for Adaptive Management of Web Service Based Workflows. In: Mařík, V., Štěpánková, O., Retschitzegger, W. (eds.) DEXA 2003. LNCS, vol. 2736, pp. 826–835. Springer, Heidelberg (2003)
7. Yu, T., Lin, K.J.: A Broker-Based Framework for QoS-Aware Web Service Composition. In: Proc. of 2005 IEEE Int'l Conf. on e-Technology, e-Commerce and e-Service, Hong Kong, China (March 2005)
8. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Trans. Softw. Eng. 30(5), 311–327 (2004)
9. Ardagna, D., Pernici, B.: Global and Local QoS Guarantee in Web Service Selection. In: Proc. of Business Process Management Workshops, pp. 32–46 (2005)
10. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: An Approach for QoS-aware Service Composition Based on Genetic Algorithms. In: Proc. of Genetic and Computation Conf., Washington, DC (June 2005)
11. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. J. Web Sem. 1(3), 281–308 (2004)
12. Serhani, M.A., Dssouli, R., Hafid, A., Sahraoui, H.: A QoS Broker Based Architecture for Efficient Web Services Selection. In: Proc. of 2005 Int'l Conf. on Web Services, Orlando, pp. 113–120 (July 2005)
13. Yu, T., Lin, K.J.: Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In: Proc. of 3rd Int'l Conf. on Service Oriented Computing, Amsterdam, The Netherlands, pp. 130–143 (December 2005)
14. Claro, D.B., Albers, P., Hao, J.-K.: Selecting Web Services for Optimal Composition. In: Proc. of ICWS 2005 2nd Int'l Workshop on Semantic and Dynamic Web Processes, Orlando (July 2005)
15. Cardoso, J.: Complexity analysis of BPEL web processes. Software Process: Improvement and Practice 12(1), 35–49 (2007)
16. Rud, D., Schmietendorf, A., Dumke, R.: Performance modeling of ws-bpel-based web service compositions. In: Services computing workshops, SCW 2006, pp. 140–147. IEEE Computer Society Press, Los Alamitos (2006)
17. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing Networks and Markov Chains. J. Wiley, Chichester (1998)
18. Cardellini, V., Casalicchio, E., Grassi, V., Mirandola, R.: A framework for optimal service selection in broker-based architectures with multiple QoS classes. In: Services computing workshops, SCW 2006, pp. 105–112. IEEE Computer Society Press, Los Alamitos (2006)
19. Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: Quantitative System Performance: Computer System Analysis Using Queueig Network Models. Prentice-Hall, Englewood Cliffs (1984)
20. Alves, A., et al.: Web service business process execution language version 2.0 Committee Draft (May 17, 2006)
21. Balsamo, S., Marzolla, M., Mirandola, R.: Efficient performance models in component-based software engineering. In: EUROMICRO 2006: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 64–71. IEEE Computer Society Press, Los Alamitos (2006)

22. D'Ambrogio, A.: A WSDL extension for performance-enabled description of web services. In: Yolum, p., Güngör, T., Gürgen, F., Özturan, C. (eds.) ISCIS 2005. LNCS, vol. 3733, pp. 371–381. Springer, Heidelberg (2005)
23. Object Management Group (OMG): UML profile for schedulability, performance and time specification. Final Adopted Specification ptc/02-03-02, OMG (March 2002)
24. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, San Francisco (2003)
25. OGSA Basic Execution Service Working Group: `http://forge.gridforum.org/projects/ogsa-bes-wg`
26. Antonioletti, M., Krause, A., Paton, N.W., Eisenberg, A., Laws, S., Malaika, S., Melton, J., Pearson, D.: The WS-DAI family of specifications for web service data access and integration. SIGMOD Record 35(1), 48–55 (2006)
27. Balsamo, S., Marzolla, M.: Performance evaluation of UML software architectures with multiclass queueing network models. In: Proc. of the Fifth International Workshop on Software and Performance (WOSP 2005), pp. 37–42. ACM Press, New York (2005)

A Appendix

A.1 BPEL for the Case Study

```
<bpws:process>
  <perf:workload type="closed" thinktime="120"/>
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/"
    location="CaseStudy.wsdl">
  <bpws:sequence>
    <bpws:invoke operation="Authenticate"/>
    <bpws:flow>
      <bpws:invoke operation="TransferISB"/>
      <bpws:invoke operation="TransferData"/>
    </bpws:flow>
    <bpws:while>
      <bpws:condition prob="0.7"/>
      <bpws:invoke operation="JobStart"/>
    </bpws:while>
    <bpws:invoke operation="TransferOSB"/>
    <bpws:invoke operation="Analyze"/>
  </bpws:sequence>
</bpws:process>
```

A.2 WSDL for the Case Study

```
<!-- Interface for Storage Element -->
<definitions>
  <portType name="DataFactory">
    <operation name="TransferData">
```

```
      <perf:PAdemand resource="DF:CPU" value="1"/>
      <perf:PAdemand resource="DF:Disk" value="120"/>
      <perf:PAdemand resource="Network" value="80"/>
    </operation>
  </portType>
</definitions>

<!-- Interface for Computing Element -->
<definitions>
  <portType name="JobFactory">
    <operation name="Authenticate">
      <perf:PAdemand resource="CE:CPU" value="10"/>
    </operation>
    <operation name="TransferISB">
      <perf:PAdemand resource="CE:CPU" value="2"/>
      <perf:PAdemand resource="Network" value="10"/>
      <perf:PAdemand resource="CE:Disk" value="120"/>
    </operation>
    <operation name="JobStart">
      <perf:PAdemand resource="CE:CPU" value="4"/>
    </operation>
    <operation name="TransferOSB">
      <perf:PAdemand resource="CE:CPU" value="1"/>
      <perf:PAdemand resource="Network" value="80"/>
      <perf:PAdemand resource="CE:Disk" value="30"/>
    </operation>
  </portType>

<!-- Interface for Analysis Element -->
<definitions>
  <portType name="DataAnalysis">
    <operation name="Analyze">
      <PAdemand resource="DA:CPU" value="100"/>
      <PAdemand resource="DA:Disk" value="30"/>
    </operation>
  </portType>
</definitions>
```