# Performance Modeling of WS-BPEL-Based Web Service Compositions

Dmytro Rud[1] Andreas Schmietendorf[1,2] Reiner Dumke[1]

[1] *Software Engineering Group, Institute for Distributed Systems, University of Magdeburg PF 4120, D-39016 Magdeburg, Germany {rud, schmiete, dumke}@ivs.cs.uni-magdeburg.de*

[2] *FHW – Berlin School of Economics, Fachbereich II – Wirtschaftsinformatik Neue Bahnhofstr. 11-17, D-10245 Berlin schmiete@fhw-berlin.de*

## Abstract

*This paper addresses quality of service aspects of web service orchestrations created using WS-BPEL from the standpoint of a web service integrator. A mathematical model based on operations research techniques and formal semantics of WS-BPEL is proposed to estimate and forecast the influence of the execution of orchestrated processes on utilization and throughput of individual involved nodes and of the whole system. This model is applied to the optimization of service levels agreement process between the involved parties.*

## 1. Introduction

In the context of the broad SOA [1] adoption, the ensuring a high quality of service (QoS) becomes very important. But estimation and management of availability, performance and other QoS aspects of web service compositions are not simple due to their distributed nature.

One of these aspects – the performance of WS-BPEL [2] based web service orchestrations – will be discussed in this paper from the standpoint of a web service integrator. The mission of the integrator is to select an optimal set of third-party services, to orchestrate a composite web service from them by filling out a business process description template with all information necessary to start the process – i.e. with partner links, addresses, etc., and finally to provide the latter to the customers. The relationship between the integrator and providers of third-party web services is based on service level agreements (SLAs) which in particular determine pricing conditions and QoS warranties. The integrator has to negotiate a SLA with every provider of selected services and to be convinced of the SLA fulfillment later on. It is not sufficient for the integrator to have only black-box view for that – specifically utilization of nodes, internal processing times and overall count of requests to a web service or one of its operations are indispensable for analysis and prediction of performance dynamics.

The main goal of this paper is to develop a performance analysis and evaluation model of processes created with WS-BPEL and to use this model to make predictions about their performance in order to optimize used service levels. We outline also an infrastructure that uses this model. The tactic goals are to be able to verify whether a new process instance will worsen the performance of involved web service provider nodes and therefore of our processes which use them, and to check whether web service providers fulfill their SLAs. We describe also WS-BPEL process profiling, which enables the operator to check whether there are bottlenecks in the system.

The rest of this paper is organized as follows: Section 2 gives a review of available related work, Section 3 represents the core of the paper and describes the mathematical model we have developed, in Section 4 further application and implementation aspects of this model are outlined, and Section 5 contains conclusions and proposals regarding subsequent work.

## 2. Related work

The book "Capacity Planning for Web Services: metrics, models, and methods" [3] describes various application aspects of the operations research techniques to the performance analysis of distributed software infrastructures and client-server systems. Questions such as capacity planning, benchmarking, performance and availability measurement, workload management and forecasting, performance modeling are discussed in-depth.

In [11] Prof. Menascé describes a generalized algorithm for SLA optimization of components involved in a distributed application. This algorithm proceeds from given maximum allowable total execution time, components' cost functions and average invocation

Proceedings of the IEEE Services Computing Workshops (SCW'06)
0-7695-2681-0/06 $20.00 © 2006 IEEE

counts, and minimizes the total cost of the application.

The Web Service Trust Center (WSTC), arranged by the authors of the present paper at the Institute for Distributed Systems of the Otto von Guericke University of Magdeburg, is a platform for development and evaluation of measurement tools and techniques in the field of SOA and web services. Its main component is Wesement – a web-based application intended for long-term measuring of performance, availability and metadata stability of web services. Collected statistics persist in a database and can be subsequently used for analysis and forecasting, imported in XML format or represented graphically. The measurement service itself as well as our experiences of its usage are described in [4], [5] and [6]. Technical aspects of the implementation are considered in [7]. The present paper constitutes the next development step of the Web Service Trust Center project.

Performance engineering and quality of service guaranteeing in the context of web service providing is also the subject of many other studies.

There are quite a number of performance-driven web service management infrastructures proposed which are in particular destined to automate negotiation, accomplishment and fulfillment validation of service level agreements [8, 9, 10, 12, 13].

Extensions of standard web service specifications can also be found in other studies which do not have a direct bearing to SLA negotiation and management infrastructures [14, 15, 16, 17].

QoS aspects of general web service-based workflows are discussed in [18, 19, 20].

Papers [21] and [22] address formal language semantics of WS-BPEL.

## 3. The proposed model

As mentioned in the introduction, we have developed a mathematical model of the performance of composed web service processes orchestrated with WS-BPEL. This model is based on operations research techniques [23, 3]. The two main actors in the model are the integrator running the WS-BPEL engine and the third-party web service providers. Both the engine and the provider nodes must be appropriately instrumented to be able to perform necessary measurements. Measurement modules must be added to both the engine and provider nodes for that purpose. They send measured data to the measurement evaluation and prediction engine that runs on the integrator side and is connected with the SLA management engine. The values measured on the provider nodes must be then transferred to the engine. Section 4 discusses various implementation possibilities of such instrumentation. There is also a database for saving measurement data obtained from measurement modules. The architecture of the proposed system is shown in Figure 1.

### 3.1. Used formalizations

Now we can turn to the description of the mathematical model itself, which gives base functioning principles of the measurement evaluation and prediction engine. We use the following notation:

- $N$ – Set of web service providers (provider nodes).
- $S[n]$ – Set of web services located on the node *n*.
- $M[s]$ – Set of operations provided by the service *s*.
- $T[m] = \{t_i[m]\}$ – Response times of the operation *m* measured by the integrator.
- $B[m] = \{b_i[m]\}$ – Compute times of the operation *m*. The corresponding provider is assumed to measure these values internally and to grant them to the integrator in accordance to the contract.
- $w[s]$ – Current waiting time that requests to web service *s* must spend in its incoming queue (for the sake of simplicity we assume that the latter is organized on the *First Come-First Served* principle) before they can be processed. This value depends on the utilization of the corresponding provider node and generally cannot be directly measured.
- $d[n]$ – Current network latency (ping time) of node *n* relative to the integrator's node. Network latencies are to be periodically measured by the integrator.
- $R[m] = \{r_i[m]\}$ – Invocation rates of the operation *m* on the corresponding web service node.
- $u[n]$ – Current utilization of the provider node *n* caused by serving web service requests from all its clients.
- $u'[n]$ – Current utilization of the provider node *n* caused by its other activities, i.e. not by serving web service requests.
- $u^o[n]$ – Current overall utilization of the node *n*.

As with compute times, we assume that the integrator can obtain utilization statistics from web service providers on regular basis.

For the performance and utilization quantities the following natural correlations are defined:

- $t[m] = b[m] + w[s] + d[n] \quad \forall\, n \in N,$ $s \in S[n],\, m \in M[s]$;
- $u^o[n] = u[n] + u'[n] \quad \forall\, n \in N$
- $u[n] = \sum_{s \in S[n]} \sum_{m \in M[s]} (r[m] \cdot b[m]) \quad \forall\, n \in N.$

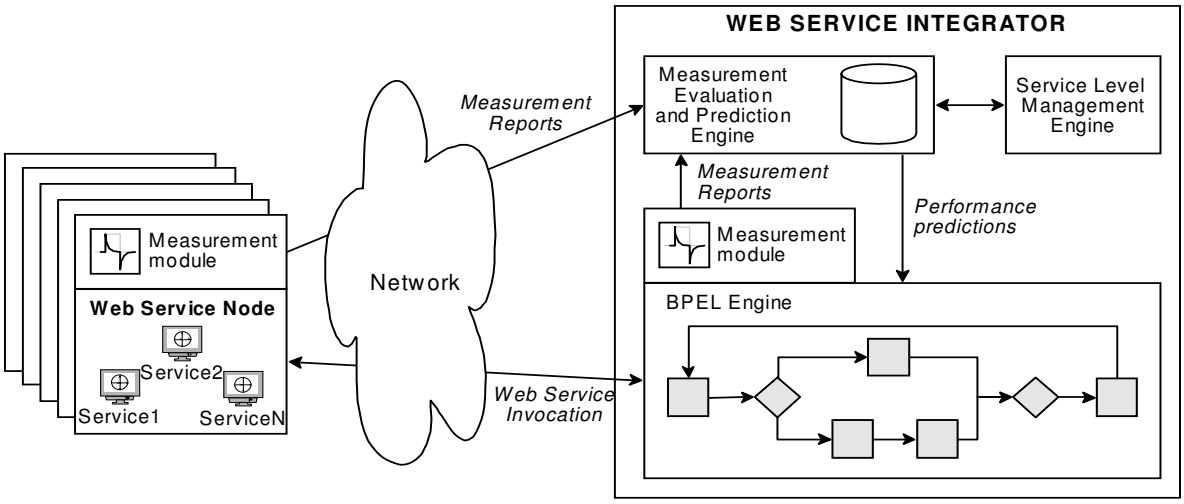


Figure 1. Performance-aware web service orchestration infrastructure

Based on these correlations and on the principles of operations research we can make the following assumption: waiting times of requests to web services hosted on a node *n* are minimal if the overall utilization of this node is not greater than 1: $u^o[n] \le 1 \Rightarrow \sum_{s \in S[n]} w[s] = \min$. Hence, these requests are served with the maximal rate achievable without reordering of requests.

If this assumption is currently true for node *n* and there come *k* new requests to operations $m_1 ... m_k \in \bigcup_{s \in S[n]} M[s]$ during one time interval, then the performance of *n* will not become worse if the expected overall execution time of these requests $b^{ex} = \sum_{i=1}^{k} b^{ex}[m_i]$, calculated as a sum of average values from expected execution times of $m_1 ... m_k$, is not greater than $(1 - u^o[n])$.

Expected values of some parameters will be used in the most formulae proposed below as well; therefore the first question we have to discuss is where there expected values can be taken from. Then (in subsections 3.3 and 3.4) we present methods for the estimation of execution times of and web service operations' invocation counts in WS-BPEL processes. Finally (in subsection 3.5) we use these values to achieve the main goal of the proposed model as mentioned in the introduction.

### 3.2. Nature of expected values

There are following methods to derive the expected value $p^{ex}$ of a parameter *p* based in particular on statistical time series $P = \{p_i\}$ of previously measured values:

1. If a correlation $\hat{p} = f_c(q)$ can be recognized or assumed between *p* and some other parameter *q*, and the current value of the latter equals $q^c$, then the expected value of *p* can be calculated as $p^{ex} = f_c(q^c)$.
2. Similarly, if there is a known autoregression association $\hat{p}_i = f_a(p_{i-k}, ..., p_{i-1})$, then the expected value of *p* can be calculated as $p^{ex} = f_a(p_{|P|-k}, ..., p_{|P|})$, where $k \in \{0..|P|-1\}$.
3. Values of some kinds of parameters – e.g. durations or probabilities – could be considered to obey a statistical (e.g. Gaussian) distribution.
4. For some parameters (e.g. waiting time with prescribed timeout) worst-case values could be determined and used as expected values, but for most parameters such values equal to the positive infinity and thus are hardly usable.
5. If none of the above methods are available (e.g. because the measured time series *P* are not representative enough to be used in the autoregression or the current value of *q* is unknown), the only possibility for getting $p^{ex}$ is to use some default values.

For every parameter of the system appropriate calculation method of the expected value must be chosen. While presenting formulae and algorithms we will specify possible choices using curly braces, e.g. "{1,2,5}". It is impossible to select the best choice

from these variants beforehand, because this selection depends on the quantity of collected statistical data, current service levels, and preferences of the integrator. Thus the final configuration must be done at the deployment time.

### 3.3. Calculation of expected duration of a business process ($\tau$)

In this subsection we give rules for calculation of expected durations of executable WS-BPEL processes.

Note that fault handlers will be not taken into account because exceptions are not part of normal behavior of the scope. Compensation handlers and <*compensate*> activities will be ignored as well, because they can be activated only from fault or another compensation handlers.

We start with simple rules for basic activities:

- **<invoke>**: if synchronous, take $\tau$ from {1,2,5}. If asynchronous, assume $\tau = 0$.
- **<receive>**: $\tau =$ expecting waiting time from {1,2,3,5}.
- **<wait>**: $\tau =$ expecting waiting time from {3,5}.
- **<assign>**, **<empty>**, **<throw>**: assume $\tau = 0$.
- **<reply>**: assume $\tau = 0$ (analogously to asynchronous <*invoke*> activity).

Then we give (recursive) formulae and algorithms for structured activities:

- **<sequence>** with nested activities $c_1 ... c_k$: $\tau = \sum_{i=1}^{k} \tau[c_i]$.
- **<switch>** with branches $c_1 ... c_{k+1}$ (where branch $c_{k+1}$ is the default branch) and probabilities $p_1 ... p_{k+1}$ {1,2,3,5} ($\sum_{i=1}^{k+1} p_i = 1$) of entering particular branches: $\tau = \sum_{i=1}^{k+1} (\tau[c_i] \cdot p_i)$.
- **<while>** with expected loops count $f$ {1,2,3,5} (can be float) and expected duration of the body of the loop $\tau_{body}$: $\tau = f \cdot \tau_{body}$.
- **<pick>** with expected probabilities $p_1 ... p_k$ {1,2,3,5} of message arrivals or alarms: We imagine each <*pick*> activity as a <*sequence*> of one <*wait*> activity and one <*switch*> activity. All <*onMessage*> and <*onAlarm*> sub-elements of the <*pick*> element are considered to represent branches $c_1 ... c_k$ of the <*switch*> activity. The duration of the <*wait*> activity $\tau_{wait}$ can be predicted from {1,2,3,4,5}. The duration of the <*switch*> activity $\tau_{switch}$ is to be calculated as described above. Thus, $\tau = \tau_{wait} + \tau_{switch} = \tau_{wait} + \sum_{i=1}^{k} (\tau[c_i] \cdot p_i)$.
- **event handlers**: same as for <*pick*>, we consider each event handler as a <*wait*> activity followed by a <*switch*> activity. The duration of the <*wait*> activity $\tau_{wait}$ may vary from zero to the overall duration of "normal" activities of the (local or global) scope {1,2,3}. All <*onMessage*> and <*onAlarm*> sub-elements of the <*eventHandlers*> element are considered to represent branches $c_1 ... c_k$ of the <*switch*> activity, with the exception that probabilities $p_1 ... p_y$ $(y \le k)$ of message events' branches can be greater than 1 because of the possibility of each message event to occur more than once. The overall duration of event handling can be estimated as $\tau = \tau_{wait} + \tau_{switch}$.
- **<scope>** with nested activity $c_1$ and event handlers $c_2 ... c_k$: $\tau = \max(\{\tau[c_i]\})$, $i \in 1...k$.
- **<flow>**: the estimation of the overall flow's duration is complicated due to possible presence of synchronization links. To take them into account, we represent the flow as a directed acyclic graph (DAG) $G := (V, E)$.

  Outgoing edges of the flow's source vertex $v^{start}$ go to start vertices of parallel branches. Outgoing edges of the parallel branches' end vertices sink into the end vertex of the flow $v^{end}$. Subgraphs of the nested activities must be constructed as follows:
  - basic activities as well as structured activities whose boundaries are not crossed by synchronization links can be represented as single vertices. In particular, <*while*> activities, serializable <*scope*>'s and event handlers can be always simplified in this way.
  - each <*sequence*> activity with boundary-crossing links becomes a chain consisting of a start vertex, nested activities' vertices and an end vertex.
  - <*switch*>, <*pick*>, <*flow*> and <*scope*> activities with boundary-crossing links are represented the same way as the "main" <*flow*>, i.e. as a source and a sink with branches' subgraphs between them (nature of the branches depends on the activity).
  - fault and compensation handlers are ignored and have no corresponding graph elements.

Synchronization links are represented as edges and form the proper subset $E^S \subset E$. All other edges constitute the subset $E^N = E \setminus E^S$.

Start and end vertices of the "main" flow and of non-simplified nested structured activities ("support vertices") constitute the subset $V^* \subseteq V$. Other vertices belong to the proper subset $V^N = V \setminus V^*$.

Figure 2 shows an example flow graph that has two branches. The first one represents a <*sequence*> with a nested <*switch*> activity, the second one is a simple <*sequence*>. One activity of this simple <*sequence*> depends on a nested activity from one of the branches in <*switch*> (thick arrow). Support vertices are marked on the picture by a dark background color.

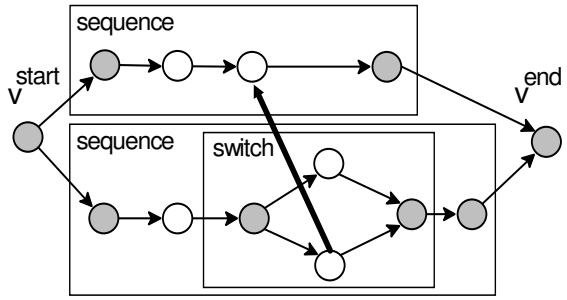


Figure 2. Example of a flow graph

For all $v \in V$ we define the following attributes:

- $\alpha^N[v]$ is the moment relative to the start time of the flow at which the activity represented by the vertex *v* will become ready to start, without consideration of incoming synchronization links: $\alpha^N[v] = \max(\{0\} \cup \{\gamma[w] \mid (w,v) \in E^N\})$.
- $\alpha^S[v]$ – same as $\alpha^N[v]$, but incoming synchronization links are being taken into account too: $\alpha^S[v] = \alpha^N[v] + \max(\{0\} \cup \{p[(w,v)] \cdot (\gamma[w] - \alpha^N[v]) \mid (w,v) \in E^S\})$, where $p[(w,v)]$ {1,2,3,5} is the probability that the status of the synchronization link *(w,v)* will be positive. It depends not only on the *transitionCondition* and *joinCondition* attributes of the corresponding source and target activities, but also on the probability of the source activity to be performed.
- $\gamma[v]$ is the moment relative to the start time of the flow at which the activity represented by the vertex *v* will complete: $\gamma[v] = \alpha^S[v] + \tau^G[v]$.
- $\tau^G[v]$ is the "duration of passing" the vertex *v*:

$$\tau^G[v] = \begin{cases} 0, & v \in V^* \\ \tau[c], & v \in V^N, \end{cases}$$

where *c* is the activity represented by the vertex *v*. Concluding formula for estimation of the duration of the entire flow is $\tau = \gamma[v^{end}]$.

- **<terminate>**: terminate calculation of the duration of the scope in which this element belongs.
- **entire process**: same as <*scope*>.

If actually *k* different processes $q_1 \ldots q_k$ are possible based on one given WS-BPEL document (this will be the case if multiple <*receive*> or <*pick*> elements exist with attribute *createInstance* set to "yes"), we need to calculate appropriate expected durations $\tau[q_1] \ldots \tau[q_k]$ for each of these processes and then take the average weighted value based on expected probabilities $p_1 \ldots p_k$ {3,5} ($\sum_{i=1}^{k} p_i = 1$) of entering each process. Hence the concluding formula is $\tau = \sum_{i=1}^{k} (\tau[q_i] \cdot p_i)$.

### 3.4. Calculation of expected web service operations' invocation counts ($\beta$)

Same as with durations, fault handlers, compensation handlers and <*compensate*> activities will be ignored.

For each operation $m \in \bigcup_{s \in S[n]} M[s],\ n \in N$ the following (recursive) formulae can be used:

- **<invoke>**: $\beta_m = 1$.
- **<sequence>** with nested activities $c_1 \ldots c_k$: $\beta_m = \sum_{i=1}^{k} \beta_m[c_i]$.
- **<scope>** with nested activity $c_1$ and event handlers $c_2 \ldots c_k$: same as sequence.
- **<flow>** with parallel branches $c_1 \ldots c_k$: same as sequence.
- **<switch>** with branches $c_1 \ldots c_{k+1}$ (where branch $c_{k+1}$ is the default branch) and probabilities $p_1 \ldots p_{k+1}$ {1,2,3,5} ($\sum_{i=1}^{k+1} p_i = 1$) of entering par-

ticular branches: $\beta_m = \sum_{i=1}^{k+1}(\beta_m[c_i]\cdot p_i)$.

- **<while>** with expected loops count $f$ {1,2,3,5} (can be float) and expected invocations count in the body of the loop $\beta_{body}$: $\beta_m = f \cdot \beta_{body}$.
- **<pick>** with expected probabilities $p_1...p_k$ of actions $c_1...c_k$ representing message arrivals or alarms: $\beta_m = \sum_{i=1}^{k}(\beta_m[c_i]\cdot p_i)$.
- **event handlers**: same as <*pick*> with the exception that probabilities of message events' branches can be greater than 1.
- **<terminate>**: terminate calculation of invocations count for the scope to which this element belongs.
- **other basic activities** (<*receive*>, <*assign*>, <*throw*>, <*reply*>, <*empty*>, <*wait*>): $\beta_m = 0$.

Same as for durations, if actually $k$ different processes $q_1...q_k$ are possible based on one given WS-BPEL document, we need to calculate appropriate expected invocations' counts $\beta_m[q_1]...\beta_m[q_k]$ for each of these processes and then take the average weighted value based on expected probabilities $p_1...p_k$ {3,5} ($\sum_{i=1}^{k} p_i = 1$) of entering each process: $\beta_m = \sum_{i=1}^{k}(\beta_m[q_i]\cdot p_i)$.

### 3.5. Analysis of expected additional utilizations of third-party web service provider nodes

After formulae to compute expected durations and web service operations' invocation counts for any given executable WS-BPEL process have been derived, we can apply them to analyze expected additional utilizations of third-party provider nodes.

Expected supplementary invocation rate of operation $m$, which is caused by the execution of the given WS-BPEL process $q$, can be calculated as $r_q^+[m] = \frac{\beta_m[q]}{\tau[q]}$. For $k$ processes $q_1...q_k$, the overall supplementary invocation rate of $m$ is $r^+[m] = \sum_{i=1}^{k} r_{q_i}^+[m] = \sum_{i=1}^{k} \frac{\beta_m[q_i]}{\tau[q_i]}$.

Expected supplementary utilization of an involved web service provider node $n$, which is caused by the execution of given WS-BPEL processes $q_1...q_k$, is $u^+[n] = \sum_{s\in S[n]}\sum_{m\in M[s]}(b^{ex}[m]\cdot r^+[m])$ ($b^{ex}[m]$ from {1,2,5}). According to the corollary mentioned previously in the introductory section, the performance of the node $n$ will not become worse if $\sum_{s\in S[n]}\sum_{m\in M[s]}(b^{ex}[m]\cdot\sum_{i=1}^{k}\frac{\beta_m[q_i]}{\tau[q_i]}) \le 1 - u^o[n]$.

This is the main criterion for the performance optimization and provision.

### 3.6. Optimization of service levels

The cooperation of the integrator and third-party web service providers is based on service level agreements. The providers place their services with certain (previously agreed or default) service levels at the integrator's disposal and the latter must pay for that. The selection of the optimal service level could be very expedient for the integrator because it helps to avoid inconsistency between actually performed and paid web service invocations and other consumed resources.

Note that we restrict ourselves with performance and price aspects of service level agreements. Other possible SLA properties – e.g. accuracy, confidentiality, etc. – are not relevant and therefore not discussed here. The SLA negotiation process is beyond the scope of this paper, too.

In SLA many variants of pricing models could be thinkable. One provider can want to be paid for every invocation (on the per-node, per-service or per-operation basis), another one can ask to pay for utilization of its nodes. There can also be complementary conditions, e.g. in regard to differentiating of charges depending on guaranteed maximal response time, etc. Providers predefine often concrete classes of service, one of which must be selected while negotiating a service level agreement.

Formulae for calculating of invocations' counts of web service operations and additional utilization of providers' nodes are already given in previous subsections. Based on these values, the integrator can switch to the most appropriate class of service.

To determine the optimal value of the "maximal response time" parameter, a kind of process profiling should be performed as follows. The fraction of time that accounts for invocations of a synchronous web service operation $m$ in the overall execution time of processes $q_1...q_k$ can be estimated as $\delta[m] = \sum_{i=1}^{k}\frac{t^{ex}[m]\cdot\beta_m[q_i]}{\tau[q_i]}$ ($t^{ex}[m]$ from {1,2,5}).

In much the same way the integrator can estimate summary fractions of all operations of a web service or of all web services of a provider. A notably large value of this fraction can warn of possible bottleneck; in this case, selection of a service level with smaller value of the maximal response time could be advisable.

Similarly the integrator can determine whether it should decrease service level (and thus the payments) for an operation whose performance is not significant due to infrequent invocations.

# 4. Application and implementation aspects

## 4.1. Possible approaches to implement measurement modules and organize data transfer

As mentioned in the above sections, both the WS-BPEL engine on the integrator side and web service engines on providers' sides must be instrumented in order to perform necessary measurements and to send their data to the integrator's measurement evaluation and prediction engine as shown in Figure 1. Here we list thinkable ways to implement these extensions:

1. Direct patching of the engine's source code. There are some open-source WS-BPEL engines available [24, 25]). Many web service engines are open-source too.
2. Instrumentation of the engine using aspect-oriented programming (AOP) [26].
3. Adding a measurement aspect to an aspect-weaving WS-BPEL engine [27].
4. Use of other possible extensibility mechanisms of the engine (e.g. plug-ins).
5. Instrumentation of the web service engine with the Application Response Measurement toolkit [28].
6. SOAP handlers with measurement and data transfer functionalities either on the integrator side or on the provider side, or both.
7. HTTP/SOAP proxies with additional measurement and data transfer functionality.
8. SOAP attachments as transport mechanism for the data exchange [29]. This approach can be used primarily on third-party provider nodes.
9. TCP packet sniffer and analyzer. This can be the last chance to measure response times of web services if we are unable to change the middleware.
10. SNMP and DTrace – these technologies can be used to measure utilization and other internal parameters of remote nodes. SNMP (simple network management protocol) is a well-known approach of system and network administration and management [30], and DTrace is a system management framework developed by Sun Microsystems and first introduced in Solaris 10 and OpenSolaris operating environments [31].
11. Internal mechanisms of the execution environment. Management functionality is embedded in J2SE and .NET execution environments and can be used to obtain necessary data.

## 4.2. Other implementation-related thoughts

It can be advisable to combine process performance estimation with initial parsing and analysis of WS-BPEL documents – i.e. the measurement evaluation and prediction engine should be a part of the WS-BPEL engine. Variants 1–4 from subsection 4.1 can be suitable for that.

It this case it would be possible to examine beforehand whether a new process can worsen the performance of involved web service nodes and thus cause the risk that the integrator will be unable to keep QoS warranties given to its clients. If this anxiety proves true, the initialization request of this new process must be either quite declined or accepted with reduced service level.

# 5. Conclusions and further work

In this paper we have proposed a performance estimation model for executable WS-BPEL processes which is based on both the formal WS-BPEL semantics and operations research techniques. It can be used to predict utilization changes on the nodes hosting involved web services, to calculate expected durations of new processes and to optimize service level agreements between web service providers and the integrator. To the best of our knowledge, this is a first proposed model of that kind.

The version 1.1 of the WS-BPEL specification was published in 2003. The new version WS-BPEL 2.0 is currently (in June 2006) being worked out by the corresponding OASIS Technical Committee and available as a draft. Our mathematical model will need to be adjusted in order to consider the changes made in the final version of the specification after the latter will be released.

Furthermore, fault and compensation handlers are not taken into account in the current version of the model. Although they are not parts of normal behavior of the process, they should be incorporated into calculations to improve the model.

The next potential improvement would be more pragmatic reaction to possible performance degradation. Currently we consider any possible performance worsening ( $w[s] \neq \min$ ) to be not acceptable, but this is a very idealistic point of view. This issue is also concerned with aspects of service level negotiation between the integrator and its clients, which is not

reflected in this paper at all.

We plan also to prototypically implement the proposed integration infrastructure and to carry out experiments in order to examine the correctness and the adaptability of the proposed mathematical model in the practice.

## 6. References


[1] Eric Newcomer and Greg Lomow. Understanding SOA with Web Services. Addison-Wesley, 2005, 444 p.

[2] Business Process Execution Language for Web Services version 1.1. – http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[3] Daniel A. Menascé, Virgílio A. F. Almeida. Capacity Planning for Web Services: metrics, models, and methods. Prentice Hall, 2001, 608 p.

[4] Schmietendorf, A.; Dumke, R.: A Measurement Service for Monitoring the Quality Behavior of Web Services offered within the Internet, in Proc. of IWSM/MetriKon 2004, Shaker-Verlag, pp. 381–390.

[5] Schmietendorf, A.; Dumke, R.; Reitz, D.: SLA Management – Challenges in the Context of Web-Service-Based Infrastructures, in Proc. of the IEEE International Conference on Web Services (ICWS 2004), pp. 606–613, San Diego, USA, July 2004.

[6] Schmietendorf, A.; Dumke, R.; Stojanov, S.: Performance Aspects in Web Service-based Integration Solutions, in Thomas, N.: Proc. of the 21st UK Performance Engineering Workshop (UKPEW 2005), No. CS-TR-916, pp. 137–152, University of Newcastle upon Tyne/England, July 2005.

[7] Dmytro Rud. Methods for Quality Assessment of Web Service Offerings (Diploma thesis, in German). Otto von Guericke University of Magdeburg, 2005. http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/diplomarbeiten/rud.pdf

[8] Web Service Level Agreements Project. http://www.research.ibm.com/wsla/

[9] Web Services Offerings Infrastructure. http://elab.njit.edu/vladimir/WSOLpublications.html

[10] Cremona – Creation, Monitoring and Management of WS-Agreement. – http://awwebx04.alphaworks.ibm.com/ettk/demos/wstkdoc/cremona

[11] Daniel A. Menascé. Mapping service-level agreements in distributed applications. IEEE Internet Computing, September-October 2004, pp. 100–102.

[12] Vijay Machiraju, Akhil Sahai, Aad van Moorsel. Web Services Management Network: An Overlay Network for Federated Service Management (HPL-2002-234). – http://www.hpl.hp.com/techreports/ 2002/HPL-2002-234.pdf

[13] Web Service QoS Project – http://www.wsqos.net/

[14] UDDI eXtension (UX). – http://www.ntu.edu.sg/home5/PG04878518/UDDIExtension.html

[15] Carolyn McGregor. A method to extend BPEL4WS to enable business performance measurement. Technical Report No. CIT/15/2003, June 2003. http://www.cit.uws.edu.au/research/reports/paper/paper03/TR-CIT-15-2003.pdf

[16] Luciano Baresi, Sam Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In Proc. of 3rd International Conference of Service-Oriented Computing. – http://www.elet.polimi.it/uload/guinea/ articles/icsoc05_baresi_guinea.pdf

[17] G. Chafle, S. Chandra, V. Mann, M. G. Nanda. Decentralized Orchestration of Composite Web Services. In Proc. of WWW'2004. – http://www.www2004.org/proceedings/docs/2p134.pdf

[18] Chintan Patel, Kaustubh Supekar, Yugyung Lee. A QoS Oriented Framework for Adaptive Management of Web Service based Workflows. – http://www.sice.umkc.edu/~leeyu/Publications/book5.pdf

[19] Ulrike Greiner, Erhard Rahm. Quality-Oriented Handling of Exceptions in Web-Service-Based Cooperative Processes. – http://akea.iwi.unisg.ch/downloads/eai2004-paper1.pdf

[20] Rohit Aggarwal, Kunal Verma, John Miller, William Milnor. Constraint Driven Web Service Composition in METEOR-S. – http://lsdis.cs.uga.edu/lib/download/aggarwal_ieee_scc_2004.pdf

[21] Nadia Busi et al. Choreography and Orchestration: a synergic approach for system design. In Proc. of 3rd International Conference of Service-Oriented Computting. – http://www.cs.unibo.it/~cguidi/Publications/ICSOC05.pdf

[22] C. Ouyang et al. Formal semantics and analysis of control flow in WS-BPEL., Technical Report BPM-05-15, 2005. – http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2005/BPM-05-15.pdf

[23] Michael W. Carter, Camille C. Price. Operations research: a practical introduction. CRC Press, 2001.

[24] ActiveBPEL an open-source WS-BPEL engine. http://www.activebpel.org

[25] PXE, Process Execution Engine. http://sourceforge.net/projects/pxe

[26] Houspanossian, C.; Cilia, M. Extending an Open-Source BPEL Engine with Aspect-Oriented Programming. – http://www.dvs1.informatik.tu-darmstadt.de/publications/pdf/ASSE05.pdf

[27] Courbis, C.; Finkelstein, A. Towards an aspect weaving BPEL engine. In: Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), 22-26 March 2004, Lancaster, UK. – http://www.cs.uvic.ca/~ycoady/acp4is04/ACP4IS-2004-papers.pdf

[28] J.D. Turner, D.A. Bacigalupo, S.A. Jarvis, D.N. Dillenberger, G.R. Nudd. Application Response Measurement of Distributed Web Services. http://www.dcs.warwick.ac.uk/research/hpsg/documents/TurnerJ.TDLARMWS.pdf

[29] Brian Connolly. Client quality reporting for J2EE Web services. Use SOAP attachments to report client response times for Web services. JavaWorld Magazine, September 2003. – http://www.javaworld.com/ javaworld/jw-09-2003/jw-0919-reporting_p.html

[30] Simple Network Management Protocol, RFC 1157. http://www.ietf.org/rfc/rfc1157.txt

[31] Sun Dynamic Tracing Framework. http://www.sun.com/bigadmin/content/dtrace/