QoS-Driven Adaptation of BPEL Scenario Execution

Kareliotis Christos
Dept. of Informatics and Telecommunications, University of Athens, Greece
ckar@di.uoa.gr

Dr. Costas Vassilakis
Dept. of Computer Science and Technology, University of Peloponnese, Greece
costas@uop.gr

Efstathios Rouvas
Dept. of Informatics and Telecommunications, University of Athens, Greece
rouvas@di.uoa.gr

Dr. Panayiotis Georgiadis
Dept. of Informatics and Telecommunications, University of Athens, Greece
p.georgiadis@di.uoa.gr

Abstract

BPEL/WSBPEL is the predominant approach for combining individual web services into integrated business processes, allowing for the specification of their sequence, control flow and data exchanges. BPEL however does not include mechanisms for considering the invoked services' Quality of Service (QoS) parameters and thus BPEL scenarios can neither tailor their execution to the individual user's needs or adapt to the highly dynamic environment of the WEB, where new services may be deployed, old ones withdrawn or existing ones changing their QoS parameters. Moreover, infrastructure failures in the distributed environment of the web introduce an additional source of failures that must be considered in the context of QoS-aware service execution. In this work we propose a framework for addressing the issues identified above; the framework allows the users to specify the QoS parameters that they require and it undertakes the task of locating and invoking suitable services. Finally, the proposed framework intercepts and resolves faults occurring during service invocation, respecting the QoS restrictions specified by the consumer.

1. Introduction

The web service paradigm has been adopted by research community and industry alike, as a standard for application communication over the Internet, guarantying independence from execution platforms, programming language and implementation details [1]. However a number of challenges still lie ahead for fully covering the needs of both service providers and consumers: [2] identifies a number of open issues in the current SOA state-of-the-art, spanning across four major categories namely *service foundations*, *service composition*, *service management and monitoring* and *service design and development*. For service governance, in particular, [1] lists "service governance" as a major research challenge, stating that *the potential composition of services into business processes across organizational boundaries can function properly and efficiently only if the services are effectively governed for compliance with QoS and policy requirements. Services must meet the functional and QoS objectives within the context of the business unit and the enterprises within which they operate.*

In this context, development procedures as well as composition and execution mechanisms need to take into account the QoS dimension of web services in order to formulate successful business processes that will satisfy users' (business or individuals) expectations. Regarding service composition into business processes, the predominant approach used nowadays is the formulation of BPEL/WSBPEL scenarios [19], in which the BPEL designer specifies the business process logic; this includes invocation of selected web services, control flow constructs and data flow arrangements in the form of result gathering and parameter passing, while provisions for exception handling (e.g. service unavailability or business logic faults) also exist. BPEL scenarios, however, do not include facilities either for specifying QoS parameters for services, or for dynamically selecting the web service to be called at run-time, therefore the BPEL scenario designer must select the concrete service implementation to be invoked in the context of the business process while creating the scenario, by examining the QoS parameters of functionally-equivalent services. This alternative, however, is not viable since (a) the same BPEL scenario may be used by different users with diverging or even contradictory requirements and (b) even if the "best choice" is made at some time point there is no guarantee that this choice will continue to be optimal in the future. Moreover, in the presence of failures, it would be desirable for the system to be able to locate and use "second best" choices automatically, provided that they deliver the required functionality and satisfy QoS restrictions.

In this paper we present a framework that complements BPEL execution with facilities for (a) specifying an *execution policy*, which comprises of restrictions for QoS attributes and defining service ranking criteria in terms of QoS characteristics (b) choosing dynamically the "most suitable" service according to the given policy and (c) automating exception handling in the presence of system faults, using QoS-aware and policy-adhering exception management techniques. This framework allows for tailoring each individual execution of a BPEL scenario (BPEL runtime instance) to the needs of the invoking user (BPEL process consumer), while relieving at the same time the BPEL scenario designer of tasks related to QoS attribute inspection and BPEL script maintenance due to changes in the available services (introductions, withdrawals or changes of QoS attributes); finally, the need for exception handling in BPEL scripts is reduced to addressing business logic faults only, since system-related faults (server unavailability, network partitioning etc) are handled automatically. The proposed framework also addresses *service selection affinity*, i.e. cases where a service selection implies the binding of subsequent selections (e.g. selecting a hotel reservation from a travel agency dictates that the payment will be made to the same travel agency).

The rest of this paper is structured as follows: section 2 overviews related work; section 3 lists definitions and outlines the QoS aspects considered in this work. Section 4 presents the overall framework architecture and gives details on the functionality of its components, while section 5 presents performance metrics. Finally, section 6 concludes the paper and outlines future work.

2. Related work

A number of research works have insofar addressed various issues related to the QoS of web services in the context of composite services. [3] and [4] present AgFlow, which revises the execution plan in order to conform the user's QoS constraints. AgFlow may operate either using *global planning*, in which the execution plan is revised in order to conform the user's QoS constraints, or using local optimization, in which optimization is made on individual task basis, using the Simple Additive Weighting [5] technique to select the optimal service for a given task. [6] presents VieDAME, which performs BPEL scenario adaptation on the basis of QoS parameters, but these QoS parameters and the selection strategy are predetermined through pluggable modules; moreover, VieDAME does not support service selection affinity and is implemented using extensions available only in the ActiveBPEL engine [7], and is thus platform-dependent. [8] introduces end-user specified policies through QoSL4BP, and BPEL transformers that incorporate the policies and appropriate monitors to the BPEL scenario before its execution. This work mainly targets at monitoring the execution and raising exceptions when the desired QoS are not met, rather than adapting the BPEL scenario so as to best match the QoS demands of the scenario consumer. [9] introduces another BPEL extension and uses an extended BPEL engine to deliver QoS-based adaptation; the use however of a BPEL extension and a custom execution engine are potential barriers to the adaptation of this solution. [13]

[10], [11], [12] and [14] consider service BPEL scenario adaptation in the context of exception resolution. [10] creates exception-aware process schemas, and the infrastructure detects invocation faults and substitutes services that have failed with alternate ones; QoS characteristics are not considered in this work. [11] includes QoS characteristics in the alternate service replacement, it does not allow however the specification of the replacement policy by the process consumer; [14] states that QoS is taken into account in the process of determining equivalence. [13] uses autonomic computing concepts for providing execution plan formulation for business processes, taking into account QoS parameters, monitors dynamically QoS violations at runtime and provides instrumentation for the handling of these exceptions.

All approaches that dynamically determine the services that will be used either in a service composition or as replacement for failed services, typically employ some semantics-based registry to determine service capabilities and QoS attributes that are required for service composition and/or service equivalence, which is required for service substitution. METEOR-S [15] [16] is a suitable infrastructure for such service discovery activities, employing ontologies where service inputs, outputs and QoS aspects are described. Execution under the METEOR-S framework is also monitored to allow for updating of QoS attributes such as response time and failure rate. WSMO [17] may provide the foundations for modeling, storing and reasoning on the relevant web service functional and non-functional aspects.

Our contribution to the works above is as follows:

- it allows the BPEL scenario designer to specify the desired QoS parameters for each service. These parameters are specified through standard BPEL variables, thus the designer may examine scenario input parameters for setting them, tuning thus the adaptation of the particular BPEL scenario execution to the desires and needs of the scenario consumer.
- it does not require any modification to the BPEL syntax or semantics.
- it takes the execution flow specified by the designer as granted, and optimizes service selection within this flow, contrary to service composition approaches which define this flow dynamically. This is an important aspect in cases where execution flow is carefully crafted by the designer to reflect particularities of the business process, specialized exception handlers are used, etc.
- it considers *service selection affinity*, enabling the conducting of multi-operation transactions with providers.
- it incorporates exception handling as an integral part of the adaptation process, allowing for switching to the "next best" solution when the originally selected candidate is unavailable.
- it does not use pre-determined alternative paths, but

selects services dynamically from an suitable registry.

3. Definitions and QoS Aspects

As stated in section 1, the adaptation of the BPEL scenario execution according to a QoS-driven policy is performed by dynamically selecting at run-time the most suitable operation to be executed for each particular operation invocation. This selection should be made among services that are *equivalent*. Formally, we define two services s_1 amd s_2 to be equivalent iff (a) they support the same operations and (b) for each operation $op_{1,i}$ of service s_1, the respective operation $op_{2,i}$ of service s_2 are either *syntactically or semantically equivalent* [6] to $op_{1,i}$. Syntactic equivalence indicates that the interfaces of $op_{1,i}$ and $op_{2,i}$ match, while semantic equivalence indicates that $op_{1,i}$ and $op_{2,i}$ only have the same functionality, but expose it using different interfaces.

	QoS provider 1	QoS provider 2	value
Cost	10 €	11 €	1
Security	DES/3DES	128 bits	3
Performance	High throughput	99%	5
Response time	0.0001 ms	Real-time	5
Availability	High	> 95%	4

Listing 1. Mapping of QoS values

In order to enable the selection of the "most suitable" operation according to some QoS specification, the QoS attributes of the operations should be represented in an unambiguous and system-processable format, while additionally means for expressing QoS-related operation selection criteria should be afforded. For brevity, in the following we will consider only the QoS parameters *cost, security, performance, response time* and *availability,* adopting the definitions in [17]. Extension of the framework to include additional attributes is straightforward, thus we have no loss of generality. Note also that different sources of qualitative parameters (e.g. service providers, independent organizations performing benchmarks or service repositories making available qualitative dimensions such as those described in [1]) may even employ different measurement domains for the same qualitative attributes, as shown in. For each such source, mappings between the domains employed by the source and numeric values are employed. These mappings include . conversions from symbolic to numeric values, and issues stemming from different value ranges (e.g. [0, 10] vs. [0, 100]) (cf. Listing 1). The latter can be accommodated by applying normalization like the one proposed for the scaling phase described in [5] or in the normalization phase of [18]. For convenience reasons and without loss of generality, in this work we normalize all qualitative attributes' values in the range [0, 5]. Thus, each operation *op* in our framework is tagged with a quintuple ($cost_{op}$, sec_{op}, $perf_{op}$, $resp_{op}$, $avail_{op}$), reflecting the QoS aspects of the particular operation.

In our approach we consider three vectors that define the QoS criteria for process invocation; in other words we define a QoS specification as a triple (MAX, MIN, W), where MAX, MIN and W are *quality vectors* (defined below). The first and the second elements specify the upper and the lower bounds –respectively– for particular QoS aspects; effectively these two vectors comprise the QoS constraints. The third element, W, represents the qualitative attributes' corresponding weights, i.e. how important each qualitative attribute is considered by the designer in the context of the particular operation invocation. Higher weights (in absolute value) indicate higher importance of the specific qualitative attribute. According to the above, the quality vector for the QoS attributes considered in this work can be defined as:

$MAX = (cost_{max}, sec_{max}, perf_{max}, resp_{max}, avail_{max})$
$MIN = (cost_{min}, sec_{min}, perf_{min}, resp_{min}, avail_{min})$
$W = (cost_{w}, sec_{w}, perf_{w}, resp_{w}, avail_{w})$

Listing 2.Quality Vectors

Thus, if a BPEL scenario designer requires for some service invocation security of level ≥3 and cost ≤2, vectors MAX and MIN will be set to *MAX = (2, 0, 0, 0, 0)* and *MIN = (0, 3, 0, 0, 0)*. A value of zero in specific a position of the MIN and MAX indicates that no higher/lower bound is defined for the respective attribute. The same vectors can be equivalently and more compactly expressed (ommiting attributes for which no constraint is specified) as *Constraints=cost:0,2;sec:3,0*. Similarly, a weight vector can be defined as *W=cost:-3,sec:1,resp:2*. Specification of negative values for elements of W may be employed to designate that services having smaller values for the specific QoS attributes are preferred against those having higher values; cost and response time are examples of QoS attributes for which negative values will be used.

4. Framework architecture

In order to accommodate the required functionality, the proposed framework introduces two additional modules in a standard BPEL execution environment. The first module is a *middleware layer* named ASOB (Alternate Service Operation Binding), which undertakes the tasks of (a) dynamically selecting the operations best matching the QoS characteristics specified by the scenario designer, (b) appropriately transforming messages and results to tackle syntactical differences between services and (c) intercepting exceptions owing to system-level causes, such as server failures or network partitionings, and resolving them by invoking equivalent operations. The second module is a *preprocessor*, which transforms BPEL scenarios created by designers so as to (a) direct invocations to the middleware layer and (b) include in each invocation all the necessary information for selecting the operation best matching the QoS characteristics specified by the BPEL designer. The overall framework architecture is depicted in Figure 1, while the framework operation and module functionality, as well as the way QoS specifications are provided by the BPEL designer are described in the following paragraphs.

4.1. QoS specification in the BPEL scenario

The ASOB framework reserves two variables through which the BPEL designer may specify the desired QoS characteristics for web service invocations. These variables are named *ASOB_QoSconstraints*, and *ASOB_QoSweight*, and correspond to the *Constraints* and *W* vectors described in section 3. Before each *invoke* construct, the designer can insert an *assign* BPEL node to appropriately set the values of these variables to the desired values, as shown in Listing 3. The preprocessor will arrange for passing these values to the ASOB middleware, where the QoS specifications will be extracted and the scenario execution will be adapted accordingly.

4.2. BPEL scenario preprocessing and deployment

The BPEL scenario (SC) as crafted by the BPEL designer is processed by the *ASOB preprocessor*, which produces an *ASOB-aware BPEL scenario* ($\mathrm{SC}_{\mathrm{ASOB}}$) as output; $\mathrm{SC}_{\mathrm{ASOB}}$ is made available for invocation by BPEL scenario consumers. $\mathrm{SC}_{\mathrm{ASOB}}$ differs from SC in the following respects:

1. $\mathrm{SC}_{\mathrm{ASOB}}$ contains an additional *partnerLink* node, which corresponds to the ASOB middleware.
2. $\mathrm{SC}_{\mathrm{ASOB}}$ includes, as its first operation, an invocation to a special web service operation of the ASOB middleware, namely *getSessionId*. This operation creates a value that is unique for a particular execution of the BPEL scenario, and returns it to the invoking scenario. Uniqueness is guaranteed by combining the requester's IP address, the current timestamp of the system and a random number from a sparse domain. As the operation name suggests, this value will be used as a session identifier for the particular execution of the BPEL scenario, in order to implement *service selection affinity*.

```
<!-- set the value of the input parameter for the web service -->
<variable name="amount" type="xsd:integer">
<assign name="assign1"> <copy>
  <from><literal>34</literal></from>
  <to variable="amount"/>
</copy></assign>
<!-- set the QoS specification -->
<assign name="QOSassign1">
  <copy>
    <from><literal>cost:0,2;sec:3,0</literal></from>
    <to variable="ASOB_QoSconstraints"/>
  </copy>
  <copy>
    <from><literal>cost:-3,sec:1,resp:2</literal></from>
    <to variable="ASOB_QoSweight"/>
  </copy>
</assign>
<invoke   partnerLink="myLink"   portType="thePort"   operation="someOp"
    inputVariable="amount" outputVariable="theOutput"/>
```

Listing 3. Original BPEL scenario excerpt

3. Each *invoke* node (i.e. each operation invocation) within SC is transformed as follows: firstly, WSDL file to which the *partnerlink* refers to is located and copied locally, and the *soapaction* address element for the particular invocation is amended to point to the ASOB middleware; the corresponding WSDL import is adjusted accordingly to point to the local copy. Secondly, the type of the *inputVariable* of the particular invocation is extended to accommodate five additional elements, namely *sessionId, origPLink, origAddress*, *ASOB_qoscons* and *ASOB_qosw*. To achieve the extension of *inputVariable*, the preprocessor downloads and appropriately modifies the files in which the type is defined (WSDL files for variables of type *messageType;* xml schemas for *element*) and amends import declarations to point to the modified files. For simple types (*type*), where the parameter is

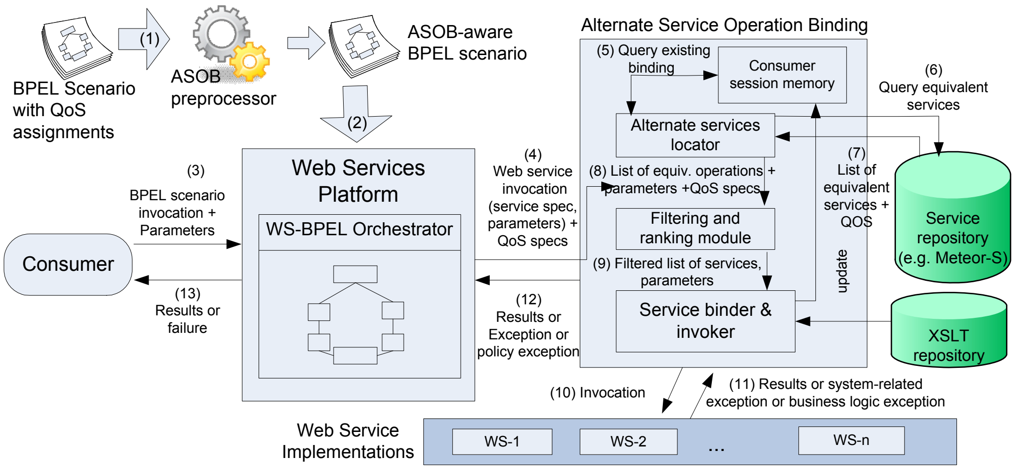
BPEL Scenario with QoS assignments
(1)
ASOB preprocessor
ASOB-aware BPEL scenario
(2)
(3) BPEL scenario invocation + Parameters
Consumer
(13) Results or failure
Web Services Platform
WS-BPEL Orchestrator
(4) Web service invocation (service spec, parameters) + QoS specs
(12) Results or Exception or policy exception
Alternate Service Operation Binding
(5) Query existing binding
Consumer session memory
(6) Query equivalent services
Alternate services locator
(7) List of equivalent services + QOS
(8) List of equiv. operations + parameters +QoS specs
Filtering and ranking module
(9) Filtered list of services, parameters
update
Service binder & invoker
Service repository (e.g. Meteor-S)
XSLT repository
(10) Invocation
(11) Results or system-related exception or business logic exception
Web Service Implementations
WS-1
WS-2
...
WS-n

Figure 1. Architecture of the ASOB framework

defined as a simple XML type (e.g. *string* or *integer*), the preprocessor creates an XML schema file defining a type containing the five aforementioned elements plus the element *ASOBvalue* of the appropriate type (*string, integer* etc), and amends the variable declaration to use the newly defined type; additionally, assignments (*copy* constructs) from/to this variable are modified accordingly. The transformed operation invocation is included in SC_{ASOB}.

Thirdly, the *invoke* node is preceded with *copy* commands assigning appropriate values to the aforementioned variables as follows:

- the value returned by the *getSessionId* operation invocation is copied to *sessionId.*
- the value of the *partnerlink* attribute in the original *invoke* construct is copied to *origPLink*
- the *soapaction* address in the original WSDL file is copied to *origAddress*.
- the values of *ASOB_QoSconstraints*, and *ASOB_QoSweight* are copied to *ASOB_qoscons* and *ASOB_qosw*, respectively.

All other constructs within SC (control flow, assignments, exception handlers etc) are left intact, and SC_{ASOB} can now be invoked by service consumers. The modifications acted upon the BPEL scenario of Listing 3 are shown in Listing 4 (changes are denoted using italics).

4.3. BPEL scenario execution

Once the preprocessed BPEL scenario has been deployed, it may be invoked by a consumer. At this stage, we assume that the Service Repository (SR) is populated with up-to-date web services specification entries (WSDL location, Endpoint addresses, Operation interfaces). In addition to this information, SR should *at least* provide (a) means for identifying functionally equivalent groups of services using semantic tagging (as, for example METEOR-S and WSMX) or any other suitable approach, and (b) information on the Quality of Service (QoS) characteristics of the operations provided by the services. Repositories such as [21] may be used to retrieve the needed QoS characteristics.

As described in the previous section, the first operation invocation will retrieve a session id from the ASOB middleware. Subsequent invocations are effectively directed to the ASOB middleware, which processes them as follows:

Step 1. It extracts from the SOAP message payload the values of the elements *sessionId, origPLink, origAddress*, *ASOB_qoscons* and *ASOB_qosw*, and constructs a new payload with these elements removed. The latter action results to a payload suitable for invoking the operation initially specified in the original BPEL scenario. Recall that *origAddress* here contains the address of the operation that was specified in the original BPEL scenario. This operation will be denoted as *op*.

```
<!-- type_amount.xsd is a preprocessor-generated file in which the type
type_amount is defined, having the parts sessionId, origPLink,
allPartnerOperations and ASOBvalue -->
<import location="type_amount.xsd" importType=
"http://www.w3.org/2001/XMLSchema" />

<!-- retrieve session id -->
<variable name="ASOBsessionId" type="xsd:string">
<invoke partnerLink="ASOB" portType="ASOBport" operation="getSessionId"
outputVariable="ASOBsessionId" />
<variable name="amount" element="type_amount">
<assign name="assign1"> <copy>
   <from><literal>34</literal></from>
   <to variable="$amount.parameters/ASOBvalue"/>
</copy></assign>
<!--assign "QOSassign1" is left intact and is not repeated here -->

<!-- plant extra fields in the input message -->
<assign name="ASOB_ASSIGN1">
   <copy> <from variable="ASOBsessionId"/>
      <to variable="$amount.parameters/sessionId"/> </copy>
   <copy> <from><literal>"myLink"</literal></from>
      <to variable="$amount.parameters/origPLink" /> </copy>
   <copy> <from><literal>"http://addr.com/path"</literal></from>
      <to variable="$amount.parameters/origAddress"> </copy>
   <copy> <from variable="ASOB_QoSconstraints" />
      <to variable="$amount.parameters/ASOB_qoscons" /> </copy>
   <copy> <from variable="ASOB_QoSconstraints" />
      <to variable="$amount.parameters/ASOB_qoscons" /> </copy>
   <copy> <from variable=" ASOB_QoSweight" />
      <to variable="$amount.parameters/ ASOB_qosw" /> </copy>
</assign>
<invoke partnerLink="ASOB" portType="ASOBport" operation="proxyInvoke"
      inputVariable="amount" outputVariable="theOutput"/>
```

Listing 4. Preprocessed BPEL scenario excerpt

Step 2. It queries the *consumer session memory* whether the particular *origPLink* has already been bound to some specific service in the context of the same session. If such a binding does exist, then the operation of the same service which is equivalent to *op* should be invoked now, to provide service selection affinity (e.g. if during a previous invocation the operation *reserveCar* of service *AgencyA* was selected, then when the operation *payCar* is invoked, the service *AgencyA* must be selected again); therefore, the appropriate operation is retrieved from the repository SR of Figure 1, and request processing continues in the filtering and ranking module (step 4, below), which will only try to invoke the equivalent operation from the selected service. If, however, no such binding is found in the consumer session memory, invocation handling continues within the alternate services locator module (step 3, below).

Step 3. The alternate services locator extracts from SR the operations *op'* that can be substituted for *op*. According to the equivalence definition in section 3, these are the services *op'* which are provided by a service equivalent to the one providing *op*, and are syntactically or semantically equivalent to *op*. These services are forwarded to the filtering and ranking module (step 4).

Step 4. the filtering and ranking module prunes from the list it receives (either from the alternate services locator or a singleton list from step (2), if a binding to some service has been made in the same session) those entries that do not satisfy the constraints specified in *ASOB_qoscons*. If no services remain in the list after the pruning step, a special type of exception *PolicyException* is returned to the BPEL engine, to signify that the specified QoS criteria cannot be met, and request processing terminates. The BPEL scenario designer may have included an appropriate exception handler which will attempt to remedy the fault (by relaxing some constraints, or by trying alternate methods e.g. "try to find a courier mail for under $10, and if this fails send by surface mail).

For each one of the qualifying services, the module completes an overall score, which is equal to

$$Sc_{op'} = \sum_{atttr \in \{\cos t, \sec, \ldots\}} attr_{op'} * ASOB\_qosw_{attr}$$

(effectively each QoS attribute of op' is multiplied by the respective weight specified in *ASOB_qosw* and the results are summed up). The services are then sorted in descending order of their overall scores, and execution continues in the service binder and invoker module.

Step 5. The service binder and invoker module extracts the first operation in the list it receives (recall that the operations with the highest overall scores are placed first in this list). For this operation, it first examines if the operation is *syntactically equivalent* to *op* (i.e. the one specified in the original BPEL scenario; this information is available in SR). If the operations are syntactically equivalent, the payload can be directly used for invoking the operation extracted from the list; if however the operations are only *semantically equivalent*, then the payload must be appropriately transformed to bridge the syntactic differences. This is performed using XSLT transforms, in a manner similar to that described in [6], according to which the XSLT files that are used for each pair of equivalent operations are pre-stored in the XSLT repository. Finally, the extracted operation is invoked.

Step 6. the service binder and invoker module intercepts the reply from the operation. If the operation was concluded successfully, the result can be sent back to the BPEL engine as a reply; before the reply is sent, it may be necessary to again apply an XSLT-based transformation to bridge any syntactic differences. In this case, the *consumer session memory* is updated to reflect the fact that the particular *partnerLink* has been bound to a specific service in the context of the current session. If, however an exception has been raised during the execution of the operation, the service binder and invoker module examines the root cause of the exception to determine whether the exception is owing to a system fault (server down, network partitioning etc) or to a business logic fault. In the latter case, no automated resolution is possible, and the exception is returned to the BPEL process (where it may be caught and handled by an exception handler specially crafted by the BPEL scenario designer). In the former case, however, it is possible to remedy the error by simply invoking some equivalent service: therefore, the service binder and invoker module extracts the next operation in the list (which is a suboptimal alternative compared to the previous one, yet a solution satisfying the QoS constraints), and iterates over steps (5) and (6). If the list is exhausted and no operation has concluded successfully, a *PolicyException* is returned to the BPEL engine. For more details on how it is system faults are distinguished from business logic faults, the interested reader is referred to [11].

5. Performance evaluation

In order to validate our approach in terms of performance, we conducted a benchmark experiment to quantify the overhead imposed by the middleware layer for performing the tasks described in subsection 4.3, i.e. (a) repository lookups, (b) operation filtering, (c) operation ranking and sorting, (d) XSLT-based transforms and (e) request interception and returning of answers. Note that, in the general case, the difference between the execution times observed when the ASOB middleware is (a) used and (b) not used, may be different from this overhead, since due to the adaptation, the response time of the operation finally selected may be different from the respective parameter of the service originally specified. Additionally, the introduction of the ASOB middleware offers functional aspects, including adaptation and exception handling, which cannot be quantified in terms of performance. This experiment aims at assuring that the approach described in the previous paragraphs is feasible.

For our experiment we used three distinct machines: the first one for executing BPEL scenarios, using the WS-BPEL 2.0 compliant JBI component [18], deployed on Glassfish v2 application server [19]; the second one was hosting the ASOB middleware (java application on Glassfish V2) and SR, implemented as a MySQL database. The third machine (with a configuration same as the second one) hosted the target web services. For benchmark data collection we used the benchmarking tool from Apache, *ab* [20] and internal profiling timers.

Figure 2 illustrates the ASOB internal process time for single web service operation invocations, against the overall service repository (SR) size and the number of equivalent services present in the repository. Figure 2 indicates that the internal process time (y-axis) mainly depends on the number of discovered equivalent services and not the overall SR size (x-axis). The very small dependency on the overall SR size can be attributed to the use of appropriate indexes within SR, which effectively exclude the non-relevant tuples from the search, thus only 1-10 extra disk page accesses are performed. The overhead increment, on the other hand, when the number of alternate services increases is considerable, mainly affecting the sorting of the candidate operation list (typically of complexity *O(n * log(n))* and the interprocess communication cost to transfer the results from SR to ASOB. It is expected that cases with more than 100-200 qualifying operations will be rare, and, as shown in Figure 2, such cases exhibit an overhead of 20-25 msec/invocation.

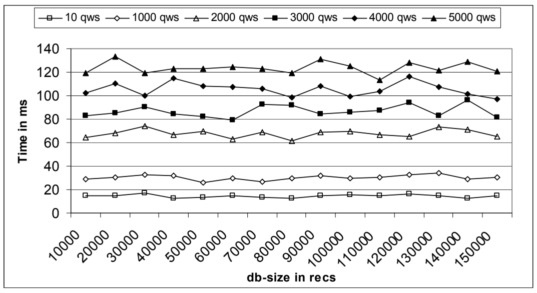

Figure 2. ASOB internal process time

concurrent ASOB invocations	20	40	60	80	100
time in msecs	17.8	18.5	34.5	46.2	61.7

Figure 3. XSLT transformation duration

Figure 3 shows the overhead incurred by applying XSLT transforms on request and response SOAP messages, to resolve syntactical differences between operations that are *semantically* but not *syntactically* equivalent. The time reported in this figure accounts for both the retrieval of the appropriate XSLT from the repository and the application of the transform. Note that such a transform is not always required (case of *syntactically equivalent* operations). As shown in the figure, when the number of concurrent connections increases, the time to execute each transform increases, since XSLT transforms are CPU-bounded and high concurrency means that less CPU share is available for each transform.

Figure 4 illustrates the number of operation invocations that can be served in a unit of time against the number of concurrent invocations when (a) services are directly invoked and (b) when invocations are made through the ASOB middleware. This diagram indicates that the introduction of the ASOB middleware is feasible, since it leads to a throughput drop of 8-16% (11%-19% when XSLT transforms are applied). At the point of 100 direct invocations the drop in performance is so sharp (the machine resources have been exhausted) that there is no point in considering more concurrent invocations. In the area between 80 and 100 concurrent invocations, the difference in throughput is gradually becoming smaller. This behavior can be attributed to the fact that while the web service execution machine has reached its limits regarding request processing at 80 direct invocations, there is still ample power available in the machine hosting the ASOB module to handle the processing required by the specific module.

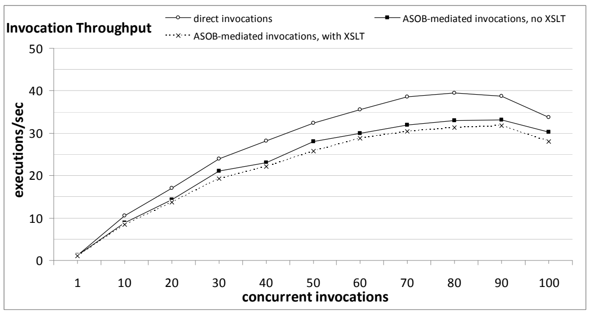

Figure 4. Invocation throughput

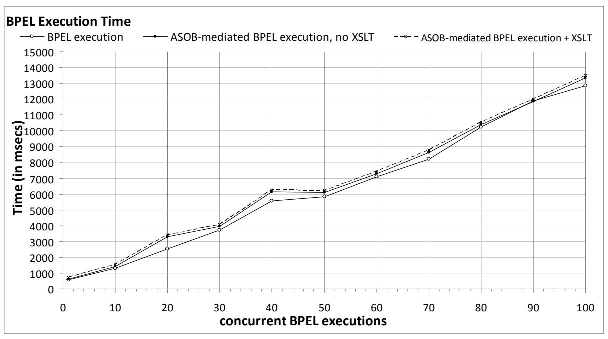

Figure 5. BPEL scenario execution time

Figure 5 illustrates the BPEL execution time of a BPEL scenario containing two web service invocations against the number of concurrent executions. The x-axis represents the concurrent BPEL process executions and y-axis the time elapsed until all of them completed successfully. It is obvious that the BPEL process time is slightly increased in the ASOB-mediated case, but the increment is very small (4%-9% without XSLT transformations, 8-16% with XSLT transformations).
Figure 6 depicts the BPEL scenario execution throughput against the number of concurrent executions. The behavior is consistent with the previous diagrams.

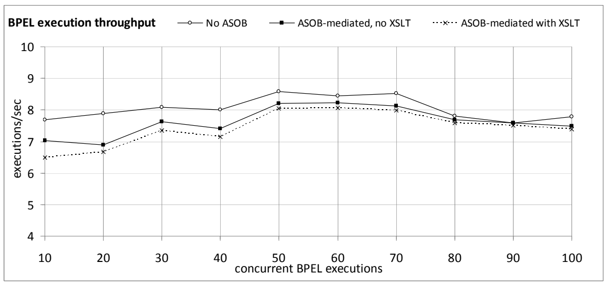

Figure 6. ASOB-mediated vs. direct invocation BPEL scenario execution throughput

6. Conclusions

Building processes that are able to cope with the dynamics of real world requirements has always been a challenging endeavor. The adoption of BPEL in the design and execution phases of business processes has already obtained gains in speed and reliability, but has not been able insofar to successfully address issues arising form the dynamic nature of the processes themselves, the diversity in user requirements and the inherent instability of distributed environments, which leads to a number of system faults.

The framework presented in this paper addresses these shortcomings employing a dynamic service selection mechanism based on QoS criteria for a BPEL process; these criteria are defined by the BPEL scenario designer and can be set to reflect the user requirements. Service attributes are stored in a repository that stores the services' functional and non-functional (qualitative) characteristics; updating the repository suffices to reflect changes in the real world (service introductions or withdrawals, changing of services' QoS aspects etc). An exception resolution mechanism for faults owing to systemic reasons is also included, easing thus the work of the BPEL designer.

Future work will focus on tighter integration with SR, providing to it updates based on the observed behavior of invoked services. Another aspect that will be investigated is the optimality of partner link bindings to specific services, when multiple operations from the same partner link are invoked: the current algorithm is *greedy*, seeking to obtain the best match for the *first* only operation invocation to any individual partner link taking place within a session, but this choice may be sub-optimal if subsequent invocations are considered. Providing ASOB with lookahead information on the services that will potentially be invoked in subsequent BPEL scenario execution steps, and the extension of the current service selection algorithm can contribute to performing more optimal partner link bindings. The removal of the need for scenario preprocessing will also be investigated.

7. References

[1] Newcomer, E., Lomow, G.: Understanding SOA with Web Services, Addison-Wesley, (2005)

[2] M. P. Papazoglou, P. Traverso, F. Leymann, Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer (40) 11, Nov. 2007, pp. 38-45.

[3] L. Zeng, B. Benatallah, A.H. Ngu, M. Dumas, J. Kalagnanam, H. Chang. QoS-aware middleware for web services composition. IEEE Trans. Softw. Eng., 30(5), 2004.

[4] L. Zeng, Dynamic Web Services Composition, PhD thesis, Univ. of New South Wales, 2003.

[5] H.C.-L and, K. Yoon, Multiple Criteria Decision Making, Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 1981.

[6] O. Moser, F. Rosenberg, S. Dustdar, Non-Intrusive Monitoring and Service Adaptation for WS-BPEL, WWW 2008, Beijing, China, pp. 815-824.

[7] Active Endpoints. ActiveBPEL Engine, 2007. http://www.active-endpoints.com/.

[8] F. Baligand, N. Rivierre, T. Ledoux. A Declarative Approach for QoS-Aware Web Service Compositions. B. Kramer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 422–428, 2007.

[9] H. Cao, H. Jin, S. Wu, L. Qi, ServiceFlow: QoS Based Service Composition in CGSP. Proceedings of IEEE EDOC'06.

[10] Liangzhao Zeng; Hui Lei; Jun-jang Jeng; Jen-Yao Chung; Benatallah, B. Policy-driven exception-management for composite Web services, E-Commerce Technology, Proceedings of CEC 05, 19-22 July 2005, pp. 355 – 363

[11] C. Kareliotis, C. Vassilakis, E. Rouvas, P. Georgiadis, Exception Resolution for BPEL Processes: a Middleware-based Framework and Performance Evaluation. Procs of iiWAS 2008, Linz, Austria.

[12] Liangzhao Zeng, Jun-Jan Jeng, Santhosh Kumaran and Jayant Kalagnanam, Reliable Execution Planning and Exception Handling for Business Process, LNCS, Springer, Technologies for E-Services, 2003. p.119-130

[13] A. E. Arpacı, A. B. Bener. Agent Based Dynamic Execution of BPEL documents. Proceedings of ISCIS 2005, LNCS 3733, pp. 332 – 341, 2005.

[14] C. Kareliotis, C. Vassilakis, P. Georgiadis, Enhancing BPEL scenarios with Dynamic Relevance-Based Exception Handling, Proceedings of the ICWS 2007, , pp.751-758.

[15] Verma, K., Sivashanmugam, K. , Sheth, A., Patil, A., Oundhakar, S. and Miller, J. METEOR–S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management, vol. 1(6), 2005, pp. 17-39.

[16] Cardoso, J. and A. Sheth. Semantic e-Workflow Composition. Journal of Intelligent Information Systems (JIIS). Vol. 21(3): pp. 191-225 (2003)

[17] J. O'Sullivan, D. Edmond, and A. Ter Hofstede: What is a Service?: Towards Accurate Description of Non-Functional Properties, Distributed and Parallel Databases, 12, 2002.

[18] JBI Team, 2008. Java Business Integration, https://open-esb.dev.java.net/Components.html

[19] Glassfish Team, 2008. Glassfish Open Source Application Server, https://glassfish.dev.java.net/

[20] Apache foundation, 2007. ab Apache Open Source benchmarking tool, http://httpd.apache.org/docs/2.0/programs/ab.html

[21] Al-Masri, E., 2008. The QWS Dataset, http://www.uoguelph.ca/~qmahmoud/qws/index.html