An integrated framework for QoS-based adaptation and exception resolution in WS-BPEL scenarios

Margaris Dionisis
Department of Informatics and Telecommunications, University of Athens, Panepistimioupoli, 15784, Athens, Greece
+302107275186
margaris@di.uoa.gr

Vassilakis Costas
Department of Computer Science and Technology, University of Peloponnese, Terma Karaiskaki, 22100, Tripoli, Greece
+30 2710372203
costas@uop.gr

Georgiadis Panagiotis
Department of Informatics and Telecommunications, University of Athens, Panepistimioupoli, 15784, Athens, Greece
+302107275235
p.georgiadis@di.uoa.gr

ABSTRACT
In this paper, we present a framework which incorporates runtime quality of service-based adaptation for BPEL scenarios, allowing for tailoring their execution to the diverse needs of individual users. The proposed framework also caters for automatically resolving system-level exceptions, such as machine outages or network partitionings, while both scenario execution adaptation and exception resolution maintain the transactional semantics that invocations to multiple services offered by the same provider may bear.

Categories and Subject Descriptors
H.3.5 [Information Systems]:Online Information Services – *Web-based services.* H.3.4 [Information Systems]: Systems and Software – *Distributed systems; Performance evaluation (efficiency and effectiveness).*

General Terms
Design, Performance.

Keywords
WS-BPEL, Adaptation, Exception resolution, Quality of Service.

1. INTRODUCTION
Web Services are considered as a dominant standard for distributed application communication over the internet. Consumer applications can remotely find and invoke complicated functionality, through established XLM-based protocols, in a technology agnostic manner. Towards this direction, Web Services Business Process Execution Language [1] (WS-BPEL) constitutes a powerful tool for composing individual web services into business processes, by composing WS-BPEL scripts. WS-BPEL clients can effectively define, in a single scenario, web service invocations, enriched with business process flow specification and data flow arrangements. WS-BPEL however does not include provisions to allow for (a) the client to specify the quality-of-service (QoS) requirements for a WS-BPEL scenario invocation and (b) for the WS-BPEL execution engine to adapt the execution of WS-BPEL scenarios, by dynamically selecting at run-time the web services best matching the client's QoS requirements and invoke them. Indeed, [2] lists governance for compliance with QoS and policy requirements as an open issue for the SOA architecture. This shortcoming has been identified by many researchers, and a number of approaches have been proposed to fill this gap. [3] classifies these categories into (i) those addressing adaptation at *horizontal level* where the adaptation involves mainly service selection that determines the binding of each task in the composite service to actual implementations, leaving unchanged the composition logic and (ii) those addressing adaptation at the vertical level, within which the composition logic can be altered.

Furthermore, as identified in [4], dynamic resolution of exceptions occurring in WS-BPEL scenario executions (as opposed to *static resolution* through fault handlers provided by WS-BPEL) is required to elevating the robustness and reliability of business processes and simplifying the maintenance of their specifications. This is especially true in environments supporting QoS-based adaptation, since the selection of an alternative service to be invoked as a replacement to the failed one should take into account the QoS specifications effective for the particular execution of the WS-BPEL scenario.

Finally, [5] identifies *service selection affinity* as an important requirement for maintaining the transactional semantics that invocations to operations offered by the same provider may bear. Service selection affinity refers to cases where a service selection in the context of adaptation 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).

In this paper we present a framework that extends BPEL execution with provisions for (a) specifying QoS requirements for invocations of web services within a WS-BPEL scenario (b) adapting the WS-BPEL scenario execution according to the QoS requirements while maintaining service selection affinity and (c) automatically resolving system-level exceptions in a QoS requirement-adhering manner (for a more detailed discussion on the distinction between system-level and business logic-level faults, the interested reader is referred to [6]). The system architecture is compliant with the SOA paradigm, while all additional information needed for adaptation purposes (i.e. the QoS specifications for individual web service invocations) are

SAC'13, March 18-22, 2013, Coimbra, Portugal.

expressed in standard WS-BPEL syntax. The proposed framework also considers both sequential and parallel execution structures of WS-BPEL (<sequence> and <flow> tags, respectively), allowing for successful adaptation of any WS-BPEL scenario. To the best of the authors' knowledge, no other work supports all the above listed features i.e. (i) QoS-based adaptation (ii) QoS-aware exception handling (iii) maintenance of service selection affinity and (iv) handling of both sequential and parallel execution structures. In this work, we adopt the horizontal adaptation approach, since (a) it respects the composition logic chosen by the designer and (b) we consider exception handling, and the horizontal adaptation enables the exploitation of exception handlers which may have been carefully crafted by the BPEL scenario designer, and integrated into the scenario.

The rest of the paper is structured as follows: In section 2 we overview related work, while in section 3 we briefly present the QoS aspects considered in this work. In section 4 we present the overall framework architecture and elaborate on the functionality of its components. In section 5 we present and discuss experiments conducted to assess the proposed framework's performance. Finally, Section 6 concludes the paper and outlines future work.

2. RELATED WORK

As stated in section 1, adaptation approaches presented insofar follow either the horizontal and vertical level adaptation [2]. AgFlow, introduced in [7] provides WS-BPEL clients with QoS constrains, by revising the execution plan either on a global or a local planning. In case of local planning, web service selection is done during execution time and conforms to a greedy strategy. On the other hand, in case of global planning, WS-BPEL scenario is partitioned in regions of collaboration and web service selection strategy is concentrated on a group of actions. Work in [8], introduces VieDAME, which adapts the execution of BPEL scenarios according to QoS parameters; however these parameters and the selection strategy are pre-determined through pluggable modules. Additionally, VieDAME does not support service selection affinity and is platform-dependent since it relies on extensions of the ActiveBPEL engine.

Work in [9] considers service selection in the presence of QoS constraints and aiming to minimize an objective function for the entire orchestration employing both brute force (OPTIM_S) and heuristic (OPTIM_HWEIGHT) algorithms. This work presents the service selection algorithms but does not propose an architecture on top of which the adaptation can be realized, while it additionally does not consider service selection affinity. The MOSES approach is introduced in [10] performs web service selection by means of formulating and solving a linear programming problem, considering by different patterns (par_or, par_and etc) and monitoring QoS execution during runtime. MOSES assumes that business processes are written as abstract compositions (contrary to our approach where business processes are specified through actual WS-BPEL scenarios, enabling the use of existing ones without any modification), while QoS requirements are stated through an SLA, giving the *average value* of QoS attributes, not allowing distinct QoS specifications per web service invocation.

Several works exist which not only optimize business processes, but also implement exception handling mechanisms in order to provide solutions in unpredicted run time environments. More specifically, framework in [11] 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. The work in [5] combines adaptation to QoS specifications and exception resolution (reverting to wherever possible to suboptimal plans, in the presence of exceptions), undertaking however a greedy strategy to service binding, which may lead to suboptimal solutions.

All composition and exception resolution approaches require some repository to describe (a) the functionalities of web services (this includes the concept of *service equivalence* [12], i.e. services implementing the same functionality) and (b) their QoS parameters. METEOR-S [13] 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 [14] may provide the foundations for modeling, storing and reasoning on the relevant web service functional and non-functional aspects. [15] presents WSMoD (Web Services MOdeling Design) for providing methodologies that address the design of Web services according to specific qualities of service (QoS) rather than functional descriptions only.

Our work extends, in terms of offered features, the work presented in [5] by adding support for parallel execution structures (<*flow*> tags) in BPEL scenarios. This extension is not trivial, since (i) since the architecture of [5] performs a *greedy-type* adaptation (intercepting and adapting individual service calls), while the optimization of parallel execution structures considered in this paper necessitates the knowledge of the overall structure of the BPEL scenario, hence the adaptation algorithm is substantially different and (ii) the aggregation functions of QoS dimensions employed for parallel executions are substantially different from those employed for sequential executions [16], dictating thus the use of different adaptation algorithms in order to achieve optimal adaptation. The difference between the optimization strategies for sequential and parallel execution structures is exemplified in section 3.

3. QOS ASPECTS AND DEFINITIONS

QoS may be defined in terms of attributes [17] [18], while typical attributes considered are price, response time, availability, reputation, security etc [19]. For conciseness purposes and without loss of generality, in this paper we will consider only the attributes *responseTime (rt), cost (c)* and *reliability (rel)*, adopting their definitions from [14]. In the proposed framework, the QoS specifications for a service within the BPEL scenario may include an upper bound and a lower bound for each QoS attribute, plus a *weight*, indicating how important each qualitative attribute is considered by the designer in the context of the particular operation invocation. Thus, the QoS specifications may be defined in the form of three vectors, namely $MAX = (rt_{max}, c_{max}, rel_{max})$, $MIN = (rt_{min}, c_{min}, rel_{min})$ and $W = (rt_w, c_w, rel_{min})$.

Weights may be negative to indicate that lower values are preferable to higher ones, as is expected for attributes such as cost and response time. For convenience reasons, we assume that all QoS attributes are normalized in the range [0, 10].

The QoS of services composed through sequential or parallel execution from constituent services s_1, …, s_n having QoS attributes equal to (rt_1, c_1, av_1), …, (rt_n, c_n, av_n) is given in the following table [20].

Table 1. QoS of composite services

	QoS attribute		
	responseTime	cost	reliability
Sequential composition	$\sum_{i=1}^{n} rt_i$	$\sum_{i=1}^{n} c_i$	$\prod_{i=1}^{n} av_i$
Parallel composition	$\max_i(rt_i)$	$\sum_{i=1}^{n} c_i$	$\prod_{i=1}^{n} av_i$

As we can see from table 1, the response time of a sequential composition is equal to the sum of its components' response time, while the response time of a parallel composition is equal to the maximum value. This difference is important in the adaptation process, since different search strategies should be employed to optimally adapt the scenario to the client's QoS specification. Consider for example the case of a BPEL scenario includes sequential invocations to A and B, which is invoked with the setting $W=(-1, -1, 0)$ for both service invocations. If the repository of available services were as listed in table 2, then the adaptation engine should select services (A_1, B_1), with this composition (scoring -17, a score higher that any other composition). In a parallel composition however, the adaptation engine should select either (A_1, B_2) or (A_2, B_2), since these provide an overall score of -13, as opposed to -16 of (A_1, B_1).

As noted in the introduction, the framework presented in this paper arranges for maintaining service selection affinity. Maintenance of service selection affinity guarantees that service invocations designated in the original BPEL scenario to be directed to the same service provider (e.g. a hotel room reservation and the related payment), are guaranteed to be also directed to the same provider through the adaptation procedure; the adaptation procedure may select a different same provider from the one designated in the original BPEL scenario, however the adaptation procedure guarantees that both the reservation and the payment invocations would be directed to the same service provider.

Table 2. Sample repository contents

Service	responseTime	cost	reliability
A_1	6	5	8
A_2	8	4	7
B_1	2	5	9
B_2	7	1	7

4. PROPOSED FRAMEWORK

The proposed framework extends the typical WS_BPEL execution scenario by accommodating two additional modules, namely the *BPEL scenario preprocessor* and an *adaptation layer*. The BPEL scenario preprocessor accepts as input the original BPEL scenario and produces as output a transformed BPEL scenario, which has been transformed to (i) transmit to the adaptation layer the QoS specifications and the necessary information regarding the scenario structure (i.e. sequential and parallel flows) to enable the adaptation of the scenario and (ii) redirect all service invocations to the adaptation layer, where they will be appropriately redirected to the most suitable service provider. The adaptation layer is deployed as a middleware, positioned between the BPEL execution engine and the web service providers and arranges for redirecting the web service invocations to the web service implementations best matching the client's QoS specifications, and intercepting and resolving system-level exceptions. The adaptation layer also offers two utility web services, the first one assigning session identifiers to BPEL scenario executions and the second one accepting the information regarding QoS specifications and the scenario structure. Figure 1 presents the overall architecture of the proposed framework; in the following section we will elaborate on (a) the specification of QoS parameters in the BPEL scenario (b) the operation of the preprocessor and (c) the operation of the adaptation layer.

4.1 Specifying QoS information in the scenario

The first step towards enabling the QoS-based adaptation is the specification of the required QoS for service invocations. In order to provide this feature in a WS-BPEL compliant fashion, , the proposed framework adopts the following conventions:

1. the designer should include in each *invoke* construct in the WS-BPEL scenario the optional attribute *name*, assigning distinct names to the *invoke* constructs.
2. for the *invoke* construct having name *invX*, the designer should use the WS-BPEL variables *QoSmax_invX*, *QoSmin_invX* and *QoSweight_invX*, which define the respective QoS specifications for the particular invocation. The BPEL designer may set the values for these variables after inspecting input parameters to the scenario (e.g. "choose=cheapest"), arranging thus for tailoring the QoS specification to the invoking user's preferences.

Figure 2 presents an excerpt of a BPEL scenario setting QoS specifications for an invocation (named *invoke1*). Since variable *QoSmin_invoke1* is not set, no lower bounds will be considered for the QoS attribute values in the process of adapting the particular web service invocation. Additionally, since the *QoSmax_invoke1* does not include a setting for the *reliability* QoS attribute, no upper bound for this attribute value will be considered.

4.2 Preprocessing the BPEL scenario

As shown in Figure 1, before the BPEL scenario is deployed on the web services platform, it is processed by the BPEL preprocessor, which creates an *enhanced BPEL scenario* as its output. The enhanced BPEL scenario differs from the original one in the following aspects:

1. it includes as its first operation an invocation to the web service *getSessionId* provided by the middleware; the result of the invocation is stored in a variable for later perusal.

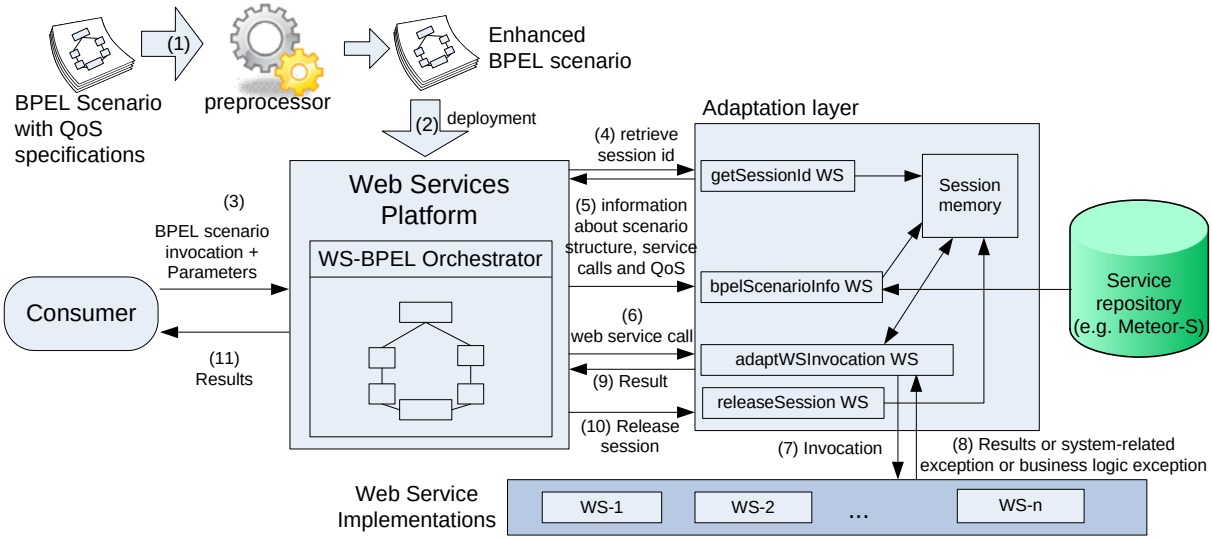

Figure 1. Proposed Framework Architecture

```
<assign>
  <copy>
    <from><literal>respTime:5;cost:3</literal></from>
    <to variable="QoSmax_invoke1"/>
  </copy>
  <copy>
    <from><literal>respTime:-1;cost:-2;reliability:1</literal></from>
    <to variable="QoSweight_invoke1"/>
  </copy>
</assign>
<invoke name="invoke1" partnerLink="lnk1" portType="port1"
operation="op1" inputVariable="input1" outputVariable="output2"/>
```

Figure 2. QoS specification in the BPEL scenario

2. it includes an invocation to the *bpelScenarioInfo* web service provided by the middleware, through which the BPEL scenario transfers to the adaptation middleware (a) the current session identifier (b) the values of all QoS-related parameters (*QoSmax_*, *QoSmin_* and *QoSweight_*) and (c) the information about the scenario structure. The latter effectively is represented as a simplified XML representation of the BPEL scenario including only the <*sequence*>, <*flow*> and <*invoke*> constructs of the original BPEL scenario; for the <*invoke*> constructs in particular, only the *name*, and *operation* attributes are transmitted, coupled with the service's endpoint address, extracted from the relevant WSDL file. This invocation is inserted before the first <*invoke*> construct of the original WS-BPEL scenario, to ascertain that the adaptation-related information have been transferred to the adaptation layer before the first invocation is intercepted and adapted.
3. The preprocessor arranges that each web service invocation is complemented with a header including the session identifier for the current scenario execution (the value returned by the getSessionId WS) and the value of the *name* attribute of the particular *invoke* construct. While header manipulation not a standard WS-BPEL feature, most contemporary BPEL engines provide means to set request headers, e.g. [21][22].
4. The BPEL scenario includes as its final operation an invocation to the *releaseSession* web service provided by the middleware.

The enhanced BPEL scenario produced by the preprocessor is then deployed to the web services platform and made available for execution.

4.3 Executing the BPEL scenario

When the BPEL scenario starts executing, it will retrieve the session identifier from the adaptation layer and subsequently will transfer to the adaptation layer all information described in the previous subsection. The adaptation layer at this stage proceeds to the creation of the current session's execution plan as follows:

1. for each web service invocation within the WS-BPEL scenario, its equivalent services are retrieved from the service repository; note that the information retrieved from the service repository includes the values for the equivalent services' QoS attributes. Only equivalent services that satisfy the QoS thresholds specified in the respective invocations' *QoSmax_* and *QoSmin_*are retrieved. If, for some service in the initial WS-BPEL scenario, no candidates satisfying the thresholds are found, then the adaptation layer returns a QoS_PolicyFault to the web services platform. The WS-BPEL designer may intercept the fault using the standard WS-BPEL mechanisms (<catch> construct) and attempt to resolve it, e.g. by relaxing the constraints and restarting the scenario, or simply notify the requesting client of the error condition.
2. the adaptation layer formulates all candidate execution plans for the particular execution of the BPEL scenario. Assuming that the scenario contains *N* invocations $\{inv_1, inv_2, \ldots, inv_N\}$ and that for each invocation inv_j there is a set of equivalent services $EQ_j=\{s_{j,1}, s_{j,2}, \ldots, s_{j,k}\}$, the maximal set of candidate execution plans is $EQ_j \times EQ_2 \times \ldots \times EQ_N$. This set is however pruned by removing elements that violate the service selection affinity principle, i.e. if invocations inv_i and inv_j are directed to the same service provider (i.e. services for which the *host* part of their endpoint address is identical) in the original

scenario, then all candidate execution plans in which replacements to inv_i and inv_j are not directed to the same service provider are removed from the candidate set. If the pruning step results in an empty candidate execution plan set, then a QoS_PolicyFault is returned to the web services platform.

3. for each execution plan within the candidate set formulated in step 2, an overall score is computed. The score computation procedure proceeds in a bottom-up fashion: initially, the score of individual invocations is computed using the formula $sc(inv_i) = rt_i * w_{rt,i} + c_i * w_{c,i} + av_i * w_{av,i}$, where rt_i, c_i and av_i are the QoS attribute values for the service replacing inv_i in the execution plan, while $w_{rt,i}$, $w_{c,i}$, $w_{av,i}$ are the weights specified in the QoS_weight_ variable for invocation inv_i. After all individual invocations' scores have been computed, the formulas in Table 1 are used to compute the overall score of the execution plan. Finally, the execution plan with the highest score is selected, and the correspondences between the original invocations and the services used in the selected execution plan are stored in the session memory (cf. Figure 1), coupled with the current session id. The correspondences are marked as *unbound*; this flag will be used for exception resolution (described below).

When an *invoke* construct is processed within the BPEL scenario, the outgoing request is redirected to the *adaptWSInvocation* web service provided by the adaptation layer; this can be accomplished either using a proxy setting in the web services platform or by using a transparent redirection router (both techniques are detailed in [4]). When the *adaptWSInvocation* intercepts a request, it processes it as follows:

1. it extracts from the request headers the session identifier and name of the *invoke* construct. Using these keys, it queries the *session memory* for the correspondence between the *invoke* construct and the actual service endpoint, selected in the execution plan formulation phase.

2. the request is forwarded to the endpoint retrieved in the previous step and the reply is received. If the reply is a normal response or a business logic-level fault (cf. [6]), then the reply is forwarded back to the web services platform. Additionally, the host part of the endpoint to which the invocation was made (denoted as h_{inv} in the following) is extracted, and the session memory is updated setting the correspondence between *invoke* construct names and endpoint address to *bound*, for *all invocations* to endpoints offered by h_{inv}. This update will prevent the exception resolution process (described below) from breaking the service selection affinity.

3. if the reply received from the invocation is a system-level fault (e.g. "host unreachable" or "connection refused"; for a full discussion the interested reader is referred to [6]), then the adaptation layer will try to resolve the fault by invoking a service equivalent to the failed one. Note however that such a resolution is possible only if no prior successful invocation was made in the same session to a service offered by h_{inv}. This restriction is applied to maintain session affinity, since if a prior invocation was made to host h_{inv} and the current invocation is directed to another host to resolve the system fault, then the service selection affinity will be broken. Taking the above into account, the adaptation layer first queries the session memory to determine if the current invocation has been marked as *bound* (recall from step 2 above that this will be performed if *any* prior invocation to services offered by h_{inv} has concluded successfully). If it has been marked as *bound*, then the fault cannot be automatically resolved and is thus returned to the web services platform. If, however, the current invocation is marked as *unbound*, then the adaptation layer first locates in the current execution plan all services s_1. s_2, …, s_k offered by host h_{inv} and then retrieves from the repository all k-tuples (s'$_1$. s'$_2$, …, s'$_k$) such that:

   a. s_i is equivalent to s'$_i$, ∀ i=1, 2, …, k; this condition guarantees the functional equivalence of the initial execution plan to the candidate exception resolution plan.

   b. s'$_i$ satisfies the QoS thresholds specified in the respective invocations' *QoSmax_* and *QoSmin_* variables, ∀ i=1, 2, …, k; this condition guarantees that the candidate exception resolution plan adheres to the restrictions specified by the client.

   c. all services s'$_i$ are offered by the same host, which must be different from h_{inv}; this condition guarantees that the candidate exception resolution plan maintains the service selection affinity and that the failed host will not be retried.

   Subsequently, for each k-tuple KT_j, an overall score is computed using the formula $sc(KT_j) = \sum_{i=1}^{k} rt_i * w_{rt,i} + c_i * w_{c,i} + av_i * w_{av,i}$ (rt_i, c_i and av_i denote the QoS attribute values for service s'$_i$ in KT_j and while $w_{rt,i}$, $w_{c,i}$, $w_{av,i}$ are the weights specified in the QoS_weight_ variable for the respective invocation). The k-tuple with the highest score is then chosen and all invocations in the execution plan to services offered by host h_{inv} are replaced by the corresponding invocations to services of the chosen k-tuple. Finally, the failed service invocation is restarted, being now directed to the newly chosen endpoint. If a system-level exception occurs at this point, the next-best k-tuple is selected, the execution plan is updated and the invocation is restarted again; this is be repeated until either a request succeeds or an administrator-defined limit is reached; in the latter case, the system-level exception is returned to the web services platform.

5. EXPERIMENTAL ANALYSIS

In order to assess the performance of our approach and validate our approach, we have conducted a set of experiments, aiming to measure and quantify the overhead incurred due to the introduction of the middleware. In these experiments we measured (a) the overhead imposed by the use of the invocations to the *getSessionId* and *bpelScenarioInfo* web services (invoked once per execution of a WS-BPEL scenario) (b) the overhead imposed for each web service invocation within the BPEL scenario and (c) the overhead imposed when the exception resolution mechanism is activated. The time taken by the preprocessor to transform the original WS-BPEL scenario into its enhanced form is not assessed, since preprocessing takes place in an offline fashion, not penalizing thus the production system performance. Finally, release session time has been found to be

negligible and metrics are not presented here due to space limitations. Moreover, the "release session" invocation may be implemented as an asynchronous web service call, having thus minimal impact on the WS-BPEL scenario execution time.

For our experiments we used two machines: the first one (a workstation equipped with one Pentium 4@2.8GHz CPU and 512MB of RAM) hosted the preprocessor and the clients, while the second one (a workstation equipped with one Pentium i7@1.6 GHz and 4 GBytes of RAM) hosted the BPEL execution engine (a Glassfish application server [23], the middleware and the target web services. The repository was implemented as an HSQLDB server, which was hosted on the second workstation ([24]). The repository was populated with synthetic data with an overall size of 2.000 web services. The machines were connected through a 100Mbps local area network.

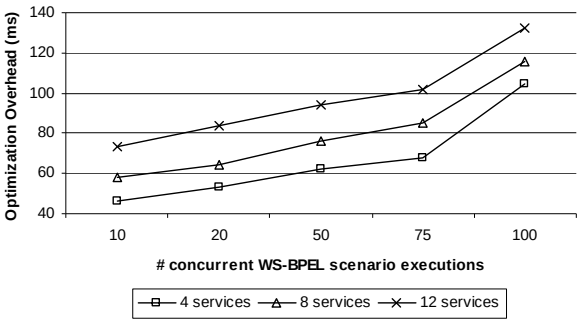

Figure 3. Optimization overhead

Figure 3 presents the optimization overhead (i.e. the overhead imposed by the use of the invocations to the *getSessionId* and *bpelScenarioInfo* web services) for varying number invocations present in the WS-BPEL scenario and different number of concurrent invocations (i.e. concurrent clients requesting the execution of the WS-BPEL scenario). The overhead increase has been found to be steeper when the number of concurrent invocations raises from 75 to 100 concurrent invocations; this is due to the depletion of the second workstation's resources at this load range; offloading specific tasks from that machine (e.g. hosting the adaptation layer and/or the target web services in a different machine than the WS-BPEL execution engine) is expected to provide smoother performance scaling.

Figure 4 presents the overhead incurred for the execution of a service invocation within the WS-BPEL scenario. This effectively accounts for (a) the two extra network messages required to transfer the request to the adaptation layer and return the reply from it and (b) the time taken to lookup in the session memory the correspondence between the particular service invocation and the endpoint determined in the optimization stage, and adjust the request message for forwarding to that endpoint. Even for high concurrency levels, the overhead for service execution is small (18.5 msec). Similarly to Figure 3, the overhead rises more steeply when the number of concurrent executions raises from 75 to 100, which is again due to the depletion of the second workstation's resources.

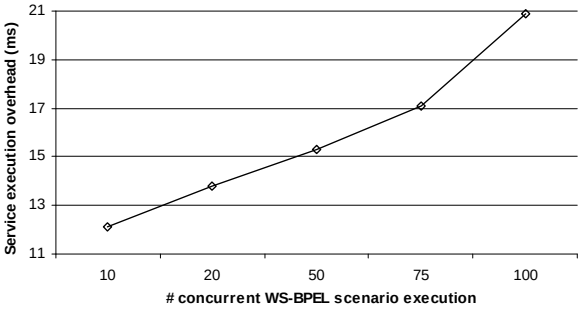

Figure 4. Service execution overhead

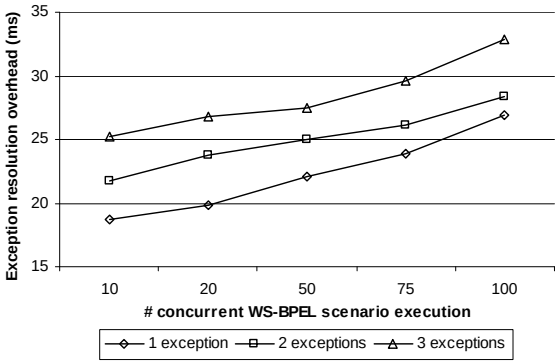

Figure 5. Exception resolution overhead

Finally, figure 5 presents the overhead incurred for resolving system level exceptions. This overhead accounts only for the time needed by the adaptation layer to perform the relevant tasks and does not include the time needed to invoke the failing services, since the latter varies significantly with the root cause of the failure (e.g. a fault owing to a network timeout leads to significantly higher delays than a fault owing to an invocation to a service that has been withdrawn), and therefore no meaningful statistics can be derived for the failing services' invocation times. Note also that the overheads illustrated in figure 5 refer to the resolution of an exception occurring in the invocation of a *single service*; the "1 exception" data series refers to the case of the exception being resolved by the first alternative service, while the data series "2 exceptions" refers to the case that the first alternative service fails and the second one succeeds (similarly for the data series "3 exceptions"). As described in subsection 4.3, only the first attempt to resolve an exception involves repository lookups and calculations of scores for alternative solutions, while subsequent attempts simply move to the "next best" solutions computed in the first attempt; this justifies the small time increments between the different data series in figure 5.

6. CONCLUSION AND FUTURE WORK

In this paper we have presented a framework enabling the WS-BPEL designers to specify the QoS requirements for the service invocations included in the WS-BPEL scenarios and the subsequent adaptation of the scenarios' execution to these specifications. The proposed framework also supports the automatic resolution of system-level exceptions, while it also caters for the maintenance of service selection affinity,

ascertaining that transactional semantics of service invocations are preserved. The proposed framework includes a scenario preprocessing step, before the scenario is deployed and made available for invocations, and an *adaptation layer*, which undertakes the tasks of optimizing the execution plan for the WS-BPEL scenario, adapting the execution and resolving exceptions.

The proposed framework has been experimentally evaluated to assess its performance. The overheads for the various phases have been quantified to be reasonable, while its performance scales acceptably with the number of concurrent BPEL scenario executions.

Our future work will focus on the integration of QoS-adherence monitoring mechanisms, such as those described in [8] and the handling of loops and conditional execution constructs in WS-BPEL scenarios; for the latter task, statistical information from prior scenario executions are foreseen as information sources to the adaptation process. Finally, integration of service selection affinity and exception resolution in vertical adaptation strategies will be considered.

7. REFERENCES

[1] OASIS WSBPEL TC. WS-BPEL 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

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

[3] Cardellini V, Di Valerio V, Grassi V, Iannucci S, Lo Presti, F. A Performance Comparison of QoS-Driven Service Selection Approaches. *In Proceedings of ServiceWave 2011*, Abramowicz W et al. (Eds.): LNCS 6994, pp. 167–178, 2011.

[4] Kareliotis, C., Vassilakis, C., Rouvas, E., Georgiadis P.: IQoS-aware exception resolution for BPEL processes: a middleware-based framework and performance evaluation. *IJWGS* 5(3): 284-320, 2009

[5] Kareliotis, C., Vassilakis, C., Rouvas, S., Georgiadis, P.: QoS-Driven Adaptation of BPEL Scenario Execution. *In Proceedings of ICWS 2009*: 271-278

[6] Kareliotis, C., Vassilakis, C. and Georgiadis, P. (2007) Enhancing BPEL scenarios with dynamic relevance-based exception handling. *In Proceedings of ICWS07*, Salt Lake City, Utah, USA, 9–13 July, pp.751–758, 2007.

[7] Zeng, L.B, Benatallah, A.H.N, Dumas, M., Kalagnanam, J., Chang, H. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5), 2004.

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

[9] Xia, Y., Chen, P., Bao, L., Wang, M., Yang, J. A QoS-Aware Web Service Selection Algorithm Based on Clustering. *In Proceedings of ICWS11*, 2011.

[10] Cardellini, V., Iannucci, S. Designing a Broker for QoS-driven Runtime Adaptation of SOA Applications. *In Proceedings of ICWS10*, Miami, Florida, USA, pp. 504–511, 2010.

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

[12] Rinderle-Ma, S., Reichert, M., Jurisch, M. Equivalence of Web Services in Process-Aware Service Compositions. *In Proceedings of ICWS'09*, 6-10 July 2009, Los Angeles, CA, USA, 2009.

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

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

[15] Comerio, M., De Paoli, F., Grega, S., Maurino A., Batini, C. WSMoD: A Methodology for QoS-Based Web Services Design. *Web services research*, Volume 4, Issue 2, 2007

[16] Jaeger, M.C., Rojec-Goldmann, G., Muehl, G. QoS Aggregation for Web Service Composition using Workflow Patterns. *In Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference*, 2004.

[17] ISO. UNI EN ISO 8402 (Part of the ISO 9000 2002): Quality Vocabulary, 2002.

[18] ITU. Recommendation E.800 Quality of service and dependability vocabulary.

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

[20] Canfora, G., Di Penta, M., Esposito, R., Villani, M.L. An Approach for QoS-aware Service Composition based on Genetic Algorithms. *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pp. 1069-1075, 2005.

[21] Apache ODE, Headers Handling. http://ode.apache.org/headers-handling.html

[22] Oracle. Manipulating SOAP Headers in BPEL. http://docs.oracle.com/cd/E14571_01/integration.1111/e10224/bp_manipdoc.htm#CIHFCBAD

[23] GlassFish Community. http://glassfish.java.net/

[24] HSQLDB. http://hsqldb.org/