On Replacement Service Selection in WS-BPEL Scenario Adaptation

Dionisis Margaris and Panagiotis Georgiadis
Department of Informatics and Telecommunications
University of Athens
Athens, Greece
margaris@di.uoa.gr, p.georgiadis@di.uoa.gr

Costas Vassilakis
Department of Informatics and Telecommunications
University of the Peloponnese
Tripoli, Greece
costas@uop.gr

***Abstract*—WS-BPEL scenario execution adaptation has been proposed by numerous researchers as a response to the need of users to tailor the WS-BPEL scenario execution to their individual preferences; these preferences are typically expressed through Quality of Service (QoS) policies, which the adaptation mechanism considers in order to select the services that will ultimately be invoked to realize the desired business process. In this paper, we consider a number of issues related to WS-BPEL scenario adaptation, aiming to enhance adaptation quality and improve the QoS offered to end users. More specifically, with the goal of broadening the service selection pool we (a) discuss the identification of potential services that can be used to realize a functionality used in the WS-BPEL scenario and (b) elaborate on transactional semantics that invocations to multiple services offered by the same provider may bear. We also describe and validate an architecture for realizing the proposed enhancements.**

Keywords— WS-BPEL; adaptation; transactional semantics; equivalent web services; quality of service; evaluation

I. INTRODUCTION

Web Services are the dominant standard for building distributed applications. Service providers make available functionalities that can be invoked through well-defined XML-based protocols, and consumer applications may locate and invoke them without any concern about implementation details or technological decisions on the side of the service provider. Individual services may be orchestrated to formulate higher level business processes using the Web Services Business Process Execution Language (WS-BPEL) [1].

WS-BPEL is an XML-based language and programs written in it (*scripts* or *scenarios*) are deployed in WS-BPEL execution engines, and made thus available for invocation. WS-BPEL has been designed to model business processes that are stable, and thus involve the invocation of web services that are known at the time the script is written. Hence, the WS-BPEL programmer specifies the exact services to be invoked for the realization of the business process. This arrangement however is not adequate in the current web: the functionalities needed within the scenario (e.g. booking a hotel room) are typically offered by numerous providers (different hotels and travel agencies), and distinct providers offer this functionality under different quality of service (QoS) parameters. Therefore, it would be desirable for consumers to be able to tailor the execution of the WS-BPEL scenario with respect to their QoS requirements; according to [2], governance for compliance with QoS and policy requirements is an open issue for the SOA architecture.

To tackle this shortcoming, numerous approaches have been proposed, following two main strategies [3]: (i) *horizontal adaptation*, where the composition logic remains intact and the main adaptation task is to select the service best matching the client's QoS requirements; the selected services are substituted for either *abstract tasks* (e.g. [3]) or *concrete service invocations* (e.g. [5]) and (ii) *vertical adaptation*, where the composition logic may be modified. Under both strategies, the adaptation mechanism, for each service used within the WS-BPEL scenario, needs to identify all services delivering equivalent functionality, so as to choose among them the service best matching the client's requirements. Typically in this process, adaptation proposals adopt the service definition presented in [12][19], according to which "a web service has been described as an aggregation of functionality published for use", thus for two services to be considered equivalent they must offer the same set of functionalities. [6] further distinguishes among *syntactical equivalence* (the interfaces of the original and the alternative services match) and *semantic equivalence* (services only have the same functionality but expose it using different interfaces); in the latter case, when the adaptation mechanism substitutes a service A with another service B, a syntactical transformation is applied to align the schema of the substitute service's request payload and reply to the ones of the substituted one [6][20]. In this paper, we argue that this equivalence definition is over-restrictive and propose the concept of *service replacement candidate* (SRC) to broaden the pool of services among which the substitute service can be selected and thus providing the opportunity to formulate better adaptations.

Another issue that must be addressed within the adaptation process is the handling of service selection affinity, identified in [5] as an important requirement for maintaining the transactional semantics that invocations to operations offered by the same provider may bear. Service selection affinity (SSA) 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). Insofar, published works either:

(a) do not support SSA (e.g. [8]), permitting thus for these invocations to be directed to different providers and running the risk to break the transactional semantics, or

(b) assume that all invocations to services offered by the same provider bear transactional semantics (e.g. [5][9]); however this may place unnecessary restrictions in cases when SSA is not required (e.g. when a scenario invokes two services offered by provider *P* to fetch football and basketball results, where no transactional semantics are involved), limiting thus the choices available to the adaptation mechanism and potentially leading to suboptimal adaptations.

In this paper we introduce additional notations, allowing the scenario designer to designate the cases that SSA is required, and present the associated actions taken by the adaptation environment to guarantee the required SSA while formulating optimal adaptations.

Finally, we describe a middleware-based architecture and associated tools, which enable the realization of the tasks presented above and validate the applicability of the proposed approach through experiments, concerning both the overhead introduced by the adaptation mechanism and the QoS of the formulated adaptations. Our overall approach follows the *horizontal adaptation* paradigm, which, as noted in [5], preserves the execution flow crafted by the designer to reflect particularities of the business process, while it also allows the exploitation of specialized exception handlers.

The contribution of this work is as follows: (a) the introduction of the SRC concept; (b) the provision of mechanisms to designate the cases where SSA should be applied to maintain transactional semantics; (c) the design of an algorithm to perform QoS-based adaptation taking into account the introduced enhancements and (d) the description, implementation and validation of a framework for realizing these features.

The rest of this paper is structured as follows: section II overviews related work, while section III presents fundamental concepts regarding the QoS and introduces the SRC concept, the specification of cases where SSA should be preserved. Section IV presents the adaptation algorithm and outlines the architecture for realizing the adaptation, while section V presents a performance, as well as qualitative evaluation study, aiming to substantiate the feasibility of the proposed approach. Finally, the last section concludes the paper and outlines future work.

II. Related Work

Insofar, WS-BPEL scenario adaptation has received considerable research attention, and existing adaptation approaches follow either the *horizontal* or the *vertical* adaptation approach [3]. VieDAME [6] adapts WS-BPEL scenario execution taking into account QoS parameters; both QoS parameters and the selection strategy are coded into pluggable modules which are attached to the WS-BPEL orchestrator. VieDAME monitors the execution of WS-BPEL scenarios and arranges for dynamic replacement of web services that fail to meet the desired QoS levels. Another use of monitoring in QoS-based adaptation can be found in [29], where the adaptation engine monitors the QoS-related performance of services to predict future performance of these services, and uses these predictions to formulate more accurate adaptations. Work in [7] examines service selection in the presence of QoS constraints and aims to minimize an objective function for the entire orchestration employing either brute force (*OPTIM_S*) or heuristic (*OPTIM_HWEIGHT*) algorithms.

[3] introduces MOSES, a methodology and a software tool implementing it to support QoS-driven adaptation of a service-oriented system. MOSES performs adaptation actions, whose goal is to determine at runtime the most suitable implementation to be bound to each abstract task S_i, selecting it from a set of available implementations; the selection among the available implementations is performed by means of formulating and solving a linear programming problem.

AgFlow [4] also performs QoS-based adaptation undertaking the middleware approach. AgFlow may operate using *global planning*, in which case it considers QoS constraints and preferences assigned to a composite service as a whole rather than to individual tasks, and uses integer programming to compute optimal plans for composite service executions; alternatively to global planning *local optimization* can be employed, which performs optimal service selection for each individual task in a composite service without considering QoS constraints spanning multiple tasks and without necessarily leading to optimal overall QoS. Additionally, AgFlow implements execution replanning in order to address issues such as services becoming unavailable or changing their predicted QoS. The work in [32] also addresses horizontal composition of services to form business transactions; this work views the determination of the optimal composition as a constraint optimization problem, allowing also for human user intervention to enhance the solving process.

[5] performs QoS-based adaptation and exception resolution. The ASOB middleware introduced in this work intercepts service invocation failures and distinguishes business logic faults from system faults; the latter category is handled by the middleware by falling back to "next best" solutions, while the resolution of exceptions falling into the first category is left to the scenario designer. [5] arranges for maintaining SSA, assuming that all requests made to the same service provider are interdependent and bear transactional semantics. [9] extends the work in [5] by including support for parallel execution structures.

Recent works, e.g. [31] tackle the issue of performance when composing (or adapting) large composition structures. [31] suggests that large-scale composition structures are decomposed into small-scale composition segments through mixed integer programming, and subsequently a QoS-optimal composite solution is found for each small-scale composition segment, leading to reduced time for solving the composition problem.

III. QoS & Adaptation Concepts

A. QoS Concepts

QoS is generally defined in terms of attributes corresponding to non-functional aspects of services [14] with typical attributes being response time, availability, price, reputation, security and so forth [15]. Without loss of generality, we will limit our discussion to attributes *response time* (rt), *availability* (av) and *cost* (c), adopting their definitions from [12]; the extension of the proposed algorithm and framework to include more QoS attributes is straightforward.

Taking the above into account, each functionality implementation (realized as a *service operation*) considered in the adaptation process has a known QoS vector $QoS_S=(rt_s, av_s, c_s)$ which is recorded in an appropriate repository (e.g.

METEOR-S [22]). The same repository should also provide information regarding which operations are equivalent.

Within a WS-BPEL scenario, individual functionalities are composed into sequential or parallel flows to implement the business process. Based on the QoS parameters of the individual functionalities invoked within the compositions, it is possible to compute the QoS value of the composition using the formulas shown in Table I [10]. As we can see from Table I, 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.

TABLE I. QOS OF COMPOSITE SERVICES

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

In the context of adaptation, selection of the concrete service that will realize some functionality is typically driven by specifications of the upper and lower bounds for QoS attributes. QoS bounds may either be defined as *global constraints* (i.e. express the desired values for the whole WS-BPEL scenario) or as *local constraints* (each such constraint expresses the desired values for a particular service invocation) [10]. When adaptation problems need to address global constraints performance is poor [24], therefore either local constraints are directly used (e.g. [8][9]) or methods for mapping global constraints to local constraints are employed. Complementary to the QoS bounds, a *weight* is assigned to each QoS attribute, indicating how important each QoS attribute is considered by the designer in the context of the particular business process modeled by the scenario. Weights always apply to the whole composition, rather than to individual services, since they reflect the perceived importance of each QoS attribute dimension on the process as a whole, and not its constituent parts [23].

In the proposed framework, the QoS specifications for a service within the WS-BPEL scenario may include an upper bound and a lower bound for each QoS attribute, i.e. for service s_j included in a WS-BPEL scenario, the designer formulates two vectors $MIN_j(min_{rt,j}, min_{av,j}, min_{c,j})$ and $MAX_j(max_{rt,j}, max_{av,j}, max_{c,j})$. Additionally the designer formulates a weight vector $W = (rt_w, av_w, c_w)$, indicating how important each QoS attribute is considered by the designer in the context of the particular operation invocation. The values of the QoS attributes are assumed to be expressed in a "larger values are better" setup, e.g. a service having *cost = 6* is *cheaper* than a service having *cost = 4*.

B. Service Replacement Candidates

As noted in the introduction, in order to formulate the pool of candidates for realizing some specific functionality in the context of WS-BPEL scenario adaptation, adaptation proposals employ the concept of service equivalence, according to which:

1. a web service has been described as an aggregation of functionality published for use [12][19]. Individual functionalities aggregated to form a web service are termed *operations* in the context of SOA [23].
2. two services S_1 and S_2 are equivalent if and only if (a) they support the same operations and (b) for each operation $op_{1,i} \in S_1$, the corresponding operation $op_{2,i} \in S_2$ is either syntactically or semantically equivalent to $op_{1,i}$ [6].

Syntactic equivalence indicates that $op_{1,i}$ and $op_{2,i}$ have the same functionality *and* their interfaces match operation match (e.g. when multiple instances of the same service are hosted on different machines to provide increased reliability), while semantic equivalence indicates that $op_{1,i}$ and $op_{2,i}$ only have the same functionality, but expose it using different interfaces. In this work, we consider that differences in the interfaces can be bridged by applying XSLT transformations [6][20].

This equivalence definition can be applied to perform WS-BPEL scenario adaptation, since each service S can be replaced by any service S' equivalent to it; however, it may prove to be over-restrictive and thus limit the potential of the adaptation algorithm to create optimal adaptations.

Consider for instance the case of two hotel booking services HB_1 and HB_2, which offer functionalities (operations) as follows:

- HB_1 offers the operations *RoomBooking*, *AirportShuttle* and *CarRental*.
- HB_2 offers only the operations *RoomBooking*, and *AirportShuttle*.

According to the equivalence definition presented above, it is clear that HB_1 and HB_2 are not equivalent, since they do not support the same operations, and therefore we cannot substitute HB_2 for HB_1 in the general case. However, in the context of adaptation performed on the execution of a WS-BPEL scenario *HotelReservation* containing only invocations to the *RoomBooking*, and *AirportShuttle* services, it is clear that HB_2 can be substituted for HB_1 since all functionalities required by the adapted WS-BPEL scenario are offered by HB_2; the inverse substitution (HB_1 for HB_2) is also possible.

In order to formalize and generalize the example given above, we introduce the concept of *service replacement candidate* (SRC) as follows:

A service S' is a *service replacement candidate* for service S *in the context of a WS-BPEL scenario BS* if and only if for each operation $op_{s,i}$ of service *S* invoked within BS, S' provides the same functionality through operation $op_{s',j}$ and $op_{s,i}$ is either syntactically or semantically equivalent to $op_{s',j}$.

Under this definition, in the example presented above service HB_2 is a SRC for service HB_1, and can thus be substituted for it.

C. Service Selection Affinity

A WS-BPEL scenario may involve multiple invocations to operations provided by the same service provider (the technical term for a service provider in WS-BPEL is *partner link*). These operations may logically depend on one another, in the sense that if –in the context of a particular scenario execution– the first service is obtained from a particular service provider then the second service must be obtained from the same service provider. Consider for instance a hotel reservation service *ReserveRoom* that provides the operations *checkAvailability, getQuote* and *performReservationAndPay*: it is obvious that if the availability of a specific hotel is checked (by invoking the *checkAvailability*

operation it offers), then the quote must be received from the *same* hotel (or we may be asking for quotes for rooms that are not available) and the reservation/payment must also be made for the same hotel (or we may be booking rooms that are not available and/or are priced differently than we have seen).

When a WS-BPEL scenario is crafted, the scenario designer specifies a concrete *partner link* for each service provider s/he uses, and subsequently binds each operation invocation to a specific service provider (by listing the partner link name in the *invoke* construct). When the scenario is executed within a non-adaptive environment, all operations are directed to the service provider chosen by the designer; therefore if two operations (e.g. *checkAvailability* and *getQuote*) need to be obtained from the same service provider, it suffices that the WS-BPEL scenario uses the same *partner link* for invoking the operations.

In environments supporting execution adaptation for matching the QoS requirements of the client, the adaptation process examines the QoS specifications designated for each operation invocation and the QoS characteristics of the available operations, and directs the invocation to the service provider best matching the QoS specifications; therefore, the invocation may be actually directed to different service providers than the ones originally specified in the WS-BPEL scenario. If such an adaptation is however performed in an uncontrolled fashion, then it is possible that operation invocations originally set to be served by the same service provider are directed to different service providers, breaking thus the transactional semantics.

Consider for example the case of adapting the above WS-BPEL scenario, which has been crafted to invoke the *checkAvailability* and *getQuote* operations offered by HotelA. Let us assume that the WS-BPEL scenario consumer has designated response time as the sole criterion for QoS-based adaptation, and that *checkAvailability* of HotelA and *getQuote* of HotelB are the operations with the smallest response time (n.b. HotelB is considered as equivalent to HotelA, therefore it is possible for the adaptation process to substitute an invocation to an operation of HotelA with an invocation to the respective operation of HotelB). If the adaptation is performed in an uncontrolled fashion, then the adaptation mechanism will direct the invocation of *checkAvailability* to HotelA, while *getQuote* will be directed to HotelB. Hence, the adaptation environment must include provisions, which will guarantee that invocations with such dependencies -which will be referred to with the term *service selection affinity*- will finally be directed to the same provider.

However, in some cases, the fact that two operation invocations are designated to be directed to the same provider is merely coincidental. Consider for instance a WS-BPEL scenario *SportsResults* which invokes the *FootballResults* and *BasketballResults* operations of the same service provider to get the respective results. Clearly in this case it is possible to use different providers to get the football and the basketball results, thus SSA need not be applied in this case.

Having only the WS-BPEL scenario at hand, however, the adaptation process has no means to distinguish between the two cases exemplified above, i.e. (a) the case of a room reservation where SSA must be maintained and (b) the case of obtaining sport results, in which case SSA need not be maintained. To tackle this issue, the WS-BPEL scenario designer should indicate which operation invocations are those bearing transactional semantics and therefore need to be directed to the same provider. To this end, we introduce an additional attribute, namely *affinityGroup*; if SSA must be maintained across a set of operation invocations, the WS-BPEL scenario designer should add to the specification of these invocations the *affinityGroup* attribute with an identical value. For instance, the *invoke* constructs of the hotel reservation room would be written as follows:

```
<invoke partnerLink="HotelA" affinityGroup="roomRes"
   operation="checkAvailability" … />
<invoke partnerLink="HotelA" affinityGroup="roomRes"
   operation="getQuote" … />
<invoke partnerLink="HotelA" affinityGroup="roomRes"
   operation="performReservationAndPay" … />
```

The addition of attributes to WS-BPEL constructs is in-line with the language's specifications [1]. A WS-BPEL scenario may designate multiple affinity groups, e.g. one affinity group *RoomRes* may pertain to a hotel room reservation and a second one *AirticketRes* to an air ticket reservation; in this case, all services belonging to the *RoomRes* affinity group will be directed to some provider S_{room} (chosen according to the QoS bounds and weights specified), all operation invocations belonging to the *AirticketRes* group will be directed to some provider $S_{Airticket}$; naturally, it is possible that $S_{room} \neq S_{Airticket}$.

In the case of the *SportsResults* scenario detailed above, the WS-BPEL designer would not use the *affinityGroup* attribute, and therefore the adaptation environment would impose no restrictions regarding the selection of the providers for operations *FootballResults* and *BasketballResults*.

Service selection affinity is analogous to realization relations used in engineering processes [34].

IV. The Adaptation Architecture And Algorithm

To realize the adaptation goals presented in section 3, the proposed approach adopts the architecture illustrated in Fig 1. The architecture introduces two additional components, the *preprocessor* and the *adaptation layer*. Adaptation layers are commonly used for WS-BPEL execution adaptation [5][6][11][13]. The *preprocessor* performs transformations on the original WS-BPEL scenario by (a) arranging for passing appropriate data to the adaptation layer to drive the adaptation and (b) redirecting service invocations to the adaptation layer, so as to be sent to the service implementations best matching the QoS specifications. The preprocessing step produces an *enhanced WS-BPEL scenario*, which is then deployed to the WS-BPEL orchestrator. The *adaptation layer* intervenes between the WS-BPEL orchestrator and the actual web service implementations, arranging for formulating the WS-BPEL scenario *execution plan*, i.e. choosing for each operation invocation in the executing scenario the most appropriate implementation with respect to the current execution's QoS policy, maintaining the SSA where required.

In the following paragraphs, we describe the operation of these two modules, detailing (a) the transformations employed by the preprocessor, (b) the WS-BPEL scenario execution procedure and (c) the algorithm employed to formulate the WS-BPEL scenario execution plan.

A. QoS Specification in the WS-BPEL Scenario

At design time, the WS-BPEL scenario designer specifies the QoS parameters that apply to each invocation. In this work, we adopt the approach introduced in [5] and refined in [9], according to which:

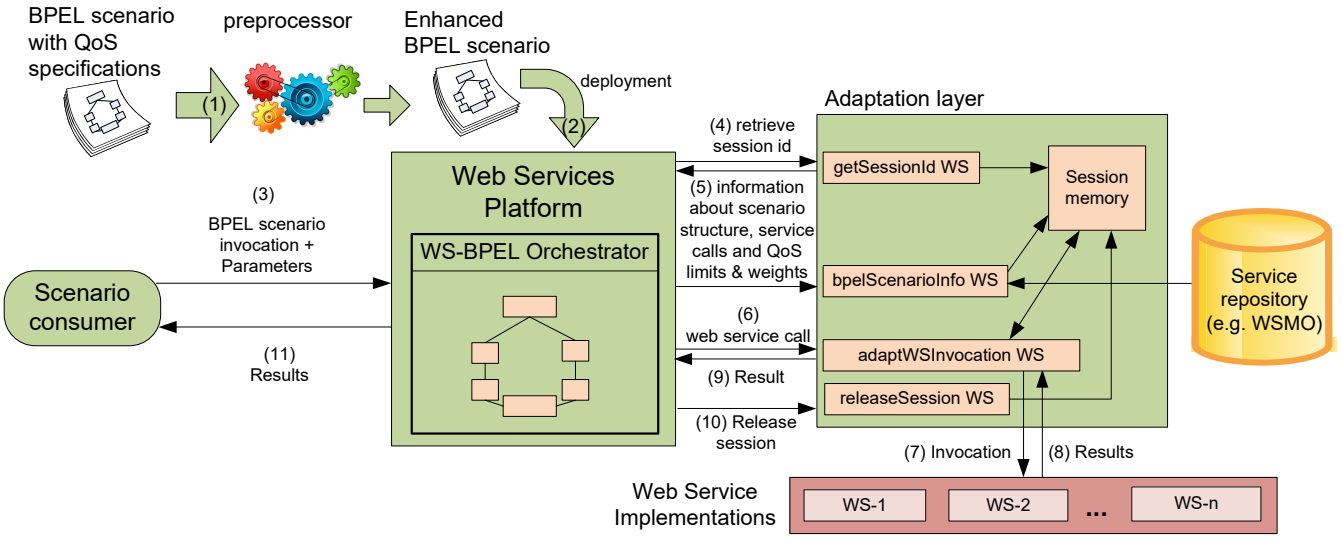

Fig. 1. Architecture of the adaptation framework

1. for each *invoke* construct, the designer provides the attribute *name*, assigning distinct names to the invoke constructs,
2. for the *invoke* construct having the *name* attribute equal to *invX*, the designer should use the WS-BPEL variables *QoSmax_invX* and *QoSmin_invX* which designate the QoS bounds for the particular invocation and
3. the designer should provide the WS-BPEL variable *QoS_weight* to specify the weight of each QoS attribute (recall that weights apply to the whole composition, rather than to individual services [23]).

The WS-BPEL designer may set the values for variables *QoSmax_invX* and *QoSmin_invX* and *QoS_weight* after examining the values of user-provided parameters, tailoring thus the QoS specification to the preferences of the user invoking the scenario. Listing. 1 shows an example of maximum QoS bounds specification for an operation invocation.

```
<assign>
    <copy>
        <from><literal>7</literal></from>
        <to variable="QoSmax_getQuote" part="respTime"/>
    </copy>
    <copy>
        <from><literal>4</literal></from>
        <to variable="QoSmax_getQuote" part="cost"/>
    </copy>
</assign>
<invoke name="getQuote" partnerLink="hotel1" operation="getQuote"
    outputVariable="quote" inputVariable="roomTypeAndPeriod" />
```

Listing 1. QoS specification in the WS-BPEL scenario

B. The Preprocessing Step

As noted above, before the WS-BPEL scenario is deployed to the WS-BPEL orchestrator it is preprocessed so as to become ready for adaptation. The transformation of the scenario to an adaptation-ready form consists of the following activities:

1. arranging for retrieving a session id (*SID*) from the middleware; SID is used to distinguish among invocations to the middleware pertaining to individual executions of WS-BPEL scenarios performed within the WS-BPEL orchestrator. This is accomplished through an invocation to the *getSessionId* operation provided by the adaptation layer, which is inserted at the beginning of the scenario.
2. providing for passing to the middleware all appropriate information regarding the operation invocations performed within the WS-BPEL scenario, their structure (parallel vs. sequential), the affinity groups, the QoS weights for the scenario (as set using the *QoS_weight* variable) and the QoS bounds for the different invocations (as specified through the *QoSmin_invX* and *QoSmax_invX* variables). This is realized through an invocation to the *bpelScenarioInfo* operation of the adaptation layer. This information will be used by the adaptation layer to formulate the execution plan, which is inserted before the first *invoke* operation within the scenario.
3. modifying individual operation invocations so that they are redirected to the adaptation layer, instead of the actual web service operation initially specified in the WS-BPEL scenario. The adaptation layer will then forward the invocation to the appropriate implementation, according to the formulated execution plan. This is accomplished by modifying the *partnerlink* specification. The preprocessor arranges so that the *headers* of each invocation [16][17] contain the session identifier, to enable the adaptation layer to distinguish the particular WS-BPEL scenario execution in the context of which this invocation is performed.
4. inserting an invocation to the *releaseSession* operation of the adaptation layer as the last activity of the WS-BPEL scenario, to allow for release of resources allocated to the particular execution.

When the modified scenario is generated, it is deployed to the WS-BPEL orchestrator and made available for invocation.

C. Executing the WS-BPEL Scenario

When the WS-BPEL scenario commences execution, it will first retrieve from the adaptation layer the session identifier, and afterwards it will invoke the *bpelScenarioInfo* operation of the adaptation layer to provide to it all the information required for the formulation of the execution plan. The adaptation layer will then compute the execution plan as follows:

1. For each operation invocation, the adaptation layer retrieves from the repository those operations that are equivalent to the operation being invoked and satisfy the QoS bounds for the particular invocation. Thus, the set of possible operation assignments for operation op_i $POA(op_i)=\{op_{i,1}, op_{i,2}, \ldots, op_{i,x}\}$ is formulated as follows: $POA(op_i) = \{ op \in Repository: op \; equivalent \; op_i \wedge [(min_{rt,i} \leq rt_s \leq max_{rt,i}) \wedge (min_{c,i} \leq c_s \leq max_{c,i}) \wedge (min_{rel,i} \leq rel_s \leq max_{rel,i})]\}$; recall that the min and max bounds per operation have been received as input to the *bpelScenarioInfo* operation.
2. The adaptation layer formulates an integer programming (IP) problem, in order to produce the execution plan, i.e. a set of concrete operation assignments $EP=\{cop_1, cop_2, \ldots, cop_n\}$ where $cop_i \in POA(op_i)$, such that this set of assignment best matches the QoS parameters specified by the user. To this end, the candidate *i* for realizing operation op_j (i.e. the i^{th} element in $POA(op_j)$) is assigned a utility value calculated by the following function [24]:

$$U(op_{j,i}) = \sum_{k=1}^{3} \frac{Q_{max}(j,k) - q_k(o_{j,i})}{Q_{max'}(k) - Q_{min'}(k)} * w_k$$

where $q_k(o_{j,i})$ is the value of the k^{th} QoS attribute of operation $o_{j,i}$ (the first QoS attribute corresponds to response time, the second one to cost and the third one to availability), w_k being the weight assigned to the k^{th} QoS attribute, $Q_{max}(j,k) = \max_{s \in POA(j)} q_k(s)$ [i.e. the maximum value of QoS attribute *k* among possible concrete operation assignments for operation *j*], and $Q_{max'}(k)$ [resp. $Q_{min'}(k)$] being the overall maximum (resp. minimum) value of QoS attribute *k* within the repository. Note that services with high QoS values have low utility function values. Given the utility function, the computation of the *m-best solutions* can be formulated as an integer programming optimization problem as follows: minimize the overall utility value given by:

$$OUV_{QoS} = \sum_{j=1}^{N} \sum_{i=1}^{|POA(op_j)|} U(op_{j,i}) * x_{j,i}$$

where *N* is the number of operation invocations within the WS-BPEL scenario, $|POA(op_i)|$ is the cardinality of $POA(op_i)$ and $x_{j,i}$ is a binary variable taking the value 1 if $op_{j,i}$ is chosen for implementing operation op_j of the WS-BPEL scenario and the value 0 otherwise.

Since each operation op_i designated in the original WS-BPEL scenario must be realized through exactly one operation in $POA(op_i)$, the constraint set

$$\sum_{i=1}^{|POA(op_j)|} x_{j,i} = 1, 1 \leq j \leq N$$

is added to the problem. In order to support transactional semantics of operations, as designated through SSA groups, the adaptation layer adds constraints to the problem as follows: let us assume that operations op_x and op_y belong in the same affinity group, and $POA(op_x) = \{o_{x,1}, o_{x,2}, \ldots, o_{x,L(x)}\}$, $POA(op_y) = \{o_{y,1}, o_{y,2}, \ldots, o_{y,L(y)}\}$, the possible operation assignments for op_x and op_y, respectively. Without loss of generality, we assume that for the first *k* services in $POA(op_x)$ and $POA(op_y)$:

$$provider(op_{x,m}) = provider(op_{y,m}) \; \forall \; 1 \leq m \leq k$$

while

$$provider(op_{x,m}) \neq provider(op_{y,m}) \; \forall \; m > k$$

Under this setting, transactional semantics are maintained if for the realization of op_x and op_y within the scenario, the corresponding services (i.e. services with the same index) from $POA(op_x)$ and $POA(op_y)$ are selected. To ensure this in the solution of the IP problem, the following constraints are added to the problem:

$$op_{x,a} - op_{y,a} = 0, 1 \leq a \leq k$$

This method is directly generalizable to cases where the affinity group contains more than two operation invocations; for example if the affinity group contained a third operation invocation op_z, the set of constraints

$$op_{x,a} - op_{z,a} = 0, 1 \leq a \leq k$$

would be added to the problem. Note that additional constraints to explicitly maintain affinity between op_y and op_z (i.e. $op_{y,a} - op_{z,a} = 0$) need not be included, since this is guaranteed by the two introduced set of constraints by virtue of transitivity.

This approach (a) exploits the concept of SRC, considering only to the operations actually involved in the scenario and (b) further broadens the options available to the middleware by removing the need for affinity maintenance among services that coincidentally were set in the original scenario to be directed to the same service provider.

Then, the IP problem is solved and the solution is saved (coupled with the session id) to the *session memory* (cf. Fig. 1).

3. When the adaptation layer intercepts an operation invocation (recall from subparagraph IV.B.2 that the preprocessor arranges so that invocations are redirected to the adaptation layer), it retrieves from the session memory the selection made for the realization of the particular invocation in the context of the current WS-BPEL scenario execution (i.e. within the execution plan formulated in step 2) and redirects the invocation to that service. The reply is then collected and returned as a reply to the WS-BPEL orchestrator.
4. Finally, when the WS-BPEL scenario reaches its end, it invokes the *releaseSession* web service, providing the session identifier as a parameter. The *releaseSession* service will then remove from the session memory all information pertaining to this session.

V. Experimental Evaluation

In this section, we report on our experiments aiming to substantiate the feasibility of the proposed approach, both in terms of execution time (quantifying the introduced overhead and performance gains) and solution quality. For our experiments we used two machines: (a) a workstation, equipped with one 6-core Intel Xeon E5-2620@2.0GHz CPU and 16 GB of RAM, which hosted the preprocessor and the clients and (b) a workstation with identical configuration to the first, except for the memory which was 64GBytes, that hosted the WS-BPEL orchestration engine, the adaptation layer, the target web services deployed on a Glassfish 4.1 application server [18] and the service repository. The machines were connected through a 1Gbps local area network. The service repository was implemented as in-memory hash-based structure, which proved more efficient than using a

separate (memory or disk-based) database. Regarding the overhead, we do not measure the time spent for preprocessing, since this is performed in an off-line fashion and does not penalize the WS-BPEL scenario execution performance.

In all experiments, the service repository was populated with synthetic data having an overall size of 1,000 web services; each web service included 3-8 operations and each operation was offered by 60 alternative providers. Each service had at least 9 other services equivalent to it (i.e. containing equivalents for *all* its operations). The QoS attribute values in this repository were uniformly drawn from the domain [0, 10]. The WS-BPEL scenarios used in the experiments were synthetically generated by randomly drawing operations from the repository, and the performance evaluation tests were run for each of the generated scenarios; 1,000 scenarios were generated in total. In the scenario generation process, two consecutive functionality invocations were selected to be executed sequentially (<sequence> construct) with a probability of 0.7 and in parallel (<flow> construct), with a probability of 0.3, while two operations were set to fall in the same affinity group with a probability of 0.2.

The first experiment quantifies the time needed to formulate the WS-BPEL scenario execution plan for varying concurrency degrees (Fig. 3); this overhead is incurred once per scenario execution. We can observe that the time needed remains low even for high degrees of concurrency (160 msec for 200 concurrent invocations) and scales linearly with the concurrency degree.

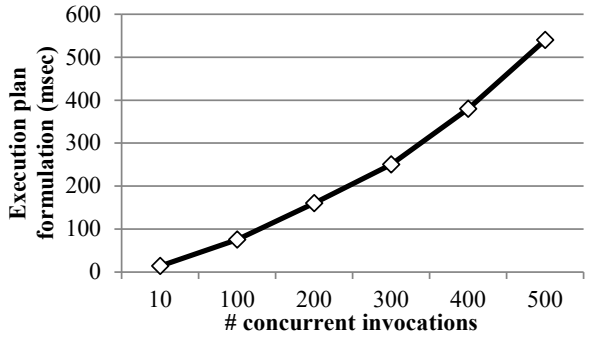

Fig. 3. Execution plan formulation overhead

In the proposed architecture, each operation invocation is redirected to the adaptation layer, which consults the WS-BPEL scenario execution plan and forwards the request accordingly to the chosen operation implementation; subsequently it receives the reply and forwards it back to the WS-BPEL orchestrator. This introduces two extra network messages and the associated processing time, and the incurred overhead, for varying degrees of concurrency, is illustrated in Fig 4. This overhead is incurred once per operation invocation. We can note that the overhead remains small and scales linearly with the concurrency degree.

Fig. 5 compares the QoS of the execution plan formulated for a number of representative trial cases by (i) the QoS-based algorithm described in [9] and (ii) the approach proposed in this paper. The diagram also shows an average which has been computed considering all 1,000 WS-BPEL scenarios used in the experiment, while the representative trial cases were chosen so as to include different number of operation invocations (scenarios 1-3 contain 3 invocations, scenarios 4-6 contain 6 invocations and scenarios 7-10 contain 8 invocations), varying settings regarding parallel flows (scenarios 1, 2, 4 and 7 contain no parallel flows, scenarios 3, 5, 8 and 9 contain one parallel flow and scenarios 6 and 10 contain two parallel flows), and varying numbers of affinity groups (scenarios 1 and 2 contained no affinity groups; scenarios 3, 4, and 5 contained one affinity group, scenarios 6, 7 and 8 contained two affinity groups and scenarios 9 and 10 contained three affinity groups).

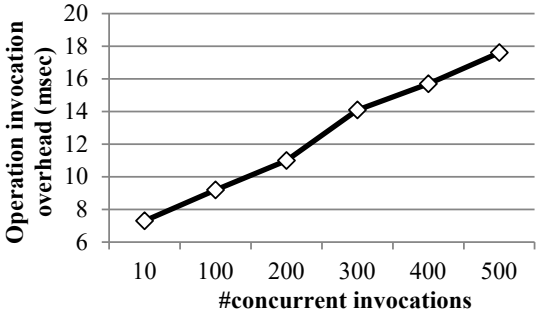

Fig. 4. Operation invocation overhead

All affinity groups contained two invocations, except for scenarios 8 and 9 where one affinity group contained three invocations). We chose to compare the proposed approach against the one described in [9], since the latter maintains SSA (albeit among *all* operations directed to the same provider) and examines the whole solution space, being thus able to always locate the optimal solution. The lower QoS bounds for the operation invocations were randomly drawn from the domain [0,4], while the upper QoS bounds for the operation invocations were randomly drawn from the domain [6,10]. The weights of the QoS attributes were randomly selected from the domain [0,1]. In all cases, a uniform distribution was used.

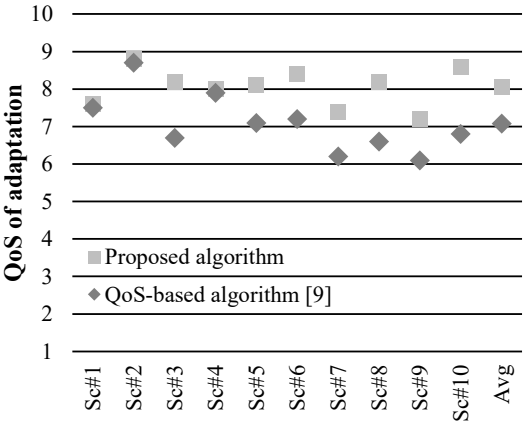

Fig.5. QoS of solutions formulated by the proposed approach and the algorithm described in [9].

The diagram shows that the algorithm proposed in this paper achieves solutions whose QoS is on average higher by 14% than the corresponding solutions formulated by the algorithm described in [9]. This is due to the facts that the proposed algorithm has a broader choice of operations to realize each functionality invocation specified in the original WS-BPEL scenario due to the concepts of SRC and the explicit specification of the affinity groups.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented a framework and associated algorithms for adapting the execution of WS-BPEL scenarios based on QoS policies specified by the users. Our framework introduces the SRC concept, which broadens the set of alternatives that the adaptation layer can use to formulate the execution plan; moreover, it allows for explicit specification of cases that SSA needs to be maintained, further removing unnecessary constraints and allowing for formulation of more efficient scenario execution plans. The proposed framework is complemented with an execution architecture for enacting the adaptation, as well as an appropriate algorithm for computing the WS-BPEL scenario execution plan. The proposed framework has been experimentally validated regarding (i) its performance, (ii) the quality of execution plans generated.

Our future work will focus on extending the framework to handle more efficiently conditional and iteration constructs in the WS-BPEL scenario; this can be supported through branch prediction and loop unrolling techniques [25], as well as by gathering statistical information from prior scenario executions and using it as input to the adaptation process. This information will quantify aspects regarding the behavior of control constructs in the scenario, e.g. the probability that a conditional branch is executed or the distribution of the number of executions of a loop [27]. We also plan to examine how the algorithm can be extended to consider different adaptation strategies [28], and to evaluate the performance of these strategies. Incorporation of collaborative filtering techniques complementary to the QoS-based ones, as suggested in [29][33] will be also considered.

REFERENCES

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

[2] MP. Papazoglou, P. Traverso, F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges", IEEE Computer vol. 40, no 11, 2007, pp. 38-45.

[3] V. Cardellini, V. Di Valerio, V. Grassi, S. Iannucci, F. Lo Presti, "A Performance Comparison of QoS-Driven Service Selection Approaches", Proceedings of ServiceWave 2011, 2011, pp. 167–178.

[4] LB. Zeng, AHN. Benatallah, M. Dumas, J. Kalagnanam, H. Chang, "QoS-aware middleware for web services composition". IEEE Transactions on Software Engineering, vol. 30, no 5, 2004.

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

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

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

[8] C. Kareliotis, C. Vassilakis, P. Georgiadis, "Enhancing BPEL scenarios with dynamic relevance-based exception handling", Proceedings of ICWS07, Salt Lake City, Utah, USA, 9–13 July 2013, pp.751–758.

[9] D. Margaris, C. Vassilakis, P. Georgiadis, "An integrated framework for QoS-based adaptation and exception resolution in WS-BPEL scenarios", Proceedings of the ACM Symposium on Applied Computing, 2013.

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

[11] Z. Zheng, H. Ma, M. Lyu, I. King, "QoS-Aware Web Service Recommendation by Collaborative Filtering", IEEE Transactions on Services Computing vol. 4 no 2, 2011, pp. 140-152.

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

[13] US. Manikrao, TV. Prabhakar, "Dynamic Selection of Web Services with Recommendation System" Proceedings of the International Conference on Next Generation Web Services Practices, 2005, pp. 117-121.

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

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

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

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

[18] GlassFish Community, http://glassfish.java.net/

[19] D. Kuebler, W. Eibach, "Metering and accounting for Web services," 2001.

[20] C. Kareliotis, C. Vassilakis, E. Rouvas, P. Georgiadis, "IQoS-aware exception resolution for BPEL processes: a middleware-based framework and performance evaluation", International Journal of Web and Grid Services, vol. 5, January 2009, pp. 284-320.

[21] X. Fei, S. Lu, "A Dataflow-Based Scientific Workflow Composition Framework", IEEE Transactions on Services Computing 5(1), 2012, pp. 45-58

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

[23] W3C. Web Services Description Language (WSDL) 1.1, 2001.

[24] M. Alrifai, T. Risse, "Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition", Proceedings of WWW '09, 2009, pp. 881-890.

[25] A.J. Bernstein, "Analysis of Programs for Parallel Processing", IEEE Trans. on Electronic Computers" Volume:EC-15(5), 1996, pp. 757–763.

[26] D. Martin, M. Paolucci, S. McIlraith, M. Burstein et al. "Bringing Semantics to Web Services: The OWL-S Approach", in Semantic Web Services and Web Process Composition, 2005, pp. 26-42.

[27] D. Ardagna, B. Pernici, "Adaptive Service Composition in Flexible Processes", IEEE Transactions on Software Engineering, vol. 33, no. 6, June 2007, 369 - 384

[28] C. Zeginis, D. Plexousakis, "Web Service Adaptation: State of the art and Research Challenges", Institute of Computer Science, FORTH-ICS, Technical Report 410, ICS-FORTH, October 2010.

[29] D. Margaris, C. Vassilakis, P. Georgiadis, "A Hybrid Framework for WS-BPEL Scenario Execution Adaptation, Using Monitoring and Feedback Data", Proceedings of the ACM SAC, 2015, Salamanca, Spain

[30] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, R. Mirandola, "MOSES: a Framework for QoS Driven Runtime Adaptation of Service-oriented Systems", IEEE Transactions on Software Engineering 38(5), 2012, pp. 1138 – 1159.

[31] L. Qi, X. Xia, J. Ni, Ch. Ma, Y. Luo, "A Decomposition-based Method for QoS-aware Web Service Composition with Large-scale Composition Structure", Proceedings of the Fifth International Conferences on Advanced Service Computing, 2013, pp. 81-86.

[32] A.B. Hassine, S. Matsubara, T. Ishida, "A Constraint-Based Approach to Horizontal Web Service Composition", Proceedings of the 5th International Semantic Web Conference, ISWC 2006, 2006, pp. 130-143.

[33] D. Margaris, C. Vassilakis, P. Georgiadis, "An integrated framework for adapting WS-BPEL scenario execution using QoS and collaborative filtering techniques", Science of Computer Programming 98, 2015, pp. 707–734.

[34] M. Sinnema, S. Deelstra, and P. Hoekstra, "The covamof derivation process," in ICSR, 2006, pp. 101–114.