University of Peloponnese
Department of Informatics and Telecommunications
Software and Database Systems Laboratory

Combining Quality of Service-based and Collaborative filtering-based techniques for BPEL scenario execution adaptation

Technical Report TR-14002

Dionisis Margaris, Costas Vassilakis, Panagiotis Georgiadis
margaris@di.uoa.gr, costas@uop.gr, p.georgiadis@di.uoa.gr

May, 2014
Tripoli, Greece

1. Introduction

In this technical report, we present an approach for combining Quality of Service (QoS)-based criteria and collaborating filtering (CF)-based techniques for performing BPEL scenario execution adaptation. We consider *horizontal adaptation* of BPEL scenario execution, i.e. the adaptation leaves the composition logic intact and focuses on selecting the most appropriate service to realize each of the functionalities invoked in the context of the BPEL scenario. The algorithms are presented in detail and a respective example on the algorithm operation is given.

The rest of this report is organized as follows: in section 2, we present the underlying foundations regarding QoS and CF. In section 3, we present the algorithms used for performing the adaptation, while section 4 gives a detailed example. Finally, section 5 concludes the report.

2. QoS concepts and collaborative filtering foundations

In the following subsections we summarize the concepts and underpinnings from the areas of QoS and collaborative filtering, which are used in our work.

2.1 QoS concepts

For conciseness purposes, in this paper we will consider only the attributes responseTime (rt), cost (c) and availability (av), adopting their definitions from [1]. This does not lead to loss of generality, since the algorithms can be straightforwardly extended to accommodate more attributes.

The QoS specifications for a service within the BPEL scenario may include an upper bound and a lower bound for each QoS attribute, i.e. for each service s_j included in a BPEL scenario, the designer formulates two vectors $MIN_j=(min_{rt,j}, min_{c,j}, min_{av,j})$ and $MAX_j=(max_{rt,j}, max_{c,j}, max_{av,j})$. Additionally the designer formulates a weight vector $W=(rt_w, c_w, av_w)$, 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* means that that it is *cheaper* that a service having *cost = 4*.

In order to compute the QoS attribute values of a service S composed from constituent services s_1, …, s_n having QoS attributes equal to $(rt_1, c_1, av_1), ..., (rt_n, c_n, av_n)$, respectively, the formulas given in Table 1 [2] can be used.

response time	cost	availability
$\sum_{i=1}^{n} rt_i$	$\sum_{i=1}^{n} c_i$	$\prod_{i=1}^{n} av_i$

Table 1. QoS of composite services

Most works dealing with QoS-based BPEL scenario execution adaptation, consider given QoS attribute values for each service, which can be for instance declared by the service provider within an SLA. However, in the real world, QoS metrics such as response time and availability may vary, due to server or network conditions (failures, overloads, bottlenecks etc). To tackle this issue, in this paper, we employ prediction models for QoS attribute values, in order to use in the recommentation process values that are closer to the actual ones, improving thus the accuracy of the adaptation. In particular, we adopt [3] and [4] for predicting the service response time and service availability, respectively. Both these algorithms predict future performance of services by examining past measurements; the platform proposed in this work collects these measurements when invoking services in the context of BPEL scenario executions and makes them available to the modules predicting the future QoS values.

2.2 Subsumption relationship representation

In order to adapt the BPEL scenario execution, the adaptation engine needs to be able to find which services offer the same functionality, and are thus candidate for invocation when this particular functionality is needed. In this work, we represent this information using *subsumption relationships* [5] which, for any pair of services S_1 and S_2 defined as follows: (i) S_1 *exact* S_2, iff S_1 provides the same functionality with S_2 (ii) S_1 *plugin* S_2, iff S_1 provides more specific functionality than S_2; in this case S_1 could be used whenever the functionality of S_2 is needed, since it delivers (a specialization of)

the functionality delivered by S_2 (iii) S_1 *subsume* S_2, iff S_2 provides more generic functionality than S_2. In this case S_1 cannot unconditionally be used whenever the functionality of S_2 is needed and (iv) S_1 *fail* S_2, in all other cases; in this case, S_2 cannot be substituted for S_2. Under these definitions, a service *A* can be unconditionally substituted by a service *B* if *(A exact B or A plugin B)*; this setup provides more flexibility as compared to strict service equivalence (*A exact B*) regarding the formulation of the adapted execution plan, and is hence adopted in this paper.

Effectively, subsumption relationships organize services in a tree, where generic services are located towards the root and more specific services towards the leaves [5]. Tree nodes, besides service identity, can accommodate QoS values for the services they represent; this information can be stored in repositories such as OPUCE [6]. Figure 1 shows an excerpt of a subsumption relationships tree.

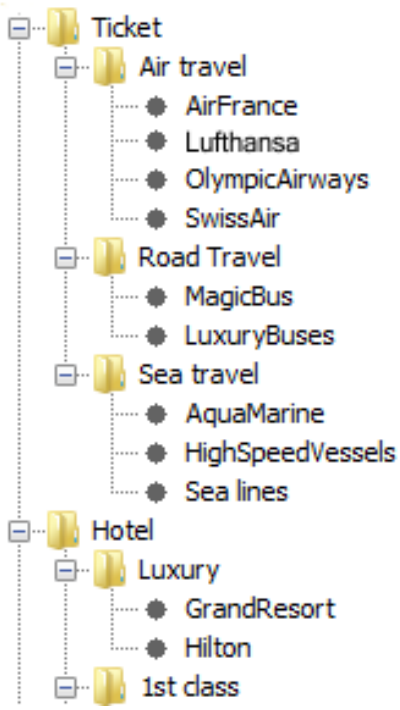

Figure 1. Example subsumption relationships

2.3 Designations on specific service bindings and functionality omissions

As noted in section 1, users may wish to designate exact services to be invoked for realizing specific functionalities, while asking for recommendations on other ones. For instance, in a travel planning scenario the consumer may request that s/he travels by "Sea Lines". Further, the consumer may also specify that some functionality optionally included in the BPEL scenario is not executed; for example, a tourist may not want to rent a car, while such a provision is present in the scenario. Typically, the BPEL code will examine input parameters and decide using a conditional execution construct (<switch>) whether to invoke the functionality or not. Finally, functionalities that are neither explicitly bound to a specific services, nor are designated as "not to be executed" are subject to adaptation. We consider that specific bindings and designations for functionality omissions are explicitly expressed in the request for scenario invocations.

2.4 Usage patterns repository

In order to perform CF-based adaptation, a repository with user ratings for services is required. In this paper, we adopt the representation used in [7], where the ratings repository is modelled as a table having a number of columns equal to the functionalities present in the BPEL scenario, and one row for each BPEL scenario execution. Cell *i,j* is filled with value S if during the i^{th} execution of the BPEL scenario, service S was used to implement functionality *j*; cell (i, j) may be also blank, if during the i^{th} execution of the BPEL scenario functionality *j* was omitted. In order to accommodate user ratings, we extend this repository by adding one column per functionality. This column stores an integer value from the domain [1, 10], corresponding to the rating given by the user that executed the particular scenario instance. For the cases that the user has not provided a rating, a *null* value is stored and the CF-based algorithm uses a default value, as explained in section 4. The BPEL scenario adaptation unit inserts new records to the usage patterns repository, when the concrete services that will be invoked in the context of a particular BPEL scenario execution are decided, while the user evaluation collection module arranges for storing the user rankings in the relevant columns.

# exec	*Travel*	*R(travel)*	*Hotel*	*R(Hotel)*	*Event*	*R(Event)*
1	OlympicAirways	8	YouthHostel	3	ChampionsLeague	7
2	SwissAir	6	Hilton	9	GrandConcert	6
3	HighSpeedVessels	*null*	YouthHostel	*null*		
4	LuxuryBuses	4		*null*	EuroleagueFinals	9
5	Lufthansa	7	GrandResort	8	OperaPerformance	6
6	AirFrance	*null*	Hilton	*null*		
7	LuxuryBuses	*null*	YouthHostel	*null*	ChampionsLeague	*null*

Table 2. Example usage patterns repository

3. The service recommendation algorithm

As stated in section 1, our approach follows the horizontal adaptation algorithm, i.e. it leaves the composition logic intact and adapts the execution by selecting which concrete service implementation will be used in each specific invocation. In order to perform this task, the algorithm takes into account the following criteria:

- The consumer's QoS specifications (bounds and weights).
- Designations on which exact services should be invoked, if such bindings are requested by the consumer (e.g. a user wanting to travel using Air France).
- Designations on which functionalities should not be invoked (e.g. a user wanting to book a trip without scheduling any event attendance).
- The QoS characteristics of the available service implementations, including monitored values of the QoS attributes of the services.
- The service subsumption relationships.
- The usage pattern repository, including ratings entered by the users.

The approach proposed in this paper incorporates two different candidate service ranking algorithms, the first examining the QoS aspects only ([7]) and the second being based on CF techniques ([8]). The algorithms run in parallel to formulate their suggestions regarding the services that should be used in the adapted execution, and subsequently their suggestions are combined, through a metasearch score combination algorithm with varying weights. An example of the algorithm operation can be found in section 4.

3.1 The QoS-based adaptation algorithm

The QoS-based adaptation algorithm initially identifies the services which are candidate to be used for delivering functionalities in the context of the current BPEL scenario, respecting the QoS-bounds set by the user, and subsequently computes the *k-best* service assignments to the functionalities requested for the particular scenario execution. In more detail, the algorithm proceeds as follows:

1. For each functionality f_i for which adaptation has been requested, the algorithm retrieves from the semantic service repository the concrete services that (a) deliver this functionality and (b) respect the QoS bounds set by the users. These are the candidates for implementing functionality f_i. Formally, this is expressed as

$$C(f_i) = \{s_{i,j}: (s_{i,j} \text{ exact } f_i \vee s_{i,j} \text{ plugin } f_i) \wedge QoS_{min}(req, f_i) \leq QoS(s_{i,j}) \leq QoS_{max}(req, f_i)\}$$

   Note that in all steps of this algorithm, the QoS values for response time and availability considered for each service are those returned by predictor methods [3] and [4], respectively.
2. Subsequently, the algorithm formulates an integer programming problem to compute the *k-best* solutions regarding the assignment of concrete services $s_{i,j}$ to each functionality f_i. To express the integer programming optimization problem in this work we adopt the concrete service utility function used in [9], which is

$$U(s_{i,j}) = \sum_{p=1}^{3} \frac{Q_{max}(i,\mathrm{p}) - q_p(s_{i,j})}{Q_{max'}(lp) - Q_{min'}(p)} * w_p \quad (1)$$

   where $q_p(s_{i,j})$ is the value of the p^{th} QoS attribute of concrete service $s_{i,j}$ (the first QoS attribute being response time, the second cost and the third one availability), w_p being the weight assigned to the p^{th} QoS attribute

$Q_{max}(i,p) = \max_{s \in C(f_i)} q_p(s)$

[i.e. the maximum value of QoS attribute p among possible concrete service assignments for functionality f_i], and $Q_{max'}(p)$ [resp. $Q_{min'}(p)$] being the overall maximum (resp. minimum) value of QoS attribute p within the service repository. In this work, we modify the utility function so as to have an increasing value with respect to the utility of the service (contrary to the function in [9], which has a decreasing value). The modified utility function used hereafter is $U(s_{i,j}) = \sum_{p=1}^{3} (1 - \frac{Q_{max}(i,p) - q_p(s_{i,j})}{Q_{max'}(lp) - Q_{min'}(p)}) * w_p$. Using the utility function, the computation of the best solution is expressed as the following integer programming problem: maximise the overall utility value given by

$$OUV_{QoS} = \sum_{i=1}^{F} \sum_{j=1}^{|C(f_i)|} U(s_{i,j}) * x_{i,j}$$

where F is the number of functionalities f_i requiring adaptation, and each $x_{j,i}$ is a binary variable taking the value 1 if $i_{j,j}$ is selected for delivering functionality f_i, and 0, otherwise. Since each functionality f_i is delivered in the final execution plan by exactly one concrete service, the maximization of the utility value is subject to the constraint

$$\sum_{j=1}^{|C(f_j)|} x_{i,j} = 1, \forall i: 1 \leq i \leq F$$

This problem is then solved and the *k-best* solutions are obtained. Note that this formulation employs the sum function to rate the availability of the composite service taking into account the availability values of the constituent services, rather than the product function, as denoted in Table 1. The transformation from the product function to the sum function is achieved by applying the logarithmic function to the computation of reliability [10], since $\log(\prod_{i=1}^{n} rel_i) = \sum_{i=1}^{n} \log(rel_i)$. Through this transformation, the problem can be expressed as an integer programming problem and solved efficiently.

The solutions are saved, together with their overall utility score, for perusal in the combination step. In order to solve the integer programming problem computing the *k-best* solutions, the IBM ILOG CPLEX optimizer was used. In our implementations, we have set k=20.

3.2 The CF-based algorithm

The CF-based algorithm employed in our proposal is an adaptation of the standard GroupLens algorithm [11], modified to take into account the semantic distance of the services realizing the same functionality. For instance, rows 2 and 5 of Table 2 are considered "semantically close", since they both list air transport for travel, a first class hotel for accommodation and classical music events; on the other hand rows 2 and 7 of the same table are considered "semantically distant", since all three services correspond to diverse real world counterparts (air travel vs. bus, 1st class hotel vs. 3rd class, concert vs. sports). Taking this into account, when a request arrives asking for travel via AirFrance and accommodation in GrandResort and requesting a recommendation for event attendance, the ratings in rows 2 and 5 must be taken more strongly into account than those in row 7, since the former two rows are "closer" to the one under adaptation.

To accommodate this adaptation, we extend the formula of cosine similarity between two rows $\vec{X}$, $\vec{Y}$ of the usage pattern repository table as follows:

$$r(\vec{X},\vec{Y}) = \frac{\sum_{k=1}^{n} (\vec{X}[k] * \vec{Y}[k] * d(\vec{X}[k],\vec{Y}[k]))}{\|\vec{X}\| * \|\vec{Y}\|} \quad (2)$$

We can observe in equation (2) that the standard cosine similarity metric has been extended to accommodate the semantic distance between the services that realize the same functionality in rows $\vec{X}$ and $\vec{Y}$; this is accomplished by multiplying each term of the sum in the nominator by a metric of the semantic distance between the two services, which is denoted as $d(s_1, s_2)$ and is computed using the formula introduced in [12]:

$$d(s_1,s_2) = C - lw*PathLength - NumberOfDownDirection \quad (3)$$

where C is a constant set to 8 [12], lw is the level weight for each path in subsumption tree (cf. Figure 1), *PathLength* is the number of edges counted from functionality s_1 to functionality s_2 and *NumberOfDownDirection* is the number of edges counted in the directed path between functionality s_1 and s_2 and whose direction is towards a lower level in the subsumption tree. For more details in the computation of the semantic distance, the interested reader is referred to [12]. We further normalize this similarity metric in the range [0, 1] by dividing the result computed in the above formula by 8; this way, the multiplication by the normalized similarity metric in equation (2) reduces the correlation coefficient between the two rows by a factor proportional to the semantic distance of the services employed in these rows to realize the same functionality.

For items not explicitly rated, we follow the rationale of [7] according to which usage of a service is an indication of preference, and we choose a rating equal to the 80% of the maximum rating. This is inline with the findings of [13], which asserts that dissatisfied users will provide negative feedback with a very high probability (≥89%). Rows that have not been rated at all (and therefore have a default value for all ratings) are the reason behind choosing the cosine similarity against the Pearson similarity, since the latter disregards rows whose ratings have no variance (i.e. are all equal).

Using the modified cosine similarity, the CF-based algorithm operates as follows:

1. It retrieves from the usage pattern repository all rows that contain a service implementing the functionality on which a recommendation is requested. For example, if a recommendation on event attendance is requested, only rows 1, 2, 4, 5 and 7 of table 1 will be retrieved.
2. The rows retrieved from step 1 are filtered to retain only those that fulfil the QoS criteria requested by the user.
3. The similarities between the request and each row are computed using the modified cosine similarity metric. The request is represented here as a vector $\vec{R}$, having a rating equal to 10 for each functionality included in the scenario and a rating equal to 0 for each functionality designated as not to be executed.
4. For each distinct service implementing the requested functionality that is included in the remaining rows, we compute its rating prediction using the standard rating prediction formula

$$p(\vec{R}[k]) = \frac{\sum_{\vec{N} \in raters(\vec{R}[k])} (\vec{N}[k]) * r(\vec{R},\vec{N})}{\sum_{\vec{N} \in raters(\vec{R}[k])} r(\vec{R},\vec{N})}$$

[11] (we again do not subtract the mean $\overrightarrow{N}$ from $\overrightarrow{N}[k]$, so as not to render useless the rows having only default values, and correspondingly we do not add the mean rating of the user for which the prediction is being made).

5. Finally, we retain the *20-best* services for each functionality requiring adaptation, for perusal in the combination step.

After the lists of candidates for each individual service that is subject to adaptation have been computed, the algorithm selects the *top-20* execution plans with respect to their *CF-score*. Given an execution plan containing services ($s_{1,i}$, …, $s_{N,k}$) with the similarity scores of the services computed in step 5 being (CFS($s_{1,i}$), …, CFS($s_{N,k}$)), then the *CF-score* of the execution plan is equal to CFS($s_{1,i}$)+…+CFS($s_{N,k}$). Computing the *top-20* execution plans is modelled as an integer programming optimization problem, formulated as follows: maximize the overall utility value given by:

$$OUV_{CF} = (\sum_{i=1}^{F} \sum_{j=1}^{L(i)} prediction(s_{i,j}) * x_{i,j})/F$$

where F is the number of functionalities $funct_k(request)$ requiring adaptation, $L(i)$ is the number of services selected by the CF-based algorithm for functionality i (normally 20, but it is possible that fewer results are retrieved, depending on the contents of the usage pattern repository), and each $x_{i,j}$ is a binary variable taking the value 1 if $s_{i,j}$ is selected for delivering functionality $funct_i(request)$, and the value 0, otherwise. The result is divided by the number of functionalities F, to normalize it in the range [0,1]. Since each functionality $funct_i(request)$ is delivered in the final execution plan by exactly one concrete service, the maximization of the utility value is subject to the constraint

$$\sum_{j=1}^{L(i)} x_{i,j} = 1, 1 \leq i \leq F$$

The solutions are saved, together with their overall utility score, for perusal in the combination step.

The CF module has been implemented using Apache Mahout (https://mahout.apache.org/), by subclassing the *UncenteredCosineSimilarity* class and reimplementing in the subclass the *UserSimilarity* method, to accommodate the semantic similarity metric described above. The integer programming problem for computing the *k-best* solutions is solved using the IBM ILOG CPLEX optimizer.

3.3 The combination step

The combination step synthesizes the results given by individual algorithms to produce to a single result. Recall from the previous two subsections that each algorithm produces a set of candidate execution plans, with each execution plan being tagged with the relevant normalized score (*QoS-score* or *CF-score*). In order to combine the scores, we use the CombMNZ metasearch algorithm, since it has been found to have the best performance [14] [the CombMNZ rating of a solution is computed by multiplying the sum of the individual scores by the number of non-zero scores, i.e. $CombMNZ_i = m_i * \sum_{j=1}^{m_i} r_j(i)$, where m_i is the number of algorithms giving non-zero rating to item i and $r_j(i)$ is the rating given by algorithm j to item i]. After computing the CombMNZ metasearch for all candidate execution plans, the combination step selects the execution plan with the highest score, which will be used to drive the adaptation process.

3.4 The execution adaptation architecture

The execution adaptation architecture, illustrated in Fig. 2, follows the middleware-based approach, with an *adaptation layer* intercepting web service invocations and appropriately directing them to the services decided by the adaptation algorithm. As

shown in Figure 2, the BPEL scenario execution initially passes to the adaptation layer the information regarding service invocations that will be performed, QoS bounds and weights as well as specific service bindings. When the adaptation layer receives this information, it applies the adaptation algorithm to formulate the execution plan for the particular scenario execution (i.e. decide the actual services that will be invoked to deliver each functionality) and stores the execution plan for later perusal. Subsequently, when a web service invocation is intercepted by the adaptation layer, the respective execution plan is retrieved from the execution plan storage, the web service decided to deliver the specific functionality is extracted and the invocation is routed accordingly to that service. Note that steps (4)-(8) depicted in Figure 2 are repeated multiple times within each BPEL scenario execution, once per web service invocation performed. When the invocation to a service implementation has concluded, the data regarding the service's response time and availability are passed to the QoS prediction and update module, which computes the predicted values for the respective QoS parameters and updates the corresponding elements in the semantic service repository.

Additionally, the BPEL scenario returns at the end of its execution, along with the result, an *evaluation token*, which the consumer may use to enter the ratings for the services s/he has used in the context of the BPEL scenario execution. The evaluation token is returned in the response headers, to retain the response payload schema intact. To accommodate this additional functionality (passing the necessary information to the adaptation layer and returning the evaluation token), the BPEL scenario is preprocessed as described in [7] before being deployed to the web services platform, with the preprocessing step injecting the necessary invocations to the adaptation layer into the scenario, and the result of the preprocessing step is then deployed and made available for invocations.

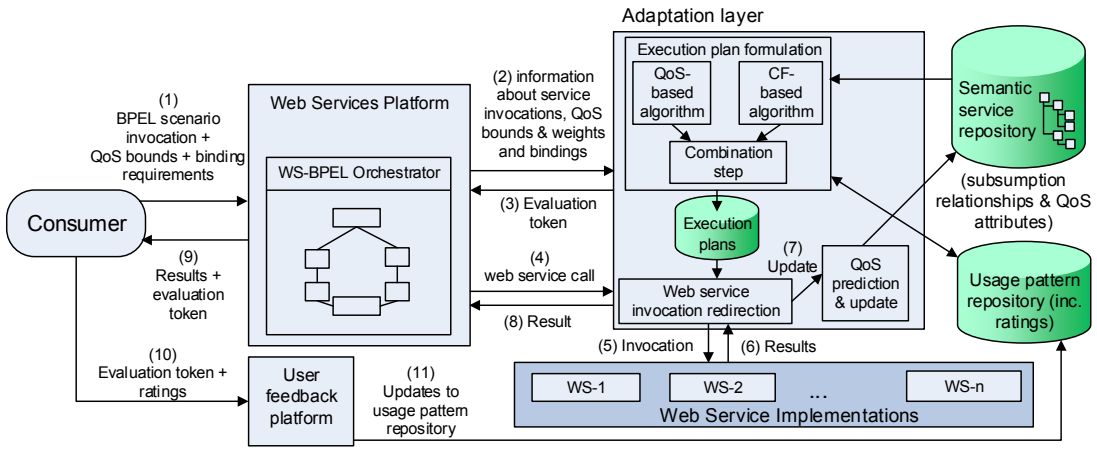

Figure 2. The Execution adaptation architecture

4. An example of the algorithm operation

In this section, we give an illustrative example on the operation of the adaptation algorithm. In this example, we consider the following:

a) The scenario to be adapted is the trip reservation application used in the examples in section 2. The scenario includes mandatory invocations to a travel reservation and a hotel reservation service, and an optional invocation to an even attendance booking service.

b) The subsumption relationships that will be used in this scenario are as depicted in Figure 3.

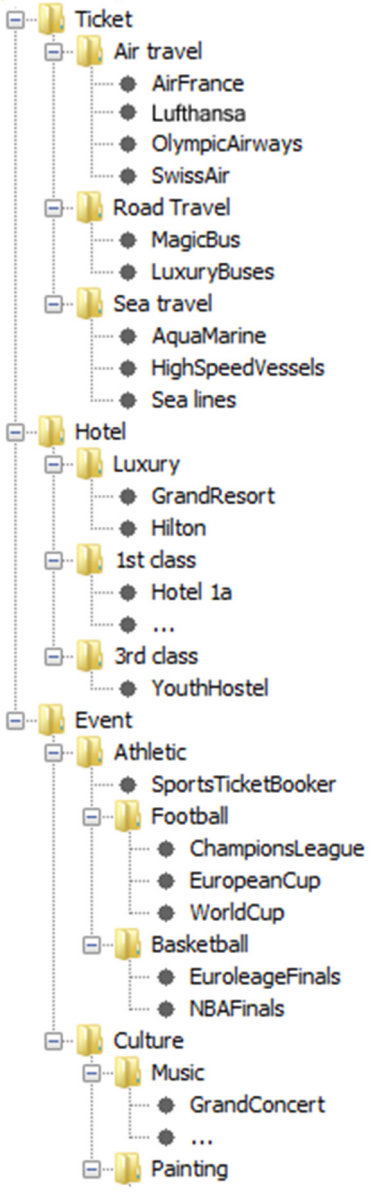

Figure 3. Subsumption relationships tree used in the example

c) The QoS values for the services implementing the "Air travel" functionality are as shown in Table 3. The table lists only the QoS values for the services implementing the "Air travel" functionality, since these are the only ones pertinent in this example.

Equivalent WS	*Cost*	*Response Time*	*Availability*
AirFrance	8	10	7
Lufthansa	9	8	7
OlympicAirways	2	5	9
Swissair	7	7	8

Table 3. QoS values for the services implementing the "Air travel" functionality

d) The contents of the usage pattern repository are as shown in Table 4.

# exec	*Travel*	*R(travel)*	*Hotel*	*R(Hotel)*	*Event*	*R(Event)*
1	OlympicAirways	8	Hilton	3	GrandConcert	5
2	Lufthansa	9	YouthHostel	9	EuropaLeague	7
3	HighSpeedVessels		YouthHostel			
4	HighSpeedVessels	4			ChampionsLeague	9
5	AirFrance	7	GrandResort	8	OperaPerformance	6
6	SwissAir	6	Hilton			
7	LuxuryBuses	9	YouthHostel	6	EuroleagueFinals	9
8	Lufthansa	9	YouthHostel	8	EuroleagueFinals	

Table 4. Usage pattern repository used in the example

e) The user request to be adapted is:
AirTravel(R), YouthHostel, ChampionsLeague
which effectively reads: *I want to stay in YouthHostel and attend the ChampionsLeague event, and I want a recommendation regarding an AirTravel service.*
The user has set a QoS weight vector equal to
W= (0.4, 0.3, 0.3)
while the MIN and MAX vectors, setting the lower and upper limits respectively for service QoS attributes, are set as follows:
$MIN_{AirTravel}$*=(4, null, 5)*
$MAX_{AirTravel}$*=(null, null, null)*
i..e. a minimum of 4 and 5 is set for the travel cost and availability respectively, while no lower limit is set for its response time. Similarly, no upper bounds are imposed for any service.

f) We assume that the minimum and maximum values for the QoS attributes within the repository are as follows (these are needed in the utility function *U*):
MIN=(2, 1, 2)
MAX=(10, 10, 9)

4.1 Applying the QoS-based algorithm

1. First, we retrieve from the service repository (c.f. Table 3) all rows implementing the "air travel" functionality. All rows of the repository qualify (since the excerpt of the repository depicted in Table 3 consists exactly of these rows)
2. Subsequently, the rows not meeting the QoS bounds are filtered out. As a consequence, row #3, corresponding to the *OlympicAirways* service, is rejected. Subsequently, we compute the utility function *U* for each of the remaining services. The values for the utility function are as follows (recall that the utility function is defined as $U(s_{i,j}) = \sum_{p=1}^{3} \left(1 - \frac{Q_{max}(i,p) - q_p(s_{i,j})}{Q_{max'}(lp) - Q_{min'}(p)}\right) * w_p$):

U(AirFrance) = $\left(1 - \frac{9-8}{10-2}\right) * 0.4 + \left(1 - \frac{10-10}{10-1}\right) * 0.3 + \left(1 - \frac{8-7}{9-2}\right) * 0.3 = 0.913$

$$U(\text{Lufthansa}) = \left(1 - \frac{9-9}{10-2}\right) * 0.4 + \left(1 - \frac{10-8}{10-1}\right) * 0.3 + \left(1 - \frac{8-7}{9-2}\right) * 0.3 = 0.888$$

$$U(\text{SwissAir}) = \left(1 - \frac{9-9}{10-2}\right) * 0.4 + \left(1 - \frac{10-8}{10-1}\right) * 0.3 + \left(1 - \frac{8-7}{9-2}\right) * 0.3 = 0.788$$

Subsequently, we formulate the integer programming problem to maximize the overall utility function

$$OUV_{QoS} = \sum_{i=1}^{F} \sum_{j=1}^{|C(f_i)|} U(s_{i,j}) * x_{i,j}$$

subject to the constraint

$$\sum_{j=1}^{L(i)} x_{i,j} = 1, 1 \leq i \leq F$$

Since now F=1 (F is the number of functionalities for which adaptation is requested), we have that the overall utility function is reduced to

$$OUV_{QoS} = \sum_{j=1}^{|C(AirTravel)|} U(s_{AirTravel,j}) * x_{AirTravel,j} =$$

$$U(\text{AirFrance}) * x_{\text{AirFrance}} + U(\text{Lufthansa}) * x_{\text{Lufthansa}} + U(\text{SwissAir}) * x_{\text{SwissAir}}$$

subject to the constraint

$$x_{\text{AirFrance}} + x_{\text{Lufthansa}} + x_{\text{SwissAir}} = 1$$

The three possible solutions to this integer programming problem are as shown in Table 5:

Solution	*QoS-score*
$x_{\text{AirFrance}}=1$, $x_{\text{Lufthansa}}=0$, $x_{\text{SwissAir}}=0$	0.913
$x_{\text{AirFrance}}=0$, $x_{\text{Lufthansa}}=1$, $x_{\text{SwissAir}}=0$	0.888
$x_{\text{AirFrance}}=0$, $x_{\text{Lufthansa}}=0$, $x_{\text{SwissAir}}=1$	0.788

Table 5. Solutions proposed by the QoS-based algorithm

We save these solutions for perusal in the combination step.

4.2 Applying the CF-based algorithm

According to the first step of the CF-based algorithm, we will retrieve from the usage pattern repository (cf. Table 4) only those rows that involve the functionality requested for adaptation. Since the functionality for which adaptation is requested is *AirTravel*, rows 3, 4 and 7 will be eliminated, since they involve other means of transportation (sea travel for rows 3 and 4 and bus travel for row 7). Therefore, the rows depicted in Table 6 will be retrieved.

1	OlympicAirways	8	Hilton	3	GrandConcert	5
2	Lufthansa	6	YouthHostel	9	EuropaLeague	6
5	AirFrance	7	GrandResort	8	OperaPerformance	6
6	SwissAir	6	Hilton	*null*		
8	Lufthansa	9	YouthHostel	8	EuroleagueFinals	*null*

Table 6. Rows of the usage pattern repository delivering the functionality under adaptation

The second step of the CF-based algorithm eliminates the rows for which the service delivering the functionality under adaptation does not meet the QoS bounds set by the

client. Row 1 fails to satisfy them so it is eliminated, and the rows retained for further processing are as shown in Table 7. At this point, we fill the *null* value of row #6 and row #8 with the default value (8).

2	Lufthansa	6	YouthHostel	9	EuropaLeague	6
5	AirFrance	7	GrandResort	8	OperaPerformance	6
6	SwissAir	6	Hilton	*8*		
8	Lufthansa	9	YouthHostel	8	EuroleagueFinals	*8*

Table 7. Rows of table 6 satisfying the QoS bounds

We now compute the similarity of each row to a request vector $\vec{R} = (10, 10, 10)$, taking into account the semantic distances between the services. The semantic distances between the services pertinent to this adaptation are computed through the formula

$d(s_1,s_2) = (8 - lw*PathLength - NumberOfDownDirection) / 8$

and their values are as follows:

d(AirTravel, Lufthansa) = (8 - 2/3 *1 – 1) / 8 = (19/3) / 8 = 19/24

d(AirTravel, AirFrance) = (8 - 2/3 *1 – 1) / 8 = (19/3) / 8 = 19/24

d(AirTravel, SwissAir) = (8 - 2/3 *1 – 1) / 8 = (19/3) / 8 = 19/24

d(YouthHostel, YouthHostel) = (8 - 1/3 *0 – 0)/ 8 = 8 / 8 = 1

d(YouthHostel, GrandResort) = (8 - 1/3 *4 – 2) / 8 = (14/3) / 8 = 14/24

d(YouthHostel, Hilton) = (8 - 1/3 *4 – 2) / 8 = (14/3) / 8 = 14/24

d(ChampionsLeague, EuropaLeague) = (8 - 1/4 *2 – 1) / 8 = (26/4) / 8 = 26/32

d(ChampionsLeague, EuroleagueFinals) = (8 - 1/4 *4 – 2) / 8 = (5) / 8 = 5/8

d(ChampionsLeague, OperaPerformance) = (8 - 1/4 *6 – 3) / 8 = (14/4) / 8 = 14/32

The third step of the CF-based algorithm is to compute the similarity between the user request vector $\vec{R} = (10, 10, 10)$ and the vectors corresponding to the raters of the functionality for which adaptation is requested. Recall from section 3 that the similarity is computed using the cosine similarity metric, using the formula

$$r(\vec{X}, \vec{Y}) = \frac{\sum_{k=1}^{n} (\vec{X}[k] * \vec{Y}[k] * d(\vec{X}[k], \vec{Y}[k]))}{\|\vec{X}\| * \|\vec{Y}\|}$$

Therefore, the similarity metric *r* between the rows of Table 7 and the user request vector $\vec{R} = (10, 10, 10)$ are as follows:

$$r(\vec{R}, \overrightarrow{row_2}) = \frac{\sum_{k=1}^{3} (R[k] * \overrightarrow{row_2}[k] * d(R[k], \overrightarrow{row_2}[k]))}{\|\vec{R}\| * \|\overrightarrow{row_2}\|} =$$

$$\frac{\left(6 * 10 * \frac{9}{24}\right) + (9 * 10 * 1) + \left(6 * 10 * \frac{26}{32}\right)}{\sqrt{6^2 + 9^2 + 6^2} * \sqrt{10^2 + 10^2 + 10^2}} = 0.869$$

$$r(\vec{R}, \overrightarrow{row_5}) = \frac{\sum_{k=1}^{3} (R[k] * \overrightarrow{row_5}[k] * d(R[k], \overrightarrow{row_5}[k]))}{\|\vec{R}\| * \|\overrightarrow{row_2}\|} =$$

$$\frac{\left(7 * 10 * \frac{9}{24}\right) + \left(8 * 10 * \frac{14}{24}\right) + (6 * 10 * \frac{14}{32})}{\sqrt{7^2 + 8^2 + 6^2} * \sqrt{10^2 + 10^2 + 10^2}} = 0.607$$

$$r\left(\vec{R}, \overrightarrow{row_6}\right) = \frac{\sum_{k=1}^{3}\left(R[k] * \overrightarrow{row_6}[k] * d(R[k], \overrightarrow{row_6}[k])\right)}{\|\vec{R}\| * \|\overrightarrow{row_2}\|} =$$

$$\frac{\left(6 * 10 * \frac{19}{24}\right) + \left(8 * 10 * \frac{14}{24}\right)}{\sqrt{6^2 + 8^2} * \sqrt{10^2 + 10^2 + 10^2}} = 0.544$$

$$r\left(\vec{R}, \overrightarrow{row_8}\right) = \frac{\sum_{k=1}^{3}\left(R[k] * \overrightarrow{row_8}[k] * d(R[k], \overrightarrow{row_8}[k])\right)}{\|\vec{R}\| * \|\overrightarrow{row_2}\|} =$$

$$\frac{\left(9 * 10 * \frac{19}{24}\right) + (9 * 10 * 1) + (8 * 10 * \frac{5}{8})}{\sqrt{9^2 + 8^2 + 8^2} * \sqrt{10^2 + 10^2 + 10^2}} = 0.844$$

Subsequently, we compute each service's rating prediction using, as discussed in section 3, the rating prediction formula

$$p\left(\vec{R}[k]\right) = \frac{\sum_{\vec{N} \in raters(\vec{R}[k])}\left(\vec{N}[k]\right) * r(\vec{R}, \vec{N})}{\sum_{\vec{N} \in raters(\vec{R}[k])} r(\vec{R}, \vec{N})}$$

And therefore we obtain

p(Lufthansa) = (6*0.869 + 0.844*9)/ (0.869+0.844) = 7.48

p(AirFrance) = 7* 0.607 / 0.607 = 7

p(SwissAir) = 6* 0.544 / 0.544 = 6

These values are then normalized to the range [0,1] by dividing by the maximum possible value of a rating, in our case 10:

p_n(Lufthansa) = (6*0.869 + 0.844*9)/ (0.869+0.844) = 0.748

p_n(AirFrance) = 7* 0.607 / 0.607 = 0.7

p_n(SwissAir) = 6* 0.544 / 0.544 = 0.6

Since the number of possible solutions is less than 20, all solutions are retained. Subsequently, similarly to the case of the QoS-based algorithm, we formulate the integer programming problem, i.e. to maximize the overall utility function

$$OUV_{CF} = (\sum_{i=1}^{F}\sum_{j=1}^{L(i)} prediction\left(s_{i,j}\right) * x_{i,j})/F$$

subject to the constraint

$$\sum_{j=1}^{L(i)} x_{i,j} = 1, 1 \leq i \leq F$$

Since the number of functionalities requiring adaptation F is equal to 1, the integer optimization problem is reduced to

$$OUV_{CF} = \sum_{j=1}^{3} prediction(s_{AirTravel,j}) * x_{AirTravel,j}) =$$

$$\text{prediction(AirFrance)} * x_{\text{AirFrance}} +$$
$$\text{prediction(Lufthansa)} * x_{\text{Lufthansa}} +$$
$$\text{prediction(SwissAir)} * x_{\text{SwissAir}}$$

subject to the constraint

$$x_{\text{AirFrance}} + x_{\text{Lufthansa}} + x_{\text{SwissAir}} = 1$$

The three possible solutions to this problem are as shown in Table 8.

Solution	CF-score
$x_{AirFrance}$=1, $x_{Lufthansa}$=0, $x_{SwissAir}$=0	0.7
$x_{AirFrance}$=0, $x_{Lufthansa}$=1, $x_{SwissAir}$=0	0.748
$x_{AirFrance}$=0, $x_{Lufthansa}$=0, $x_{SwissAir}$=1	0.6

Table 8. Solutions proposed by the CF-based algorithm

4.3 Combining the results

Finally, we apply the CombMNZ metasearch algorithm to compute the final score of each solution. The CombMNZ algorithm adds the individual scores and multiplies the result by the number of algorithms proposing each solution. In our case, all solutions are proposed by both the QoS-based algorithm and the CF-based algorithm, hence the results are as follows:

$CombMNZ_{AirFrance}$ = (0.913 + 0.7) * 2 = 1.613 * 2 = 3.226
$CombMNZ_{Lufthansa}$ = (0.888 + 0.748) * 2 = 1.636 * 2 = 3.272
$CombMNZ_{SwissAir}$ = (0.788 + 0.6) * 2 = 1.388 * 2 =2.776

We can observe that the $CombMNZ_{Lufthansa}$ score is the maximum among all scores, hence the Lufthansa service will be chosen for the particular adaptation, so the user scenario execution plan becomes «Lufthansa, YouthHostel, ChampionsLeague».

5. Conclusions

In this technical report we have presented an approach to the adaptation of BPEL scenario execution, combining QoS-based criteria with a CF-based approach. The CF-based approach allows for considering the ratings of the users for individual services, hence complementing the objective and measurable QoS-based criteria with user views, reflecting the real-world experience from using the services. We have also given a running example on the algorithm, to facilitate the understanding of its operation.

6. References

[1] 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)

[2] Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An Approach for QoS-aware Service Composition based on Genetic Algorithms. In: 2005 Cnference on genetic and evolutionary computation, H-G. Beyer, U-M. O'Reilly (eds.), 1069-1075 (2005)

[3] Shao, L., Guo, Y., Chen, X., He, Y.: Pattern-Discovery-Based Response Time Prediction. In: Advances in Automation and Robotics, vol. 2 LNEE, vol. 123, 355-362 (2012)

[4] Duan, Y., Huang, Y.: Research on availability prediction model of web service. In: 2011 International Conference on Computer Science and Service System, 1590–1594 (2011)

[5] Paolucci, M., Kawamura, T., Payne, T., Sycara, T.,: Semantic Matching of Web Services Capabilities. In: International Semantic Web Conference, 333-347 (2002)

[6] Yu, J., Sheng, Q., Han, J., Wu, Y., Liu, C.: A semantically enhanced service repository for user-centric service discovery and management. In: Data & Knowledge Engineering, vol. 72, 202-218 (Feb. 2012)

[7] Margaris, D., Vassilakis, C., Georgiadis, P.: Adapting WS-BPEL scenario execution using collaborative filtering techniques. In: IEEE 7th International Conference on Research Challenges in Information Science, R. Wieringa, et al. (eds), Paris, France (2013)

[8] Arpacı, A.E., Bener, A.B.: Agent Based Dynamic Execution of BPEL documents. In: ISCIS 2005, LNCS 3733, P. Yolum, et al. (eds.), 332 – 341 (2005)

[9] Alrifai, M., Risse, T.: Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition. In: 18th international conference on World wide web (WWW '09), Th. Karagiannis and M. Vojnovic (Eds.), 881-890 (2009)

[10] Yu, T., Lin, K.J.: Service selection algorithms for Web services with end-to-end QoS constraints. In: Information systems and e-business management vol. 3(2), 103-126 (2005)

[11] Saric, A., Hadzikadic, M., Wilson, D: Alternative Formulas for Rating Prediction Using Collaborative Filtering. In: Proceedings of the 18th International Symposium on Foundations of Intelligent Systems, 301-310 (2009)

[12] Bramantoro, A., Krishnaswamy, S., Indrawan, M.: A semantic distance measure for matching web services. In: 2005 International Conference on Web Information Systems Engineering,. Ngu, A.H.H et al. (eds.), 217-226 (2005)

[13] Chelminski, P., Coulter, R.: An examination of consumer advocacy and complaining behavior in the context of service failure. In: Journal of services marketing, vol. 25(5), 361–370 (2011)

[14] Montague, M., Aslam, J.A.: Relevance score normalization for metasearch. In: CIKM 2001, H. Paques et al. (eds), 427-433 (2001)