A Collaborative Filtering Algorithm with Clustering for Personalized Web Service Selection in Business Processes

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*— Recommender systems aim to propose items that are expected to be of interest to the users. As one of the most successful approaches to building recommender systems, collaborative filtering exploits the known preferences of a group of users to formulate recommendations or predictions of the unknown preferences for other users. In many cases, collaborative filtering algorithms handle complex items, which are described using hierarchical tree structures containing rich semantic information. In order to make accurate recommendations on such items, the related algorithms must examine all aspects of the available semantic information. Thus, when collaborative filtering techniques are employed to adapt the execution of business processes, they must take into account the services' Quality of Service parameters, so as to generate recommendations tailored to the individual user needs. In this paper, we present a collaborative filtering-based algorithm which takes into account the web services' QoS parameters in order to tailor the execution of business processes to the preferences of users. An offline clustering technique is also introduced for supporting the efficient and scalable execution of proposed algorithm under the presence of large repositories of sparse data.**

Keywords*— *collaborative filtering; clustering; business processes; web services; quality of service; hierarchical tree; performance

I. INTRODUCTION

A web service is an accessible application that other applications and humans can discover and trigger to satisfy multiple needs [25]. One of the major strengths of web services is their ability to be composed into *composite services*, which model high-level business processes. In general, providing complete business processes, rather than making available individual services, is essential and suits better user needs. Composition addresses the situation of a user request that cannot be satisfied by any available service, whereas a composite service obtained by combining a set of available services might be used to satisfy the user request [26].

The predominant way of specifying executable business processes in the context of service oriented architecture nowadays is the WS-BPEL language [1]. Executable business process specifications include the designations of the services that will be invoked to realize the business process, control flow constructs (e.g. conditional execution and loops) as well as variables to store, manipulate and communicate data. In executable business process specifications however, the services that will be invoked are selected at design time, i.e. when the specification is created; in the current era however, business processes need to be agile and adapt to the needs of the user. This is particularly true in the context of service oriented architecture, where each service may be offered by different providers under different quality of service (QoS) terms: for example, many banks may offer the "money transfer" service with different execution time, commission cost, security levels etc., and the user would want to either directly specify a specific provider to carry out the task or define a selection policy, according to which the runtime environment would select the "best matching" service implementation.

QoS-driven service selection leverages agility and tailorability of business processes, it however fails to consider the satisfaction of service users in the "real world". For instance, a courier service may offer cheap rates and short delivery time, however customers may be dissatisfied because parcels arrive broken or wet, and this aspect is not reflected in the QoS parameters. To this end, collaborative filtering-based techniques have been proposed to drive the adaptation process [19][27] or as complementary means to QoS-based service selection [28]. Collaborative filtering (CF) synthesizes the informed opinions of humans (i.e. opinions that encompass the aspect of satisfaction), to make personalized and accurate predictions and recommendations [29]. In the context of collaborating filtering, personalization is achieved by considering ratings of "similar users" (in our case, uses of individual services by similar users), under the collaborative filtering's fundamental assumption that if users X and Y have similar behaviors (e.g., buying, watching, listening – in our case, selecting the same services) on some items, they will act on other items similarly [5]. Many variations of collaborative filtering algorithms and systems exist; however, most of them usually take two steps: (a) look for users who share the same rating patterns with the active user (the user who the prediction is for) and (b) use the ratings from those like-minded users found in (a) to calculate a prediction for the active user.

Collaborative filtering techniques have been proved to exhibit degraded performance in the presence of large volumes

of data; to tackle this shortcoming, clustering schemes have emerged [30][31]. Clustering schemes organize users and/or items into clusters based on appropriate characteristics, and in order to make a recommendation they first compute the similarity between the target user and clusters, allowing for rapidly locating similar elements and limiting processing to these only.

In this paper we contribute to the state of the art of business process adaptation through service recommendation by presenting a novel approach for performing personalized web service selection in business processes realized in the service-oriented architecture; the presented approach employs collaborative filtering techniques enhanced with a clustering scheme to promote scalability, while it additionally takes into account QoS specifications provided by the user to further refine and personalize service selection. The QoS aspects are taken into account in a twofold fashion, firstly by considering user-defined bounds for service selection and secondly by incorporating QoS attribute closeness into the similarity metric employed by the collaborative filtering algorithm; the latter aspect extends the practices used in published works such as [20][27], where similarity is computed taking into account only the service functionality. The presented approach considers the particular characteristics of web services and SOA business processes, such as service functional equivalence, QoS aspects of the services and explicit specification of desired service providers, and these characteristics are considered both in the clustering scheme and the recommendation procedure.

The rest of the paper is structured as follows: section II overviews related work, while section III presents the QoS and collaborative filtering concepts used in this paper. Section IV presents the clustering method employed, while section V presents the algorithm for service recommendation. Section VI evaluates the proposed approach in terms of performance and quality of solutions formulated, in order to validate it and substantiate its feasibility. Finally, section VII concludes the paper and outlines future work.

II. Related Work

Personalized web service selection in the context of business processes has been an item research for almost a decade now. Most works allow the user to specify a QoS-based service selection policy [3][4][6][7][8][9][10][11], typically providing bounds and weights of QoS attributes such as response time, cost, availability [14] and so forth; then the adaptation mechanism tries to find a suitable service composition, to realize the requested business process in a fashion best suiting the designated policy. To accomplish this task, existing approaches follow two main strategies [2]: (i) *horizontal adaptation*, where the composition logic has been specified beforehand (e.g. by means of an executable business process specification) and the main adaptation task is to select and invoke the service best matching the user's QoS policy and (ii) *vertical adaptation*, where the adaptation mechanism may choose the composition logic, under the constraint of delivering the requested functionality. QoS bounds may be either defined at *local level* (i.e. constraints pertain to the individual operations invoked within the business process) or at *global level* (i.e. constraints pertain to the composition as a whole).

Within all personalized web service selection approaches, the adaptation mechanism needs to represent the functionality offered by the individual service implementations, so as to identify the pool of services that can be used to realize a specific task. Some approaches only maintain information about *service equivalence* [12], i.e. matching among services that deliver exactly the same functionality. A more sophisticated scheme is presented in [13], according to which a service S_1 may be related to service S_2 through one of the following *subsumption relations*: (i) S_1 *exact* S_2, for services having identical functionality, e.g. they both reserve a hotel room, (ii) S_1 *plugin* S_2 where service S_1 is more specific than S_2 and can be therefore used in its place; for example if S_1 is "reserve a hotel room" while S_2 is "reserve accommodation" (including hotels, apartments, guesthouses etc.), the goal of S_2 can be accomplished through S_1 since S_1 does book accommodation, (iii) S_1 *subsume* S_2 when S_1 is more generic than S_2 and therefore cannot be always used in its place (e.g. if S_1 reserves accommodation and S_2 reserves a hotel room, we cannot unconditionally use S_1 instead of S_2 since the use of S_1 may result in booking an apartment instead of a hotel room. Note that S_1 *subsume* S_2 does not always preclude the use of S_1 instead of S_2, however this can only be done under certain conditions. In this work, we will not consider this case) and (iv) S_1 *fail* S_2 if none of the *exact*, *plugin*, *subsume* relation applies. A service matchmaking mechanism relying on subsumption relations is presented in [50]; this mechanism operates over an OWL-S ontology representing subsumptions and computes the degree of semantic matching for a given pair of service advertisement and request. Using subsumption relations offers the advantage of broadening the set of choices regarding service selection (a task S_2 may be delivered by any service S_1 for which *[*S_1 *exact* S_2 *or* S_1 *plugin* S_2*]*, while equivalence relations would limit the choices to services S'_1 for which *[*S'_1 *exact* S_2*]*). Furthermore, subsumption relations organize services in a semantic tree, which enables the use of standard similarity metric computations that are essential for the operation of clustering schemes and collaborative filtering techniques; hence in this work we will adopt the use of subsumption relations.

The QoS attributes of the available services and their equivalence or subsumption relations need to be stored in a suitable repository, in order to be made available to the adaptation mechanism. METEOR-S [14], WSMO [15] and OPUCE [36] can be used to implement this repository. All these schemes adopt ontologies for the representation of service-related information, hence they provide the necessary expressive power to encompass all necessary information.

In the domain of recommender systems, various approaches for generating recommendations have been developed. It has been proven that the CF-based recommendation approach is the most successful and widely used approach for recommendation systems [5]. CF recommends items for a particular user using the opinions of other people based on the assumption that people with similar tastes will rate things similarly [16]. It can be further

divided into user-based and item based CF approaches [17]. User-based CF first finds a set of nearest neighbors of a target user by computing correlations or similarities between users. The prediction value of unknown items to the target user is then computed according to his/her nearest users. In contrast, item-based CF attempts to find a set of similar items that are rated by different users in some similar way. Then, for a target item, predictions can be generated, for example, by taking a weighted average of the active user's item ratings on these neighbor items. Item-based CF has been shown to achieve prediction accuracies that are comparable to or even better than user-based CF algorithms [18].

The basic assumption in CF approaches is that there are sufficient historical data for measuring similarity between items. This assumption does not hold however in various application domains, therefore, CF approaches exhibit the problems of sparsity (a situation that transactional data are lacking or are insufficient) and cold-start (a situation in which the system cannot draw any inferences for users or items about which it has not yet gathered sufficient information). To improve the prediction accuracy, hybrid recommender systems which integrate other information sources with CF approaches have been developed. Because the computation of item similarities is independent of the methods used for generating predictions, multiple knowledge sources, including structured semantic information about items, can be brought to bear in determining similarities among items [32]. The integration of semantic similarities for items allows the system to make inferences based on the underlying reasons for which a user may or may not be interested in a particular item [32]. The approach taken in [33] uses a domain ontology, and maps items to a set of concepts corresponding to the ontology's nodes; in order to find similar users, semantic match against the domain ontology is employed. [34] integrates the techniques of semantic similarity and the traditional item-based collaborative filtering to handle recommendation issues of one-and-only items in e-government services, for instance in suggesting trade exhibitions. [35] proposes a hybrid semantic recommendation system to provide personalized government to business (G2B) e-services, in particular, business partner recommendation e-services for Australian small to medium enterprises (SMEs); this is accomplished by first formulating a product semantic relevance model and subsequently developing a hybrid semantic recommendation approach which combines item-based collaborative filtering (CF) similarity and item-based semantic similarity techniques.

Works in [11][19][27][28] use CF techniques for service selection. The goal of [11] is to predict Web service QoS values based on past Web service QoS information collection from different service users. [19] examines the use of CF for suggesting web services to the users; the work in [19] however considers individual services, not business processes (it is however capable of suggesting service compositions delivering a specific functionality). [27] employs CF to adapt the execution of WS-BPEL processes, while [28] extends the work of [27] by combining CF-based recommendations with a QoS-based recommendations using the metasearch paradigm. These approaches however employ standard user-based CF-techniques, hence they exhibit limited scalability.

As far as clustering is concerned, it has been extensively studied by researchers in psychology, statistics, biology and other domains. [37] and [38] provide surveys of clustering algorithms, while surveys for specific clustering algorithm categories also exist, such as [39] which focuses on partitional clustering algorithms. Recent developments in the field are clustering algorithms for mining large databases or *big data*; relevant surveys can be found in [40] and [41].

III. QOS AND CF CONCEPTS

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

A. QoS concepts

The QoS aspects of web services are typically described by means of attributes, to which values are assigned. [42] surveys numerous web service QoS models, encompassing as much as 161 attributes. In this work, for conciseness purposes we will consider only the QoS attributes responseTime (rt), cost (c) and availability (av), adopting their definitions from [14]. The presented algorithms can be easily extended to consider more attributes, hence no loss of generality occurs.

The QoS policy for service selection within the business process specification may include an upper and a lower bound for each QoS attribute, i.e. for each service s_i invoked in the context of the business process two vectors are defined $MIN_j=(min_{rt,j}, min_{c,j}, min_{av,j})$ and $MAX_j=(max_{rt,j}, max_{c,j}, max_{av,j})$. Additionally a weight vector $W=(rt_w, c_w, av_w)$, indicating how important each QoS attribute is considered in the context of the particular operation invocation is supplied. Weights apply to the whole business process, 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 [3]. The values of QoS attributes are assumed to be encoded in a "larger values are better" scheme, e.g. a service having *responseTime= 8* is actually *faster* than a service having *responseTime = 4*.

B. Subsumption relation representation

In order to perform adaptation we need to use a formal representation of the services' functionality, and in particular a representation of which services are able to realize a specific task. As noted in section II, the use of subsumption relations offers advantages over the use of plain equivalence relationships, hence the use of subsumption relations is adopted in this paper. The representation of subsumption relationships between service categories (or *abstract tasks*, in horizontal adaptation terminology) is addressed in [13] and [20] through the use of trees; according to this approach, generic service categories are located towards the tree root and specific service categories are placed towards the leafs, and generic service categories are connected with their specialization categories through *is-a* links. Since in this work we are interested not only in service categories but in concrete services also (because concrete services will be the ones invoked during business process execution), we extend the tree scheme used in [13] and [20] by accommodating *instance-of* arcs, complementary to the *is-a* arcs. An *instance-of* arc is drawn in the subsumption relationship tree between service category C and concrete

service *S*, if and only if *S* implements exactly the functionality specified by category C.

To illustrate this representation, let us consider the case of a travel planning scenario containing the following activities: ticket booking, hotel booking, and room service drink ordering. In this case the subsumption relationships, including categories and concrete services could be arranged as shown in Fig 1 (categories are denoted using a folder icon; concrete services are denoted using a bullet mark).

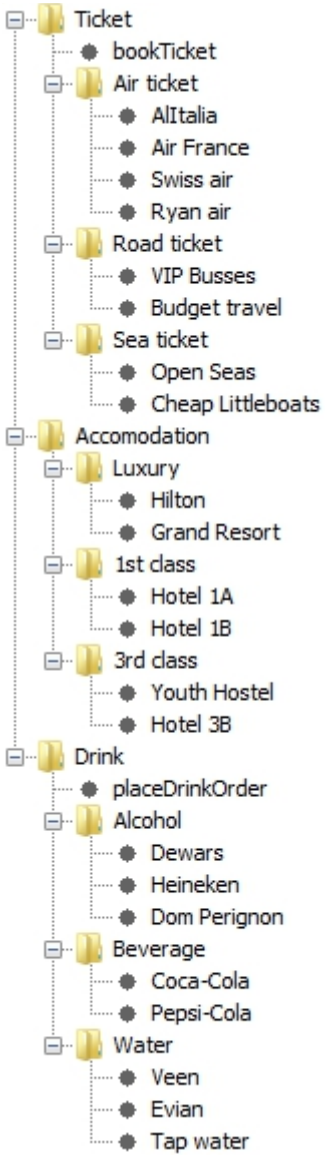

Fig. 1. Subsumption relationships for the travel planning scenario

In order to perform adaptation, we need to locate, using the subsumption relationship representation, the services that are able to realize a specific task, and then choose among them the most prominent one, according to the adaptation policy. As noted in section II, a service can be used to realize a task if it is connected to the task with either the *exact* or the *plugin* relation. In the presence of service categories (*abstract services*) and concrete services, the four rules listed below can be used to compute the services that are candidate to deliver a specific task, which is designated either via a category (abstract service) or a specific service. (In the following, *c* represents a category, while s_1 and s_2 represent services.)

- *Rule* C_{ex}*: c exact* s_1 iff *c* is the immediate parent of s_1 (e.g. Air travel and Swiss air in Fig. 1)
- *Rule* C_{pl}*: c plugin* s_1 iff *c* is an ancestor of s_1 (e.g. Ticket and Swiss air in Fig. 1)
- *Rule* S_{ex}*:* s_1 *exact* s_2 iff ∃ *c*: *c* is the immediate parent of s_1 and *c* is the immediate parent of s_2 (e.g. Air France and Swiss air in Fig. 1).
- *Rule* S_{plg}*:* s_1 *plugin* s_2 iff ∃ *c*: *c* is the immediate parent of s_1 and *c* plugin s_2 (e.g. *bookTicket* and *VIP Busses* in Fig. 1).

In all other cases it is not possible to perform an unconditional substitution, therefore a *fail* result is computed.

QoS attributes of concrete service implementations can be straightforwardly accommodated in the representation requested in Fig. 1: it suffices to attach to each node corresponding to a concrete service a vector of the form $QoS_S=(rt_s, c_s, av_s)$, to express the values of the specific service's QoS attributes.

C. Explicit service invocations bindings

In real-world business processes, the consumer is allowed to explicitly choose which implementations will carry out some service(s) s/he needs in the context of the business process. For example, in our travel planning scenario, the user may request that accommodation booking is realized through the Hilton service.

It is also possible that the user requests to skip the execution of certain functionalities, e.g. a user may specify that no drink is ordered; this is typically handled via a conditional execution construct within the executable business process specification. For simplicity purposes, in this paper we will assume that if a service should not be invoked in the context of a particular execution, this is designated through a specific input parameter and therefore the condition within the conditional execution construct has the simplified form

<condition>$orderDrink = false</condition>

Finally, functionalities that are neither explicitly bound to a specific service implementation, nor are designated as "not to be executed" are subject to adaptation, using the algorithm described in section IV, below.

D. Past executions repository

Collaborative filtering algorithms rely on the existence of user evaluations (ratings) or choices on items. A *ratings matrix* [16] is used to store this information; each row in the matrix corresponds to a user, while each column corresponds to an item. Given that the goal of the proposed algorithm is to perform personalized service selections in the context of business process executions, a row in the matrix corresponds to a particular execution of the business process and a column in the matrix corresponds to a concrete service implementation. We will use the term *past executions repository* to refer to this matrix. Cell *(i,j)* in the past executions repository will be set to *true* if service s_j was used in the i^{th} execution of the business process, otherwise it will be set to the value of *false*.

Ratings matrixes tend to be sparse [16]: for representational compactness purposes in this paper we will use a modified notation according to which each column in the past executions repository corresponds to a functionality included in the scenario (e.g. room booking, ticket booking, drink ordering), each row corresponds to a past execution and the value of cell *(i,j)* designates the service that was used to realize task *j* within the i^{th} execution of the business process. If task *j* was skipped in the context of the i^{th} business process execution (because the user designated that the functionality should not be executed), the value of cell *(i,j)* will be set to *null*.

Table I illustrates a service usage pattern repository for the travel planning scenario. In executions 1-6 all functionalities were invoked, while in execution 7 the hotel accommodation service was skipped.

TABLE I. EXAMPLE PAST EXECUTIONS REPOSITORY

# exec	Travel	Hotel	Drink
1	Swiss	Hilton	Heineken
2	Alitalia	Hilton	Heineken
3	Ryanair	Hotel_1a	Heineken
4	Alitalia	Youth_Hostel	Dom_Perignon
5	Ryanair	Youth_Hostel	Tap_Water
6	Budget_Travel	Hotel_3B	Tap_Water
7	Open_Seas		Evian

E. Similarity and distance metrics for services

Both in the clustering method and in the collaborative filtering algorithm, it is necessary to compute how similar (or distant) two services are. Service similarity and distance metrics will be used in these contexts to compute the similarity and dissimilarity of two executions (an execution *e* of a business process is effectively an n-dimensional vector of services e=(s_1, s_2, …, s_n), where the i^{th} element e[i]=s_i corresponds to the service that realized the i^{th} task within the business process), either to decide whether they should be clustered together or to determine if a past execution is a prominent recommender for the current adaptation. In this paper, we consider two dimensions to measure similarity and distance between two services:

(a) their *semantic similarity*, corresponding to whether they perform the same task; for instance the semantic similarity between *Ryan Air* and *Swiss Air* (cf. Fig. 1) is high, since both services realize the same functionality (air travel), while services *Evian* and *Coca-Cola* are less similar, since the first realizes the functionality *Water* whereas the second realizes the functionality *Beverage*. Note however that they do bear some similarity, since they both realize the functionality *Drink*.

(b) their QoS characteristics likeness, corresponding to how close are the non-functional parameters under which the services realize the task. In regards to this aspect, *Ryan Air* –being an economy airline– is not similar to *Swiss air* (a full-service airline), although they deliver the same functionality, since their cost and availability deviate highly. On the other hand, Veen (an expensive, Finish water) is similar to Dom Perignon (an expensive champagne) since among the services in their categories (water and alcohol, respectively) these services exhibit the highest cost, the worst response time and comparable ranks regarding their availability (cf. Table II).

TABLE II. SAMPLE QOS VALUES WITHIN THE REPOSITORY

Service	responseTime	cost	availability
Dewars	6	3	8
Heineken	7	8	7
Dom Perignon	6	1	9
Coca cola	8	8	7
Pepsi	8	8	8
Veen	8	2	9
Evian	8	5	8
Tap water	8	10	6
Hilton	7	2	7
Grand Resort	7	3	7
Hotel_1A	8	5	5
Hotel_1B	8	6	5
Youth_Hostel	5	9	5
Hotel_3B	5	8	5
Alitalia	8	7	4
AirFrance	8	6	9
Swiss	10	3	10
Ryanair	9	9	3
VIP_Buses	7	3	7
Budget_Travel	6	9	7
Open_Seas	6	6	7
Cheap_LittleBoat	5	10	7

Regarding the semantic dimension, we adopt the semantic similarity distance metric between two services proposed in [20]; according to [20], the semantic similarity between two services is:

$$ssim(s_1,s_2) = C - lw * PathLength - NumDownDirection$$

where:

- *C* is a constant set to 8 [20][24]
- *lw* is the level weight for each path within the subsumption relation tree, and it depends on the depth of the tree. To compute *lw*, we use the formula $= \frac{(n-ln-1)}{n}$, where ln is the level of the service in the subsumption relation tree and *n* is the tree height.
- *PathLength* is the number of edges counted from service s_1 to service s_2 and
- *NumDownDirection* is the number of edges counted in the directed path between service s_1 and s_2 and whose direction is towards a lower tree level.

We further normalize this semantic similarity metric by dividing it by *C* (i.e. 8), hence its value is always in the range [0, 1]. Combining all the above the semantic similarity between services s_1 and s_2 is given by the formula $ssim(s_1, s_2) = \frac{8 - lw * PathLength - NumDownDirection}{8}$.

Regarding the QoS aspect, the distance between two services is computed using the Euclidean distance metric; in the computation, each QoS dimension is weighted using the QoS attribute weight specified for the current adaptation (c.f. section III.A). Furthermore, the attributes values are normalized by dividing them with the maximum value of the attribute within the corresponding category, in order to reflect how close to the maximum value within the category the specific value is; this is

analogous to the score normalization performed in metasearch algorithms [45]. Combining all the above,

$$qdist(s_1, s_2) = \sqrt{\sum_{q \in \{cost, av, respTime\}} \left(\frac{q(s_1)}{q_{max}(cat(s_1))} - \frac{q(s_2)}{q_{max}(cat(s_2))} \right)^2 * w_q}$$

where $q(s_i)$ denotes the value of QoS attribute q (cost, availability, response time) for service s_i, and w_q is the weight assigned to QoS attribute q. $q_{max}(cat(s_i))$ is the maximum value present in the repository regarding QoS attribute q under the category in which s_i is a direct child; for example, to compute $cost_{max}(Ryan\ air)$ we (a) locate the category of *Ryan air* (Air ticket), (b) find the children of the category (AlItalia, Air France, Swiss air and Ryan Air), (c) extract their costs (7, 6, 3, 9) and (d) compute the maximum value (which is equal to 9).

The QoS-based similarity of two services is then computed as

$$qsim(s_1, s_2) = 1 - qdist(s_1, s_2).$$

Combining the semantic similarity with the QoS-based similarity, we compute the overall similarity metric of two services which is

$$sim(s_1, s_2) = ssim(s_1, s_2) * qsim(s_1, s_2)$$

while the overall distance is calculated as

$$dist(s_1, s_2) = 1 - sim(s_1, s_2)$$

IV. Clustering Method

As noted in the previous sections, the approach proposed in this paper uses clustering to cater for scalability with the size of the past executions repository. The computation of the clusters is performed in an off-line fashion, and the clustered repository is made available to the recommendation algorithm (described in section V) as soon as the computation is complete; therefore, the performance of the clustering technique does not penalize the recommendation process. The cluster computation method uses the CLARA clustering algorithm [44] to formulate clusters; since the number of clusters K that will deliver the optimal clustering performance is not however known a priori, the iterated local search paradigm [47] is used to reduce the search range for K, using the Silhouette coefficient [44] as a solution quality metric. The steps of the clustering method are described in the following paragraphs.

Step 1: The potential range of the optimal cluster number is determined and the initial starting points of the iterated local search are computed. Since for the optimal number of clusters C_{opt} it is expected that $Copt \approx \sqrt{N/2}$ [43], where N is in our case the number of elements in the past executions repository *PER*, the range $[\frac{\sqrt{N/2}}{2}, 2 * \sqrt{N/2}]$ will be tested to find the value of C_{opt}. We then extract the initial starting points of the iterated local search with logarithmic cardinality from the above range as follows:

1. The distance between the starting points is set to
$$d = \log_{10}(2 * \sqrt{N/2} - \frac{\sqrt{N/2}}{2}) * 10$$
2. The set of initial starting points is set to
$$ISP = \{\frac{\sqrt{N/2}}{2} + \frac{d}{2}, \frac{\sqrt{N/2}}{2} + \frac{3d}{2}, \frac{\sqrt{N/2}}{2} + \frac{5*d}{2}, \ldots, 2 * \sqrt{\frac{N}{2}} - \frac{d}{2}\}.$$

Step 2. A hill climbing algorithm is executed for each $isp \in ISP$. The hill climbing procedure is depicted in Fig. 2:

```
hillClimbing(PER, numClusters, d)
    K = numClusters
    optClustering = CLARA(PER, numClusters)
    optShilouetteValue = Silhouette(optClustering)
    range = d
    DO
        sampleDistance = log10(range)
        neighbors = {K – range / 2 + sampleDistance / 2,
                     K – range / 2 + 3 * sampleDistance / 2,
                     …
                     K + range / 2 - sampleDistance / 2}
        improvement = FALSE
        FOR EACH n IN neighbors
            testClustering = CLARA(PER, n)
            testShilouetteValue = Silhouette(testClustering)
            if (testShilouetteValue > optShilouetteValue)
                improvement = TRUE
                optClustering = testClustering
                optShilouetteValue = testShilouetteValue
                K = n
            END IF
        END FOR
        range = sampleDistance
    UNTIL improvement==FALSE
    RETURN K, optClustering, optShilouetteValue
```

Fig. 2. Hill climbing algorithm

Effectively, for each starting point *isp*, the CLARA clustering algorithm [44] is used to cluster the elements of *PER* into *isp* clusters, and the Silhouette coefficient [44] of the computed clustering is computed (the computation procedure is described below). Subsequently, the neighborhood of the starting point is searched for a better solution, and the best one found replaces the implementing the *steepest ascent* hill climbing paradigm [48], and the procedure continues until no improvement to the solution is possible. The candidate solutions examined at each step are generated by uniformly sampling points around the current best solution. Initially, the range from which the points are sampled is set to the distance d between starting points calculated in step 1, this range is divided into $log(d)$ intervals, and the center of each interval is then tested to determine if it provides a better solution. If the neighborhood search leads to the discovery of a better solution, the search continues in the neighborhood of the newly found solution, decreasing however the range logarithmically, so as to elaborate on the close neighborhood of the discovered solution.

When running the CLARA algorithm, it is required that a metric is provided to calculate the dissimilarity between two nodes (in our case, past executions in the *PER*). This metric calculation is based on the Sørensen similarity index [21] (alternatively known as Dice's coefficient [22]), according to which the similarity of two sets $A=\{a_1, a_2, \ldots, a_n\}$, $B=\{b_1, b_2, \ldots, b_m\}$, is equal to $S(A,B) = \frac{2*|A \cap B|}{|A|+|B|}$; the metric is suitably modified to fit a domain with similarities, such as those defined

in section III.E. The modification follows the approach used in the fuzzy set similarity index calculation, where the cardinality of the intersection of two sets (i.e. the nominator in the Sørensen similarity index formula) is computed as the sum of the probabilities that a member belongs in both sets [23]. Correspondingly, when set member similarity between two past executions of a business process $pe_1=\{s_{1,1}, s_{1,2}, \ldots, s_{1,n}\}$ and $pe_2=\{s_{2,1}, s_{2,2}, \ldots, s_{2,n}\}$ is considered, the nominator of the fraction is replaced by $2 * \sum_i sim(s_{1,i}, s_{2,i})$; therefore the formula for computing the similarity between two past executions is shaped as $similarity(pe_1, pe_2) = \frac{2*\sum_i sim(s_{1,i},s_{2,i})}{|pe_1|+|pe_2|}$. In this computation, if exactly one of $s_{1,i}$ and $s_{2,i}$ is null (signifying that in the corresponding task was not executed in the specific past execution), then $sim(s_{1,i}, s_{2,i})$=0. If both $s_{1,i}$ and $s_{2,i}$ are null (i.e. none of the tasks was executed), then $sim(s_{1,i}, s_{2,i})$=1.

Having the $similarity(pe_1, pe_2)$ metric available, the corresponding dissimilarity metric is computed as $dissimilarity(pe_1, pe_2) = 1 - similarity(pe_1, pe_2)$.

Step 3: The clusterings that have been produced by the execution of the *hillClimbing* procedure of step 2 are collected, and the one having the greatest Silhouette coefficient value is chosen.

As noted above, the Silhouette coefficient [44] is used as a quality metric to compare different clusterings. The Silhouette coefficient of a clustering $C=\{C_1, C_2, \ldots, C_n\}$, this is computed as follows:

1. For each past execution $pe \in PER$, let *pe* belong to cluster C_i. We compute the average dissimilarity *a(pe)* of *pe* to all other past executions belonging to the same cluster C_i, i.e. $a(pe) = \frac{\sum_{pe' \in C_i, pe' \neq pe} dissimilarity(pe,pe')}{|C_i|-1}$. Dissimilarity between two past executions (pe, pe') is computed using the formula described in step 2.
   Additionally, for each cluster $C_j \in C$: $C_j \neq C_i$, we compute the average dissimilarity $d_j(pe)$ to all past executions belonging to cluster C_j, and we define $b(pe) = \min_j d_j(pe)$.
   Finally, the silhouette value for *pe* is defined as
   $$s(pe) = \frac{b(pe) - a(pe)}{\max(a(pe), b(pe))}$$
   With the range of the silhouette weights being [-1, 1], a high value of *sw(pe)* (close to 1) means that *pe* has been correctly clustered, a low value (close to -1) means that *pe* would be better clustered in a neighbor cluster while a value close to 0 indicates a borderline assignment.
2. Having computed the silhouette value of all past executions *pe*, we compute the silhouette value for each cluster $C_i \in C$ as the average of the silhouette values of past executions within C_i, i.e. $s(C_i) = average_{pe \in C_i}(s(pe))$.
3. Finally, the silhouette value of clustering *C* is computed as the average of the silhouette values of all clusters C_i within *C*, i.e. $s(C) = average_{C_i \in C}(s(C_i))$.

After the clusters have been built, an index is created in each cluster. This index maps the elements of the cluster to the leaf nodes of the subsumption relation tree (c.f. fig 1), so as to facilitate fast retrieval of cluster elements which refer to a particular service implementation.

V. The Service Recommendation Algorithm

Having available the information listed in section III above, upon each execution of a business process the adaptation algorithm determines the concrete services that will realize the tasks for which recommendations are requested for. The selection of the services is performed using a collaborative filtering algorithm, arranging in parallel to satisfy the QoS restrictions specified for the QoS attributes of each task. In the following paragraphs we elaborate on the steps of the algorithm; to illustrate the functionality of the algorithm, we will use the example request excerpt to the travel planning business process depicted in Fig. 3:

```
<businessProcess id="TravelPlanning">
    <bindings>
        <task id="travelTicket" bind="recommend">Air ticket</task>
        <task id="accommodation" bind="explicit">Hilton</task>
        <task id="drink" bind="explicit">Heineken</task>
    </bindings>
    <QoSLimits>
        <QoSBound taskid="travelTicket" attr="cost"
            bound="min">4</QoSBound>
    </QoSLimits>
    <QoSWeights>
        <QoSWeight attr="responseTime">0.1</QoSWeight>
        <QoSWeight attr="cost">0.7</QoSWeight>
        <QoSWeight attr="availability">0.2</QoSWeight>
    </QoSWeights>
</businessProcess>
```

Fig. 3. Example business process execution request

which is essentially read as follows: "I want to stay at Hilton Hotel, order Heineken from room service and I want a recommendation for my air ticket booking. The recommended service's cost must be over 4 (recall that since attributes are coded in a "larger values are better" scheme, a lower bound for cost effectively filters out the most expensive ones) and the QoS weights are response time=10%, cost=70% and reliability=20%". The repository of available services is as listed in table 2. Throughout the example, we will consider that the services' QoS attribute values are as illustrated in Table II, the subsumption relation tree is as depicted in Fig. 1 and the contents of the contents of the past executions repository are as shown in Table I.

Step 1: the adaptation algorithm formulates a task vector $T=(t_1, t_2, \ldots, t_n)$, where each t_i corresponds to a task that is part of the business process. The values of the elements t_i are determined as follows:

- if the task corresponding to element t_i is explicitly bound to a specific service, then the value of t_i is set to the identifier of this service.
- if either the corresponding task will not invoked in the context of the specific business process execution then the value of t_i is set to *null*.

- if a recommendation is requested for the task, then the value of t_i is set to the category of the service for which the recommendation is requested (e.g. *Air Ticket* or *Ticket*). Subsequently, steps 2 and 3 below are executed for each requested recommendation.

  In our example, the task vector would be instantiated to *T=(Air Ticket, Hilton, Heineken)*.

Step 2: In order to formulate the recommendation for $task_i$, the *k-nearest neighbors* to the current request are retrieved from the clustered past executions repository. Following the results of [46], we have set *k=50* (the maximum value of *k* used in [46]). To retrieve the k-nearest neighbors, the similarity of the task vector *T* with the cluster medoids is initially computed. The similarity of the functionality vector with each medoid (which corresponds to a past execution) is computed using the modified version of the Sørensen similarity index [21] described in step 2 of section IV. The cluster with the highest similarity is selected and searched for past executions *pe* that fulfill the criterion

$task_i(request)$ *exact* $task_i(pe)$ *or* $task_i(request)$ *plugin* $task_i(pe)$

and additionally satisfy the QoS bounds set by the user. These are the only rows that are useful for formulating a recommendation for $task_i(request)$, since only these include services that can be unconditionally used to realize the selected task and additionally conform to the user-defined QoS restrictions. Then, their similarity score with the task vector *T* is computed and the past executions attaining the 50 highest scores are the ones retained to be used as "recommenders" in the subsequent steps. In order to retrieve the cluster elements fulfilling the above mentioned criteria, the branch of the subsumption relation tree corresponding to the requested functionality is first located in the cluster's index; subsequently, its descendants satisfying the QoS bounds are found and finally the cluster elements are retrieved through the index pointers. If less than 50 recommenders are found, the search continues to the remaining clusters, in descending order of similarity of the task vector *T* with the cluster medoids. The output of this step is the list of past executions retrieved (up to 50), with each past execution being tagged with the similarity metric between itself and the task vector *T*.

In our example, rows 2-5 of Table 1 would be retrieved, since (a) rows 6-7 result to a *fail* subsumption relation, regarding the ticket functionality "Air ticket" and row 1 does not satisfy the QoS bound (the cost is lower than the specified threshold). The similarity metrics between the retrieved rows and the task vector *T* would be computed as follows (we first detail the computation of the sum in the nominator or the $similarity(pe_1, pe_2)$ metric, which is the complex part of the formula, and then proceed to the computation of the modified Sørensen similarity index value):

$$\sum_i sim(T_i, row2_i) =$$

$$\left(1 - \sqrt{\left(\frac{8.75}{10} - \frac{8}{10}\right)^2 * 0.1 + \left(\frac{6.25}{9} - \frac{7}{9}\right)^2 * 0.7 + \left(\frac{6.5}{10} - \frac{4}{10}\right)^2 * 0.2}\right) * \frac{8 - \frac{2}{3} * 1 - 1}{8}$$

$$+ \left(1 - \sqrt{\left(\frac{7}{8} - \frac{7}{8}\right)^2 * 0.1 + \left(\frac{2}{9} - \frac{2}{9}\right)^2 * 0.7 + \left(\frac{7}{7} - \frac{7}{7}\right)^2 * 0.2}\right) * \frac{8 - \frac{1}{3} * 0 - 0}{8}$$

$$+ \left(1 - \sqrt{\left(\frac{7}{8} - \frac{7}{8}\right)^2 * 0.1 + \left(\frac{8}{10} - \frac{8}{10}\right)^2 * 0.7 + \left(\frac{7}{9} - \frac{7}{9}\right)^2 * 0.2}\right) * \frac{8 - \frac{1}{3} * 0 - 0}{8} =$$

$$0.87 * 0.79 + 1.0 * 1.0 + 1.0 * 1.0 = 2.69$$

and analogously for the remaining rows:

$$\sum_i sim(T_i, row3_i) = 1.95$$

$$\sum_i sim(T_i, row4_i) = 1.35$$

$$\sum_i sim(T_i, row5_i) = 1.39$$

Consequently, the similarity metrics, computed via the modified Sørensen similarity index, between *T* and these rows are:

$similarity(T, row2) = 2 * 2.69 / (3+3) = 0.896$

$similarity(T, row3) = 2 * 1.95 / (3+3) = 0.65$

$similarity(T, row4) = 2 * 1.35 / (3+3) = 0.45$

$similarity(T, row5) = 2 * 1.39 / (3+3) = 0.46$

Step 3: Finally, the algorithm groups the rows retrieved in step 2 by the value of the service implementing the $task_i(request)$ functionality and computes the sum of the modified Sørensen similarity index values within each group. The service corresponding to the group having the greatest sum is then selected to realize the specific task in the context of the current execution.

In our example, rows 2 and 4 form one group corresponding to service *Alitalia* and achieving an overall score of 1.346, while rows 3 and 5 form a second group corresponding to service *Ryanair* with an overall score of 1.11. Thus, service *Alitalia* is selected to realize the *AirTravel* task in the context of the current scenario execution.

Steps 2-3 are repeated for each functionality $funct_i(request)$ for which a recommendation is requested.

VI. Experimental Evaluation

In this section, we report on our experiments aiming to evaluate the performance of the proposed approach, both in terms of optimization time (the time needed to compute the requested recommendations) and the quality of the solution. For our experiments we used two machines. The first machine was equipped with one 6-core Intel Xeon E5-2620@2.0GHz CPU and 16 GB of RAM, which hosted the clients. The second machine's configuration was identical to the first, except for the memory which was 64GBytes; this machine hosted the algorithm's executable, the semantic service repository and the past executions repository. The machines were connected through a 1Gbps local area network. The clustered semantic service repository was implemented as in-memory structure; the memory footprint of the repository was less than 50MB, therefore this approach is feasible. Within the repository, the semantic similarities between services (cf. section III.E) were precomputed, in order to speed up the similarity calculation process performed in the execution of step 2 of the recommendation algorithm described in section V.

In order to evaluate our algorithm, we ran two sets of experiments. In the first set, the past executions repository was not clustered, implementing effectively an extension of the

algorithm proposed in [27] that considers the QoS aspect in the service similarity metric (the algorithm in [27] uses only the semantic distance as a similarity metric and does not employ clustering). In the second set, the past executions were clustered as described in section IV. In the experiments we have varied the following parameters:

1. the number of concurrent invocations,
2. the size of the past executions repository,
3. the number of functionalities in the scenario and
4. the number of recommendations requested.

In all experiments, the semantic service repository was populated with synthetic data having an overall size of 2.000 web services; these services account for 20 different tasks, with each task having 100 alternative providers. The QoS attribute values in this repository were uniformly drawn from the domain [0,10]. When conducting a test for a particular number of tasks, we synthetically generated 20 business processes, randomly drawing implementations of distinct functionalities from the repository, and the performance evaluation tests were run for each of the generated scenarios. Each unique performance evaluation test was run 100 times, and the average value was computed and is shown in the following diagrams. The lower QoS bounds for the tasks were randomly drawn from the domain [0,4], while the respective upper QoS bounds were randomly drawn from the domain [7.5,10]. The weights of the QoS attributes were again randomly selected from the domain [0,1]. In all cases, a uniform distribution was used.

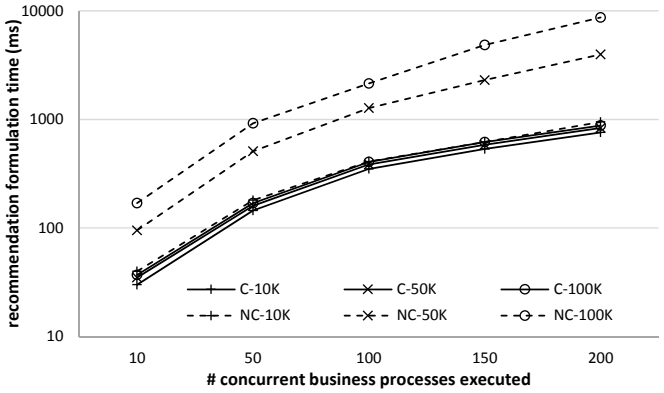

Fig. 4. Recommendation formulation time for varying levels of concurrency

Fig. 4 presents the time needed to generate a recommendation under various concurrency level and past executions repository sizes. Curves C-10K and NC-10K indicate the performance of the clustering and the non-clustering algorithm respectively when the past execution repository contains 10K entries, and similarly for the other past execution repository sizes (50K and 100K). For this experiment, a business process with five tasks was used and one recommendation was requested, while the remaining four tasks were explicitly bound to specific service implementations. In this diagram, we observe that the recommendation time needed by the clustering algorithm increases linearly with the concurrency level (please note that a base-10 logarithmic scale is used for the vertical axis). Moreover, Fig. 4 shows that the time needed by the clustering algorithm to formulate its recommendation increases slowly when the size past execution repository increases, since when the size of the repository increases by a factor of 10, the required time increases only by 15% to 25%. On the contrary, the non-clustering algorithm exhibits limited scalability, since the recommendation time increases between 4.25 and 9.15 times when the size of the repository increases by 10.

Fig. 5 illustrates the time needed to formulate the recommendation when the number of tasks in the business process varies. In these experiments, the concurrency level was set to one and for each business process a single recommendation was requested, while the remaining tasks were explicitly bound to specific services. Curve naming follows the pattern described for Fig. 4. In this diagram we can notice that the clustered algorithm exhibits an increase in recommendation time of 70% when the number of tasks in the business process increases by a factor of 2, while the corresponding recommendation time increment exhibited by the non-clustering algorithm ranges between 205% and 288%. In both cases the increment is owing to the higher dimensionality of the problem, necessitating more computations in similarity comparisons. Again we can observe that the clustering algorithm exhibits better scalability with the number of entries in the past executions repository, as compared with the non-clustering algorithm; indeed, the time needed by the clustering algorithm to formulate its recommendations increases by 15% to 21% when the number of entries in the past executions repository increases by a factor of 10, while the respective time for the non-clustering algorithm increases between 545% and 780%.

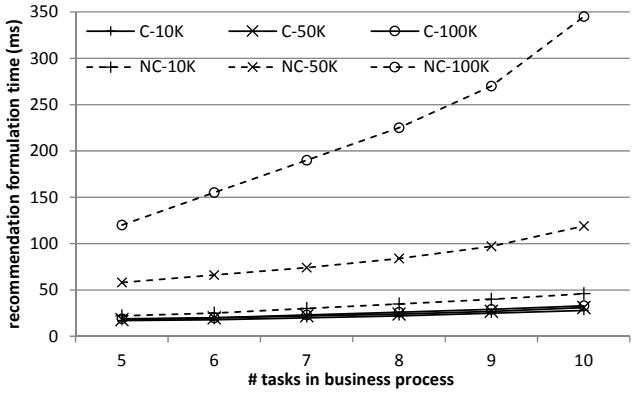

Fig. 5. Recommendation formulation time for varying number of tasks in the business process

Fig. 6 presents the time needed to formulate the recommendations when the number of recommendations requested per business process varies. In these experiments, a business process containing six tasks was used and the concurrency level was set to one. Curve naming follows the pattern described for Fig. 4. In Fig. 6 we can notice that the time needed for each recommendation is fairly stable, i.e. the time needed to formulate two recommendations within a business process is approximately double than the time needed to formulate a single recommendation. This is expected, since the algorithm presented in section V handles each recommendation independently. In absolute times, the clustered implementation has a clear performance advantage over the non-clustered one. This diagram also reaffirms that the clustered algorithm exhibits

high scalability with the number of entries in the past executions repository, since the time needed for formulation recommendation is found to increase by 15% to 24% when the size of the repository increases by a factor of 10. On the contrary, the corresponding time increment for the non-clustered algorithm ranges between 655% and 771%.

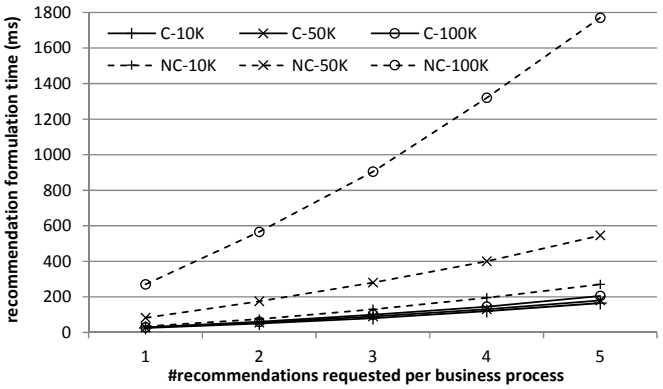

Fig. 6. Recommendation formulation time for varying number of requested recommendations

Finally, Fig. 7 depicts the QoS of the services recommended by different algorithms, for ten representative trial cases and on average. The representative trial cases were drawn from a pool of 1,000 synthetically generated business processes; to draw the representative cases, the business processes were clustered according to the scheme presented in section IV and the cluster medoids were chosen; the average however has been computed using all 1,000 test business processes, and not only the representative ones. The goal of this diagram is to provide insight on how the QoS services recommended by the algorithm in section V compares with the QoS of the services proposed by plain QoS-based algorithms (the QoS-based algorithm described in [9] has been used), and the QoS of service recommendations made by the non-clustered version of the algorithm presented in section V. The latter comparison aims to assess whether the introduction of clustering may lead to inability to find some matching entries (because they have been classified into a different cluster than the one best matching the current request). Finally, a "random" algorithm has been included, to provide insight on how the QoS of the recommended services compares to the average case.

In Fig. 7 we can observe that expectedly the QoS-only algorithm achieves the highest QoS; however, as noted in the introduction, this algorithm completely disregards the satisfaction of service users in the "real world". The non-clustered algorithm produces recommendations having QoS in the range [87%, 100%] of the optimal ones with an average equal to 93%, while the QoS of the clustered algorithm's recommendations fall in the range [84%, 100%] with an average of 90.9%. We can thus conclude that the CF-based algorithms succeed in formulating recommendations with high QoS, while additionally the introduction of clustering leads to a very small QoS degradation (a maximum of 3% and an average of 2.1%), offering on the other hand significant performance gains. The recommendations offered by the CF-based algorithms are also significantly better than those of the random algorithm, whose QoS falls in the range [58%, 100%] of the optimal ones, with an average of 68%.

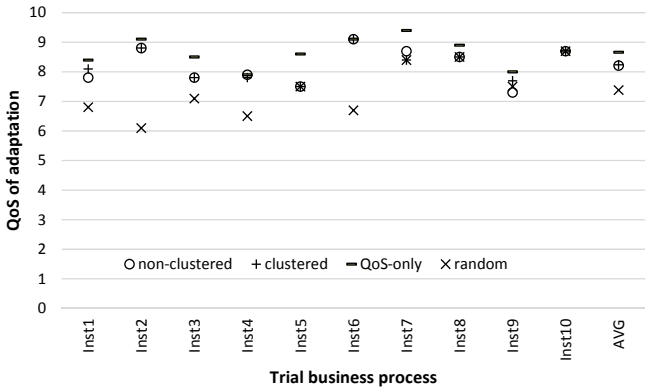

Fig. 7. QoS of solutions proposed by individual algorithms and the combined ones (with and without clustering)

Elaborating on the qualitative performance of the clustered algorithm, the experiments have shown that the *precision at position n* metric [49] for the algorithm ranges from 86% to 100% with an average of 94%. This means that the clustering scheme achieves to retrieve, on average, 47 out of the 50 nearest neighbors to the current request. Therefore, the recommendations of the clustered and the non-clustered algorithms are based on nearest neighbor datasets which are the same to a great extent, and hence the actual recommendations produced are identical in most cases, and very similar in the remaining ones.

VII. Conclusion and Future Work

In this paper we have presented a collaborative filtering-based algorithm with clustering for recommending web services in order to realize personalization and tailoring of business process execution. The proposed algorithm takes into account (i) user-defined QoS weights and limits, (ii) the available services' functionality and QoS attributes values and (iii) the history of past executions of the business process, so as to select the most appropriate recommenders and formulate the recommendations. The similarity metric used in the CF process extends the ones used in existing works by considering not only the functionality resemblance, but additionally the closeness of the QoS attributes. The proposed algorithm employs a clustering scheme to improve recommendation time and leverage scalability. The algorithm have been experimentally validated regarding both its performance and the quality of recommendations offered, and it has been found to be efficient (i.e. it introduces low overhead), scalable with the size of the past executions repository and able to formulate recommendations with high QoS.

Our future work will focus on considering incremental clustering techniques such as BIRCH [51] and the one presented in [52]. Incremental clustering will remove the need to construct the clusters anew in order to accommodate the stream of new execution traces into the past executions repository. Additionally, different clustering algorithms will be evaluated and compared with the CLARA algorithm used in the current paper. Finally, we plan to conduct a user survey, in order to measure the degree to which users are satisfied by the recommendations generated by adaptation algorithm.

REFERENCES

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

[2] 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, Abramowicz W et al. (Eds.): LNCS 6994, 2011, pp. 167–178.

[3] 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.

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

[5] CL. Hwang, K. Yoon, "Multiple Criteria Decision Making", Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 1981.

[6] O. Moser, F. Rosenberg, S. Dustdar, "Non-Intrusive Monitoring and Service Adaptation for WS-BPEL", 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", Proceedings 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, Coimbra, Portugal.

[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] S. Rinderle-Ma, M. Reichert, M. Jurisch, "Equivalence of Web Services in Process-Aware Service Compositions", Proceedings of ICWS'09, 6-10 July 2009.

[13] M. Paolucci, T. Kawamura, T. Payne, T. Sycara, "Semantic Matching of Web Services Capabilities", Proceedings of the International Semantic Web Conference, Sardinia, 2002, pp. 333-347.

[14] 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.

[15] J. Cardoso, A. Sheth, "Semantic e-Workflow Composition", Journal of Intelligent Information Systems, vol. 21 no 3, pp. 191-225, 2003.

[16] JB. Schafer, D. Frankowski, J. Herlocker, S. Sen, "Collaborative Filtering Recommender Systems", in "The Adaptive Web", Lecture Notes in Computer Science Volume 4321, 2007, pp 291-324.

[17] JL. Herlocker, JA. Konstan, LG. Terveen, JT. Riedl, "Evaluating collaborative filtering recommender systems", ACM Transactions on Information Systems vol. 22, no 1, January 2004, pp. 5-53.

[18] M. Balabanovic, Y. Shoham. "Fab: content-based, collaborative recommendation", Communications of the ACM, vol. 40, issue 3, 1997, pp 66-72.

[19] 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.

[20] A. Bramantoro, S. Krishnaswamy, M, Indrawan, "A semantic distance measure for matching web services", Proceeding of the 2005 international conference on Web Information Systems Engineering, 2005, pp 217-226.

[21] TA. Sorensen, "A method of establishing groups of equal amplitude in plant sociology based on similarity of species content, and its application to analyses of the vegetation on Danish commons", K dan Vidensk Selsk Biol Skr 5, 1948, pp. 1-34.

[22] LR. Dice, "Measures of the Amount of Ecologic Association Between Species", Ecology vol. 26, no 3, 1945, pp. 297–302, doi:10.2307/1932409

[23] E. Hullermeier, M. Rifqi, S. Henzgen, R. Senge, "Comparing Fuzzy Partitions: A Generalization of the Rand Index and Related Measures", IEEE Transactions on Fuzzy Systems, vol. 20, no 3, June 2012, pp. 546-556.

[24] G. Hirst, D. St-Onge, "Lexical Chains as Representations of Context for the Detection and Correction of Malapropisms", chapter in "WordNet: An Electronic Lexical Database", Christiane Fellbaum (ed), chapter 13, 1998, pp. 305-332, The MIT Press, Cambridge, MA.

[25] M. Papazoglou and D. Georgakopoulos, "Introduction to the Special Issue on Service-Oriented Computing", Communications of the ACM, 46,10, October 2003.

[26] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, " A Foundational Vision of e-Services", Web Services, E-Business, and the Semantic Web, Lecture Notes in Computer Science Volume 3095, 2004, pp 28-40

[27] D. Margaris, C. Vassilakis, P. Georgiadis, "Adapting WS-BPEL scenario execution using collaborative filtering techniques", Proceedings of the IEEE 7th International Conference on Research Challenges in Information Science, RCIS 2013, R. Wieringa, S. Nurcan, C. Rolland, J-L. Cavarero (eds), Paris, France, 2013

[28] 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.

[29] M. Claypool, A. Gokhale, Y. Miranda, P. Murnikov, D. Netes, M. Sartin, "Combining Content-Based and Collaborative Filters in an Online Newspaper". SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation, I. Soboroff, C. Nicholas, M. Pazzani (eds), Berkeley, California, 1999

[30] S. Gong, "A Collaborative Filtering Recommendation Algorithm Based on User Clustering and Item Clustering", Journal of Software, Vol 5, No 7, 2010, pp. 745-752.

[31] A. Das, M. Datar, A. Garg, S. Rajaram, "Google News Personalization: Scalable Online Collaborative Filtering", Proceedings of the 16th international conference on World Wide Web, 2007, pp. 271-280.

[32] B. Mobasher, X. Jin, Y. Zhou, "Semantically Enhanced Collaborative Filtering on the Web", in Web Mining: From Web to Semantic Web Lecture Notes in Computer Science Volume 3209, 2004, pp 57-76

[33] J. Lee, K. Nam, S. Lee, "Semantics based collaborative filtering," R. Lee, N. Ishii (Eds.): Software Engineering, Artificial Intelligence, SCI 209, 2009, pp. 201-208

[34] X. Guo, and J. Lu, (2007) "Intelligent e-government services with personalized recommendation techniques," on the special issue on Eservice Intelligence of International Journal of Intelligent Systems, vol. 22, no. 5, pp. 401-417.

[35] J. Lu, Q. Shambour, Y. Xu, Q. Lin, and G. Zhang, "BizSeeker: A hybrid semantic recommendation system for personalized government-to business e-services," Internet Research, Vol. 20 (3), 2010, pp. 342 - 365.

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

[37] A.K. Jain, R.C. Dubes, "Algorithms for Clustering Data", Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[38] C.C. Aggarwal, C.K. Reddy, "Data Clustering: Algorithms and Applications", Chapman and Hall/CRC, 2013

[39] M.E. Celebi, "Partitional Clustering Algorithms", Springer; 2015 edition (November 7, 2014)

[40] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A.Y. Zomaya, S. Foufou, A. Bouras, "A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis", IEEE Transactions on Emerging Topics in Computing, 2(3), 2014, pp. 267 – 279.

[41] A.S. Shirkhorshidi, S. Aghabozorgi, T.Y. Wah, T. Herawan, "Big Data Clustering: A Review", Computational Science and Its Applications – ICCSA 2014, LNCS 8583, 2014, pp 707-720

[42] S. Benbernou, I. Brandic, C. Cappiello et al., "Modeling and Negotiating Service Quality", Service Research Challenges and Solutions for the Future Internet, Lecture Notes in Computer Science Volume 6500, 2010, pp 157-208

[43] K. Mardia, J.T. Kent, J.M. Bibby, “Multivariate Analysis”, Academic Press, 1980, ISBN: 0124712525

[44] L. Kaufman, P.J. Rousseeuw, “Finding Groups in Data: an Introduction to Cluster Analysis”, John Wiley & Sons, ISBN: 0471735787

[45] D. He, D. Wu, “Toward a Robust Data Fusion for Document Retrieval”, IEEE 4th International Conference on Natural Language Processing and Knowledge Engineering - NLP-KE, 2008.

[46] G. Xue, C. Lin, Q. Yang, W. Xi, H. Zeng, Y. Yu, Z. Chen, “Scalable collaborative filtering using cluster-based smoothing”. In Proc. of SIGIR ’05, 2005.

[47] H.R. Lourenco, O.C. Martin, T. Stutzle, “Iterated Local Search”, n "Handbook of Metaheuristics", edited by F. Glover and G. Kochenberger, ISORMS 57, 2002, p 321-353.

[48] B. Coppin, “Artificial Intelligence Illuminated”, Jones & Bartlett Learning, 2004, ISBN: 0763732303

[49] T-Y. Liu, J. Xu, T. Qin, W. Xiong, H. Li, “ LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval”, Proceedings of the SIGIR 2007 Workshop “Learning to rank for information retrieval”, 2007.

[50] M. Klusch, B. Fries, K. Sycara, “Automated Semantic Web Service Discovery with OWLS-MX”, Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems, H. Nakashima, M.P. Wellman, G. Weiss, P.Stone (eds), 2006, pp. 915 - 922.

[51] T. Zhang, R. Ramakrishnan, M. Livny, “BIRCH: an efficient data clustering method for very large databases”, Proceedings of the 1996 ACM SIGMOD international conference on Management of data (SIGMOD '96), 1996, pp. 103-114.

[52] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, “Incremental clustering and dynamic information retrieval”, Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (STOC '97), 1997, pp. 626-635.