Top-k Web Service Compositions using Fuzzy Dominance Relationship

Karim Benouaret, Djamal Benslimane, Allel Hadjali, Mahmoud Barhamgi

Top-k Web Service Compositions using Fuzzy Dominance Relationship

Karim Benouaret [1], Djamal Benslimane [1], Allel Hadjali [2], Mahmoud Barhamgi [1]

[1]*LIRIS, Claude Bernard Lyon1 University, 69622 , Villeurbanne, France*
firstname.lastname@liris.cnrs.fr
[2]*Enssat, University of Rennes 1, Lannion, France*
allel.hadjali@enssat.fr

Abstract

Data Web service composition is a powerful means to answer users' complex queries. User preferences are a key aspect that must be taken into account in the composition scheme. In this paper, we present an approach to automatically compose Data Web services while taking into account the user preferences. User preferences are modeled thanks to fuzzy sets. We use an RDF query rewriting algorithm to determine the relevant services. The fuzzy constraints of the relevant services are matched to those of the query using a set of matching methods. We rank-order services using a fuzzification of Pareto dominance, then compute the top-k service compositions. We propose also a method to improve the diversity of returned compositions while maintaining as possible the compositions with the highest scores. Finally, we present a thorough experimental study of our approach.

1. Introduction

Recent years have witnessed a growing interest in using Web services as a reliable means for data publishing and sharing among enterprises [5]. This type of services is known as *Data Web services*, where services correspond to calls over the *business objects* (e.g., *Customer*) in the underlying data sources. In this context, *Data Web Service Composition* is a powerful solution to answer the user's complex queries by combining primitive simple Data Web services to realize value-added services on top of existing ones.

Users' *preferences* is another key aspect that must be considered in the composition process. In this respect, the fuzzy sets theory [4] has been proved to be a viable solution to model preferences. Fuzzy sets are very well suited to the interpretation of linguistic terms, which constitute a convenient way for users to express their preferences. For example, when expressing preferences about the "price", users often employ fuzzy terms like " *cheap*", "*affordable*", etc.

As services and providers proliferate, a large number of candidate compositions that would use different services may be used to answer the same query. It is thus important to set up an effective composition framework that would identify and retrieve the most relevant services and return the top-k compositions according to the user preferences.
Example: Consider the services from the car e-commerce in Table-1. The symbols "$" and "?" denote inputs and outputs, respectively. Services providing the same functionality belong to the same service class. For instance, S_{21}, S_{22}, S_{23} and S_{24} belong to the same class $\mathcal{S}_2$. Each service has its constraints on the data it manipulates. For instance, the cars returned by S_{21} are of *cheap* price and *short* warranty.

Table 1. Example of Data Services

Service	Functionality	Constraints
$S_{11}(\$x, ?y)$	Returns the automakers y whose country is x	-
$S_{21}(\$x, ?y, ?z, ?t)$	Returns the cars y along with their prices z and warranties t for a given automaker x	z is $cheap$, t is $short$
$S_{22}(\$x, ?y, ?z, ?t)$		z is $accessible$, t is $[12, 24]$
$S_{23}(\$x, ?y, ?z, ?t)$		z is $expensive$, t is $long$
$S_{24}(\$x, ?y, ?z, ?t)$		z is $[9000, 14000]$, t is $[6, 24]$
$S_{31}(\$x, ?y, ?z)$	Returns the power y and the consumption z for a given car x	y is $weak$, z is $small$
$S_{32}(\$x, ?y, ?z)$		y is $ordinary$, z is $approximately$ 4
$S_{33}(\$x, ?y, ?z)$		y is $powerful$, z is $high$
$S_{34}(\$x, ?y, ?z)$		y is $[60, 110]$, z is $[3.5, 5.5]$

Let us now assume that the user Bob would like to submit the following query Q_1: "return the French cars, *preferably* at an affordable price with a warranty around 18 months and having a normal power with a medium consumption".

Bob will have to invoke S_{11} to retrieve the French automakers, he then invoke one or more of the services

$S_{21}, S_{22}, S_{23}, S_{24}$ to retrieve the French cars along with their prices and warranties, finally he will invoke one or more of the services $S_{31}, S_{32}, S_{33}, S_{34}$ to retrieve the power and the consumption of retrieved cars. This manual process is painstaking. It raises the following challenges: (i) how to understand the semantics of the published services to select the relevant ones that can contribute to answering the query at hand; (ii) how to retain the most relevant services that better satisfy the user preferences; and (iii) how to generate the best compositions that satisfy the whole user query.
Contributions: We already tackled the first challenge by proposing in [2] an RDF query rewriting approach that generates automatically the Data service compositions (which does not include any preference constraints). In this paper, we focus on the second and third challenges. We select services that cover a part of the query even if their constraints match only partially the user preference constraints. Different methods are investigated to compute the matching degrees between the services' constraints and the preferences involved in the query. In order to select the most relevant services, a multicriteria fuzzy dominance relationship is proposed to rank-order services. The selected services are then used to find the top-k compositions that answer the user query. To avoid returning similar compositions, we also propose a diversified top-k service composition method that aims to both improve the diversity of top-k selection and maintain as possible the services with better scores.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formally defines the studied problem. Section 4 describes the proposed fuzzy dominance and ranking criteria. Section 5 is devoted to the both top-k and diversified top-k compositions generation methods. Section 6 presents the global architecture of our implemented composition system and reports a thorough experimental evaluation. Finally, Section 7 concludes the paper.

2 Related work

Preferences in Web service selection/composition have received much attention by many researchers. In [12], the authors use a qualitative graphical representation of preference, CP-nets, to deal with services selection in terms of user preferences. This approach can reason about a user's incomplete and constrained preference. In [7], a method to rank semantic web services is proposed. It is based on computing the matching degree between a set of requested NFPs (Non-Functional Properties) and a set of NFPs offered by the discovered Web services. NFPs cover QoS aspects, but also other business-related properties such as pricing and insurance. Semantic annotations are used for describing NFPs and the ranking process is achieved using some automatic reasoning techniques that exploit the annotations. However, the problem of composition is not addressed in these works.

Agarwal and Lamparter [1] proposed an approach for an automated selection of Web service for composition. Web service combinations can be compared with each other and ranked according to the user preferences. Preferences are modeled as a fuzzy IF-THEN rules. The IF part contains fuzzy descriptions of the various properties of a service and the THEN part is one of the fuzzy characterizations of a special concept called *Rank*. A fuzzy rule describes which combination of attribute values a user is willing to accept to which degree, where attribute values and degree of acceptance are fuzzy sets. ServiceRank [13] considers the QoS aspects as well as the social perspectives of services. Services that have good QoSs and are frequently invoked by others are more trusted by the community and will be assigned high ranks. In [11], the authors propose a system for conducting qualitative Web service selection in the presence of incomplete or conflicting user preferences. The paradigm of CP-nets is used to model user preferences. The system utilizes the history of users to amend the preferences of active users, thus improving the results of service selection.

The work the most related to our proposal is [10], where the authors consider dominance relationships between web services based on their degrees of match to a given request in order to rank available services. Distinct scores based on the notion of dominance are defined for assessing when a service is objectively interesting. However, that work considers only selection of single services, without dealing with neither the problem of composition nor the user preferences.

Finally, in [9], the authors propose a method to diversify Web service search results in order to deal with that have different, but unknown, preferences. The proposed method focuses on QoS parameters with non-numeric values, for which no ordering can be defined. However, this method provides the same services to all users, also the problem of composition is not addressed. In our approach the diversified compositions vary according to the user.

3 Preference-based composition model

3.1 Preference Queries

Users express their preference queries over domain ontologies using a slight modification of SPARQL. For instance, query Q_1 given in Section 1 is expressed as follows:

```
URL=http://vm.liris.cnrs.fr:36880/MembershipFunctions/
SELECT  ?n ?pr ?w ?pw ?co
WHERE{?Au rdf:type AutoMaker  ?Au hasCountry 'France'
?Au makes ?C  ?C rdf:type Car ?C hasName ?n  ?C hasPrice ?pr
?C hasWarranty ?w  ?C hasPower ?pw  ?C hasConsumption ?co}
Preferring {?pr is 'URL/AffordablesService',
?w is 'URL/around(18)Service',?pw is 'URL/NormalService',
?co is 'URL/MediumService'}
```

Preferences are modeled using fuzzy sets [4]. Formally, a fuzzy set F on a referential X is characterized by a membership function $\mu_F : X \longrightarrow [0,1]$, where $\mu_F(x)$ represents the grade of membership of x in F. Namely having $x, y \in F$, x is more preferable than y iff $\mu_F(x) > \mu_F(y)$. In practice, membership functions are often of trapezoidal form represented by the quadruplet (A, B, a, b) as shown in Figure 1. A regular interval $[A, B]$ can be seen as a fuzzy set represented by the quadruplet $(A, B, 0, 0)$.

Membership functions are implemented as Web services and can be shared by users. They are used in the Preferring clause of the query by mentioning the URI of the implementing service. More details are provided in section 6.

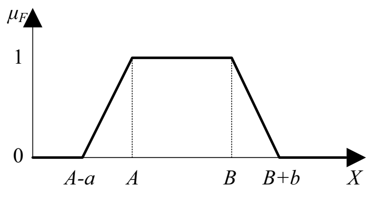

Figure 1. Fuzzy value representation

3.2 Data services

Data services are partitioned into different classes. A class $\mathcal{S}_j$ compromises services providing the same functionality. A Data service S_{ji} of class $\mathcal{S}_j$ is described as a predicate $S_{ji}(\$X_j, ?Y_j) : - < G_j(X_j, Y_j, Z_j), C_{ji} >$ where: X_j and Y_j are the sets of input and output variables of S_{ji}, respectively. Input and output variables are also called distinguished variables. They are prefixed with the symbols "\$" and ?, respectively. $G_j(X_j, Y_j, Z_j)$ represents the functionality of the service. This functionality is described as a semantic relationship between input and output variables. Z_j is the set of existential variables relating X_j and Y_j. $C_{ji} = \{C_{ji_1}, ..., C_{ji_n}\}$ is a set of constraints expressed as intervals or fuzzy sets on X_j, Y_j or Z_j variables.

X_j and Y_j variables are defined in the WSDL description of services. Functionality G_j and constraints C_{ji} of a service S_{ji} are added to the WSDL descriptions in the form of annotations. The annotations are represented in the form of SPARQL queries. For instance, the following query illustrates the functionality and constraints of S_{21}:

```
URL=http://vm.liris.cnrs.fr:36880/MembershipFunctions/
RDFQuery{SELECT ?y ?z ?t
WHERE{?Au rdf:type AutoMaker  ?Au name $x
?Au makes ?C  ?C rdf:type Car ?C hasName ?y
?C hasPrice ?z  ?C hasWarranty ?t}}
CONSTRAINTS{?z is 'URL/CheapService'
 ?t is  'URL/ShortService'}
```

The `SELECT` and `WHERE` clauses define the functionality of S_{21}. The `CONSTRAINTS` clause gives the constraints of S_{21}.

3.3 Discovering Relevant Services

Let Q be a preference query. We use our RDF query rewriting algorithm [2] to discover the parts of Q that are covered by each service—recall that in the general case services may match only parts (referred to by q_j) of Q. A part q_j is covered by one or more services that constitute a class of relevant services $\mathcal{S}_j$. A service $S_{ji} \in \mathcal{S}_j$ is said to be relevant to Q iff the functionality of S_{ji} completely matches a part q_j and its constraints match completely or partially the preference constraints of q_j. We use a set of methods $M = \{m_1, ..., m_{n'}\}$ (e.g., constraints inclusion operators) resulting in different degrees of match for each service. Two classes of constraints inclusion operator are considered. Let $C \equiv x$ is E and $C' \equiv x$ is F be two constraints.

- *Quantitative method (QM).* $Deg(C \subseteq C') = \frac{|E \cap F|}{|E|} = \frac{\sum_{x \in X} T(\mu_E(x), \mu_F(x))}{\sum_{x \in X} \mu_E(x)}$ where the intersection is interpreted by a t-norm operator T [4]. For instances, $T =$"min" (M-QM) and $T =$"$product$" (P-QM).

- *Logic method (LM).* $Deg(C \subseteq C') = min_{x \in X} (\mu_E(x) \rightarrow_f \mu_F(x))$ where $\rightarrow_f$ stands for a fuzzy implication [4]. For instances, Godel (G-LM), and Lukasiewicz (L-LM).

Table 2. Services and their matching degrees

S_{ji}	q_j	M-QM	P-QM	G-LM	L-LM
S_{11}	q_1	-	-	-	-
S_{21}	q_2	$(1, 0.57)$	$(0.98, 057)$	$(1, 0)$	$(0.80, 0)$
S_{22}		$(0.89, 1)$	$(0.77, 1)$	$(0, 1)$	$(0.50, 1)$
S_{23}		$(0.20, 0.16)$	$(0.13, 0.13)$	$(0, 0)$	$(0, 0)$
S_{24}		$(0.83, 0.88)$	$(0.83, 0.88)$	$(0.60, 0.50)$	$(0.60, 0.50)$
S_{31}	q_3	$(0.50, 0.36)$	$(0.46, 0.32)$	$(0, 0)$	$(0, 0)$
S_{32}		$(0.79, 0.75)$	$(0.69, 0.72)$	$(0, 0.25)$	$(0.40, 0.50)$
S_{33}		$(0.21, 0.64)$	$(0.17, 0.61)$	$(0, 0)$	$(0, 0)$
S_{34}		$(0.83, 0.85)$	$(0.83, 0.85)$	$(0.50, 0.50)$	$(0.50, 0.50)$

Table 2 shows the matching degrees between each service S_{ji} from Table 1 and its corresponding part q_j of Q_1. The service S_{11} covering the part q_1 does not have a matching degree as there are no constraints imposed by the user on q_1. However, each service covering the part q_2 is associated with four (the number of methods) degrees. Each matching degree is formulated as a pair of real values within the range $[0, 1]$, where the first and second values are the matching degrees of the constraints *price* and *warranty*, respectively. Similarly, for the matching degrees of the services covering q_3, the first and second values represent respectively the inclusion degrees of the constraints *power* and *consumption*.

3.4 Problem statement

Given a preference query $Q : - < q_1, ..., q_n >$. Each part q_j is a tuple $(\overline{q}_j, P_{q_j})$, where $\overline{q}_j$ represents q_j without its preferences P_{q_j}. Given a set of services classes $\mathcal{S} = \{\mathcal{S}_1, ..., \mathcal{S}_n\}$, where a class $\mathcal{S}_j$ regroups the services that are relevant to the part q_j and given a set $M = \{m_1, ..., m_{n'}\}$ of matching methods. The problem is how to rank services in each class $\mathcal{S}_j$ to select the most relevant ones and how to rank generated compositions to select the top-k ones.

4 Fuzzy dominance and fuzzy scores

4.1 Dominances: Pareto vs Fuzzy

Services of the same class $\mathcal{S}_j$ have the same functionality, they only differ in terms of constraints, providing thus different matching degrees. Individual matching degrees of services could be aggregated. One method is to assign weights to individual degrees and, for instance, compute a weighted average of degrees. In doing so, users may not know enough to make trade-offs between different relevancies using numbers (average degrees). Users thus lose the flexibility to select their desired answers by themselves. Computing the skyline [3] is as a natural solution to overcome this limitation. The skyline consists of the set of points which are not dominated by any other point.

Definition 1 *(Pareto dominance) Let u and v be two d-dimensional points. We say that u dominates v, denoted by $u \succ v$, iff $\forall i \in [1, d], u_i \geq v_i \wedge \exists k \in [1, d], u_k > v_k$.*

Pareto dominance is not always significant to rank-order points. To illustrate this situation, let $u = (u_1, u_2) = (1, 0)$ and $v = (v_1, v_2) = (0.90, 1)$ be two matching degrees. In Pareto order, u and v are incomparable. However, one can consider that v is better than u since $v_2 = 1$ is too much higher than $u_2 = 0$, contrariwise $v_1 = 0.90$ is almost close to $u_1 = 1$. It is thus interesting to fuzzify the dominance relationship to express the extent to which a matching degree (more or less) dominates another one. We define below a fuzzy dominance that relies on particular membership function of a graded inequality of the type *strongly larger than*.

Definition 2 *(fuzzy dominance) Given two d-dimensional points u and v, we define the fuzzy dominance to express the extent to which u dominates v such as:*

$$deg(u \succ v) = \frac{\sum_{i=1}^{d} \mu_{\gg}(u_i, v_i)}{d}$$

Where $\mu_{\gg}(u_i, v_i)$ expresses the extent to which u_i is more or less (strongly) greater than v_i. $\mu_{\gg}$ is defined as:

$$\mu_{\gg}(x, y) = \left\{ \begin{array}{cl} 0 & if x - y \leq \varepsilon \\ 1 & if x - y \geq \lambda + \varepsilon \\ \frac{x-y-\varepsilon}{\lambda} & otherwise \end{array} \right\}$$

Where $\lambda > 0$, i.e., $\mu_{\gg}$ is more demanding than the idea of "strictly greater" and $\varepsilon \geq 0$ in order to ensure that $\mu_{\gg}$ agrees with the idea of "greater" in the usual sense. The semantics of $\mu_{\gg}$ is as follows: if $x - y$ is less than ε, then x is not at all strongly greater than y; if $x - y$ is larger than $\lambda + \varepsilon$, then x is all much greater then y; if $x - y$ is between ε and λ, then x is much greater than y is a matter of degree.

Let us reconsider the previous instances $u = (1, 0)$, $v = (0.90, 1)$. With $\varepsilon = 0$ and $\lambda = 0.2$, we have $deg(u \succ v) = 0.25$ and $deg(v \succ u) = 0.5$. This is more significant than u and v are incomparable provided by Pareto dominance. In the following sections, we use the defined fuzzy dominance to compute scores of services and compositions.

4.2 Associating score with a service

Under a single matching degree, the dominance relationship is unambiguous. When multiple methods are applied, resulting in different matching degrees for the same constraints, the dominance relationship becomes uncertain. The model proposed in [8], namely probabilistic skyline overcomes this problem. Contrariwise, Skoutas et al. show in [10] the limitations of the probabilistic skyline to rank services and introduce the Pareto dominating score of individual services. We generalize this score to fuzzy dominance and propose a fuzzy dominating score (FDS). An FDS of a service S_{ji} indicates the average extent to which S_{ji} dominates the whole services of its class $\mathcal{S}_j$.

Definition 3 *(Fuzzy dominating score) The fuzzy dominating score of a service S_{ji} in its class $\mathcal{S}_j$ is defined as:*

$$FDS(S_{ji}) = \frac{1}{(|\mathcal{S}_j| - 1)\,|M|^2} \sum_{h=1}^{|M|} \sum_{\substack{S_{jk} \in \mathcal{S}_j \\ k \neq i}} \sum_{r=1}^{|M|} deg(S_{ji}^h \succ S_{jk}^r)$$

where S_{ji}^h is the matching degree of S_{ji} obtained by applying the h^{th} method. The term $(|\mathcal{S}_j| - 1)$ is used to normalize the FDS in the range $[0, 1]$. Table 3 shows the fuzzy dominating scores of the services of our example.

4.3 Associating score with a composition

Different compositions can be generated from different classes. To rank such compositions, we extend the previous FDS definition to composition and associate each one with an FDS as an aggregation of different FDSs of its component services. Let $C = \{S_{1i_1}, ..., S_{ni_n}\}$ be a composition of n services and $d = d_1 + ... + d_n$ be the number of preferences, where d_j is the number of constraints involved in the service S_{ji_j}. The FDS of C is then computed as follows:

$$FDS(C) = \frac{1}{d} \sum_{j=1}^{n} d_j \cdot FDS(S_{ji_j})$$

Table 3. Top-k services

Services	Class	Score	Top-k
S_{11}	$\mathcal{S}_1$	-	S_{11}
~~S_{21}~~	$\mathcal{S}_2$	0.527	S_{22} S_{24}
S_{22}		0.657	
~~S_{23}~~		0.027	
S_{24}		0.533	
~~S_{31}~~	$\mathcal{S}_3$	0.083	S_{32} S_{34}
S_{32}		0.573	
~~S_{33}~~		0.187	
S_{34}		0.717	

5 Top-k service composition

5.1 Efficient generation

A straightforward method to find the top-k compositions that answer a query is to generate all possible compositions, compute their scores, and return the top-k ones. However, this approach results in a high computational cost, as it needs to generate all possible compositions, whereas, most of them are not in the top-k. The following theorem[1] provides an optimization technique to find quickly the top-k compositions : *the top-k services of the different service classes are sufficient to compute the top-k compositions*.

Theorem 1 *Let $C = \{S_{1i_1}, ..., S_{ni_n}\}$ be a composition and top-$k.\mathcal{S}_j$ (resp. top-$k.\mathcal{C}$) be the top-k services of the class $\mathcal{S}_j$ (resp. the top-k compositions). Then, $\exists S_{ji_j} \in C; S_{ji_j} \notin$ top-$k.\mathcal{S}_j \implies C \notin$ top-$k.\mathcal{C}$.*

Table 3 shows the top-k ($k = 2$) services in each service class using the FDS. Thus, relevant services that are not in the top-k of their classes are eliminated. They are crossed out in Table 3. The other services are retained. The top-k compositions are generated from the different top-$k.\mathcal{S}_j$ classes. Table 4 shows the possible compositions along with their scores and the top-k compositions of our example.

Table 4. Top-k composition

Compositions	Score	Top-k
$C_1 = \{S_{11}, S_{22}, S_{32}\}$	0.615	C_2 C_4
$C_2 = \{S_{11}, S_{22}, S_{34}\}$	0.687	
$C_3 = \{S_{11}, S_{24}, S_{32}\}$	0.553	
$C_4 = \{S_{11}, S_{24}, S_{34}\}$	0.625	

5.2 Top-k compositions algorithm

The algorithm, hereafter referred to as $\mathcal{TKSC}$, computes the top-k compositions according to the fuzzy scores. It proceeds as follows.

[1]Proof excluded due to lack of space

Algorithm 1: $\mathcal{TKSC}$

```
Input: $Q$ preference query; $\mathcal{S}$ set of service classes; $k \in \mathbb{N}$;
       $M = \{m_1, ..., m_{n'}\}$ set of methods, $\varepsilon \geq 0; \lambda > 0$;
1  foreach $\mathcal{S}_j$ in $\mathcal{S}$ do
2      $S \leftarrow random(\mathcal{S}_j, 1)$;
3      if $\exists q_j \in Q; cover(S, q_j)$ then
4          $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{S}_j$;
5          if $P_{q_j} \neq \emptyset$ then
6              foreach $S_{ji}$ in $\mathcal{S}_j$ do
7                  foreach $m$ in $M$ do
8                      ComputeDegree($C_{ji}, P_{q_j}, m$);
9  foreach $\mathcal{S}_j$ in $\mathcal{R}$ do
10     if $P_{q_j} = \emptyset$ then
11         top-$k.\mathcal{S}_j \leftarrow random(\mathcal{S}_j, k)$;
12     else
13         foreach $S_{ji}$ in $\mathcal{S}_j$ do
14             ComputeSScore($S_{ji}$);
15         top-$k.\mathcal{S}_j \leftarrow$ top($k, \mathcal{S}_j$);
16 $\mathcal{C} \leftarrow$ ComposeServices(top-$k.\mathcal{S}_{j_1}$, ..., top-$k.\mathcal{S}_{j_m}$);
17 foreach $C$ in $\mathcal{C}$ do
18     ComputeCScore($C$);
19 return top($k, \mathcal{C}$);
```

Step.1 *compute the matching degrees (lines 1-8).* Each class whose services cover a query part is added into the list of relevant classes. If its services touch the user preferences, we compute its different matching degrees according to the number of methods.

Step.2 *eliminating less relevant services (lines 9-15).* For each class whose services do not touch the user preferences, we select randomly k services since they are all equal with respect to user preferences. Otherwise, i.e., its services touch the user preferences, we first compute the score of its services and then retain only the top-k ones.

Step.3 *returning top-k compositions (lines 16-19).* We first compose the retained services, i.e., the top-k of each class, we then compute the score of generated compositions. Finally we provide the user with the top-k ones.

5.3 Diversity-aware top-k compositions

Similar services could exist in each class $\mathcal{S}_i$ leading to similar top-k compositions. A little variety in the top-k compositions list will probably lead to the user frustration. Diversification is then needed to improve the quality of the top-k compositions. We tackle this issue by proposing a method for maximizing the diversity of compositions while maintaining an acceptable accuracy (expressed in terms of FDS) of compositions. We propose to diversify the top-k compositions by firstly diversifying the top-k services of each class $\mathcal{S}_j$, by diversifying the compositions themselves.

The diversity of the top-k of a class $\mathcal{S}_j$ means that the services it includes should be dissimilar amongst each other. A principled way to improving diversity while maintaining accuracy, is to explicitly use both diversity and accuracy of during the top-k services selection. We use the following quality metric that combines diversity and accuracy:

$$Quality(S_{ji}) = FDS(S_{ji}) \times RelDiv(S_{ji}, dtopk\mathcal{S}_j)$$

The quality of a service S_{ji} in its class $\mathcal{S}_j$ is proportional to its accuracy w.r.t. FDS and to its relative diversity to those diversified top-k services so far selected $dtopk\mathcal{S}_j$. Initially, $dtopk\mathcal{S}_j$ is an empty set, and its first element will be necessary one of the services with higher FDS. The relative diversity of a service S_{ji} to the current set $dtopk\mathcal{S}_j$ is defined as the average dissimilarity between S_{ji} and the so far selected service [6] as described in the following equation:

$$RelDiv(S_{ji}, dtopk\mathcal{S}_j) = \frac{\sum_{S_{jr} \in dtopk\mathcal{S}_j} Dist(S_{ji}, S_{jr})}{|dk\mathcal{S}_j|}$$

The relative diversity of a service S_{ji} to an initial empty set, i.e., $|dtopk\mathcal{S}_j| = 0$, is set to 1. The quantity $Dist(S_{ji}, S_{jr})$ represents the distance (i.e., dissimilarity) between the services S_{ji} and S_{jr}. Recall that Data services of the same class have the same functionality and only differ in their constraints, therefore the distance can be then reduced to the distance between their constraints.

Given two services S_{ji} and S_{jr} in $\mathcal{S}_j$ having the constraints $C_{ji} \equiv x_1 \text{ is } E_1, ..., x_{d_j} \text{ is } E_{d_j}$ and $C_{jr} \equiv x_1 \text{ is } F_1, ..., x_{d_j} \text{ is } F_{d_j}$, respectively, where d_j is the number of constraints involved in the services S_{ji} and S_{jr}. The distance between S_{ji} and S_{jr} can be measured by $Dist(S_{ji}, S_{jr}) = max_{h \in \{1,...,d_j\}} Dist(E_h, F_h)$, where $Dist(E_h, F_h) = max_{x \in X} |\mu_E(x) - \mu_F(x)|$ is the distance between the fuzzy sets E_h and F_h.

Algorithm 2: $\mathcal{DTKS}$

```
Input: $k \in \mathbb{N}$; $s \in \mathbb{N}$; $\mathcal{S}_j$ a service class;
1 $\mathcal{S}'_j \leftarrow$ top($k \cdot s$,$\mathcal{S}_j$);
2 $dtopk.\mathcal{S}_j \leftarrow \emptyset$;
3 for $i$=1 to $k$ do
4     ComputeQuality($\mathcal{S}'_j$);
5     $dtopk.\mathcal{S}_j \leftarrow dtopk.\mathcal{S}_j \cup$ {MaxQuality($\mathcal{S}'_j$)};
6     $\mathcal{S}'_j \leftarrow \mathcal{S}'_j -$ {MaxQuality($\mathcal{S}'_j$)};
7 return $dtopk\mathcal{S}_j$;
```

Diversified top-k services computing strategy: The above quality measure guides the construction of the diversified top-k services of a class $\mathcal{S}_j$ in an incremental way as described in Algorithm 2 ($\mathcal{DTKS}$). During each step the remaining services of $\mathcal{S}_j$ are rank-ordered according to their quality and the highest quality service is added to $dtopk\mathcal{S}_j$. The first service of the diversified top-k of $\mathcal{S}_j$ to be selected is always the one with the highest FDS. The initial service class $\mathcal{S}_j$ can be bounded to a smaller size equivalent to $k \cdot s$ to decrease the search space especially when $\mathcal{S}_j$ is too large.

Diversified top-k service compositions computing: The top-k compositions set is diversified by diversifying its component compositions and maintaining acceptable compositions scores. The $Quality$ of a composition C is an aggregation of different $Qualities$ of its component services.

Let $C = \{S_{1i_1}, ..., S_{ni_n}\}$ be a composition of n services and $d = d_1 + ... + d_n$ be the number of preferences, where d_j is the number of constraints involved in the service S_{ji_j}. The $Quality$ of C is then computed as follows:

$$Quality(C) = \frac{1}{d} \sum_{j=1}^{n} d_j \cdot Quality(S_{ji_j})$$

The diversified top-k compositions ($\mathcal{DTKSC}$) is obtained from $\mathcal{TKSC}$ by applying the following modifications:
line 15: instead of taking the top-k services in each class based on their scores, we take them based on their qualities, i.e., we take the diversified top-k ones, by applying Algorithm 2. Line 15 becomes: *top-k.*$\mathcal{S}_j \leftarrow DTKS(k, s, \mathcal{S}_j)$;
line 18: we compute the quality of compositions, instead of their scores. This line becomes: computeCQuality(C);
line 19: instead of returning the top-k compositions, i.e., the top-k with the highest scores, we return the diversified top-k ones, i.e., the ones having the best Qualities. So the line 19 becomes: **return** Dtop($k, \mathcal{C}$);

6 Architecture and experimental evaluation

6.1 System Architecture

In this section we outline the basic components of our system, their roles and how they interact with each other. A high-level architecture of our system is illustrated in Figure 2. The system consists of the following components:
The *Fuzzy Membership Functions Manager* is useful to manage fuzzy linguistic terms. It enables users and service providers to define their desired fuzzy terms along with their membership functions. The defined terms are stored in a local fuzzy terms knowledge base which can be shared by users, and are linked to their implementing Web services. This link[2] describes a set of fuzzy terms and their implementing Web services. Users and providers can directly test the proposed membership functions and use the associated fuzzy terms. For each fuzzy term we provide a shape that gives a graphical representation of the associated membership function, a form that helps users to compute the degree

[2]http://vm.liris.cnrs.fr:36880/FuzzyTerms/

to which a given value is in the fuzzy set of the considered fuzzy term, and a WSDL description of the Web service that implements the membership function.

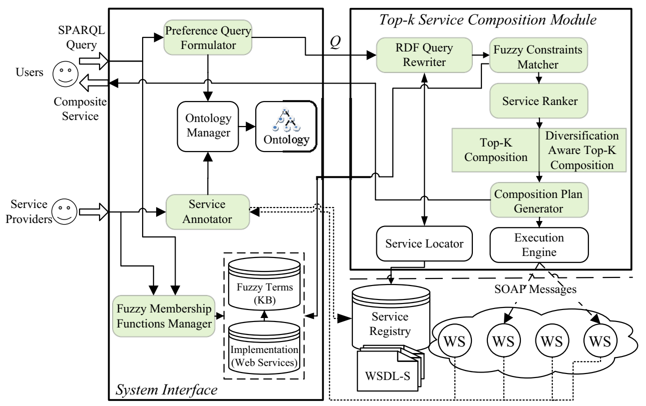

Figure 2. System Architecture

The *Service Annotator* allows providers to annotate WSDL description files of services with fuzzy terms to represent the services constraints and with SPARQL queries expressed over a domain ontology to represent the semantic definition of the service functionality in the form of RDF graph. This annotation is implemented by adding a new element called "rdfQuery" to the XML Schema of WSDL as in WSDL-S approach. The WSDL files are then published on a service registry. The ontology manager uses Jena API to manage domain ontology (i.e., to add/delete concepts).
The *Preference Query Formulator* provides users with a GUI implemented with Java Swing to interactively formulate their queries over a domain ontology. Users are not required to know any specific ontology query languages to express their queries.
The *Top-k Service Compositions* consists of five components. The *RDF Query Rewriter* implements an RDF query rewriting algorithm [2] to identify the relevant services that match (some parts of) a user query. For that purpose, it exploits the service annotation. The *Service Locator* feeds the *Query Rewriter* with services that most likely match a given query. The *Top-K Compositions* component computes (i) the matching degrees of relevant services, (ii) the fuzzy dominating scores of relevant services, (iii) the top-k services of each relevant service class and (iv) the fuzzy compositions scores to return the top-k compositions. The *diversification-aware Top-k Compositions* component implements the proposed quality metric to compute a diversified top-k service composition. The (diversified) top-k service compositions are then translated by the *composition plan generator* into execution plans expressed in the XPDL language. They are executed by a workflow execution engine; we use the Sarasvati execution engine from Google.

6.2 Experimental evaluation

This section presents an extensive experimental study of our approach. Our objective is to prove the efficiency and the scalability of our proposed Top-k service composition algorithms as the number of the considered services increases. For this purpose, we implemented a Web service generator. The generator takes as input a set of (real-life) model services (each representing a class of services) and their associated fuzzy constraints and produces for each model service a set of synthetic Web services and their associated synthetic fuzzy constraints. In the experiments we evaluated the effects of the following parameters: (*i*) the number of services per class, (*ii*) the service classes number, (*iii*) the number of fuzzy constraints per class, (*iv*) the number of matching methods and (*v*) the effects of ε and λ.

The algorithms $\mathcal{TKSC}$ and $\mathcal{DTKSC}$ were implemented in Java and the experiments were conducted on a Pentium D 2:4GHz with 2GB of RAM, running Windows XP.

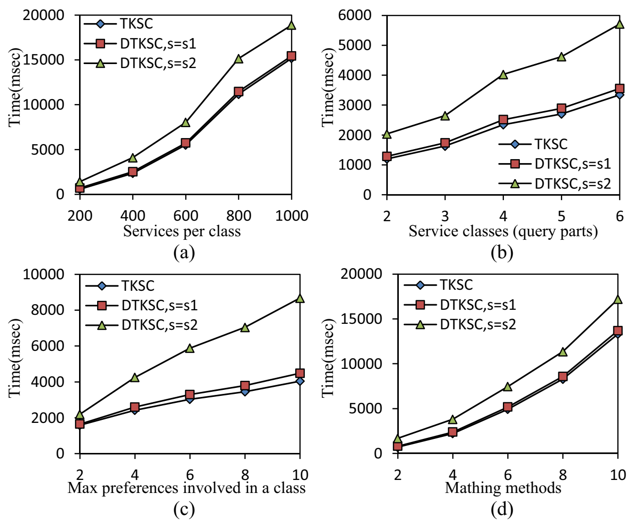

Figure 3. Performance results[3] **($k = 5$)**

Performance vs number of services per class: we varied the number of services per class from 100 to 1000. Figure 3-(a) show that our framework can handle hundreds of services in a reasonable time. The results show also that computing the diversified top-k composition introduces an insignificant cost when the factor s is small (e.g., $s = s_1$).

Performance vs number of classes: We varied the classes number from 1 to 6. Figure 3-(b) show that the execution time is proportional to the classes number.

Performance vs number of constraints per service: We varied the fuzzy constraints number from 1 to 10. Figure 3-(c) shows that when the factor s is small (e.g., $s = s_1$)

[3] $s_1 = \left\lfloor \sqrt{\frac{|\mathcal{S}_j|}{k}} \right\rfloor$ and $s_2 = \left\lfloor \frac{|\mathcal{S}_j|}{k} \right\rfloor$ where $\lfloor x \rfloor$ is the integer part of x

Table 5. Effects of (ε, λ)

(ε, λ)	Top-k Compositions			Diversified Top-k Compositions			
	Component Services	Score	Diversity	Component Services	Quality	Score	Diversity
$(0.002, 0.05)$	$\{S_{1318}, S_{2292}, S_{3154}, S_{4154}\}$	0.74703556	0.6121456	$\{S_{1318}, S_{2292}, S_{3154}, S_{4154}\}$	0.74703556	0.74703556	0.6995363
	$\{S_{1318}, S_{259}, S_{3154}, S_{4154}\}$	0.7441032		$\{S_{1318}, S_{2292}, S_{3154}, S_{4134}\}$	0.6972428	0.7426259	
	$\{S_{1318}, S_{2152}, S_{3154}, S_{4154}\}$	0.7441032		$\{S_{1318}, S_{2134}, S_{3154}, S_{4154}\}$	0.6972428	0.7426259	
$(0.02, 0.2)$	$\{S_{1318}, S_{2292}, S_{3154}, S_{4154}\}$	0.6563174	0.59373885	$\{S_{1318}, S_{2292}, S_{3154}, S_{4154}\}$	0.6563174	0.6563174	0.6995363
	$\{S_{1318}, S_{2132}, S_{3154}, S_{4154}\}$	0.655371		$\{S_{1318}, S_{2292}, S_{3154}, S_{4134}\}$	0.612067	0.6519956	
	$\{S_{1318}, S_{259}, S_{3154}, S_{4154}\}$	0.65328693		$\{S_{1318}, S_{2134}, S_{3154}, S_{4154}\}$	0.6098658	0.6515922	
$(0.1, 0.3)$	$\{S_{1318}, S_{2292}, S_{3154}, S_{4154}\}$	0.53315574	0.62760955	$\{S_{1318}, S_{2292}, S_{3154}, S_{4154}\}$	0.53315574	0.53315574	0.71135545
	$\{S_{1318}, S_{2132}, S_{3154}, S_{4134}\}$	0.5312762		$\{S_{1318}, S_{2292}, S_{3154}, S_{4134}\}$	0.49845165	0.5312762	
	$\{S_{1318}, S_{2292}, S_{3154}, S_{4154}\}$	0.53008974		$\{S_{1318}, S_{2134}, S_{3154}, S_{4154}\}$	0.49460968	0.5256555	

the cost incurred in computing the diversified top-k compositions is insignificant as the constraints number increases.

Performance vs number of matching methods: we varied the number of matching methods from 1 to 10. The results of this experiment are shown in Figure 3-(d). Once again the cost incurred in computing the diversified top-k compositions remains insignificant as the methods number increases if the factor *s* has a reasonable value (e.g., $s = s_1$).

The effects of ε **and** λ**:** varying ε and λ change the scores/qualities for the top-k/diversified-top-k compositions. This may consequently lead to the inclusion or to the exclusion of a composition from top-k/diversified top-k compositions. Table 5 shows the top-k/diversified-top-k compositions for different values of ε and λ; the higher the values of these parameters are the higher the global diversity of the diversified top-k compositions is. The global diversity of the diversified top-k compositions set described in the following equation is the average of the diversities between each couple of compositions in the compositions set: $div(top-k) = \frac{\sum_{i=1}^{k}\sum_{j=i+1}^{k} div(C_i,C_j)}{(k^2-k)/2}$, where $div(C_i, C_j) = Dist(C_i, C_j)$. Note that the global diversity of the diversified top-k compositions is always higher than that of the top-k compositions.

7 Conclusions

In this paper, we proposed an approach to compute the top-k Data service compositions for answering fuzzy preference queries. We introduced the concept of *fuzzy dominance relationship* to measure to what extent a service (represented by its vector of matching degrees) dominates another one. This new concept allowed us to rank-order candidate services in their respective classes and compositions to compute the top-k ones. We propose also a method to improve the diversity of returned compositions while maintaining as possible the compositions with the highest scores. Further, we developed and evaluated suitable algorithms for computing the top-k/diversified-top-k compositions. As a future work, we intend to apply the proposed fuzzy approach to top-k QoS-based service composition.

References

[1] S. Agarwal and S. Lamparter. User preference based automated selection of web service compositions. In K. V. A. S. M. Z. C. Bussler, editor, *ICSOC Workshop on Dynamic Web Processes*, pages 1–12, Amsterdam, Netherlands, Dezember 2005. IBM.

[2] M. Barhamgi, D. Benslimane, and B. Medjahed. A query rewriting approach for web service composition. *IEEE T. Services Computing*, 3(3):206–222, 2010.

[3] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[4] D. Dubois and H. Prade. *Fundamentals of fuzzy sets*. Kluwer, Netherlands, 2000.

[5] A. Jhingran. Enterprise information mashups: Integrating information, simply. In *VLDB*, pages 3–4, 2006.

[6] D. McSherry. Diversity-conscious retrieval. In *ECCBR*, pages 219–233, 2002.

[7] M. Palmonari, M. Comerio, and F. D. Paoli. Effective and flexible nfp-based ranking of web services. In *ICSOC/ServiceWave*, pages 546–560, 2009.

[8] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[9] D. Skoutas, M. Alrifai, and W. Nejdl. Re-ranking web service search results under diverse user preferences. In *VLDB, Workshop on Personalized Access, Profile Management, and Context Awareness in Databases*, pages 898–909, 2010.

[10] D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. K. Sellis. Top- dominant web services under multi-criteria matching. In *EDBT*, pages 898–909, 2009.

[11] H. Wang, S. Shao, X. Zhou, C. Wan, and A. Bouguettaya. Web service selection with incomplete or inconsistent user preferences. In *ICSOC/ServiceWave*, pages 83–98, 2009.

[12] H. Wang, J. Xu, and P. Li. Incomplete preference-driven web service selection. In *IEEE SCC (1)*, pages 75–82, 2008.

[13] Q. Wu, A. Iyengar, R. Subramanian, I. Rouvellou, I. Silva-Lepe, and T. A. Mikalsen. Combining quality of service and social information for ranking services. In *ICSOC/ServiceWave*, pages 561–575, 2009.