Top-k web services compositions: A fuzzy-set-based approach

Karim Benouaret, Djamal Benslimane, Allel Hadjali

Top-k Service Compositions: A Fuzzy Set-Based Approach

Karim Benouaret
Lyon 1 University
69622, Villeurbanne, France
karim.benouaret@liris.cnrs.fr

Djamal Benslimane
Lyon 1 University
69622, Villeurbanne, France
djamal.benslimane@liris.cnrs.fr

Allel Hadjali
Enssat-University of Rennes 1
22305, Lannion, France
allel.hadjali@enssat.fr

ABSTRACT

Data as a Service (DaaS) is a flexible way that allows enterprises to expose their data. Composition of DaaS services provides bridges to answer queries. User preferences are becoming increasingly important to personalizing the composition process. In this paper, we propose an approach to compose DaaS services in the context of preference queries where preferences are modeled by means of fuzzy sets that allow for a large variety of flexible terms such as 'cheap', 'affordable' and 'fairly expensive'. The proposed approach is based on RDF-based query rewritings to take into account the partial matching between individual DaaS services and parts of the user query. Matching degrees between DaaS services and fuzzy preference constraints are computed by means of different constraints inclusion methods. Such degrees express to which extent a service is relevant to the resolution of the query. A fuzzification of Pareto dominance is also proposed to better rank composite services by computing the score of services. The resulting scores are then used to compute the top-k DaaS service compositions that cover the user query.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query formulation, Search process, Selection process; H.3.5 [**Online Information Services**]: Web-based services

General Terms

Measurement, Theory, Performance

Keywords

Fuzzy dominance, Web service, Top-k compositions, fuzzy preferences queries.

1. INTRODUCTION

Nowadays, companies are increasingly moving towards a service-oriented architecture for data sharing on the Web by putting their data sources behind *Web services*, thereby providing a well-documented and interoperable method of interacting with their data [4, 5, 7]. In particular, if no single Web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfill the request. In this context, we talk about *DaaS (Data as a Service) Service Composition*, where services correspond to calls, i.e. queries, over the data sources.

When searching for items to purchase over the Web, customer *preferences* are critical in the search. A more general and suitable approach to model preferences is based on fuzzy sets theory [6]. Fuzzy sets are very well suited to the interpretation of linguistic terms, which constitute a convenient way for a user to express her/his preferences. For example, when expressing preferences about the 'price' of a car, users often employ fuzzy terms like 'rather cheap', 'affordable' and 'not expensive'.

However as DaaS services and providers proliferate, a large number of candidate compositions that would use different -most likely competing- services may be used to answer the same query. Hence, it is important to set up an effective service composition framework that would identify and retrieve the most relevant services and return the top-k compositions according to the user preferences.

The following example illustrates a typical scenario related to our previous discussion, showing the challenges in finding the top-k service compositions.

Running example. Let us consider an e-commerce system supporting users to buy cars. The system has access to the set of services described in Table 1. The symbols '$' and '?' denote inputs and outputs of DaaS services, respectively. Services providing the same functionality belong to the same service class. For instance, $\{S_{21}, S_{22}, S_{23}, S_{24}\}$ belong to the same class $\mathcal{S}_2$. Each service has its (fuzzy) constraints on the data it manipulates. For instance, the cars returned by S_{21} are of *cheap* price and *short* warranty.

Assume that a user Bob wants to buy a car. He sets his preferences and submits 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 can 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 painful and tedious. It raises the follow-

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

Table 1: Example of DaaS 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 $[3.5, 5.5]$

ing challenges: (i) how to understand the semantics of the published DaaS services to select the relevant ones that can contribute to answering the query at hand; (ii) how to retain the most relevant services (several similar DaaS services offer the same functionality but are associated with different constraints) that better satisfy the user's fuzzy preferences (preferences based on fuzzy terms); and (iii) how to generate the best k DaaS service compositions that satisfy the whole user query.

Contributions. We tackled the first challenge by proposing a semantic annotation of DaaS services and an efficient RDF-based query rewriting approach that generates automatically the DaaS compositions that cover a user query (which does not include any preference constraints) [2]. In this paper, we focus on the second and third challenges. We allow approximate query containment by selecting DaaS services that cover a part of the query even if their constraints match only partially the user preference constraints. Different constraints inclusion methods are investigated to compute the matching degrees between the services' fuzzy constraints and the fuzzy 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 scores computed are then leveraged to find the top-k DaaS service compositions that answer the user query.

The rest of the paper is organized as follows. In Section 2, we formally describe the studied problem. Section 3 describes the ranking approach of DaaS services which is mainly based on the fuzzy dominance relationship. Section 4 is devoted to the top-k DaaS service compositions generation method for answering queries. Section 5 presents the global architecture of our service composition-based preference query answering and shows some experimental results. We review related work in Section 6. Section 7 concludes the paper.

2. SERVICE COMPOSITION-BASED PREFERENCE QUERIES ANSWERING

2.1 Preference Queries

Users express their conjunctive preference queries over domain ontologies using a slightly modified version of SPARQL query language. For instance, query Q_1 given in Section 1 is expressed as follows:

```
SELECT ?n ?pr ?w ?pw ?co
WHERE  ?Au rdf:type AutoMaker  ?Au hasNationality 'French'
       ?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'affordable', ?w is 'around 18',
            ?pw is 'normal', ?co is 'medium' }
```

In our approach, user preferences are modeled using fuzzy sets [6]. 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. In particular, $\mu_F(x) = 1$ reflects full membership of x in F, $\mu_F(x) = 0$ absolute non-membership and $0 < \mu_F(x) < 1$ partial membership. Namely having $x, y \in F$, x is more preferable than y iff $\mu_F(x) > \mu_F(y)$. If $\mu_F(x) = \mu_F(y)$, x and y are equally preferred. In practice, membership functions are often of trapezoidal form represented by the quadruplet (A, B, a, b). A regular interval [A, B] can be seen as a fuzzy set represented by the quadruplet $(A, B, 0, 0)$. The semantics of all fuzzy terms of Table 1 are available in the `URL: http://vm.liris.cnrs.fr:36880/FuzzyTerms/.`

2.2 DaaS services

Let us assume that DaaS services are partitioned into different classes of services. A class S_j represents services with the same inputs, outputs, and providing the same functionality but different (fuzzy) constraints. A DaaS 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 DaaS services. Functionality G_j and constraints C_{ji} of a DaaS service S_{ji} are added to the standard WSDL descriptions in the form of annotations. The annotations are represented in the form of SPARQL queries. For instance, the following SPARQL query illustrates the functionality and service constraints of the DaaS service S_{21}:

```
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  ?z is 'cheap'  ?t is 'short'}
```

2.3 Discovering Relevant Services

Let Q be a conjunctive preference query. We use an efficient RDF query rewriting algorithm 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. The same part q_j is in general covered by one or more services that constitute a class of relevant services and is designated as class $\mathcal{S}_j$. A service $S_{ji} \in \mathcal{S}_j$ is said to be relevant to a query Q iff the functionality of S_{ji} completely matches the part query q_j and its constraints match completely or partially the preference constraints of q_j (i.e., the matching degree is strictly higher than 0).

As preference constraints of a query are expressed in the rich fuzzy sets framework, their matching degrees with DaaS services constraints may differ from one constraints inclusion method (CIM) to another. Each relevant service is then associated with $|M|$ matching degrees (if $M = \{m_1, ..., m_{|M|}\}$ is the set of the used methods). For instance, Table 2 shows the matching degrees between each service S_{ji} from Table 1 and its corresponding component q_j (of the query Q_1). The degrees are computed by applying the following CIMs:

- *Cardinality-based method (CBM)* [15]. Let C and C' be two fuzzy constraints, $Deg(C \subseteq C') = \frac{|C \cap C'|}{|C|}$.
- *Implication-based method (IBM)* [1]. $Deg(C \subseteq C') = min_x(\mu_C(x) \rightarrow_f \mu_{C'}(x))$ where $\rightarrow_f$ stands for a fuzzy implication. The following IBMs are used for our running example: Godel (G-IBM), Lukasiewicz (L-IBM) and Kleene-Diennes (K-IBM).

Due to lack of space, we do not present these methods. The service S_{11} covering the component q_1 does not have a matching degree because there are no constraints imposed by the user on q_1. However, each service covering the component 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 the component q_3, the first and second values represent the inclusion degrees of the constraints *power* and *consumption*, respectively.

Table 2: Matching Degrees between services and fuzzy preference constraints of Q_1

S_{ji}	q_j	CBM	G-IBM	L-IBM	K-IBM
S_{11}	q_1	-	-	-	-
S_{21}	q_2	$(1, 0.57)$	$(1, 0)$	$(1, 0)$	$(0.80, 0)$
S_{22}		$(0.89, 1)$	$(0, 1)$	$(0.90, 1)$	$(0.50, 1)$
S_{23}		$(0.20, 0.16)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
S_{24}		$(0.83, 0.88)$	$(0.60, 0.50)$	$(0.60, 0.50)$	$(0.60, 0.50)$
S_{31}	q_3	$(0.50, 0.36)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
S_{32}		$(0.79, 0.75)$	$(0, 0.25)$	$(0.60, 0.50)$	$(0.40, 0.50)$
S_{33}		$(0.21, 0.64)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
S_{34}		$(0.83, 0.85)$	$(0.50, 0.50)$	$(0.50, 0.50)$	$(0.50, 0.50)$

2.4 Problem Statement

Given a query $Q : - < q_1, ..., q_n >$ where each q_j is a subquery (query component). 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 $\{\mathcal{S}_1, ..., \mathcal{S}_n\}$ where a class $\mathcal{S}_j$ regroups the services that are relevant to a query part q_j. Given $|M|$ CIMs to compute the matching degrees between the constraints of relevant services and the user's preference. The problem we are interested in is how to rank DaaS services in each class $\mathcal{S}_j$ to select the most relevant services and how to rank generated DaaS service compositions to select the top-k ones that answer the fuzzy query Q.

3. FUZZY DOMINANCE AND FUZZY SCORES

3.1 Fuzzy dominance v.s. Pareto dominance

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 so doing, 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 that, however, seem comparable from a user point of view. 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, we have neither $u \succ v$ nor $v \succ u$, i.e. the instances 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 $u_1 = 1$. This is why it is interesting to fuzzify the dominance relationship to express the extent to which a matching degree (more or less) dominates another one. In line with the general fuzzification dominance approach discussed in [8], we define below a fuzzy dominance relationship 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:

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

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

$$\mu_{\gg}(x, y) = \left\{ \begin{array}{ll} 1 & if x - y > \lambda + \varepsilon \\ 0 & if x - y \leq \varepsilon \\ \frac{x-y-\varepsilon}{\lambda} & otherwise \end{array} \right\} \quad (2)$$

Where $\lambda > 0$, i.e. $\gg$ is more demanding than the idea of 'strictly greater'. We should also have $\varepsilon \geq 0$ in order to ensure that $\gg$ is a relation that agrees with the idea of 'greater' in the usual sense.

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 $|u \succ v| = |v \succ u| = 0$ provided by Pareto dominance, where $|u \succ v| = 1$ if $u \succ v$, 0 otherwise. In the following sections, we use the defined fuzzy dominance to compute scores of services and compositions.

3.2 Associating fuzzy score with a service

It is well known that under a single matching degree method (mono criteria), the dominance relationship is unambiguous. When multiple CIM are applied (multi-criteria), resulting in different matching degrees for the same couple of constraints, the dominance relationship becomes uncertain. The model proposed in [10], namely probabilistic skyline overcomes this problem. Contrariwise, Skoutas et al. show in [11] the limitations of the probabilistic skyline to rank Web 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 S_j.

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

$$FDS(S_{ji}) = \frac{1}{(|\mathcal{S}_j|-1)\,|M|^2} \sum_{h=1}^{|M|} \sum_{\substack{S_{jk}\in\mathcal{S}_j \\ k\neq i}} \sum_{r=1}^{|M|} deg(S_{ji}^h \succ S_{jk}^r) \quad (3)$$

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

3.3 Associating fuzzy score with a composition

Different DaaS service compositions can be generated from different $\mathcal{S}_j$ service classes to answer a user query. To rank such generated compositions, we extend the previous FDS definition to service composition and associate each composition with an FDS. The FDS of a composition C is an aggregation of different FDSs of its component services.

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

$$FDS(C) = \frac{1}{d} \sum_{j=1}^{n} d_j \cdot FDS(S_{ji_j}) \quad (4)$$

It is important to note that not all compositions are valid. A composition C of services is valid if (i) it covers the user query Q, (ii) it contains one and only one service from each service class $\mathcal{S}_j$, and (iii) it is executable. A composition is said to be executable if all input parameters necessary for the invocation of its component services are bound.

4. TOP-K DAAS SERVICE COMPOSITIONS

4.1 An Efficient Generation of Top-k Compositions

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. In the following, we provide an optimization technique to find the top-k compositions. This technique allows eliminating relevant services S_{ji} from their classes $\mathcal{S}_j$ before generating the compositions, i.e. services that we are sure that if they are composed with other ones, the obtained compositions are not in the top-k. The idea is: we first compute the score of each service in its class, then only the best services in each class are retained, after that we compose the retained services, finally, we compute the score of the obtained compositions and return the top-k ones. To this end, we introduce the following theorem[1].

Table 3: Services' scores and top-k services

Services	Class	Score	Top-k
S_{11}	$\mathcal{S}_1$	-	S_{11}
~~S_{21}~~	$\mathcal{S}_2$	0.487	S_{22} S_{24}
S_{22}		0.653	
~~S_{23}~~		0.035	
S_{24}		0.538	
~~S_{31}~~	$\mathcal{S}_3$	0.094	S_{32} S_{34}
S_{32}		0.593	
~~S_{33}~~		0.130	
S_{34}		0.743	

THEOREM 1. *Let $C = \{S_{1i_1}, ..., S_{ni_n}\}$ be a composition. Let top-k($\mathcal{S}_j$) (resp. top-k(compositions)) 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($\mathcal{S}_j$) $\Longrightarrow C \notin$ top-k(compositions)*

This theorem means that the top-k sets of the different service classes $\mathcal{S}_j$ are sufficient to compute the top-k compositions that answer the query at hand.

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

Table 4: Compositions' scores and top-k ones

Compositions	Score	Top-k
$C_1 = \{S_{11}, S_{22}, S_{32}\}$	0.623	C_2 C_4
$C_2 = \{S_{11}, S_{22}, S_{34}\}$	0.698	
$C_3 = \{S_{11}, S_{24}, S_{32}\}$	0.566	
$C_4 = \{S_{11}, S_{24}, S_{34}\}$	0.640	

4.2 Top-k DaaS Service Compositions Algorithm

The algorithm, hereafter referred as $TKDSC$, computes the top-k DaaS service compositions according to the fuzzy scores. The algorithm proceeds as following.

Step.1 compute the matching degrees (lines 1-13). Each class whose services cover a query component is added to the list of relevant classes. If its services touch the query's user preferences, we compute its different matching degrees according to the number of methods.

[1]Proofs excluded due to lack of space

Algorithm 1 TKDSC

Require: Q a preference query, $\mathcal{S}$ a set of service classes, $M = \{m_1, ..., m_{|M|}\}$ a set of methods, $k \in \mathbb{N}$, ε, λ

1: **for** all $\mathcal{S}_j$ 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: **for** all S_{ji} in $\mathcal{S}_j$ **do**
7: **for** all m in M **do**
8: MatchingDegree(C_{ji}, P_{q_j}, m);
9: **end for**
10: **end for**
11: **end if**
12: **end if**
13: **end for**
14: **for** all $\mathcal{S}_j$ in $\mathcal{R}$ **do**
15: **if** $P_{q_j} = \emptyset$ **then**
16: $top\text{-}k.\mathcal{S}_j \leftarrow random(\mathcal{S}_j, k)$;
17: **else**
18: **for** all S_{ij} in $\mathcal{S}_i$ **do**
19: ServiceScore(S_{ji});
20: **end for**
21: $top\text{-}k.\mathcal{S}_j \leftarrow$ top$(k, \mathcal{S}_j)$;
22: **end if**
23: **end for**
24: $\mathcal{C} \leftarrow$ ComposeServices$(top\text{-}k.\mathcal{S}_{j_1}, ..., top\text{-}k.\mathcal{S}_{j_m})$;
25: **for** all C in $\mathcal{C}$ **do**
26: CompositionScore(C);
27: **end for**
28: **return** top$(k, \mathcal{C})$;

Step.2 eliminating less relevant services (lines 14-23). 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 24-28). First, we compose services from only the retained ones, i.e. the top-k in each class. Then we compute the score of generated compositions and finally we provide the user with the top-k ones.

Figure 1 depicts some results obtained for queries with 2, 3 and 4 subqueries (query components). These results show that the number of subqueries adds only a slight increase in the execution time of the top-k algorithm. However, this execution time is increased significantly when the number of relevant services per class is large.

5. SYSTEM ARCHITECTURE

Figure 2 presents our implemented top-k DaaS service compositions system. The system consists of the three major modules: *Annotation Module*, *Interactive Query Formulation Module* and *Top-k Service Compositions Module*.

The *Annotation Module* allows service providers to annotate WSDL description files of services with fuzzy sets to represent linguistic terms and with SPARQL queries to represent the RPV and constraints of services. This annotation is implemented by adding a new element called 'rdfQuery' to the XML Schema of WSDL as in WSDL-S approach.

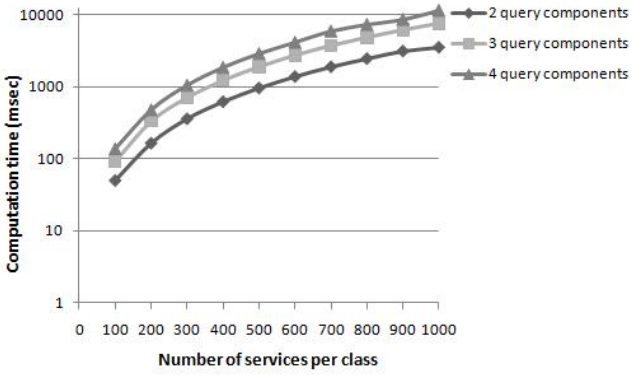

Figure 1: experimental results

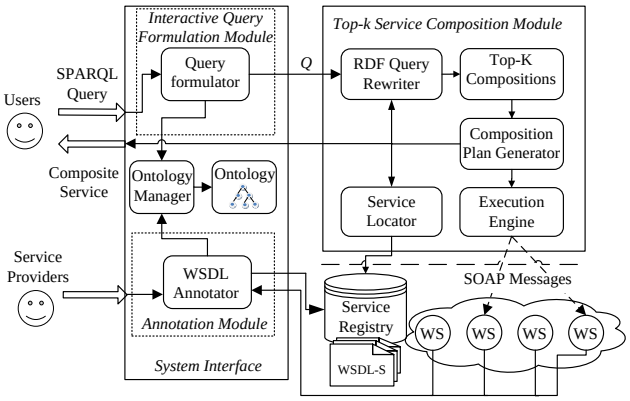

Figure 2: DaaS Service Composition Architecture

The annotated 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 *Interactive Query Formulation Module* 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 Module* consists of five components. The *RDF Query Rewriter* implements an efficient RDF query rewriting algorithm (*RDF Query Rewriter*) to identify the relevant services that match (some parts of) a user query. For that purpose, it exploits the RPVs in the service description files. 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 top-k 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 Saras-vati execution engine from Google.

6. RELATED WORK

Preferences in Web service selection/composition have re-

ceived much attention in the service computing community during the last years. Taking user preferences into account allows to rank candidate services/ compositions and return only the best k ones to the user. Hereafter, we review some works for ranking and selecting Web services. Authors in [9] propose a method to rank semantic web services. 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. Wang et al.[12] 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 information (stored in the system) of users to amend the preferences of active users, thus improving the results of service selection. In [13], the authors present a new ranking method, called ServiceRank. It considers the QoS aspects as well as the social perspectives of services (such as how they invoke each other via service composition). A service that provides good QoS and is invoked more frequently by others is more trusted by the community and will be assigned a higher rank.

The works the most related to our proposal are those proposed in [11, 14]. In [11], Skoutas et al. address ranking and clustering Web services and propose methods based on the notion of dominance; these methods apply multiple matching criteria without aggregating the match scores of individual services parameters. The notion of Web service dominance introduced relies on the one of uncertain dominance discussed in [10]. Two distinct scores are defined for assessing when a service is objectively interesting: (*i*) the dominated score of a service to indicate the average number of services by which it is dominated (*ii*) the dominating score of a service to indicate the average number of services that it dominates. Algorithms for computing the top-k Web services according to each score are provided. To achieve a balance between dominated and dominating scores, a third ranking score for a service that combines both, is introduced.

In [14], Yu and Bougettaya address the problem of uncertainty inherent in the Quality of Web Service (QoWS) and compute the skylines from service providers (referred to as service skylines). A service skyline can be regarded as a set of service providers that are not dominated by others in terms of QoWS attributes that interest all users. To this end, a concept called p-dominant service skyline is defined. A provider S belongs to the p-dominant skyline if the probability that S is dominated by any other provider is less than p. The authors provide also a discussion about the interest of p-dominant skyline w.r.t. the notion of p-skyline proposed in [10]. They show also that the former, on one hand, avoids requiring from users to transform personal preferences into numerical weights and, on the other hand, will not miss any 'really good' providers (i.e., the ones that provide consistently high QoWS).

7. CONCLUSIONS

In this paper, we addressed the top-k service compositions problem to answer fuzzy preference queries. The key notion of our approach is the fuzzy dominance relationship introduced to measure the extent to which a vector of matching degrees more or less dominates another one. This kind of fuzzy dominance is then used to both rank DaaS services in their classes and DaaS compositions in order to compute the top-k service compositions. Although we have focused on the DaaS services to answer a query, the proposed approach could be applied in other cases to compute, for instance, the top-k QoS-based service compositions.

8. REFERENCES

[1] W. Bandler and L. kohout. Fuzzy power and fuzzy implication operators. Fuzzy Sets and Systems, 4:13–30, 1980.

[2] M. Barhamgi, D. Benslimane, and B. Medjahed. A query rewriting approach for web service composition. IEEE Transactions on Services Computing (TSC), 15(5):795–825, jan 2010.

[3] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In IEEE International Conference on Data Engineering, ICDE 2001, pages 421–430, 2001.

[4] D. Butler. Mashups mix data into global service. Nature, 2006.

[5] M. Carey. This is your data on soa. In In IEEE Int. Conference on Services-Oriented Computing and Applications, 2007.

[6] D. Dubois and H. Prade. Fundamentals of fuzzy sets, volume 7 of The Handbooks of Fuzzy Sets. Kluwer Academic Pub, Netherlands, 2000.

[7] A. Jhingran. Enterprise information mashups: integrating information, simply. In In Proccedings of the 2006 VLDB, pages 3–4, 2006.

[8] M. Koppen and R. Vicente Garcia. A fuzzy scheme for teh ranking of multivariate data and its application. In In Proccedings of the 2004 Annual Meeting of the NAFIPS, pages 140–145, 2004.

[9] M. Palmonari, M. Comerio, and F. De Paoli. Effective and flexible nfp-based ranking of web services. In ICSOC-ServiceWave '09, pages 546–560, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In In Proccedings of the 2007 VLDB Conference, pages 20–27, 2007.

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

[12] H. Wang, J. Xu, and P. Li. Incomplete preference-driven web service selection. In SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing, pages 75–82, Washington, DC, USA, 2008. IEEE Computer Society.

[13] Q. Wu, A. Iyengar, R. Subramanian, I. Rouvellou, I. Silva-Lepe, and T. Mikalsen. Combining quality of service and social information for ranking services. In ICSOC-ServiceWave '09, pages 561–575, Berlin, Heidelberg, 2009.

[14] Q. Yu and A. Bouguettaya. Computing service skyline from uncertain qows. IEEE T. Services Computing, 3(1):16–29, 2010.

[15] L. Zadeh. A computational approach to fuzzy quantifiers in natural languages. Computer Mathematics with Applications, 9:149–183, 1983.