FuDoCS: A Web Service Composition System Based on Fuzzy Dominance for Preference Query Answering

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

[1] *Claude Bernard Lyon1 University, 69622 , Villeurbanne, France*
{karim.benouaret, djamal.benslimane, mahmoud.barhamgi}@liris.cnrs.fr
[2] *Enssat, University of Rennes 1, 22305, Lannion, France*
allel.hadjali@enssat.fr

1. INTRODUCTION

Modern enterprises are increasingly moving towards a service oriented architecture for data sharing by putting their data sources behind services, thereby providing an interoperable way to interact with their data. This class of services is known as DaaS *(Data-as-a-Service)* services. DaaS Composition is a powerful solution to answer the user's complex queries by combining primitive DaaS services. User preferences are a key aspect that must be considered in the service composition process. A more general and suitable approach to model preferences is based on fuzzy sets theory [3]. Fuzzy sets are very well suited to the interpretation of linguistic terms and 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*, etc. However as DaaS services 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.

Challenges. Consider the services from the car e-commerce in Table 1 (i.e., typical services that can be provided by systems like the e-Bay). The symbols "\$" and "?" denote inputs and outputs of 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 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 the 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". Answering Q_1 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 (several services offer the same functionality but are associated with different constraints) that better satisfy the user's preferences; and (iii) how to generate the best k compositions that satisfy the whole user query.

Table 1: Example of DaaS Services.

Service	Functionality	Constraints
$S_{11}(\$x, ?y)$	Returns the automakers y in a given country 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 $[3.5, 5.5]$

Contributions. We address the above challenges by proposing an automatic Web service compositions framework for preference query answering. This framework is based on a semantic annotation of DaaS services in the form of RDF graphs and an RDF-based query rewriting algorithm that generates automatically the relevant DaaS compositions that cover a user query (which does not include any preference constraints) [1]. In order to select the most relevant services, a fuzzy dominance relationship and fuzzy scores are proposed and implemented to rank-order both individual and composite services. Finally, the framework implements a first algorithm that efficiently compute the top-k service compositions and a second algorithm that diversify the top-k service compositions by using a quality metric that combines both diversity and service accuracy.

2. SYSTEM ARCHITECTURE

This section outlines the main components of our system architecture illustrated in Figure 1.

2.1 Fuzzy Membership Functions Manager

The *Fuzzy Membership Functions Manager* (FMFM) is useful to manage fuzzy linguistic terms. These terms are created and used to express user preferences and service

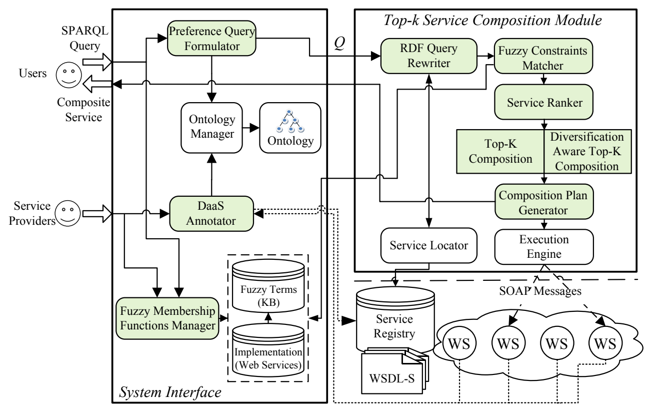


Figure 1: DaaS Service Composition Architecture.

constraints, they are modeled using fuzzy sets. Formally, a fuzzy set F on a referential X is characterized by a membership function $\mu_F : X \longrightarrow [0, 1]$, where $\mu_F(x)$ is the grade of membership of x in F. μ_F offers a convenient way for ordering the elements of X. 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. Fuzzy terms are stored in a knowledge base, their associated fuzzy membership functions are implemented as Web services. They are identified by the URL of their implementing Web services and can be shared among users.

2.2 Preference Queries

The *Preference Query Formulator* provides users with a GUI implemented with Java Swing to interactively formulate their conjunctive fuzzy preference queries over a domain ontology. The ontology manager uses Jena API to manage domain ontology. The resulting user query is represented using a slightly modified version 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}
PREFERING{?pr is ‘URL/AffordablesService’,
        ?w is ‘URL/around(18)Service’,?pw is ‘URL/NormalService’,
        ?co is ‘URL/MediumService’}
```

The “is” operator in the `PREFERING` clause is used to place fuzzy constraints (preferences) on the query's variables (e.g., the price ?pr should be *Affordable* as defined by the service on `URL/AffordableService`).

2.3 DaaS Annotator

The *DaaS Annotator* allows service providers to semantically annotate WSDL description files of services with (*i*) the semantic description of the service functionality and (*ii*) the (fuzzy) constraints of a service. This annotation is represented in the form of SPARQL queries. For instance, the following SPARQL query illustrates the functionality and constraints of the DaaS service 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’}
```

2.4 RDF Query Rewriter

The *RDF Query Rewriter* leverages an RDF query rewriting algorithm described in [1] to identify the relevant services that match (some parts of) a user query. For that purpose, it exploits the semantic description of the functionality of a service. The same subquery q_j of the initial query is in general covered by different 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 its functionality completely matches the sub query q_j. The preference constraints are not taken into account by this component.

2.5 Fuzzy Constraints Matcher

The *Fuzzy Constraints Matcher* (FCM) component is used to compute the matching degrees between (fuzzy) preference constraints and (fuzzy) service constraints for each relevant service. As preference constraints are expressed in the rich fuzzy sets framework, the computed matching degrees may differ from one Constraints Matching Method (CMM) to another. The FCM component associates to each relevant service M matching degrees (4 methods are implemented, i.e., $M = 4$) as depicted in Table 2. The service S_{11} covering the subquery q_1 does not have a matching degree because there are no constraints imposed by the user on q_1. Each service covering the q_2 is associated with four ($M = 4$) degrees 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 the matching degrees of the constraints *power* and *consumption*, respectively.

Table 2: Matching Degrees between Services' Constraints and Preference Constraints of Q_1.

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}		$(0.83, 0.85)$	$(0.83, 0.85)$	$(0.50, 0.50)$	$(0.50, 0.50)$

2.6 Services Ranker

The role of the *Service Ranker* component is to rank both individual and composite services. The ranking of services based on their associated matching degrees is useful to compute the top-k service compositions. We show below the limitation of Pareto dominance[1] and highlight why its fuzzification is important. We then introduce a particular fuzzy dominance and a way to compute fuzzy scores of services.

[1]For a d-dimensional dataset, a point u dominates another point v iff u is at least as good as v in all dimensions and (strictly) better than v in at least one dimension.

Let $u = (u_1, u_2) = (1, 0)$ and $v = (v_1, v_2) = (0.90, 1)$ be two matching degrees. In Pareto order, the points 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 [2]. We define below a fuzzy dominance relationship that relies on particular membership function of a graded inequality.

Fuzzy dominance. Given two d-dimensional points u and v, the fuzzy dominance expresses the extent to which u dominates v, it is defined as:

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

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

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

where $\lambda > 0$ and $\varepsilon \geq 0$. With this fuzzy dominance, the previous matching degrees u and v become comparable by computing $deg(u \succ v) = 0.25$ and $deg(v \succ u) = 0.5$. Further v is better than u.

Associating fuzzy score with a service. Under a single matching degree, the dominance relationship is unambiguous. When multiple CMM are applied, resulting in different matching degrees for the same couple of constraints, the (fuzzy) dominance relationship becomes uncertain. The probabilistic skyline proposed in [4] overcomes this problem, i.e., the problem of uncertainty. Contrariwise, Skoutas et al. show in [5] 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) for individual services. An FDS of a service S_{ji} indicates the average extent to which S_{ji} dominates the whole services of its class $\mathcal{S}_j$. It is defined as:

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

Where S_{ji}^h is the matching degree of the service S_{ji} obtained by applying the h^{th} CMM. The term $(|\mathcal{S}_j| - 1)$ is used to normalize the FDS score by making it in the range $[0, 1]$.

Associating fuzzy score with a composition. To be able to rank-order DaaS compositions, we associate each composition with an FDS. The FDS of a composition C is an aggregation of the 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 user preference constraints where d_j is the number of constraints involved in the service S_{ji}. The FDS of C is then computed as follows:

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

2.7 Top-k Service Compositions

Our system provides two ways of computing the top-k service compositions.

Top-k Compositions. The top-k compositions component is used to efficiently generate the compositions that better answer a query. A straightforward method to find the top-k compositions is to generate all possible compositions, compute their scores, and return the top-k ones. However, this approach incurs a high computational cost. 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 top-k sets of the different service classes $\mathcal{S}_j$ are sufficient to compute the top-k compositions that answer the user query.

Diversification-aware Top-k Compositions. Multiple similar services could exist in each class $\mathcal{S}_j$ leading to similar compositions. Diversification, which means that the considered services in each class must be dissimilar from each other, is then needed to improve the quality of the top-k compositions. In our proposed system, diversifying the top-k compositions is achieved by firstly diversifying the top-k services in classes $\mathcal{S}_j$ and then by diversifying the compositions themselves. To diversify the top-k of $\mathcal{S}_j$, we use the following quality metric that combines diversity and accuracy during the generation of the top-k of a class $\mathcal{S}_j$:

$$Quality(S_{ji}, \mathcal{S}_j^D) = FDS(S_{ji}) * RelDiv(S_{ji}, \mathcal{S}_j^D) \tag{5}$$

The quality of a service S_{ji} in its class $\mathcal{S}_j$ is proportional to its fuzzy score and to its relative diversity to those services so far selected $\mathcal{S}_j^D$. The relative diversity of S_{ji} to the set $\mathcal{S}_j^D$ is defined as following:

$$RelDiv(S_{ji}, \mathcal{S}_j^D) = \frac{\sum_{S_{jr} \in \mathcal{S}_j^D} (1 - Sim(S_{ji}, S_{jr})}{|\mathcal{S}_j^D|} \tag{6}$$

$Sim(S_{ji}, S_{jr})$ represents a similarity measure between two services S_{ji} and S_{jr}. Since services of the same class have the same functionality, the similarity measure is only computed from thier (fuzzy) constraints. The above quality measure guides the construction of the diversified top-k of $\mathcal{S}_j$ in an incremental way. During each step the remaining services are rank-ordered according to their quality and the highest quality service is added to $\mathcal{S}_j^D$.

2.8 Composition Plan Generation

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 Sarasvati execution engine from Google.

3. DEMO SCENARIOS

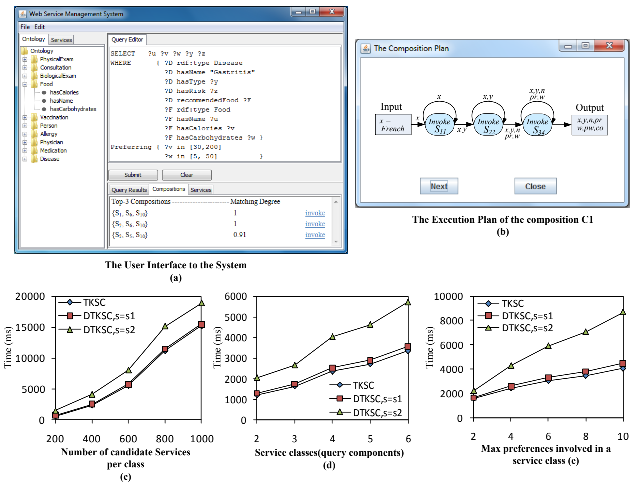
Demo 1. We implemented a set of /120/ DaaS services on top of /7/ databases holding synthetic commercial data about cars, clients, invoices, etc. Each DaaS service retrieves information about a set of cars satisfying some fuzzy constraints on their attributes (e.g., price, warranty, etc). We created an ontology with Protege too. Our first demonstration will illustrate how users can test the provided fuzzy terms, formulate their fuzzy preference queries, and how DaaS providers can annotate the description files of their services. Figure 2, plot a, presents the user interface to the composition system. Users edit their queries in the *Query Editor*. The panel on the left-hand side gives a tree view of

Table 3: 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	

Table 4: Effects of the Used Similarity Measure.

Diversified Top-k Compositions				
Component Services	Score	Quality		
		M	*L*	*N*
$\{S_{1356}, S_{2372}, S_{3285}, S_{4214}, S_{5183}\}$	0.6919484	0.6919484	0.6919484	0.6919484
$\{S_{1356}, S_{2372}, S_{3283}, S_{4214}, S_{5183}\}$	0.68804884	0.6744621	0.6615082	0.6780993
$\{S_{1356}, S_{2372}, S_{3360}, S_{4214}, S_{5183}\}$	0.69165516	0.6713853	0.6594182	0.6809209



```
SELECT   ?u ?v ?w ?y ?z
WHERE      { ?D rdf:type Disease
             ?D hasName "Gastritis"
             ?D hasType ?y
             ?D hasRisk ?z
             ?D recommendedFood ?F
             ?F rdf:type Food
             ?F hasName ?u
             ?F hasCalories ?v
             ?F hasCarbohydrates ?w }
Preferring { ?v in [30,200]
             ?w in [5, 50]          }
```

```
Top-3 Compositions ---------------------- Matching Degree
{S1, S6, S10}                             1              invoke
{S2, S6, S10}                             1              invoke
{S2, S5, S10}                             0.91           invoke
```

Figure 2: Demo.

the domain ontology. Executing the query specified in the query editor results in the compositions shown in the "Compositions" tab. We will demonstrate the query answering of Q_1 described in Section 1. Q_1 will be executed without its fuzzy constrains to demonstrate how DaaS services can be discovered and composed to answer the query. We will show the execution plan of the selected composition as depicted in Figure 2, plot b. The service S_{11} is invoked first with the required nationality for automakers x which is relayed also to its outputs. The inputs with which a service is invoked are always relayed to its outputs; this allows to get the query's requested data by joining the outputs of services that are *leaves* (e.g. S_{34}) in the execution plan. Then S_{22} is invoked with the retrieved automakers, and finally S_{34} is invoked to retrieve the consumption and the power of retrieved cars. The query Q_1 will be then re-executed with its fuzzy preferences and we will show how the top-k and diversified top-k compositions are generated to answer the query. In addition, we will compare the returned compositions when the fuzzy dominance is applied to those obtained if the Pareto dominance were applied and show to what extent the fuzzy dominance improves the quality of returned compositions. Furthermore, for the same fuzzy term (e.g. Affordable), we will vary its membership functions both in Q_1 and the considered set of DaaS services and show to what extent this may influence the returned compositions.

Demo 2. We will present an experimental study of our approach conducted on a Pentium D 2:4GHz with 2GB of RAM, running Windows XP to highlight its efficiency and scalability. We will show our implemented Web service generator. It 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. We considered a set of /15/ model Web services from the domain of car commerce having about /20/ attributes (e.g. *Price*, *Warranty*, *Power*, *Consumption*, etc.). We will show the effects of the following parameters on the performance of our system and on the quality of the computed compositions: (*i*) the number of services per class (Figure 2, plot c), (*ii*) the service classes number (Figure 2, plot d), (*iii*) the number of fuzzy constraints per class (Figure 2, plot e), (*iv*) the number of considered matching methods, (*v*) the considered value of *k*, (*vi*) the effects of the used similarity measure SIM (three measures were implemented denoted by M, L and N in Table 4), and (*vii*) the effects of ε and λ (Table 3).

4. REFERENCES

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

[2] K. Benouaret, D. Benslimane, and A. Hadjali. Top-k service compositions: A fuzzy set-based approach. In *SAC*, pages 1033–1038, 2011.

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

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

[5] 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.