Web Service Compositions with Fuzzy Preferences: A Graded Dominance Relationship Based Approach

KARIM BENOUARET, LIRIS, Université Claude Bernard Lyon 1
DJAMAL BENSLIMANE, LIRIS, Université Claude Bernard Lyon 1
ALLEL HADJALI, LIAS, ENSMA – Poitiers
MAHMOUD BARHAMGI, LIRIS, Université Claude Bernard Lyon 1
ZAKARIA MAAMAR, Zayed University
Quan Z. SHENG, The University of Adelaide

Data-driven Web services build on service-oriented technologies to provide an interoperable method of interacting with data sources on top of the Web. Data Web services composition has emerged as a flexible solution to answer users' complex queries on the fly. However, as the number of Web services on the Web is growing quickly, a large number of candidate compositions that would use different (most likely competing) services may be used to answer the same query. User preferences are a key factor that can be used to rank candidate services/compositions and retain only the best ones. In this paper, we present a novel approach to compute the top-k data service compositions based on user preferences. In our approach, we model user preferences using fuzzy sets and incorporate them into the composition query. We use an efficient RDF query rewriting algorithm to determine the relevant services that may be used to answer the composition query. We match the (fuzzy) constraints of the relevant services to those of the query and determine their matching degrees using a set of matching methods. We then rank-order the candidate services based on a fuzzification of Pareto dominance and compute the top-k Data service compositions. In addition, we introduce a new method to increase the diversity of returned top-k compositions while maintaining as much as possible the compositions with the highest scores. Finally, we describe the architecture of our system and present a thorough experimental study of our proposed techniques and algorithms. The experimental study demonstrates the efficiency and the effectiveness of our techniques in different settings.

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

General Terms: Algorithms, Measurement, Performance

Additional Key Words and Phrases: Web services, service composition, preference queries, top-k, diversity

ACM Reference Format:
Benouaret, K., Benslimane, D., Hadjali, A., Barhamgi, M., Maamar, Z., and Sheng, Q. Z. 2014. Web Service Compositions with Fuzzy Preferences: A Graded-Dominance-Relationship-Based Approach *ACM Trans. Internet Technol.* x, x, Article 34 (October 2014), 34 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

Author's addresses: Karim Benouaret, Computer Science Department, University of Lyon 1, Villeurbanne, France; email: karim.benouaret@liris.cnrs.fr; Djamal Benslimane, Computer Science Department, University of Lyon 1, Villeurbanne, France; email: djamal.benslimane@liris.cnrs.fr; Mahmoud Barhamgi, Computer Science Department, University of Lyon 1, Villeurbanne, France; email: mahmoud.barhamgi@liris.cnrs.fr; Allel Hadjali, ENSMA – Poitiers, Chasseneuil, France; email: allel.hadjali@ensma.fr; Zakaria Maamar, College of Information Technology, Zayed University, United Arab Emirates; email: zakaria.maamar@zu.ac.ae; Quan Z. Sheng, School of Computer Science, the University of Adelaide, Australia; email: michael.sheng@adelaide.edu.au.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2014 ACM 1533-5399/2014/10-ART34 $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Recent years have witnessed a growing interest in the use of Web services as a reliable means for e-commerce, content publication and management, enabling users to perform several operations, like searches, purchases and data uploads [Carey et al. 2012; Dustdar et al. 2012]. This type of Web services is known as *data-driven Web services* [Deutsch et al. 2004] or data services for short, where Web services are typically powered by databases. Moreover, Web users often need to compose different Web services to achieve a more complex task that cannot be fulfilled by an individual Web service.

User preferences play a major role in the customization of the composition process. Therefore, it is important to help users formulate their preferences since they may not be able to identify their concrete preferences, i.e., their preferences are with some fuzziness. A more general and crucial approach to represent this kind of preferences is based on the fuzzy sets theory [Dubois and Prade 2000][Hadjali et al. 2008]. Fuzzy sets are very appropriate for the interpretation of linguistic terms, which constitute a convenient way for users to express their preferences. For example, when expressing preferences about the "price" of a car, users often employ linguistic terms like "*rather cheap*", "*affordable*" and "*not expensive*".

One of the most challenging problems in data service composition is that due to the proliferation of data services and service providers, a large number of candidate compositions that would use different (most likely competing) data services may be used to answer the same query. It is therefore important to set up an effective data service composition framework that would identify and retrieve the most relevant data services and return the top-k data service compositions according to the user preferences.

The following example presents a typical scenario that clearly shows the different challenges involved in finding the top-k data service compositions.

1.1. Motivating example

Consider a set of car trading Web services in Table I (i.e., typical data services that can be provided by systems like the e-Bay). The symbols "$" and "?" denote inputs and outputs of data services, respectively. Services providing the same functionality belong to the same service class. For instance, the services s_{21}, s_{22}, s_{23} and s_{24} belong to the same class $\mathcal{S}_2$. Each data 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.

Table I: Example of data services

Data 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 $[3.5, 5.5]$

Let us now assume that a user, Bob, wants to buy a car. He sets his preferences and submits the following query $\mathcal{Q}_1$: "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 uses the services described in Table I to obtain such information. He will have to invoke s_{11} to retrieve the French automakers, then invoke one or more

of the data 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 data services $s_{31}, s_{32}, s_{33}, s_{34}$ to retrieve the power and the consumption of retrieved cars.

To select the car that better satisfies his requirements, Bob needs to go through a series of trial-run processes. If the number of available services is large, this manual process would be very painstaking and raises the following challenges:

— how to understand the semantics of the published data services to select the relevant ones that can contribute to answering the query at hand.
— how to retain the most relevant data services (several similar data services offer the same functionality but are associated with different constraints) that better satisfy the user's fuzzy preferences (i.e., preferences based on fuzzy terms).
— how to generate the best k data service compositions that satisfy the query.

1.2. Contributions

We already tackled the first challenge by proposing in [Barhamgi et al. 2010] a semantic annotation of data services that describes the services functionality and an efficient RDF-based query rewriting approach that generates automatically the data service compositions for a given query (which does take into account any user preference). In this paper, we focus on the second and third challenges. We leverage our RDF query rewriting algorithm to find the relevant data services that can contribute to the resolution of a given preference query. Since the number of candidate data services for a composition may be still large, performing an exhaustive search, i.e., generate all possible combinations, to find the best data service compositions is not practical as the problem of composition using query rewriting techniques is known to be NP-hard [Ludäscher and Nash 2004; Deutsch et al. 2007; Li and Chang 2001], i.e., any exact solution to this problem has an exponential cost. Therefore, reducing the search space by focusing only on the best data services of each service class is crucial for reducing the computational cost. Our main contributions include the following:

— As data services of the same class have the same functionality and only differ in their constraints, the relevance of each service w.r.t. a given query can be reduced to the relevance of their constraints w.r.t. the user preferences. For this purpose, we investigate multiple methods for computing the matching degrees between the preferences involved in the query and the data services' constraints.
— We present a method for further reducing the search space by examining only the top-k data services of each service class. In particular, we define a ranking criterion based on a fuzzy dominance relationship in order to select the top-k data services in each service class, we then compose these data services and return only the top-k data service compositions.
— To avoid returning similar data service compositions, i.e., those returning similar informations, we also propose a diversified top-k data service compositions method that aims to both improve the diversity of top-k selection and maintain as possible top-k highest ranked ones.
— We propose a comprehensive architecture of our composition system and evaluate our approach through a set of thorough experiments.

The rest of the paper is organized as follows. In Section 2, we provide the necessary background on fuzzy sets. In Section 3, we formally define the studied problem. Section 4 describes the proposed fuzzy dominance relationship and a ranking approach for data services. Section 5 is devoted to both top-k and diversified top-k data service composition methods for answering preference queries. Section 6 presents the architecture of our implemented composition system for preference query answering and

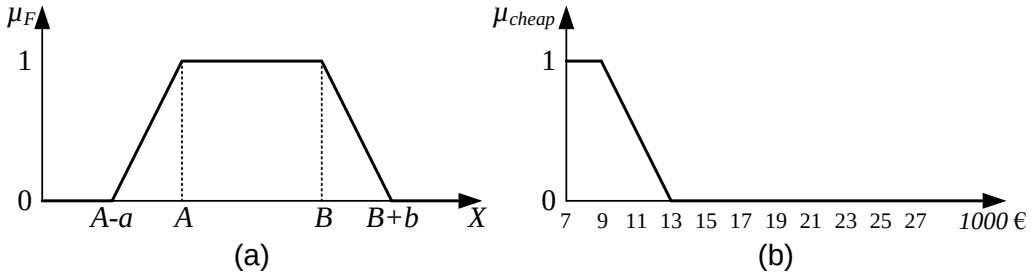

Fig. 1: Trapezoidal Membership Functions Examples

reports the results of a set of thorough experimental evaluations. In Section 7, we review the related work in the literature. Finally, Section 8 concludes the paper and outlines some perspectives for future work.

2. BACKGROUND ON FUZZY SETS

2.1. Basic Notions

Fuzzy set theory was introduced by Zadeh [Zadeh 1965] to model sets whose boundaries are not well defined. Typical examples are those described using adjectives of the natural language, such as "cheap", "affordable" and "expensive". For such sets, the transition between full membership and full mismatch is gradual rather than crisp.

Formally, a fuzzy set $\mathcal{F}$ on a referential $\mathcal{X}$ is characterized by a membership function $\mu_{\mathcal{F}} : \mathcal{X} \rightarrow [0, 1]$ where $\mu_{\mathcal{F}}(x)$ denotes the grade of membership of x in $\mathcal{F}$. In particular, $\mu_{\mathcal{F}}(x) = 1$ reflects full membership of x in $\mathcal{F}$, while $\mu_{\mathcal{F}}(x) = 0$ means absolute non-membership. When $0 < \mu_{\mathcal{F}}(x) < 1$, x has partial membership in $\mathcal{F}$. $\mathcal{F}$ is normalized if $\exists x \in \mathcal{X}, \mu_{\mathcal{F}}(x) = 1$.

Two crisp sets are of particular interest when defining a fuzzy set $\mathcal{F}$:

— The core $C(\mathcal{F}) = \{x \in \mathcal{X} \mid \mu_{\mathcal{F}}(x) = 1\}$, which gathers the *prototypes* of $\mathcal{F}$.
— The support $S(\mathcal{F}) = \{x \in \mathcal{X} \mid \mu_{\mathcal{F}}(x) > 0\}$, which contains the elements that belong to some extent to $\mathcal{F}$.

In practice, the membership function associated with $\mathcal{F}$ has often a trapezoidal shape. Then, $\mathcal{F}$ is expressed by the quadruplet (A, B, a, b) where $C(\mathcal{F}) = [A, B]$ and $S(\mathcal{F}) = (A - a, B + b)$ (see Figure 1). A regular interval [A, B] can be seen as a fuzzy set represented by the quadruplet $(A, B, 0, 0)$.

Let $\mathcal{F}$ and $\mathcal{G}$ be two fuzzy sets in the universe (i.e., referential) $\mathcal{X}$, $\mathcal{F} \subseteq \mathcal{G}$ iff $\forall x \in \mathcal{X}, \mu_{\mathcal{F}}(x) \leq \mu_{\mathcal{G}}(x)$. The complement of $\mathcal{F}$, denoted by $\mathcal{F}^c$, is defined by $\mu_{\mathcal{F}^c}(x) = 1 - \mu_{\mathcal{F}}(x)$. The cardinality of $\mathcal{F}$ is defined by $|\mathcal{F}| = \sum_{x \in \mathcal{X}} \mu_{\mathcal{F}}(x)$. Furthermore, $\mathcal{F} \cap \mathcal{G}$ (resp. $\mathcal{F} \cup \mathcal{G}$) is defined in the following way:

— $\mu_{\mathcal{F} \cap \mathcal{G}} = \top(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))$ where $\top$ is a t-norm operator that generalizes the conjunction operation (e.g., $\top(x, y) = min(x, y)$ and $\top(x, y) = x \cdot y$).
— $\mu_{\mathcal{F} \cup \mathcal{G}} = \bot(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))$ where $\bot$ is a co-norm operator that generalizes the disjunction operation (e.g., $\bot(x, y) = max(x, y)$ and $\bot(x, y) = x + y - x \cdot y$).

As usual, the logical counterparts of the theoretical set operators $\cap$, $\cup$ and complementation correspond respectively to conjunction $\wedge$, disjunction $\vee$ and negation $\neg$. Interested readers are referred to [Dubois and Prade 2000] for more details.

A fuzzy implication is a mapping $\mathcal{I} : [0, 1]^2 \rightarrow [0, 1]$ satisfying the boundary conditions $\mathcal{I}(0, x) = 1$ and $\mathcal{I}(1, x) = x$ for all x in [0, 1]. Moreover, it is required that $\mathcal{I}$ be decreasing

in its first, and increasing in its second component. Two families of fuzzy implications are studied in the fuzzy literature (due to their semantic properties and the fact that their results are similar with the ones of usual implications, material implications, when the arguments are 0 or 1):

— ***R-implications***: are defined by $\mathcal{I}(x,y) = sup\{\beta \in [0,1], \top(x,\beta) \leq y\}$, where $\top$ is a t-norm operator. The two most used R-implications are *Godl* implication ($\mathcal{I}_{Gd}(x,y) = 1$ if $x \leq y$, 0 otherwise) and *Goguen* implication ($\mathcal{I}_{Go}(x,y) = 1$ if $x \leq y$, y/x otherwise).
— ***S-implications***: are defined by $\mathcal{I}(x,y) = \perp(1-x,y)$, where $\perp$ is a co-norm operator. The two most popular S-implications are *Kleene-Dienes* implication ($\mathcal{I}_{Kl}(x,y) = max((1-x,y))$ and *Lukasiewicz* implication ($\mathcal{I}_{Lu}(x,y) = min(1-x+y,1)$).

For a complete presentation on fuzzy implications, readers are invited to check [Dubois and Prade 2000].

2.2. Modeling Preferences

Fuzzy sets provide a suitable tool to express user preferences [Dubois and Prade 1996][Hadjali et al. 2008]. The notion of membership functions is used to describe these preferences for each attribute domain involved in the query. The more the degree of an element x is close to 1, the preferred is. Appendix A shows the membership functions of the preferences involved in the query of our example.

In this context, the user does not specify crisp (Boolean) criteria, but fuzzy (gradual) ones like "affordable", "very cheap" and "fairly expensive", whose satisfaction is a matter of degree. In complex queries, individual satisfaction degrees associated with elementary conditions are combined using a panoply of fuzzy set connectives, which may go beyond conjunctive and disjunctive aggregations. Such connectives can capture the different user's attitudes concerning the way the different criteria present in his/her query compensate or not.

Then, the result of a given query is no longer a flat set of elements but a set of rank-ordered elements according to their global satisfaction w.r.t. the fuzzy criteria appearing in the query. So, a complete pre-order is obtained. One can limit the number of answers by using a quantitative calibration (i.e., return the top-k answers) or a qualitative calibration (i.e., return the answers that satisfy the query with a degree above a threshold η).

3. PREFERENCES-BASED DATA SERVICE COMPOSITION MODEL

3.1. Preference Queries

We adopt a declarative approach to Web services composition, i.e., instead of selecting and composing Web services manually, users formulate their composition queries over domain ontologies. We consider conjunctive preference queries expressed over domain ontologies using a slightly modified version of SPARQL, the de facto query language for the Semantic Web. Figure 2 depicts a portion of the mediated ontology in an e-commerce domain, in particular the automobile domain.

Formally, a conjunctive preference query $\mathcal{Q}$ has the form $\mathcal{Q}(X)$:-$<\varphi(X,Y), \mathcal{P}>$, where:

— $\mathcal{Q}(X)$ is the head of $\mathcal{Q}$, has the form of a relational predicate and represents the result of the query.
— $\varphi(X,Y)$ is the body of $\mathcal{Q}$, contains a set of RDF triples where each triple is of the form (`subject.property.object`). X and Y are called *distinguished* and *existential* variables, respectively. Distinguished variables appear in the query head $\mathcal{Q}(X)$ (they may also appear in the query body $\varphi(X,Y)$). In contrast, existential variables appear only in the query body.

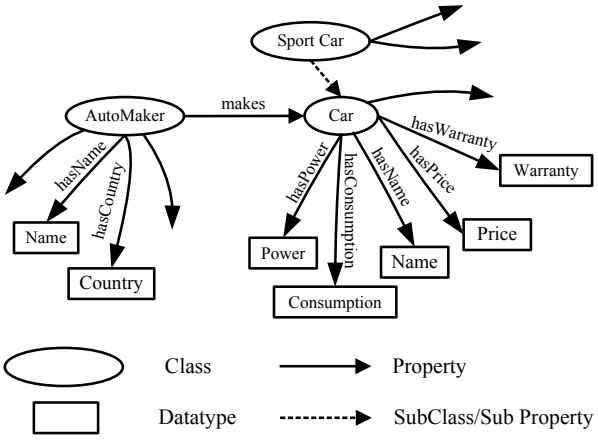

Fig. 2: Sample of Ontology

— $\mathcal{P} = \{p_1, ..., p_d\}$ is a set of preferences expressed using fuzzy sets on X and Y variables.

Membership functions of fuzzy terms are implemented as Web services and can be shared by users. They are used in the `PREFERRING` clause of the query where the URL of the implementing Web service is mentioned. More details are provided in Section 6. The head and body of $\mathcal{Q}$ are defined in `SELECT` and `WHERE` clauses, respectively.

For instance, $\mathcal{Q}_1$ of the example given in Section 1 is expressed as follows:

```
URL=http://soc.univ-lyon1.fr:8080/FunctionsDescription/index.jsp
SELECT ?a ?b ?c ?d ?e ?f
WHERE {?Au rdf:type AutoMaker  ?Au hasCountry 'France'  ?Au makes ?C
       ?Au hasName ?a  ?C rdf:type Car  ?C hasName ?b  ?C hasPrice ?c
       ?C hasWarranty ?d  ?C hasPower ?e  ?C hasConsumption ?f}
PREFERING {?c is 'URL/Affordable', ?d is 'URL/around(18)',
           ?e is 'URL/Normal', ?f is 'URL/Medium'}
```

As said earlier, the query head $\mathcal{Q}(X)$ corresponds to the `SELECT` clause, thus the distinguished variable set X includes "$?a$", "$?b$", "$?c$", "$?d$", "$?e$" and "$?f$". The existential variable set Y includes "$?Au$", and "$?C$".

The predicate `?d is 'URL/around(18)'` means that the user prefers data services that provide cars with a warranty of around 18 months. The semantics of "around 18" is given in http://soc.univ-lyon1.fr:8080/FunctionsDescription/index.jsparound(18). `SELECT` and `WHERE` clauses define the head and body of $\mathcal{Q}$, respectively. `PREFERING` clause indicates the preferences in $\mathcal{Q}$. Figure 3 is the graphical representation of query $\mathcal{Q}_1$. *Automaker* and *Car* ovals are concepts in the ontology. Arcs (e.g., *Constructs*, *hasPrice*, etc) are properties in the ontology. *A* and *C* ovals are existential variables, and *a*, *b*, *c*, *d*, *e* and *f* are distinguished variables.

3.2. Data Services

The functionalities of data services, as opposed to traditional Web services that encapsulate software artifacts, can be only captured when representing the semantic relationship between inputs and outputs [Barhamgi et al. 2010; Martin et al. 2004]. Therefore, we modeled data services as RDF Parameterized Views (RPVs) over do-

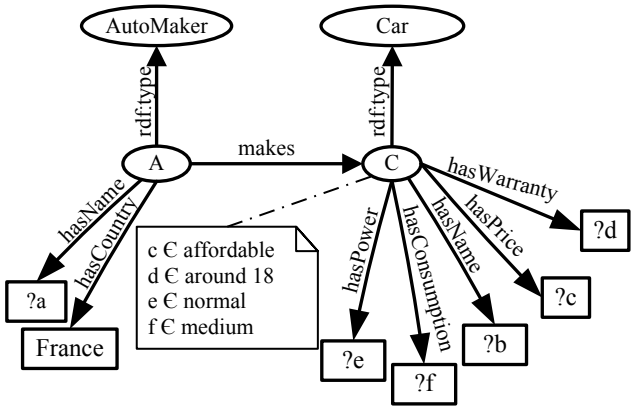

Fig. 3: Graphical Representation of the (Fuzzy) Composition Query

main ontologies. Each view captures the semantic relationships between input and output sets of a data service using concepts and relations whose semantics are formally defined in ontologies. Functionalities of data services are provided under some data constraints. For example, z is $cheap$, t is $short$ (for data service s_{21} in Table I).

Formally, a data service s_{ij} is described as a predicate $s_{ij}(\$X_i, ?Y_i)$:- $<\phi_i(X_i, Y_i, Z_i), \mathcal{C}_{ij}>$ where:

— X_i and Y_i are the sets of input and output variables of s_{ij}, respectively. Input and output variables are also called distinguished variables. They are prefixed with the symbols "$" and "?", respectively.
— $\phi_i(X_i, Y_i, Z_i)$ represents the functionality of the data service. This functionality is described as a semantic relationship between input and output variables. Z_i is the set of existential variables relating X_i and Y_i.
— $\mathcal{C}_{ij} = \{C_{ij_1}, ..., C_{ij_i}\}$ is a set of constraints expressed as intervals or fuzzy sets on X_i, Y_i or Z_i variables.

Each data service requires a particular set of inputs (parameter values) to retrieve a particular set of outputs; i.e., outputs cannot be retrieved unless inputs are bound. For example, one cannot invoke data service s_{31} without specifying the car for which it need to know the power and the consumption. Inputs and Outputs are prefixed with "$" and "?", respectively in the head of the view ($s_{ij}(\$X_i, ?Y_i)$). We annotate the service description files (e.g., HTML pages for RESTful Web services and WSDLs for SOAP- based Web services) with the defined RDF views and (fuzzy) constraints. Annotations have concretely the form of SPARQL queries (extended with preferences). For instance, the following SPARQL query represents the functionality and constraints on data service s_{21} in Table I:

```
URL=http://soc.univ-lyon1.fr:8080/FunctionsDescription/index.jsp
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/Cheap', ?t is 'URL/Short'}
```

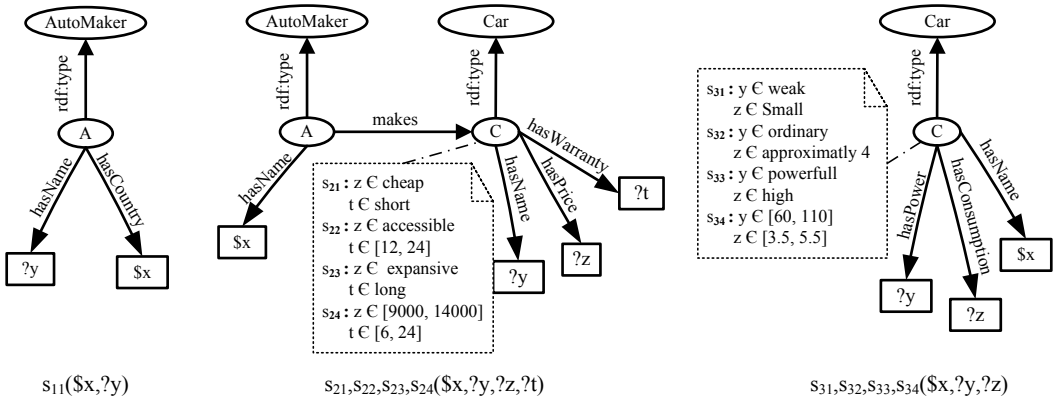

Fig. 4: Functionalities of Data Services in the Running Example

SELECT and WHERE clauses define the functionality of s_{21} and CONSTRAINTS clause gives the fuzzy constraints on service s_{21} given in Table I. Figure 4 gives the graphical representation of the data services given in Table I.

Annotations can be straightforwardly added to service description files. For-SOAP based services, WSDL files can be annotated using the extensibility feature of WSDL. In fact, the specification of WSDL allows the addition of new XML elements and attributes in certain locations to a WSDL file. This is known as the extensibility feature of WSDL. We exploit this feature as follows: for each "operation" element, we define a new child element called "RDFView" to hook the operation with the SPARQL query capturing its semantics. Appendix B, gives an example of an annotated WSDL file.

For RESTful Web services, the semantic annotations can be added to their HTML pages, as most of these services have HTML pages that describe to users what the service does and how to invoke it. The SA- REST proposal [Sheth et al. 2008] already defined different RDF predicates to annotate HTML pages of RESTful services with semantic information about their inputs, outputs, operations and faults. We exploit '*sarest:operation*" to link the operation to its associated SPARQL query. Appendix B, gives an example of an annotated HTML page.

3.3. Rationale and Benefits of Using Service Constraints

In this section, we discuss the reasons and benefits of adopting constraints to enrich the semantic descriptions of data services.

From a modeling point of view, service providers can define their data services in different ways. They can adopt a restrictive way in which explicit constraints are specified on service parameters (i.e., on inputs and outputs). For example, a data service $s_{21}(\$automaker, ?car, ?price, ?warranty)$ can be defined in a restrictive way by specifying a constraint on the "$?price$" parameter to return only cheap cars for a given automaker, i.e. the service is implemented in such a way to return only cheap cars. They can also adopt a non-restrictive way with no constraints specified at all. For example, the service s_{21} can be defined as follows $s_{21}(\$automaker, \$price1, \$price2, ?car, ?warranty)$ to return the cars whose prices are in the supplied range $[price1, price2]$ and made by a given automaker; i.e., there is no constraint on the price parameter. Note that the price is an output in the first case and an input in the second.

When all services are defined in a non-restrictive way, service composition algorithms (such as our query rewriting algorithm [Barhamgi et al. 2010]) are able to

generate all service compositions that cover a user query. In this case, there is no way to select the best services (and the compositions thereof), as we have no knowledge about the data manipulated by services. In the absence of such knowledge, the user could select a composition returning an empty result set at the composition execution time. For example, service s_{21} could be employed in the selected composition and could return no data for a given range of prices (supplied by the user). This will be known only at the composition execution time.

Data services' constraints, when specified in a service description, allow the differentiation between data services based on their manipulated data and help users select the best services and compositions that do return relevant data at the service/composition execution time. In this paper, we consider data services enriched with constraints characterizing their accessed data and focus on selecting the best services that do return relevant data at execution time.

There are benefits of defining data services in a restrictive way with constraints on the manipulated data, as assumed in this paper.

— It makes the contextual information about a service's provided data explicit. Such contextual information could not be learned from service signature alone, and often are not stored in the provider's database. For example, all cars in $s(\$automaker, ?car, ?price)$ are cheap with *implicitly two months of warranty*. This information about warranty is only known by the service provider and his regular data consumers.
— It allows for representing the dependency relationships between service parameters. For example, it is hard to represent in the case of a non-restrictive service definition the fact that the warranty depends on the price (the warranty is low when the price is low; it is high when price is high). Defining such a dependency is easier in the case of restrictive service definition by having two constraints on price and warranty, respectively. For example, $s(\$automaker, ?price, ?warranty, ?cars)$ annotated with the constraints warranty $< 6moths$ and price < 3000 shows that the low warranty of the cars returned by s is limited since the price is low too.
— It allows for avoiding empty query responses. Defining services with ranges given as input parameters will not allow for qualifying the data that the service can return. In some cases, the set of returned data can be simply empty. Defining data services in a restrictive way will certainly reduce the risk of empty responses since only services whose constraints more or less match the user constraints are selected.
— It allows for discriminating data that can be provided by different and may be competitive providers.

Constraint derivation and learning are an important issue. One approach is to query data services in an exhaustive manner. However, this is unpractical and may not be authorized by service providers. In this paper, we suppose that service providers describe the semantics and constraints of their services. This may not necessarily make service providers lose their potential service consumers, as there is no restriction on the number of services that the same provider can provide. For example, a provider may provide three semantically identical services s, s', and s'', but with different price ranges: $[16000, 24000]$, $[9000, 17000]$, and $[3000, 11000]$, respectively. In practice, when the stored data objects set is very large (as is in our example), service providers tend to provide, for performance concerns, multiple data services accessing different portions (or clusters) of the data set and optimized to provide fast and real-time data access instead of providing one single data service accessing the whole data set with a degraded performance. A rich number of clustering techniques were proposed in the literature to cluster data based on data constraints [Wagstaff and Cardie 2000; Davidson 2009; Wagstaff 2010]. Note that the fuzzy constraints that service providers use to describe

their data services could be simply the clustering functions that are used to segregate the data set into different portions.

3.4. Discovering Relevant Services

Given a preference query $\mathcal{Q}$ and a set of candidate data services, we show in this section how we select relevant data services to $\mathcal{Q}$ and classify them into a set of classes $\mathbb{S}$. We rely on an efficient RDF query rewriting algorithm [Barhamgi et al. 2010] to select services based on their functionality. Our rewriting algorithm has two phases.

The first phase is to find relevant services. The algorithm compares the RDF composition query with the RDF views of available data services and determines the parts covered by each of the views. The algorithm stores information about covered nodes and object properties as a partial containment mapping in a mapping table. The mapping table points out the different possibilities of using an RDF view to cover a part of $\mathcal{Q}$. We illustrate this phase based on our example (RDF representations of the query and the services are given in Figure 3 and Figure 4, respectively). Data service s_{11} covers node $A_{\mathcal{Q}}(?a,$"France") with the partial mapping $A_{\mathcal{Q}} \rightarrow A_{s_{11}}$, France $\rightarrow x$, $a \rightarrow y$. Note that it covers the functional datatype property hasName (i.e., identifier property) of the node $A_{\mathcal{Q}}$ which could be used to make the connection with the other parts of $\mathcal{Q}$ that are not covered by s_{11}. This containment mapping is inserted in the first line of Table II. Data service s_{21} covers the property makes $(A_{\mathcal{Q}}, C_{\mathcal{Q}})$ and partially the nodes $A_{\mathcal{Q}}$ and $C_{\mathcal{Q}}$. It covers from these two nodes the functional property hasName that could be used to make the connection with the uncovered parts of $\mathcal{Q}$. The same discussion applies to data services s_{22}, s_{23} and s_{24}, hence their insertion in the third, the fourth and the fifth row of Table II. Similarly, data services s_{31} through s_{34} cover partially node $C_{\mathcal{Q}}$.

Table II: Partial Containment Mapping Table

Data service	Partial containment mapping	Covered nodes & object properties
s_{11}("France", $?a$)	$A_{\mathcal{Q}} \rightarrow A_{s_{11}}$, France $\rightarrow x$, $a \rightarrow y$	$A_{\mathcal{Q}}(?a,$"France")
$s_{21}(\$a, ?b, ?c, ?d)$	$A_{\mathcal{Q}} \rightarrow A_{s_{21}}$, $C_{\mathcal{Q}} \rightarrow C_{s_{21}}$, $a \rightarrow x$, $b \rightarrow y$, $c \rightarrow z$, $d \rightarrow t$	$A_{\mathcal{Q}}(?a)$, makes$(A_{\mathcal{Q}}, C_{\mathcal{Q}})$, $C_{\mathcal{Q}}(?b, ?c, ?d)$
$s_{22}(\$a, ?b, ?c, ?d)$	$A_{\mathcal{Q}} \rightarrow A_{s_{22}}$, $C_{\mathcal{Q}} \rightarrow C_{s_{22}}$, $a \rightarrow x$, $b \rightarrow y$, $c \rightarrow z$, $d \rightarrow t$	$A_{\mathcal{Q}}(?a)$, makes$(A_{\mathcal{Q}}, C_{\mathcal{Q}})$, $C_{\mathcal{Q}}(?b, ?c, ?d)$
$s_{23}(\$a, ?b, ?c, ?d)$	$A_{\mathcal{Q}} \rightarrow A_{s_{23}}$, $C_{\mathcal{Q}} \rightarrow C_{s_{23}}$, $a \rightarrow x$, $b \rightarrow y$, $c \rightarrow z$, $d \rightarrow t$	$A_{\mathcal{Q}}(?a)$, makes$(A_{\mathcal{Q}}, C_{\mathcal{Q}})$, $C_{\mathcal{Q}}(?b, ?c, ?d)$
$s_{24}(\$a, ?b, ?c, ?d)$	$A_{\mathcal{Q}} \rightarrow A_{s_{24}}$, $C_{\mathcal{Q}} \rightarrow C_{s_{24}}$, $a \rightarrow x$, $b \rightarrow y$, $c \rightarrow z$, $d \rightarrow t$	$A_{\mathcal{Q}}(?a)$, makes$(A_{\mathcal{Q}}, C_{\mathcal{Q}})$, $C_{\mathcal{Q}}(?b, ?c, ?d)$
$s_{31}(\$b, ?e, ?f)$	$C_{\mathcal{Q}} \rightarrow C_{s_{31}}$, $a \rightarrow x$, $e \rightarrow y$, $f \rightarrow z$	$C_{\mathcal{Q}}(?b, ?e, ?f)$
$s_{32}(\$b, ?e, ?f)$	$C_{\mathcal{Q}} \rightarrow C_{s_{32}}$, $a \rightarrow x$, $e \rightarrow y$, $f \rightarrow z$	$C_{\mathcal{Q}}(?b, ?e, ?f)$
$s_{33}(\$b, ?e, ?f)$	$C_{\mathcal{Q}} \rightarrow C_{s_{33}}$, $a \rightarrow x$, $e \rightarrow y$, $f \rightarrow z$	$C_{\mathcal{Q}}(?b, ?e, ?f)$
$s_{34}(\$b, ?e, ?f)$	$C_{\mathcal{Q}} \rightarrow C_{s_{34}}$, $a \rightarrow x$, $e \rightarrow y$, $f \rightarrow z$	$C_{\mathcal{Q}}(?b, ?e, ?f)$

The second phase is to generate service classes. The algorithm groups the services according to the parts of $\mathcal{Q}$ they cover; i.e., data services covering the same part are put in the same group or class (for a definition and detailed discussion about the coverage notion, we refer the reader to our previous work [Barhamgi et al. 2010]). In our example we have three service classes. Then, the algorithm considers only the classes whose union of covered nodes and object properties cover the entire query (covers its different nodes and object properties). In our example classes $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ cover the query entirely; see Table III. These classes will then be used to generate the compositions in the subsequent sections. Note that in the general case we may have different class combinations covering $\mathcal{Q}$, in which case we will need to generate the compositions for each class combination.

Table III: Generated Service Classes

Service class	Data service	Covered nodes & object properties
$\mathcal{S}_1$	s_{11}(“France”, $?a$)	$A_{\mathcal{Q}}(?a$,“France”)
$\mathcal{S}_2$	$s_{21}(\$a, ?b, ?c, ?d)$ $s_{21}(\$a, ?b, ?c, ?d)$ $s_{21}(\$a, ?b, ?c, ?d)$ $s_{21}(\$a, ?b, ?c, ?d)$	$A_{\mathcal{Q}}(?a)$, makes$(A_{\mathcal{Q}}, C_{\mathcal{Q}})$, $C_{\mathcal{Q}}(?b, ?c, ?d)$
$\mathcal{S}_3$	$s_{31}(\$b, ?e, ?f)$ $s_{32}(\$b, ?e, ?f)$ $s_{33}(\$b, ?e, ?f)$ $s_{34}(\$b, ?e, ?f)$	$C_{\mathcal{Q}}(?b, ?e, ?f)$

3.5. Computing the Matching Degrees

In the previous section, we computed the different service classes $\mathbb{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ that could be combined together to answer the query. In the following, we compute the matching degrees of data services in the different classes.

Candidate data services in each service class $\mathcal{S}_i$ cover the same part of $\mathcal{Q}$, referred to as q_i. The constraints on each service s_{ij} in $\mathcal{S}_i$ may match completely or partially the preference constraints involved in q_i. Therefore, to differentiate the most relevant data services, we need to compute the matching degrees between the preference constraints involved in q_i and the data services' constraints.

To determine the matching degree of a service s_{ij}, traditional approaches assign a matching degree to each constraint corresponding to a preference in q_i. Then, this degree can be computed as an aggregation of individual matching degrees (i.e., the matching degree of each constraint). One direction is to assign weights to individual matching degrees [Dong et al. 2004]. However, users may not know how to set trade-off between different relevancies using numbers and an imprecise specification of weights could miss their desired services. They thus lose the flexibility to select their desired services. Computing the skyline from services [Alrifai et al. 2010; Yu and Bouguettaya 2010b; Yu and Bouguettaya 2010a] comes as a natural solution to overcome this limitation. Skyline computation has received significant consideration in database research [Börzsönyi et al. 2001; Tan et al. 2001; Kossmann et al. 2002; Papadias et al. 2003; Chomicki et al. 2003; Godfrey et al. 2005]. For a d-dimensional dataset, the skyline consists of the set of points which are not dominated by any other point. A point u dominates another point v if and only if u is at least as good as v in all dimensions and (strictly) better than v in at least one dimension.

However, as shown in [Skoutas et al. 2009; Skoutas et al. 2010b] considering a single matching method for evaluating services is a very coarse metric. For this purpose, we investigate multiple methods from the fuzzy set theory [Dubois and Prade 2000] to compute the matching degrees between user preferences and data services' constraints, namely, constraints inclusion methods that measure the to what extent the items returned by a given data service satisfy the user preferences.

Let $C \equiv x$ is $\mathcal{F}$ and $C' \equiv x$ is $\mathcal{G}$ be two fuzzy constraints, i.e., a constraint where authorized (or possible) values are restricted by means of a fuzzy set. Two classes of constraint inclusion methods are considered.

— ***Quantitative Method (QM)***. The inclusion degree between C and C' is computed in the following way: $Deg(C \subseteq C') = \frac{|\mathcal{F} \cap \mathcal{G}|}{|\mathcal{F}|} = \frac{\sum_{x \in X} \top(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))}{\sum_{x \in X} \mu_{\mathcal{F}}(x)}$ where the intersection is interpreted by a t-norm operator $\top$. In the following, the methods that rely on $\top$ =*“min”* and $\top$ = *“product”* are denoted by M-QM and P-QM, respectively.

— *Logic Method (LM)*. Now, the degree of inclusion is given by the following expression: $Deg(C \subseteq C') = min_{x \in X}(\mu_{\mathcal{F}}(x) \longrightarrow_f \mu_{\mathcal{G}}(x))$ where $\longrightarrow_f$ stands for a fuzzy implication.

In our example, we make use of two fuzzy implications: Gödel ($a \rightarrow_G b = 1$ if $a \leq b$, 0 otherwise) and Lukasiewicz ($a \rightarrow_L b = 1$ if $a \leq b$, $1 - a + b$ otherwise) implications. The methods based on these two implications are denoted by G-LM and L-LM, respectively.

Each relevant data service is then associated with a set of matching degrees. For instance, Table IV shows the matching degrees between each service s_{ij} in Table I and its corresponding component q_i (of the query $\mathcal{Q}_1$). Service s_{11} covering component q_1 does not have a matching degree because there are no user preferences involved in q_1. However, each data service covering component q_2 is associated with four (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 data services covering component q_3, the first and second values represent the matching degrees of the constraints `power` and `consumption`, respectively.

Table IV: Matching Degrees between Services' Constraints and Preference Constraints of $\mathcal{Q}_1$

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

3.6. Problem Statement

Assume we are given a preference query $\mathcal{Q}$:-$<q_1, ..., q_n>$. Each part (query component) q_i is a tuple $(\overline{q}_i, \mathcal{P}_{q_i})$, where $\overline{q}_i$ represents q_i without its preferences $\mathcal{P}_{q_i}$. Given a set of services classes $\mathbb{S} = \{\mathcal{S}_1, \cdots, \mathcal{S}_n\}$ where a class $\mathcal{S}_i$ regroups data services that are relevant to a query part q_i and a set $\mathbb{M} = \{M_1, \cdots, M_m\}$ of matching methods to compute the matching degrees between the constraints on relevant services and the user's preference. The problem to address is how to rank data services in each class $\mathcal{S}_i$ to select the most relevant ones and how to rank generated data service compositions to select the top-k ones that can answer the preference query $\mathcal{Q}$.

4. FUZZY DOMINANCE AND FUZZY SCORES

In this section, we introduce the notion of fuzzy dominance relationship considered between data services. To further motivate why the fuzzy dominance is needed, we first investigate the difference between fuzzy dominance and Pareto dominance. We then define the scores associated with both the data services and the data service compositions based-on the fuzzy dominance relationship.

It is well known that under a single matching degree method (mono criteria), 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 [Pei et al. 2007], namely probabilistic skyline overcomes this problem. Contrariwise, Skoutas et al. show in [Skoutas et al. 2009; Skoutas et al. 2010b] the limitations of the probabilistic skyline to rank services and

introduce the Pareto dominating score of individual services. There is, however, still some problems when applying the Pareto dominance as shown bellow.

4.1. Fuzzy Dominance vs Pareto Dominance

We start by defining formally the Pareto dominance, then discuss the reasons that motivate to make it fuzzy.

Definition 4.1. (Pareto dominance)
Given two d-dimensional points u and v, we say that u dominates[1] v, denoted by $u \succ v$, iff u is at least as good as v in all dimensions and (strictly) better than v in at least one dimension, i.e., $\forall i \in [1,d], u_i \geq v_i \wedge \exists j \in [1,d], u_j > v_j$. □

One can see that Pareto dominance does not allow discrimination between points with a large variance, i.e., points that are very good in some dimensions and very bad in other ones (e.g., $(1,0)$ and $(0.80,0)$ in Table IV) and good points, i.e., points that are (moderately) good in all dimensions (e.g., $(0.89,1)$ and $(0.77,1)$ in Table IV). To further illustrate this situation, let $u = (u_1,u_2) = (1,0)$ and $v = (v_1,v_2) = (0.90,1)$ be two matching degrees (or two points in general). 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 Pareto dominance relationship to express the extent to which a matching degrees vector (more or less) dominates another one [Benouaret et al. 2011d; Benouaret et al. 2011a]. We define below a fuzzy dominance relationship that relies on particular monotone comparison function expressing a graded inequality of the type "strongly greater than", as the higher the value, the better is the matching degree.

Definition 4.2. (fuzzy dominance)
Given two d-dimensional points u and v, we define the fuzzy dominance to express the extent to which u dominates v as:

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

Where $\mu_{\gg}$ is a membership function of the fuzzy relation $\gg$ that expresses the extent to which u_i is more or less (strongly) greater than v_i. The membership function $\mu_{\gg}$ can be defined in an absolute way (i.e., in terms of $x - y$) as follows:

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

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. □

Figure 5 gives the graphical representation of $\mu_{\gg}$ in terms of $x - y$ where H is a fuzzy parameter associated with the relation $\gg$ such that $\mu_{\gg}(x,y) = \mu_H(x - y)$. One can easily check that the trapezoidal membership function of H is $(\lambda + \varepsilon, \infty, \lambda, 0)$.

One can explain the semantics of $\mu_{\gg}$ in the following way:

— 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 than y.
— if $x - y$ is between ε and $\lambda + \varepsilon$, then x is much greater than y to some extent.

[1] Without loss of generality, we assume here the greater the value u_i, the better is.

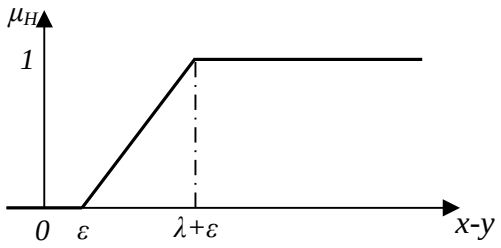

Fig. 5: Graded inequality representation in terms of $x - y$

It is worth to note that λ and ε values are subjective parameters, and then user-defined and domain-specific. They express the semantics of the (gradual) relation $\mu_{\gg}$ in a given domain for a given user.

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 $|u \succ v| = |v \succ u| = 0$ (i.e., u and v are incomparable) provided by Pareto dominance, where $|u \succ v| = 1$ if $u \succ v$, 0 otherwise. In the following sections, we will use the defined fuzzy dominance to compute scores of data services and their compositions.

4.2. Associating Fuzzy Score with a Data Service

We generalize the (Pareto) dominating score defined in [Skoutas et al. 2009; Skoutas et al. 2010b] to fuzzy dominance and propose the fuzzy dominating score (DS_f) of a data service. The DS_f of a data service s_{ij} indicates the average extent to which s_{ij} dominates the whole data services of its class $\mathcal{S}_i$.

Definition 4.3. (Fuzzy dominating score of a service)
The fuzzy dominating score (DS_f) of a data service s_{ij} in its class $\mathcal{S}_i$ is defined as:

$$DS_f(s_{ij}) = \frac{1}{(|\mathcal{S}_i| - 1)m^2} \sum_{\imath=1}^{m} \sum_{\substack{s_{ik} \in \mathcal{S}_i \\ k \neq j}} \sum_{\jmath=1}^{m} deg(s_{ij}^{\imath} \succ s_{ik}^{\jmath}) \tag{3}$$

where $s_{ij}^{\imath}$ is the matching degree of the data service s_{ij} obtained by applying the $\imath^{th}$ matching method and m stands for the number of matching methods applied. The term $(|\mathcal{S}_j| - 1)$ is used to normalize the fuzzy dominating score and make it in the range $[0, 1]$. □

Table V shows the fuzzy dominating scores of the data services of our running example (see Section 1).

4.3. Associating Fuzzy Score with a Composition

Different data service compositions can be generated from service classes $\mathcal{S}_i$ to answer a user query. To rank such generated compositions, we extend the previous defined score, i.e., the fuzzy dominating score (DS_f) to data service composition and associate each composition with a DS_f. The fuzzy dominating score of a composition $\mathcal{CS}$ is an aggregation of different DS_f scores of its component data services. It indicates the average number of possible compositions that $\mathcal{CS}$ more or less dominates.

Definition 4.4. (Fuzzy dominating score of a composition)
Let $\mathcal{CS} = \{s_{1j_1}, ..., s_{nj_n}\}$ be a composition of n services and $d = d_1 + ... + d_n$ be the

number of preference constraints in $\mathcal{Q}$, where d_i is the number of constraints (resp. preferences) involved in the service s_{ij_i} (resp. in the query component q_i). The DS_f of $\mathcal{CS}$ is then computed as follows:

$$DS_f(\mathcal{CS}) = \frac{1}{d}\sum_{i=1}^{n} d_i \cdot DS_f(s_{ij_i}) \quad (4)$$

It is important to note that not all compositions are valid. A composition $\mathcal{CS}$ of data services is valid if (i) it covers the user query $\mathcal{Q}$; (ii) it contains one and only one data service from each service class $\mathcal{S}_i$ and (iii) it is executable. A composition is said to be executable if all input parameters necessary for the invocation of its component data services are bound or can be made bound by the invocation of primitive data services whose input parameters are bound. For example, the composition $\{s_{11}(\$x, ?y)$,$s_{21}(\$x, ?y, ?z, ?t)$,$s_{31}(\$x, ?y, ?z)\}$) is executable since the inputs parameters of its component data services are all bound (the value of the variable *x* is supplied by the user). More details are provided in [Barhamgi et al. 2010].

Table V: Services' scores and top-k services

Data service	Service 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 DATA SERVICE COMPOSITIONS

In this section we show how we compute efficiently the top-k compositions. Note that in the general case, users may be interested in computing the top-k compositions ($k >$ 1) for many reasons:

— The selected composition may become un-executable; for instance, if one of its component services becomes unavailable or temporarily inaccessible. Computing top-k compositions provides a user with alternative compositions that he/she can exploit even if they are less good than the top-1 one.
— The top-k compositions are computed based on user's preferences about the accessed data sets. However, a user may have also other preferences that are not explicitly represented in the user's constraints. For instance, a user may prefer the services provided by a particular provider (e.g., eBay), and thus would be interested in selecting the best composition including a service of his preferred provider even if it is less better than the top-1 composition. Computing the top-k compositions provides users with the best compositions along with the necessary information (i.e., scores and component services) that helps them choose the one that suits them the best.
— As services could be offered by independent service providers managing different data sets (with different, sometimes overlapping, constraints), a user may need to select more than one composition to entirely satisfy his/her preferences. For instance, assume a user looking for cars with prices in [5000, 12000] and two cars retrieving services s_1 and s_2 with two constraints on price attribute [5000, 10000] and [9000, 12000], respectively. The top-1 composition will include the top-1 service s_1 as its

constraint overlaps with the user's preference more than that of s_2, which is the top-2 service and forms the top-2 composition. However, the user will need to select both of these two compositions to satisfy his/her constraint, and thus compute the top-2 compositions.

5.1. Efficient Generation of Top-k Compositions

The problem of top-k data service compositions entails computing the scores of each data service composition and returning the top-k highest ranked ones.

A straightforward method to find the top-k data service compositions that answer a query is to generate all possible compositions, compute their scores, and return the top-k ones. Clearly, 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 data service compositions. This technique allows for eliminating data services from their classes before generating the compositions, i.e., data services that we are sure that if they are composed with others, the obtained compositions are not in the top-k. The basic idea is to compute the score of each data service in its class, then only the best ones in each class are retained. The retained data services are then composed, and the scores of obtained compositions are computed, the top-k ones can be then returned to users. To this end, we introduce the following Lemma and Theorem.

LEMMA 5.1. *Let $\mathcal{CS} = \{s_{1j_1}, ..., s_{nj_n}, s\}$ and $\mathcal{CS}' = \{s_{1j_1}, ..., s_{nj_n}, s'\}$ be two similar data service compositions that only differ in the data services s and s'. Then, the following statement holds: $DS_f(s) > DS_f(s') \Longrightarrow DS_f(\mathcal{CS}) > DS_f(\mathcal{CS}')$.*

PROOF. Denoting by d' the number of constraints contained in s and s', we have: $DS_f(\mathcal{CS}) = \frac{1}{d}\sum_{i=1}^{n} d_i \cdot DS_f(s_{ij_i}) + \frac{d'}{d} \cdot DS_f(s)$ and $DS_f(\mathcal{CS}') = \frac{1}{d}\sum_{i=1}^{n} d_j \cdot DS_f(s_{ij_i}) + \frac{d'}{d} \cdot DS_f(s')$. Then, $DS_f(\mathcal{CS}) - DS_f(\mathcal{CS}') = \frac{d'}{d}(DS_f(s) - DS_f(s'))$. Since $\frac{d'}{d} > 0$ and $score(s) - score(s') > 0$, we have $DS_f(\mathcal{CS}) > DS_f(\mathcal{CS}')$. □

Lemma 5.1 indicates that the best data services in their classes will generate the best compositions.

THEOREM 5.2. *Let $\mathcal{CS} = \{s_{1j_1}, ..., s_{nj_n}\}$ be a composition of n data services. Let top-$k.\mathcal{S}_i$ and top-$k.\mathbb{CS}$ be the top-k data services of the service class $\mathcal{S}_i$ and the top-k data service compositions, respectively. Then, $\exists s_{ij_i} \in \mathcal{CS}; s_{ij_i} \notin$ top-$k.\mathcal{S}_i \Longrightarrow \mathcal{CS} \notin$ top-$k.\mathbb{CS}$.*

PROOF. Assume that $\exists s_{ij_i} \in \mathcal{CS}; s_{ij_i} \notin$ top-$k.\mathcal{S}_i$ but $\mathcal{CS} \in$ top-$k.\mathbb{CS}$. This means that $\exists s'_{ij_1}, ..., s'_{ij_k} \in \mathcal{S}_i$ such as $DS_f(s'_{ij_\ell}) > DS_f(s_{ij_i})$. Now, by replacing s_{ij_i} in $\mathcal{CS}$ with the services $s'_{ij_1}, ..., s'_{ij_k}$, we obtain k compositions $\mathcal{CS}_1, ..., \mathcal{CS}_k$ such as $DS_f(\mathcal{CS}_i) > DS_f(\mathcal{CS})$ according to Lemma 5.1. This contradicts our hypothesis. Hence, $\mathcal{CS} \notin$ top-$k.\mathbb{CS}$. □

Theorem 5.2 means that the top-k sets of the different service classes are sufficient to compute the top-k data service compositions that answer the considered query.

The fourth column of Table V shows the top-k ($k = 2$) data services in each service class according the fuzzy dominating scores. Thus, relevant data services that are not in the top-k of their classes are eliminated. They are crossed out in Table V. The other data services are retained. The top-k data service compositions are generated from different top-$k.\mathcal{S}_i$ classes. Table VI shows the possible compositions along with their fuzzy dominating scores, as well as the top-k compositions (i.e., $\mathcal{CS}_2$, $\mathcal{CS}_4$) of our running example.

Table VI: Compositions' scores and top-k ones

Composition	Composition score	Top-k
$\mathcal{CS}_1 = \{s_{11}, s_{22}, s_{32}\}$	0.615	
$\mathcal{CS}_2 = \{s_{11}, s_{22}, s_{34}\}$	0.687	$\mathcal{CS}_2$
$\mathcal{CS}_3 = \{s_{11}, s_{24}, s_{32}\}$	0.553	$\mathcal{CS}_4$
$\mathcal{CS}_4 = \{s_{11}, s_{24}, s_{34}\}$	0.625	

5.2. Top-k Service Compositions Algorithm

The algorithm, hereafter referred to as TKSC, computes the top-k data service compositions according to the fuzzy scores (see Algorithm 1). The algorithm proceeds as the following steps.

ALGORITHM 1: TKSC

Input: $\mathcal{Q}$ preference query; $\mathbb{S} = \{\mathcal{S}_1, \cdots, \mathcal{S}_n\}$ set of service classes; $\mathbb{M} = \{\mathcal{M}_1, \cdots, \mathcal{M}_m\}$ set of matching methods; $k \in \mathbb{N}$; $\varepsilon \geq 0$; $\lambda > 0$;
Output: the top k compositions

```
1  begin
2      foreach S_i in 𝕊 do
3          if P_{q_i} ≠ ∅ then
4              foreach s_ij in S_i do
5                  foreach M_ℓ in 𝕄 do
6                      ComputeMatchingDegree(C_ij, P_{q_i}, M_ℓ);
7                  end
8              end
9          end
10     end
11     foreach S_i in 𝒮 do
12         if P_{q_i} = ∅ then
13             top-k.S_i ← random(S_i, k);
14         else
15             foreach s_ij in S_i do
16                 ComputeServiceScore(s_ij);
17             end
18             top-k.S_i ← top(k, S_i);
19         end
20     end
21     ℂ𝕊 ← ComposeServices(top-k.S_1, ··· , top-k.S_n);
22     foreach CS in ℂ𝕊 do
23         ComputeCompositionScore(CS);
24     end
25     return top(k, ℂ𝕊);
26 end
```

Step 1: *compute the matching degrees (lines 2-10):*
After applying our query rewriting algorithm, relevant data service are found and service classes are generated. For each service class $\mathcal{S}_i$ touching the query's user preferences, i.e., there is one or more preference constraint involved in the query part covered by the data services of $\mathcal{S}_i$, we compute its different matching degrees, between the constraints of the data services C_{ij} and the user preferences P_{q_i}, according to the number of methods.

Step 2: *eliminate less relevant data services (lines 11-20):*
For each relevant service class $\mathcal{S}_i$ whose data 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 data services touch the user preferences, we first compute the score of its data services, we then retain only the top-k ones.

Step 3: *return top-k compositions (lines 21-25):*
We first compose the retained data services, i.e., the top-k in each relevant service class, then, we compute the scores of generated data service compositions. Finally, we provide the user with the top-k ones.

Taking the top-k data services of each service class (Theorem 5.2, TKSC generates in the worst case k^n data service compositions. However, the baseline algorithm needs to generate all possible data services composition, i.e., $\prod_{i=1}^{n} |\mathcal{S}_i|$. Thus, our algorithm reduces significantly the number of generated data service compositions.

5.3. Diversity-aware Top-k Compositions

Different similar data services could exist in each class $\mathcal{S}_i$ leading to similar data services compositions. A little variety in the top-k data services compositions list will probably lead to the user frustration. For this reason, it is crucial to provide users with the data service compositions that are still relevant to their preferences but less similar to each other, i.e., as diverse as possible. Diversification is then needed to improve user satisfaction. Diversification allows for finding compositions that cover many aspects of users information needs. Consider, for instance, a user who wants to buy a car and submits the query $\mathcal{Q}_1$ given in Section 1. A diverse result, i.e., a result that contains various prices and warranties with different horsepower and other technical characteristics, is intuitively more informative than a result that contains a homogeneous result containing only cars with similar features.

The diversity problem has attracted a lot of attention in the context of recommender systems, information retrieval and case-based reasoning systems. Some research works highlight that the diversity can be considered as important as similarity to the target query [McSherry 2002; Ziegler et al. 2005]. Two main definitions of a set diversity are introduced: (i) average dissimilarity of all pairs of elements and (ii) average rarity of the elements in the set. Different similarity/dissimilarity and rarity measures were defined and used in different heuristic algorithms for computing the diversified set that maximizes the diversity without loss of similarity (see for instance [Drosou and Pitoura 2010]).

In the context of our top-k data service compositions approach, we challenge and tackle the lack of top-k data service compositions variety by proposing a method for maximizing the diversity of data service compositions while maintaining an acceptable satisfaction level (expressed in terms of fuzzy scores) of data service compositions. We propose to diversify the top-k data service compositions by firstly diversifying the top-k data services of each class $\mathcal{S}_i$, and then by diversifying the data service compositions themselves. The diversity of the top-k data services of a class $\mathcal{S}_i$ means that the data services should be dissimilar each other.

A principled way to improving diversity of the top-k data services of a class $\mathcal{S}_i$, while at the same time maintaining satisfaction of data services, is to explicitly use both diversity and satisfaction of data services during the top-k data services selection. To this end, we make use of the following quality metric that combines diversity and satisfaction:

$$Quality(s_{ij}) = DS_f(s_{ij}) \times RelDiv(s_{ij}, dtopk.\mathcal{S}_i) \tag{5}$$

The quality of a data service s_{ij} in its class $\mathcal{S}_i$ is proportional to its satisfaction, and to its relative diversity to those diversified top-k data services so far selected $dtopk\mathcal{S}_i$. Initially, $dtopk\mathcal{S}_i$ is an empty set, and its first element will be necessary one of the data services s_{ij} with higher DS_f. The relative diversity of a data service s_{ij} to the current set $dtopk.\mathcal{S}_i$ is defined as the average dissimilarity between s_{ij} and the so far selected data services [McSherry 2002] as described in the following equation:

$$RelDiv(s_{ij}, dtopk.\mathcal{S}_i) = \begin{cases} 1 & dtopk.\mathcal{S}_i = \emptyset \\ \frac{\sum_{S_{i\ell} \in dtopk.\mathcal{S}_i} Dist(s_{ij}, s_{i\ell})}{|dtopk.\mathcal{S}_i|} & otherwise \end{cases} \quad (6)$$

The relative diversity of a service s_{ij} to an initial empty set, i.e., $|dtopk.\mathcal{S}_i| = 0$, is set to 1. The quantity $Dist(s_{ij}, s_{i\ell})$ represents the distance (i.e., dissimilarity) measure between the two services s_{ij} and $s_{j\ell}$. Recall that data services of the same class have the same functionality and only differ in their constraints, therefore the data services dissimilarity can be reduced to the dissimilarity of their constraints to quantify the extent to which two data services have similar constraints on their variables (i.e., they provide the same information about the same variable).

Given two data services $s_{ij}, s_{i\ell} \in \mathcal{S}_i$ having the constraints $\mathcal{C}_{ij} = \{x_1 \text{ is } \mathcal{F}_1, ..., x_{d_i} \text{ is } \mathcal{F}_{d_i}\}$ and $\mathcal{C}_{i\ell} = \{x_1 \text{ is } \mathcal{G}_1, ..., x_{d_i} \text{ is } \mathcal{G}_{d_i}\}$, respectively, . The distance between s_{ij} and $s_{i\ell}$ can be measured by $Dist(s_{ij}, s_{i\ell}) = max_{\imath \in \{1,...,d_i\}} Dist(\mathcal{F}_\imath, \mathcal{G}_\imath)$, where $Dist(\mathcal{F}_\imath, \mathcal{G}_\imath) = max_{x \in X_\imath} |\mu_{\mathcal{F}_\imath}(x) - \mu_{\mathcal{G}_\imath}(x)|$ is the distance between the fuzzy sets $\mathcal{F}_\imath$ and $\mathcal{G}_\imath$ [Dubois and Prade 2000]. Of course, the distance between two fuzzy sets can be measured by others distance metrics. We provide the effects of the distance metric in Section 6.

5.3.1. Diversified Top-k data services computing strategy in a given class. The above quality measure guides the construction of the diversified top-k data services of each relevant service class $\mathcal{S}_i$. This construction is achieved in an incremental way as described in Algorithm 2 refereed to as DTKS. During each step, the remaining data services of a class $\mathcal{S}_i$ are rank-ordered according to their quality and the data service with the highest quality is added to $dtopk.\mathcal{S}_i$. The first data service of the diversified top-k of a service class $\mathcal{S}_i$ to be selected is always the one with the highest DS_f. The initial service class $\mathcal{S}_i$ can be bounded to a smaller size equivalent to $k \cdot \eta$ ($\eta > 1$) to decrease the search space especially when $\mathcal{S}_i$ is too large. Also, to give the user the flexibility to make a tradeoff between the score of data services and their diversity, i.e., the quality. Specifically, when η increases the diversity of the top-k data services increases but their scores decrease and vice versa. It is worth to note that for the service classes whose services do not meet the user preferences, we just select randomly one data service, as they are all strictly similar.

5.3.2. Diversified Top-k data service compositions computing. The top-k data service compositions set is made more diverse (by applying a diversification on its component compositions) while maintaining acceptable compositions scores. The quality of a data service composition $\mathcal{CS}$ is an aggregation of qualities of its component services. Let $\mathcal{CS} = \{s_{1j_1}, ..., s_{nj_n}\}$ be a composition of n data services and $d = d_1 + ... + d_n$ be the number of user preference involved in the query, where d_i is the number of constraints involved in the service s_{ij_i}. The quality of the composition $\mathcal{CS}$ is then computed using a weighted average as follows:

$$Quality(\mathcal{CS}) = \frac{1}{d} \sum_{i=1}^{n} d_i \cdot Quality(s_{ij_i}) \quad (7)$$

ALGORITHM 2: DTKS

Input: $k \in \mathbb{N}$; $\eta \in \mathbb{N}$; $\mathcal{S}_i$ service class;
Output: $dtopk.\mathcal{S}_i$ diversified top-k data services of the class $\mathcal{S}_i$;

```
1  begin
2      S'_j ← top(k · η,S_i);
3      dtopk.S_i ← ∅;
4      for i=1 to k do
5          ComputeQuality(S'_i);
6          dtopk.S_i ← dtopk.S_i ∪ {MaxQuality(S'_i)};
7          S'_i ← S'_i − {MaxQuality(S'_i)};
8      end
9      return dtopkS_i;
10 end
```

The diversified top-k data service compositions algorithm referred as DTKSC is obtained from TKSC (the top-k data service compositions algorithm) by applying the following modifications:

— **line 13:** for relevant service classes whose data services do not meet user preference, we select randomly one data service instead of k data services as motioned above. So line 13 writes: $top\text{-}k.\mathcal{S}_i \leftarrow random(\mathcal{S}_i,1)$.
— **line 18:** instead of taking the top-k data 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, so line 18 writes: $top\text{-}k.\mathcal{S}_i \leftarrow$ DTKS$(k, \eta, \mathcal{S}_i)$.
— **line 23:** we compute the quality of the data service compositions instead of their scores. This line writes: ComputeCompositionQuality($\mathcal{CS}$).
— **line 25:** instead of returning the top-k data service 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 line 25 writes: **return** Diversifiedtop$(k, \mathbb{CS})$;

5.4. Impact of User Constraints on Composition

In the previous section we showed how user constraints are exploited to select the best compositions. In this section, we show the impact of constraints on the execution of a selected composition.

The number of results returned by executing a given composition may be very large which may mean missing the ones that are most relevant to a user's needs. User constraints can be exploited to resolve this problem by ranking the data returned by component services and the composition based on their relevance to a user's constraints. For this purpose, we propose a grade-aware composition algebra to orchestrate the data services in a selected composition.

The proposed algebra relies on the mature fuzzy database foundations [Dubois and Prade 1996] to rank data, and allows for ranking the returned results based on how well they satisfy the user's constraints. We describe below our proposed ranking aware orchestration operators we use to orchestrate data services in a composition, we then explain them based on the composition CS_2 selected in the previous sections.

Grade-aware Composition Algebra: our defined orchestration operators assume that each manipulated tuple is associated with a grade computed as the aggregation of the different grades associated with its attributes that are involved in constraints. We define the following operators:

Table VII: Implemented norms and conorms

Name	$TNorm : \top(x, y)$
Zadeh	min(x,y)
Probabilistic	xy
Lukasiewicz	$max(x + y - 1, 0)$
Hamacher	$\frac{xy}{\gamma+(1-\gamma)(x+y-xy)}$
Weber	$\begin{cases} x & \text{if} \quad y = 1 \\ y & \text{if} \quad x = 1 \\ 0 & \text{else} \end{cases}$

Name	$Conorm : \perp (x, y)$
Zadeh	$max(x, y)$
Probabilistic	$min(x + y, 1)$
Lukasiewicz	$max(x + y - 1, 0)$
Weber	$\begin{cases} x & \text{if} \quad y = 0 \\ y & \text{if} \quad x = 0 \\ 1 & \text{else} \end{cases}$

- *Grade-ware Invocation* $Invoke^g(S, t^g_{in}, O^g)$: Let S be a service, t^g_{in} be the graded input tuple with which S is invoked, O^g be the graded output set, and $S.O$ be the output returned by S. The $Invoke^g$ operator relays the tuples from $S.O$ to O^g, and for each relayed tuple t_i this operator computes the grade $g(t_i)$ as follows. First, assume t_i is involved in n fuzzy constraints P_j(where 1$\leq j \leq$n), the operator computes $g_1(t_i)$= $\top(\mu_{P_1(t_i)}, \mu_{P_2(t_i)}, ..., \mu_{P_n(t_i)})$ where $\top$ is a t-norm operator (that generalizes the conjunction operation) and μ_{P_i} the membership function associated with P_i. We implemented the T-norms presented in Table-VII. Zadeh t-norm is the greatest t-norm, thus leading to an optimistic aggregation strategy. Lukasiewicz and Weber t-norms yield a pessimistic aggregation strategy. Second, this operator computes $g(t_i)$ as follows: $g(t_i)$=$\top(g(t_{in}), g_1(t_i))$.
- *Graded Join*: $\infty^g(I^g_1, I^g_2)$, where I^g_1 and I^g_2 are two graded data sets. The grade of an output tuple is given by:
$g(\infty^g(t, t'))$ = $\top(g(t), g(t'))$ where $\top$ is a t-norm, and t and t' are joined tuples from I^g_1 and I^g_2 respectively.
- *Graded Projection* $\prod^g_A$. The projection is an operation that selects specified attributes A=$\{a_1, a_2, ...\}$ from a results set. The grade of an output tuple t is: $g(t)$ =$\perp$ $(g(t'_1), .., g(t'_i), .., g(t'_n))$ where $t = \prod_A(t'_i)_{i=1:n}$ and $\perp$ is the co-norm corresponding to the t-norm $\top$ used in the graded join.
- *Graded Union* $\cup^g$. The grade of an output tuple t is:
$g(t)$= $\perp$ $(g(t'_1), .., g(t'_i), .., g(t'_n))$, where $t'_i = t$ and $i = 1 : n$

Example: We explain the previous operators based on our running example. The composition CS_2 involves s_{11}, s_{22} and s_{34}. They can be orchestrated as follows (Figure 6). s_{11} is invoked with the desired country (e.g., France) to return the automakers (denoted by variable a). Then, service s_{22} can be invoked with automakers to return their different cars along with their prices and warranties. Service s_{34} can be invoked then with the car name to return its characteristics such as power and fuel consumption. All of these operators compute the tuples' rankings according to our defined equations.

Figure 7 shows the results (along with their rankings) at the output of each of these operators, and the final results at the composition's output. Note that the outputs of operator $Invoke^g(s_{11})$ have all the grade "1" as they are not involved in any constraint. The grades of car "c1" at the output of $Invoke^g(s_{22})$ are computed as follows: based on the membership functions associated with the price and warranty constraints, the grade of *price* attribute (i.e., c) is 0.1 and the grade of *warranty* attribute (i.e., d) is 0.25. Hence, $g_{1_{Zadeh}}(c1) = min(0.1, 0.25) = 0.1$, $g_{1_{Probabilistic}}(c1) = 0.1 * 0.25 = 0.025$, and $g_{1_{Lukasiewicz}}(c1) = max(0.1 + 0.25 - 1, 0) = 0$; and the grades are $T_{Zadeh}(c1) = min(1, 0.1) = 0.1$, $T_{Probabilistic}(c1) = 1 * 0.025 = 0.025$, and $T_{Lukasiewicz}(c1) = max(1 + 0 - 1, 0) = 0$. The grades of the other tuples are computed similarly. The final results are ordered based on their grades.

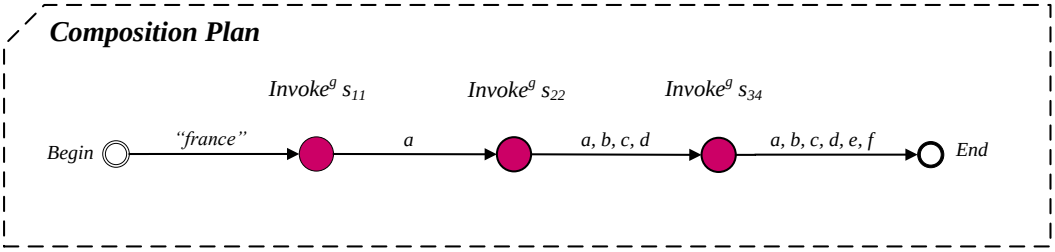

Fig. 6: The Composition Plan

***Invoke*^{g} s_{11}**

Country	a	Grade		
		T_Z	T_P	T_L
France	Peugeot Inc.	1	1	1
France	Reno Inc.	1	1	1

***Invoke*^{g} s_{22}**

a	b	c	d	Grade		
				T_Z	T_P	T_L
Peugeot Inc.	c1	19000	3	0.1	0.025	0
Peugeot Inc.	c2	12000	9	0.75	0.6	0.55
Reno Inc.	c7	8000	18	1	1	1
Reno Inc.	c9	16000	6	0.4	0.2	0

***Invoke*^{g} s_{34}**

a	b	c	d	e	f	Grade		
						T_Z	T_P	T_L
Peugeot Inc.	c1	19000	3	40	7	0.1	0.0015	0
Peugeot Inc.	c2	12000	9	60	6	0.75	0.3375	0.05
Reno Inc.	c7	8000	18	80	5	1	1	1
Reno Inc.	c9	16000	6	50	6.5	0.4	0.05	0

Final Results (ordered)

a	b	c	d	e	f	Grade		
						T_Z	T_P	T_L
Reno Inc.	c7	8000	18	80	5	1	1	1
Peugeot Inc.	c2	12000	9	60	6	0.75	0.3375	0.05
Reno Inc.	c9	16000	6	50	6.5	0.4	0.05	0
Peugeot Inc.	c1	19000	3	40	7	0.1	0.0015	0

Fig. 7: The intermediate and final results along with their grades

6. SYSTEM ARCHITECTURE AND EXPERIMENTAL EVALUATION

Our different technique and algorithms have been evaluated and tested in the context of a French project called PAIRSE [Benslimane et al. 2013; Barhamgi et al. 2013], whose objective was to propose Web service composition as a viable solution to answer data integration needs on the fly. In the rest of this section, we first describe the architecture of our system. Then, we focus on the efficiency aspect of our algorithms, as their performance impacts directly the query resolution time.

6.1. System Architecture

In this section we outline the basic components of our implemented system [Benouaret et al. 2011c], describe their roles and how they interact with each other. A high-level overview of our system is presented in Figure 8. The system consists of the following major components:

The *Fuzzy Membership Functions Manager* is used to manage fuzzy linguistic terms. It enables users and service providers to define their desired fuzzy terms along with the associated fuzzy 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. Examples of fuzzy terms along with their implementing services can be found on http://vm.liris.cnrs.fr:36880/FuzzyTerms. Users and service providers can directly test the proposed membership functions on that link 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 to which a given value is in the fuzzy set of the considered fuzzy

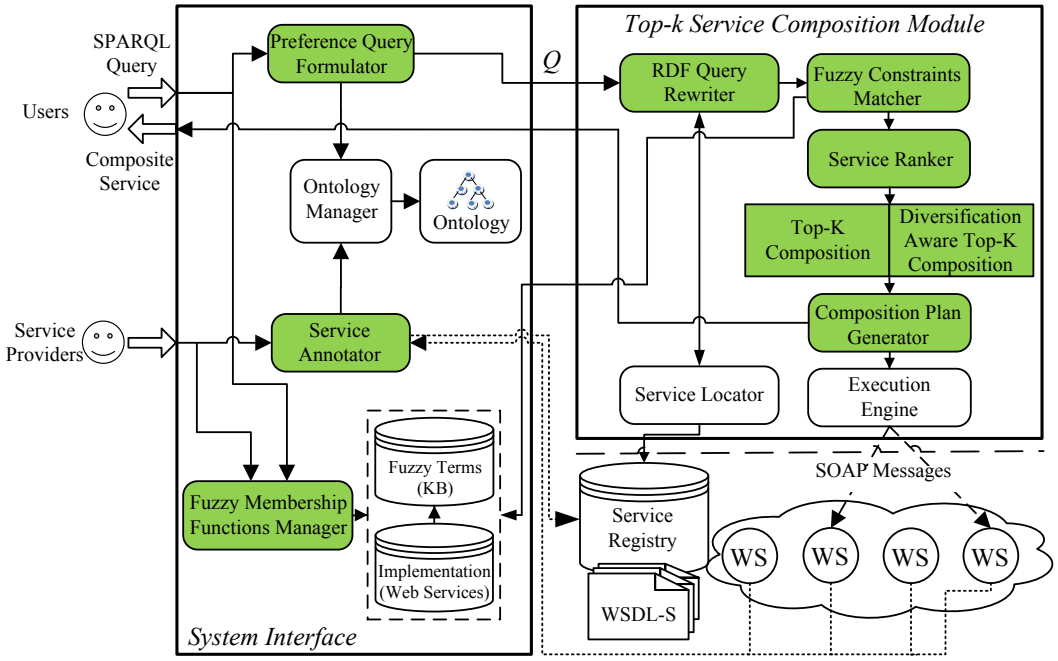

Fig. 8: Data Service Composition Architecture

term, and a WSDL description of the Web service that implements the membership function.

The *Service Annotator* allows service providers to (*i*) define the functionalities of their data services in the form of RDF parameterized views (RPVs) and specify the defined views with the desired fuzzy terms to represent the services' constraints and, (*ii*) annotate the services description files (e.g., WSDL files) with the defined views. This annotation is implemented by adding a new XML element called "rdfQuery" to the "Operation" elements in the XML Schema of WSDL as in the WSDL-S approach. The annotated WSDL files are then published to 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 have knowledge about SPARQL (or any specific ontology query languages) to express their queries, they are assisted interactively in formulating their queries and specifying the desired fuzzy terms.

The *Top-k Service Composition Module* consists of five components. The *RDF Query Rewriter* implements an efficient RDF query rewriting algorithm from [Barhamgi et al. 2010] to identify the relevant data services that match (some parts of) a user query. For that purpose, it exploits the annotations that were added to the service descrition files (e.g., WSDls). The *Service Locator* feeds the *Query Rewriter* with data services that most likely match a given query. The *Top-K Composition* component computes (*i*) the matching degrees of relevant data services, (*ii*) the fuzzy dominating scores of relevant data services, (*iii*) the top-k data 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 metrics to compute the diversified top-k data service compositions. The (diversified) top-k data service com-

positions are then translated by the *composition plan generator* into execution plans expressed in the XML Process Definition Language (XPDL)[2]. They are executed by a workflow execution engine. In our implementation, we use the Sarasvati[3] execution engine from Google.

6.2. Experimental Evaluation

This section presents an extensive experimental evaluation of our approach, focusing on : (i) the efficiency of our algorithms in terms of execution time, (ii) the effects of the used distance measure on the retrieved diversified top-k data service compositions, (iii) the effects of ε and λ on the top-k data service compositions/diversified top-k data service compositions and the benefits in terms of diversity, resulting from the use of the diversity aspect and (iv) the effectiveness of the use of the fuzzy dominating score for ranking Data services.

6.2.1. Experiment setting. Due to the limited availability of real data services, we implemented a Web service generator. The generator takes as input a set of (real-life) model data services (each representing a class of services) and their associated fuzzy constraints and produces for each model service a set of synthetic data services and their associated synthetic fuzzy constraints. The generated data services satisfy some fuzzy constraints on the attributes of the implemented model service. The generation of the synthetic data services is controlled by the following parameters: (i) the number of candidate data services per service class, (ii) the number of service classes, (iii) the number of max preferences in a service class, (iv) the number of matching methods and (v) the values of the parameters k, ε and λ. The default values of these parameters are : 400, 4, 4, 4, 5, 0.02 and 0.2, respectively.

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

6.2.2. Performance vs number of services per class. We measured the average execution time required to solve the composition problem as the number of data services per class increases. We varied the number of data services per class from 200 to 1,000. The results of this experiment are presented in Figure 9 (plot-a). The results show that our framework can handle hundreds of services in a reasonable time. The results also show that computing the diversified top-k composition introduces an insignificant cost when the factor η is small (e.g., $\eta = \eta_1$); this cost increases as η increases (e.g., $\eta = \eta_2$) since the search space for the diversified services in each class becomes larger.

6.2.3. Performance vs number of classes. We measured the average execution time required to solve the composition problem as the number of service classes increases. We varied the classes number from 1 to 6. The results of this experiment in Figure 9 (plot-b) show that the execution time is proportional to the classes number.

6.2.4. Performance vs number of constraints per service. To evaluate the effects of the constraints number on the efficiency of our TKSC and DTKSC algorithms, we varied the number of fuzzy constraints from 2 to 10 for each service in all service classes. Figure 9 (plot-c) shows the time required to compute the top-k / diversified top-k data service compositions. The result shows that time required to compute the top-k compositions increases linearly with the number of service constraints. In the case of diversified top-k compositions, when the search space in each service class is small (i.e., when the factor η is small (e.g., $\eta = \eta_1$)) the elapsed time in computing the diversified top-k

[2]http://www.xpdl.org/
[3]http://code.google.com/p/sarasvati/

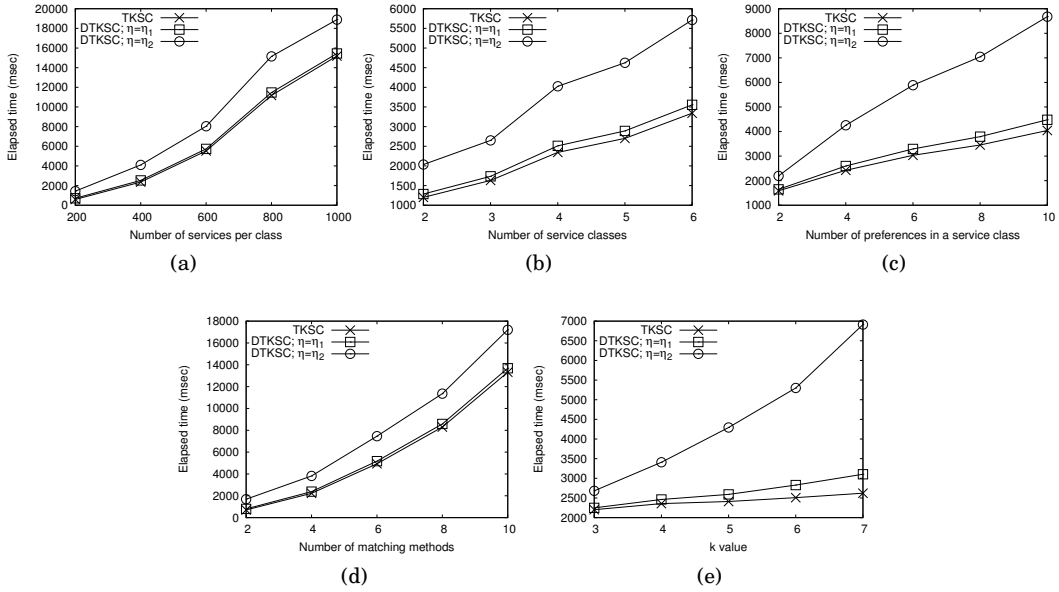

Fig. 9: Performance Results; case of $\eta_1 = \left\lfloor \sqrt{\frac{|\mathcal{S}_j|}{k}} \right\rfloor$ and $\eta_2 = \left\lfloor \frac{|\mathcal{S}_j|}{k} \right\rfloor$

compositions is almost as that required to compute the top-k compositions, the difference between the two is insignificant. However, when η is high, the time to compute the diversified top-k compositions increases, as more constraints should be processed to compute the diversity degrees.

6.2.5. 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 9 (plot-d). Once again the cost incurred in computing the diversified top-k compositions remains insignificant as the methods number increases if the factor η has a reasonable value (e.g., $\eta = \eta_1$).

6.2.6. Performance vs k. The results in Figure 9 (plot-e) show that as k increases, the cost incurred in computing the diversified top-k compositions increases slightly relative to the time needed to compute the top-k compositions.

6.2.7. The effects of the used distance measure. To compute the diversified top-k compositions we implemented all of the three distance measures:

$$M(\mathcal{F},\mathcal{G}) = \begin{cases} 0 & if \mathcal{F} = \mathcal{G} = \emptyset \\ 1 - \frac{\sum_{x \in X} min(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))}{\sum_{x \in X} max(\mu_{\mathcal{F}}(x), \mu_{\mathcal{G}}(x))} & otherwise \end{cases} \tag{8}$$

$$L(\mathcal{F},\mathcal{G}) = max_{x \in X} \left| \mu_{\mathcal{F}}(x) - \mu_{\mathcal{G}}(x) \right| \tag{9}$$

$$N(\mathcal{F},\mathcal{G}) = \begin{cases} 0 & if \mathcal{F} = \mathcal{G} = \emptyset \\ \frac{\sum_{x \in X} |\mu_{\mathcal{F}}(x) - \mu_{\mathcal{G}}(x)|}{\sum_{x \in X} (\mu_{\mathcal{F}}(x) + \mu_{\mathcal{G}}(x))} & otherwise \end{cases} \tag{10}$$

The membership functions used in computing the distance measures were discretized with a step of the order $(B + b - A + a)/1000$ (see Figure 1).

Table VIII: The effects of the used distance (dissimilarity) measure

Diversified Top-k Compositions				
Composite Services	Score	Quality		
		M	L	N
$\mathcal{CS}_1$: $\{s_{1356}, s_{2372}, s_{3285}, s_{4214}, s_{5183}\}$	0.6919484	0.6919484	0.6919484	0.6919484
$\mathcal{CS}_2$: $\{s_{1356}, s_{2372}, s_{3283}, s_{4214}, s_{5183}\}$	0.68804884	0.6744621	0.6615082	0.6780993
$\mathcal{CS}_3$: $\{s_{1356}, s_{2372}, s_{3360}, s_{4214}, s_{5183}\}$	0.69165516	0.6713853	0.6594182	0.6809209

Table IX: Effects of (ε, λ) for top-k compositions

(ε, λ)	Top-k Compositions		
	Component Services	Score	Diversity
$(0.002, 0.05)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.74703556	0.6121456
	$\{s_{1318}, s_{259}, s_{3154}, s_{4154}\}$	0.7441032	
	$\{s_{1318}, s_{2152}, s_{3154}, s_{4154}\}$	0.7441032	
$(0.02, 0.2)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.6563174	0.59373885
	$\{s_{1318}, s_{2132}, s_{3154}, s_{4154}\}$	0.655371	
	$\{s_{1318}, s_{259}, s_{3154}, s_{4154}\}$	0.65328693	
$(0.1, 0.3)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.53315574	0.62760955
	$\{s_{1318}, s_{2132}, s_{3154}, s_{4134}\}$	0.5312762	
	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.53008974	

Table X: Effects of (ε, λ) for diversified top-k compositions

(ε, λ)	Diversified Top-k Compositions			
	Component Services	Quality	Score	Diversity
$(0.002, 0.05)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.74703556	0.74703556	0.6995363
	$\{s_{1318}, s_{2292}, s_{3154}, s_{4134}\}$	0.6972428	0.7426259	
	$\{s_{1318}, s_{2134}, s_{3154}, s_{4154}\}$	0.6972428	0.7426259	
$(0.02, 0.2)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.6563174	0.6563174	0.6995363
	$\{s_{1318}, s_{2292}, s_{3154}, s_{4134}\}$	0.612067	0.6519956	
	$\{s_{1318}, s_{2134}, s_{3154}, s_{4154}\}$	0.6098658	0.6515922	
$(0.1, 0.3)$	$\{s_{1318}, s_{2292}, s_{3154}, s_{4154}\}$	0.53315574	0.53315574	0.71135545
	$\{s_{1318}, s_{2292}, s_{3154}, s_{4134}\}$	0.49845165	0.5312762	
	$\{s_{1318}, s_{2140}, s_{3154}, s_{4154}\}$	0.49460968	0.5256555	

Changing the used distance measure may change the quality of a composition, leading thus to its exclusion or inclusion to the diversified top-k compositions. Table VIII shows the diversified top-3 compositions of a given query along with their qualities when applying each of the previously seen distance measures. The composition $\mathcal{CS}_2$, for example, has a quality higher than that of $\mathcal{CS}_3$ if the distance measures M and L were applied; however its quality is lower than that of $\mathcal{CS}_3$ if the distance measure N was applied, thus leading to its exclusion if k was 2.

6.2.8. The effects of ε and λ. Changing the used values of the parameters ε and λ change the scores and the qualities for both the top-k/diversified top-k data service compositions. This may consequently lead to the inclusion or to the exclusion of a composition from top-k/diversified top-k data service compositions. Table IX and Table X show the top-k/diversified top-k data service 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 Equation 11 is the average of the diversities between each couple of compositions in the compositions set. Note that the global diversity of the diversified top-k compositions is always higher than that of the top-k compositions and the applicability of $DTKSC$ produce an average gain of $9,22\%$.

$$div(top-k) = \frac{\sum_{i=1}^{k}\sum_{j=i+1}^{k} Dist(\mathcal{CS}_i, \mathcal{CS}_j)}{(k^2-k)/2} \tag{11}$$

6.2.9. Effectiveness of the fuzzy dominating score. To evaluate the quality of results returned by applying our approach, we have focussed on one service class $\mathcal{S}_0$ containing a small set of 100 Data services. We have considered 3 matching methods M_1, M_2 and M_3 and 2 preferences involved in this class of services. For comparison, we also computed the top-k Data service in this service class by applying the Pareto dominating score proposed in [Skoutas et al. 2009; Skoutas et al. 2010b].

Table XI: Top-5 Data services using Pareto dominating score and fuzzy dominating score

Data service	Matching degrees			Rank	
	M_1	M_2	M_3	DS	DS_f
s_{04}	$(0.64433336, 0.7146259)$	$(0.5761205, 0.8961848)$	$(0.7063923, 0.8739688)$	3	2
s_{09}	$(0.80101556, 0.649472)$	$(0.6462966, 0.83784556)$	$(0.71121365, 0.9996651)$	2	1
s_{22}	$(0.045464505, 0.4498961)$	$(0.7529509, 0.9747615)$	$(0.88940185, 0.88275194)$	4	–
s_{057}	$(0.8508446, 0.944788)$	$(0.38846737, 0.16781723)$	$(0.95769846, 0.9885982)$	1	5
s_{072}	$(0.8809465, 0.96614444)$	$(0.38846737, 0.16781723)$	$(0.9934364, 0.3117907)$	5	3
s_{093}	$(0.8508446, 0.944788)$	$(0.896329, 0.8598233)$	$(0.71121365, 0.9996651)$	–	4

Table XI lists the top-5 Data services using Pareto dominating score (DS) and fuzzy dominating score (DS_f). Table XI shows that almost all the top-5 w.r.t DS are also in the top-5 w.r.t DS_f except for S_{22} witch is replaced by S_{093}. This is because S_{22} is very bad according to M_1, in particular for the first constraint. In addition, Table XI shows that the rank of the Data services S_{04}, S_{09}, S_{057} and S_{072} is different in the two top-5 sets. S_{057}, the best Data service w.r.t. DS is ranked last (i.e., fifth) w.r.t. DS_f. on the other side, the Data services S_{04}, S_{09} and S_{072} are in the top-3 (w.r.t. DS_f). This is because S_{057} is very bad according to M_2, in particular, for the second constraint. However, S_{04}, S_{09} and S_{072} are good or moderately good according to all matching methods. This is consistent with our motivation to fuzzify the Pareto dominance relationship illustrated in Section 4.1.

7. RELATED WORK

Preferences in Web service selection/composition have received much attention. Taking user preferences into account allows for for ranking candidate services/compositions and return only the ones that best satisfy the user's requirements. *ServiceTrust* [He et al. 2009] calculates reputations of services from users. It introduces transactional trust to detect QoS abuse, where malicious services gain reputation from small

transactions and cheat at large ones. However, *ServiceTrust* models transactions as binary events (success or failure) and combines reports from users without taking their preferences into account. In [Wang et al. 2008], 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 over a user's incomplete and constrained preference. In [Palmonari et al. 2009], 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 by using some automatic reasoning techniques that exploit the annotations. Unfortunately, all these works do not deal with the problem of composition.

Agarwal and Lamparter [Agarwal and Lamparter 2005] propose an automated Web service selection approach for composition. Web service combinations can be compared and ranked according to user preferences. Preferences are modeled as a set of fuzzy IF-THEN rules. The IF part contains fuzzy descriptions of the various properties of a service (i.e., a concrete Web service composition) and the THEN part is one of the fuzzy characterizations of a special concept called *Rank*. A fuzzy rule describes the combination of attribute values that a user is willing to accept and to which degree, where attribute values and degrees of acceptance are defined in a fuzzy way. ServiceRank [Wu et al. 2009] considers the QoS aspects as well as the social perspectives of services. Services that have good QoS and are frequently invoked by others are more trusted by the community and will be assigned high ranks. In [Wang et al. 2009], 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.

Recent approaches focused on computing the skyline from Web services. All these approaches focus on selecting Web services based on QoS parameters. The work in [Alrifai et al. 2010] focuses on the selection of skyline services for QoS based Web service composition. A method for determining which QoS levels of a service should be improved so that it is not dominated by other services is also discussed. In [Yu and Bouguettaya 2010b], the authors propose a skyline computation approach for service selection. The resulting skyline, called multi-service skyline, enables services users to optimally and efficiently access sets of service as an integrated service package. In the robust work [Yu and Bouguettaya 2010a], authors address the problem of uncertainty inherent to QoS and compute the skylines from service providers. A service skyline can be regarded as a set of service providers that are not dominated by others in terms of QoS aspects 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 [Pei et al. 2007]. However, these works do not take user preferences into account and except for [Alrifai et al. 2010] the problem of composition is not addressed. In [Benouaret et al. 2011a] we addressed the problem of selecting the best atomic services based on their QoS criteria and user preferences. In that work, each atomic each atomic service is associated with a QoS vector, and users specify their preferences in their service queries. The proposed approach extends existing skyline algorithms by proposing a new concept called α-dominant service skyline. This new concept is based on a fuzzy dominance relationship, and allows to better control the skyline size (in order to make it smaller or bigger), and to include atomic services that provide good compromise between the elements of the QoS vector. Even though that approach uses

the fuzzy dominance, the addressed problem is still different as in this paper we focus on compositions rather than on services, and consider constraints on the accessed data, rather than QoS constraints.

The work most related to ours are [Skoutas et al. 2009; Skoutas et al. 2010b; Benouaret et al. 2011b]. In [Skoutas et al. 2009; Skoutas et al. 2010b], 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 only considers selection of single services, without dealing with the problem of composition nor the user preferences. Furthermore, their proposed algorithms (e.g. TKDD [Skoutas et al. 2010b]) uses the Pareto dominance relationship which is not adapted when the compared services have large variance, i.e., services that are very good in some dimensions and very bad in other ones. In [Benouaret et al. 2011b], we consider the problem of top-k Web service compositions. In this paper, we propose an unified approach for selecting the top-k Web service compositions and the diversified top-k Web service compositions.

One the other hand, Result diversification has recently attracted much attention as a means of increasing user satisfaction in recommender system and Web research [Drosou and Pitoura 2010]. In [Skoutas et al. 2010a], the authors propose a method to diversify Web service search results in order to deal with users on the Web 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 without considering their personal preferences. In addition, the problem of composition is not addressed. In our approach both the service composition with preferences and the result diversification are considered. In [McSherry 2002], Mc Sherry proposes an approach to WSs retrieval that incrementally selects a diverse set of cases from a larger set of similarity-ordered cases. The same principle is adapted in our work for the diversification of the top-k Web service compositions but with different measurements.

A large body of recent research works have addressed the problems of modeling and querying the *deep Web* (a.k.a. the *hidden Web*). The authors in [Sheng et al. 2012] proposed a set of algorithms to extract all tuples from a hidden database using the minimum number of queries. The work presented in [Madhavan et al. 2008] addressed the problem of automatically and efficiently selecting the inputs values that can be used to query and extract the contents of deep-Web databases through their online forms. In [Raghavan and Garcia-Molina 2001], the authors develop a generic operational model of a hidden Web crawler, and present a prototype crawler that can extract semantic information from search forms and response pages. Senellart et al in [Senellart et al. 2008a; Senellart et al. 2008b] present an interesting approach to automatically construct data Web services on top of deep-Web databases. The basic idea is to automatically wrap Web forms by data Web services and to exploit current machine-learning techniques and heuristics to automatically infer the semantics of the service's inputs and outputs.

While these previous research works provide interesting algorithms to query forms-accessed Web databases, we believe that they are not appropriate for querying Web data services for the following two reasons. First, as Web forms target human users, they usually contain textual semantic information about the queried deep-Web database, and the expected form's input and output values. Such information is exploited in most of the cited approaches to model and query the deep-Web databases. In contrast, data Web services target machines (i.e., they are usually exploited by software consumers), and therefore their interfaces are poor in semantics. Service signatures (in service description files) are not sufficient alone to infer what the service is

supposed to do; i.e., different services may have the same signature, but completely different semantics. Therefore, human intervention from the side of service providers is still required to provide semantic descriptions about their provided services. Second, the previous data extraction algorithms cannot be used to learning the service constraints, as most of the time these constraints refer to some implicit/contextual information that are not necessary stored in the provider's databases. As an example, the service $s(\$autoMaker, ?cars, ?price)$ could be a service that returns cars along with their prices with a warranty > 1. The warranty does not appear in the service signature nor in the returned data. Discovering such implicit information is not trivial and could constitute a real challenge which is beyond the scope of this paper (it will be the subject of our future works).

8. CONCLUSION

In this paper we proposed a novel approach to compute the top-k data-driven Web service compositions based on user (fuzzy) preferences. We introduced the concept of *fuzzy dominance relationship* to measure the extent to which a data 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 data service compositions to compute the top-k ones. In addition, we proposed a new quality metric to assess the diversity of a composition relative to a set of compositions and an algorithm to select the diversified top-k compositions based on that metric. We conducted a thorough experimental study on a large set of data-driven Web services and demonstrated the efficiency and the effectiveness of our techniques in different settings. We identify the following directions for future research. First, we intend to combine this work with QoS preferences. Second, we plan to rank the data returned by the data service compositions. We also intend to study the constraints learning problem.

REFERENCES

Sudhir Agarwal and Steffen Lamparter. 2005. User Preference based Automated Selection of Web Service Compositions. In *ICSOC Workshop on Dynamic Web Processes*, Kunal Verma; Amit Sheth; Michal Zaremba; Christoph Bussler (Ed.). IBM, Amsterdam, Netherlands, 1–12.

Mohammad Alrifai, Dimitrios Skoutas, and Thomas Risse. 2010. Selecting skyline services for QoS-based web service composition. In *WWW*. 11–20.

Mahmoud Barhamgi, Djamal Benslimane, Youssef Amghar, Nora Cuppens-Boulahia, and Frédéric Cuppens. 2013. PrivComp: a privacy-aware data service composition system. In *EDBT*. 757–760.

Mahmoud Barhamgi, Djamal Benslimane, and Brahim Medjahed. 2010. A Query Rewriting Approach for Web Service Composition. *IEEE T. Services Computing* 3, 3 (2010), 206–222.

Karim Benouaret, Djamal Benslimane, and Allel Hadjali. 2011a. On the Use of Fuzzy Dominance for Computing Service Skyline Based on QoS. In *ICWS*. 540–547.

Karim Benouaret, Djamal Benslimane, and Allel Hadjali. 2011b. Top-k service compositions: A fuzzy set-based approach. In *SAC*. 1033–1038.

Karim Benouaret, Djamal Benslimane, Allel Hadjali, and Mahmoud Barhamgi. 2011c. FuDoCS: A Web Service Composition System Based on Fuzzy Dominance for Preference Query Answering. In *VLDB*. 1430–1433.

Karim Benouaret, Djamal Benslimane, Allel Hadjali, and Mahmoud Barhamgi. 2011d. Top-k Web Service Compositions Using Fuzzy Dominance Relationship. 144–151.

Djamal Benslimane, Mahmoud Barhamgi, Frederic Cuppens, Franck Morvan, Bruno Defude, Ebrahim Nageba, Michael Mrissa, Francois Paulus, Stephane Morucci, Nora Cuppens, Chirine Ghedira, Riad Mokadem, Said Oulmakhzoune, and Jocelyne FAYN. 2013. PAIRSE: A Privacy-Preserving Service-Oriented Data Integration System. *SIGMOD Record* 41, 3 (2013), 5–14.

Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *ICDE*. 421–430.

Michael J. Carey, Nicola Onose, and Michalis Petropoulos. 2012. Data services. *Commun. ACM* 55, 6 (2012), 86–97.

Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with Presorting. In *ICDE*. 717–816.

Ian Davidson. 2009. Clustering with Constraints. In *Encyclopedia of Database Systems*. 393–396.

Alin Deutsch, Bertram Ludäscher, and Alan Nash. 2007. Rewriting queries using views with access patterns under integrity constraints. *Theor. Comput. Sci.* 371, 3 (2007), 200–226.

Alin Deutsch, Liying Sui, and Victor Vianu. 2004. Specification and Verification of Data-Driven Web Services. In *Proc. of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2004)*. Paris, France.

Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. 2004. Simlarity Search for Web Services. In *VLDB*. 372–383.

Marina Drosou and Evaggelia Pitoura. 2010. Search result diversification. *SIGMOD Record* 39, 1 (2010), 41–47.

Didier Dubois and Henri Prade. 1996. Using Fuzzy Sets in Database Systems: Why and How?. In *FQAS*. 89–103.

Didier Dubois and Henri Prade (Eds.). 2000. *Fundamentals of Fuzzy Sets* . Kluwer, Boston, Mass.

Schahram Dustdar, Reinhard Pichler, Vadim Savenkov, and Hong Linh Truong. 2012. Quality-aware service-oriented data integration: requirements, state of the art and open challenges. *SIGMOD Record* 41, 1 (2012), 11–19.

Parke Godfrey, Ryan Shipley, and Jarek Gryz. 2005. Maximal Vector Computation in Large Data Sets. In *VLDB*. 229–240.

Allel Hadjali, Souhila Kaci, and Henri Prade. 2008. Database Preferences Queries - A Possibilistic Logic Approach with Symbolic Priorities. In *FoIKS*. 291–310.

Qiang He, Jun Yan, Hai Jin, and Yun Yang. 2009. ServiceTrust: Supporting Reputation-Oriented Service Selection. In *ICSOC/ServiceWave*. 269–284.

Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB*. 275–286.

Chen Li and Edward Y. Chang. 2001. On Answering Queries in the Presence of Limited Access Patterns. In *ICDT*. 219–233.

Bertram Ludäscher and Alan Nash. 2004. Web Service Composition Through Declarative Queries: The Case of Conjunctive Queries with Union and Negation. In *ICDE*. 840.

Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Y. Halevy. 2008. Google's Deep Web crawl. *PVLDB* 1, 2 (2008), 1241–1252.

David Martin, Massimo Paolucci, Sheila Mcilraith, Mark Burstein, Drew Mcdermott, Deborah Mcguinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. 2004. Bringing Semantics to Web Services: The OWL-S Approach. Springer, 26–42.

David McSherry. 2002. Diversity-Conscious Retrieval. In *ECCBR*. 219–233.

Matteo Palmonari, Marco Comerio, and Flavio De Paoli. 2009. Effective and Flexible NFP-Based Ranking of Web Services. In *ICSOC/ServiceWave*. 546–560.

Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2003. An Optimal and Progressive Algorithm for Skyline Queries. In *SIGMOD Conference*. 467–478.

Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. 2007. Probabilistic Skylines on Uncertain Data. In *VLDB*. 15–26.

Sriram Raghavan and Hector Garcia-Molina. 2001. Crawling the Hidden Web. In *VLDB*. 129–138.

Pierre Senellart, Serge Abiteboul, and Rémi Gilleron. 2008a. Understanding the Hidden Web. *ERCIM News* 2008, 72 (2008).

Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. 2008b. Automatic wrapper induction from hidden-web sources with domain knowledge. In *WIDM*. 9–16.

Cheng Sheng, Nan Zhang, Yufei Tao, and Xin Jin. 2012. Optimal Algorithms for Crawling a Hidden Database in the Web. *PVLDB* 5, 11 (2012), 1112–1123.

Amit P. Sheth, Karthik Gomadam, and Ajith Ranabahu. 2008. Semantics enhanced Services: METEOR-S, SAWSDL and SA-REST. *IEEE Data Eng. Bull.* 31, 3 (2008), 8–12.

Dimitrios Skoutas, Mohammad Alrifai, and Wolfgang Nejdl. 2010a. Re-ranking web service search results under diverse user preferences. In *VLDB, Workshop on Personalized Access, Profile Management, and Context Awareness in Databases*. 898–909.

Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, Verena Kantere, and Timos K. Sellis. 2009. Top-dominant web services under multi-criteria matching. In *EDBT*. 898–909.

Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, and Timos K. Sellis. 2010b. Ranking and Clustering Web Services Using Multicriteria Dominance Relationships. *IEEE T. Services Computing* 3, 3 (2010), 163–177.

Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. 2001. Efficient Progressive Skyline Computation. In *VLDB*. 301–310.

Kiri Wagstaff and Claire Cardie. 2000. Clustering with Instance-Level Constraints. In *AAAI/IAAI*. 1097.

Kiri L. Wagstaff. 2010. Constrained Clustering. In *Encyclopedia of Machine Learning*. 220–221.

Hongbing Wang, Shizhi Shao, Xuan Zhou, Cheng Wan, and Athman Bouguettaya. 2009. Web Service Selection with Incomplete or Inconsistent User Preferences. In *ICSOC/ServiceWave*. 83–98.

Hongbing Wang, Junjie Xu, and Peicheng Li. 2008. Incomplete Preference-driven Web Service Selection. In *IEEE SCC (1)*. 75–82.

Qinyi Wu, Arun Iyengar, Revathi Subramanian, Isabelle Rouvellou, Ignacio Silva-Lepe, and Thomas A. Mikalsen. 2009. Combining Quality of Service and Social Information for Ranking Services. In *ICSOC/ServiceWave*. 561–575.

Qi Yu and Athman Bouguettaya. 2010a. Computing Service Skyline from Uncertain QoWS. *IEEE T. Services Computing* 3, 1 (2010), 16–29.

Qi Yu and Athman Bouguettaya. 2010b. Computing Service Skylines over Sets of Services. In *ICWS*. 481–488.

Lotfi A. Zadeh. 1965. Fuzzy Sets. *Information and Control* 8, 3 (1965), 338–353.

Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *WWW*. 22–32.

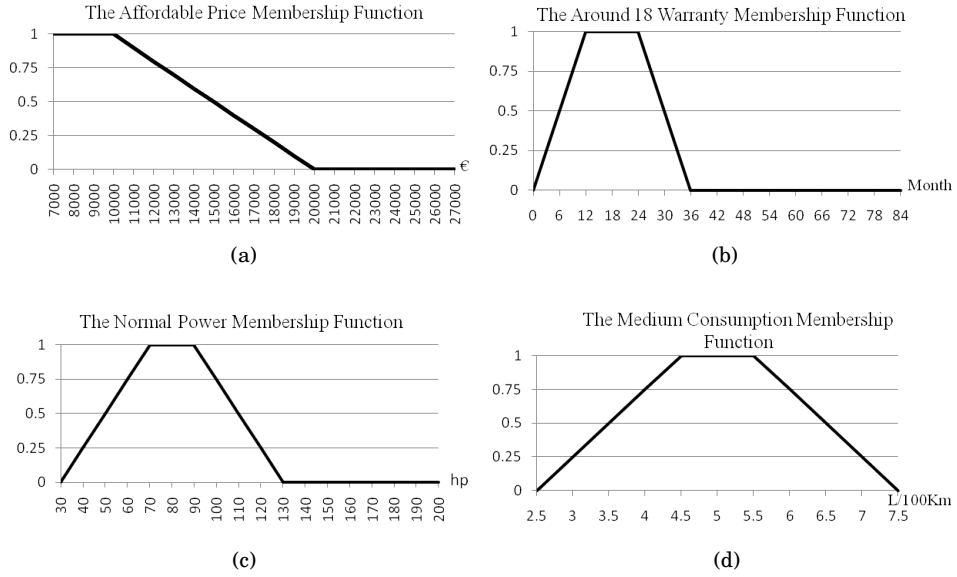

Fig. 10: Membership Functions

APPENDIX

A. MEMBERSHIP FUNCTIONS

Figure 10 shows the membership functions on the preferences (linguistic terms) involved in the query of our running example.

B. DATA SERVICES' ANNOTATIONS

We show in this appendix how we annotate the descriptions of data services with the RDF views.

The following listing gives an example of how the WSDL description file of s_{21} is annotated with its corresponding RDF view. The "rdfannot:rdfquery" element is used to hook the operation to its associated RDF view.

```
<interface name="s21">
 <operation name=CarByAutoMaker pattern=wsdl:in-out
 wssem:modelReference="RDFSCarOntology:AutoMaker"
 wssem:modelReference="RDFSCarOntology:Car">
 <! –RDF View is added as extensible element on an operation –>
 <rdfannot:rdfquery name="query1" value=
  URL=
  "http://soc.univ-lyon1.fr:8080/FunctionsDescription/index.jsp"
  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/Cheap', ?t is 'URL/Short'}"/>
   ...
```

```
  </operation>
</interface>
```

The listing hereafter gives an example of how an HTML page of a RESTful service implementing the functionality of s_{21} is annotated with its corresponding RDF view. We exploited the "*sarest:operation*" predicate of SA-REST proposal to hook the service to its associated RDF view.

```
<html xmlns:sarest="http://lsdis.cs.uga.edu/SAREST#">
  ...
   <meta about="http://carbyautomaker.com/search/">
    <meta property="sarest:input"
    content="http://liris.cnrs.fr/car.owl#AutoMaker"/>
    <meta property="sarest:output"
    content="http://liris.cnrs.fr/car.owl#Car"/>
    <meta property="sarest:action" content="HTTP GET"/>
    <meta property="sarest:lifting"
    content="http://carbyautomaker.com/api/lifting.xsl"/>
    <meta property="sarest:lowering" content=
    "http://carbyautomaker.com/api/lowering.xsl"/>
    <meta property="sarest:operation"
    content=URL=
    "http://soc.univ-lyon1.fr:8080/FunctionsDescription/index.jsp"
    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/Cheap', ?t is 'URL/Short'}"/>
</meta>
...
```

Received October 2011; revised January 2013; accepted —