An Extension of SPARQL with Fuzzy Navigational Capabilities for Querying Fuzzy RDF Data
Olivier Pivert, Olfa Slama, Virginie Thion

To cite this version:
Olivier Pivert, Olfa Slama, Virginie Thion. An Extension of SPARQL with Fuzzy Navigational Capabilities for Querying Fuzzy RDF Data. IEEE International Conference on Fuzzy Systems, Jul 2016, Vancouver, Canada. hal-01290932

HAL Id: hal-01290932
https://inria.hal.science/hal-01290932
Submitted on 8 Jul 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
An Extension of SPARQL with Fuzzy Navigational
Capabilities for Querying Fuzzy RDF Data

Olivier Pivert, Olfa Slama, Virginie Thion
Rennes 1 University / IRISA
Lannion, France
Email: {Olivier.Pivert,Olfa.Slama,Virginie.Thion}@irisa.fr

Abstract—The Resource Description Framework (RDF) is the
graph-based standard data model for representing semantic web
information, and SPARQL is the standard query language for
querying RDF data. Because of the huge volume of linked open
data published on the web, these standards have aroused a large
interest in the last years. This paper proposes a fuzzy extension
of the SPARQL language that improves its expressiveness and
usability. This extension allows (1) to query a fuzzy RDF data
model, and (2) to express fuzzy preferences on data and on the
structure of the data graph, which has not been proposed in any
previous fuzzy extensions of SPARQL.

I. INTRODUCTION

The Resource Description Framework (RDF) [25] is the
standard data model promoted by the W3C for representing
information about resources available on the Web. Nowadays,
within the semantic web realm, the way we query RDF data is
a crucial subject due to the huge quantity, the heterogenous,
the vagueness, and the wide connectivity of such data. SPARQL
[20], the official W3C recommendation as an RDF query
language, plays the same role for the RDF data model as SQL
does for relational data model. It provides basic functionalities
(such as, union and optional queries, value filtering and
ordering results, etc.) in order to query RDF data through
graph patterns, i.e., RDF graphs containing variables data.

But classical SPARQL lacks of some expressiveness and
usability capabilities as it follows a crisp (Boolean) querying
of RDF data for which the response is either false or true.
As a result, it lacks the ability to deal with flexibility aspects
(including queries with user preferences or vagueness), which
is significant in real-word applications.

From one perspective, the need to query about the structure
of data and then extract relationships between resources in
RDF graph, has motivated research into extending SPARQL
languages to be more expressive than before. In [11], [3], [1]
and [17], authors mainly extend SPARQL by allowing to query
crisp RDF through graph patterns using regular expressions
but they do not address the fuzziness in their approaches.
From another perspective, in order to make the expression of
flexible queries (involving user preferences) in RDF databases
possible, [7] and [13] propose a flexible extension of SPARQL
query language. The objective is to support user preferences
and provide users with useful results ranked according to
their preferences brought by the query. The expression of path
queries is not allowed in these extensions.

There is also a real need for a flexible SPARQL that takes
into account RDF graphs where data is described by intrinsic
weighted values, attached to edges or nodes. This weight may
denote any gradual notion like a cost, a truth value, an intensity
or a membership degree. For instance, in the real world, the
information stored on the Web, as well as its metadata are
far from being perfect and are represented by vague/imprecise
knowledge. An imprecise information may be of the form “An
album is recent with the degree of truth 0.9”.

Driven by these approaches, the RDF data model should be
enriched in order to represent such information, and new query
languages should be defined. Our aim in this paper is to extend
SPARQL in order to support flexible querying of crisp and also
fuzzy RDF graph databases (when data are imprecise).

The contribution of the paper is twofold. First, we extend
the concept of SPARQL graph pattern defined over a crisp
RDF data model, to the concept of fuzzy graph patterns that
allows: (1) to query a fuzzy RDF data model, and (2) to express
fuzzy preferences on data (through fuzzy conditions) and on the
structure of the data graph (through fuzzy regular expressions),
which has not been proposed in any previous fuzzy extensions
of SPARQL. Second, we propose FURQL, an extension of the
SPARQL language based the previous theoretical foundations.

The paper is organized as follows. Section II presents
background notions. In Section III, which is the core of the
contribution, we introduce the fuzzy graph pattern notion.
Based on this notion, we propose the FURQL language, for
which we then discuss implementation issues in Section IV.
Related work is presented in Section V. At last, Section VI
recalls the contributions and outlines some perspectives.

II. BACKGROUND NOTIONS

The Resource Description Framework (RDF) [25] uses pair-
wise disjoint infinite sets of resource names, literals and blank
nodes (i.e., unknown or anonymous resources) respectively
denoted by \( U \), \( L \) and \( B \) in the following.

Let us consider an album as a resource of the Web. A
characteristic may be attached to the album, like a title, an
artist, a date or tracks. In order to express such a characteristic,
the RDF data model uses a statement of the form of a triple
\( (s, p, o) \in (U \cup B) \times (U \cup L \cup B) \times (U \cup L \cup B) \). The subject \( s \) denotes
the resource being described, the predicate \( p \) denotes the property
of the resource and the object \( o \) denotes the property value. A
triple states that the subject \( s \) has a property \( p \) with a value \( o \).
For instance, the triple \( (\text{Beyonce, creator, B’Day}) \) states that \text{Beyonce has B’Day as a creator} property, which can be interpreted as \text{Beyonce is a creator of B’Day}.

A set of RDF triples can be modeled by a directed labeled graph (called \text{RDF graph} or \text{simply graph} in the following) where for each triple \( (s, p, o) \), the subject \( s \) and the object \( o \) are nodes, and the predicate \( p \) corresponds to an edge from the subject node to the object one. RDF is then a graph-structural data model that makes it possible to exploit the basic notions of graph theory (such as node, edge, path, neighborhood, connectivity, distance, in-degree, out-degree, etc.).

RDF provides a \text{schema definition} language called RDF Schema (RDFS), which allows specifying semantic deductive constraints on objects, subjects and relationships of an RDF data graph. It permits to declare objects and subjects as \text{instances} of given classes, and \text{inclusion statements} between classes and between relationships. It is also possible to relate the \text{domain} and \text{range} of a relationship to classes. RDF also declares entailment rules that allow to derive new triples from the explicit triples appearing in an RDF graph. Such implicit triples are part of the RDF graph even if they do not explicitly appear in it. They can be explicitly added to the graph. When all implicit triples are made explicit in the graph then the graph is said to be \text{saturated}. In the following, we only consider saturated graphs.

Unfortunately, the classical crisp RDF model is only capable of representing Boolean notions whereas real-world concepts are often of a vague or gradual nature. This is why several authors proposed fuzzy extensions of the RDF model. Throughout this paper, we consider the data model based on Definition 1 which synthesize the existing fuzzy RDF models of literature ([15], [23], [14], [12], [22], [24], [27]), whose common principle consists in adding a fuzzy degree to edges, modeled either by a value embedded in each triple or by a function associating a satisfaction degree with each triple, expressing the extent to which the fuzzy concept attached to the edge is satisfied. For instance, the fuzzy triple \("(\text{Beyonce, recommends, Euphoria}, 0.8)\) states that \text{Beyonce strongly recommends} \text{Euphoria}.

\text{Definition 1 (Fuzzy RDF (F-RDF) graph): A F-RDF graph is a tuple \((T, \zeta)\) such that (i) \(T\) is a finite set of triples of \((U \cup B) \times (U \cup L \cup B)\), (ii) \(\zeta\) is a membership function on triples \(\zeta : T \rightarrow [0, 1]\).}

According to the classical semantics associated with fuzzy graphs, \(\zeta(t)\) qualifies the intensity of the relationship involved in the statement \(t\). Intuitively, \(\zeta\) attaches fuzzy degrees to “edges” of the graph. Having a value of 0 for \(\zeta\) is equivalent to not belonging to the graph. Having a value of 1 for \(\zeta\) is equivalent to fully satisfy the associated concept. In the graph \(G_{MB}\) of Figure 1, such edges appear as classical ones, i.e. with no degree attached.

The fuzzy degrees associated with edges are given or calculated. In its simplest form, each degree may be based on a simple statistical notion, e.g. the intensity of friendship between two artists may be computed as the number of their common friends over the total number of friends with respect to each artist.

\text{Remark 1: A classical crisp RDF data graph is a special case of F-RDF data graph where the co-domains of \(\zeta\) are \(\{0, 1\}\). A fundamental implication is that the concepts and the flexible query language defined over a F-RDF graph in the following, remain relevant over a RDF graph.}

A F-RDF graph is said to be \text{ground} if it contains no blank nodes. Such a graph may be ground at the beginning or made
Example 1 (Fuzzy RDF graph): Figure 1 is an example of Fuzzy RDF graph inspired by MusicBrainz\footnote{\url{https://musicbrainz.org/}}. This graph, denoted $G_{MB}$ in the following, is the running example of the article. It mainly contains artists and albums as nodes. For readability reasons, each URI node contains the value of its name instead of the URI itself. Literal values may be attached to URI, like the age of an artist, the release date or the global rating of an album. The graph contains fuzzy relationships (e.g. friend, likes, recommends, memberOf) as well as crisp ones (e.g. creator, date,...). We limit our example to some entities including artists and albums and omit URI prefixes to avoid overcrowding the figure.

In order to create this graph, we started from a MusicBrainz nonfuzzy subgraph for which every relationship between nodes is Boolean and, then, we made it fuzzy by adding satisfaction degrees denoting intensity on some relationships. Here for instance, the degree associated with an edge of the form Art → memberOf → Group is the number of years the artist stayed in this group over the number of years this group has been existing. \hfill \textcircled{♦}

In the following, we rely on classical notions from fuzzy graph theory [21], which are the path, the distance and the strength (ST) of the connection between two nodes.

Let $G$ be a F-RDF data graph. Classically, a path $p$ in $G$ is denoted by a possibly empty sequence of triples $(t_1, \ldots, t_k, \ldots, t_n)$ such that $\{t_i \mid 1 \leq i \leq n\} \subseteq G$ and for all $1 \leq k \leq n-1$, the object of $t_k$ is the subject of $t_{k+1}$.

Given two nodes $x$ and $y$, Paths($x,y$) denotes the set of cycle-free paths\footnote{Considering paths containing cycle would not change the result of the following expressions (1) and (2).} in $G$ connecting $x$ to $y$ (the set of paths of the form $(t_1, \ldots, t_k, \ldots, t_n)$ such that $x$ is the subject of $t_1$ and $y$ is the object of $t_n$).

The distance between two nodes $x$ and $y$ is defined by

$$\text{distance}(x,y) = \min_{p \in \text{Paths}(x,y)} \text{length}(p) \quad (1)$$

where $\text{length}(p)$ is the length of a path $p$ in a fuzzy graph [21], calculated by $\text{length}(p) = \sum_{t \in p} \frac{1}{\zeta(t)}$.

The strength between two nodes $x$ and $y$ is defined by

$$\text{ST}(x,y) = \max_{p \in \text{Paths}(x,y)} \text{ST}_p(p) \quad (2)$$

where $\text{ST}_p(p)$ is the strength of the path connecting $x$ and $y$ in a fuzzy graph [21], calculated by $\text{ST}_p(p) = \min(\{\zeta(t) | t \in p\}$.

Example 2: Let us consider the cycle-free paths from $G_{MB}$ connecting Beyonce to Euphoria, depicted in Figure 2, and let us compute the distance and the strength between the pair of nodes (Beyonce, Euphoria). The distance between the pair of nodes (Beyonce, Euphoria) is calculated as $\text{distance}(\text{Beyonce}, \text{Euphoria}) = \min(\text{length}(p_1), \text{length}(p_2), \text{length}(p_3))$, with $\text{length}(p_1) = 0.8 + 1.25 = 1.25$, $\text{length}(p_2) = 1.06 + 1/0.2 + 1 = 7.7$ and $\text{length}(p_3) = 1/0.8 + 1/0.3 + 1/0.5 + 1 = 7.5$. Finally, the length of the shortest path between the pair of nodes (Beyonce, Euphoria) is $\text{distance}(\text{Beyonce}, \text{Euphoria}) = 1.25$.

SPARQL [20] is the standard query language promoted by the W3C for querying RDF Data. It is a declarative query language based on graph pattern matching, in the sense that the query processor searches for sets of triples in the data graph that satisfy a pattern (i.e., set of triples containing variables) expressed in the query. A SPARQL query has the general form given in Listing 1, where the clause select is for specifying which variables should be returned, the clause from defines the datasets to be queried, and the clause where contains the triple of the researched pattern.

```
#Result
SELECT ...  #Dataset definition
FROM ...  #Graph pattern
WHERE ...  #Modifiers
ORDER BY ..., DISTINCT ..., ...
```

Listing 1. Skeleton of a SPARQL query

Roughly speaking, a graph pattern is defined as being triples where variables can occur, composed by binary operators UNION, FILTER, OPTIONAL \ldots (concatenation). Listing 2 gives an example of SPARQL query that retrieves the albums of 2012 that are either created or liked by Shakira, with the associated rating if it is available.

```
#Modifiers
SELECT ?album ?r WHERE {
  { ?artist name "Shakira".
    ?artist creator ?album. }
  UNION
  { ?artist name "Shakira".
    ?artist likes ?album. }
  OPTIONAL { ?album rating ?r. }
} ?album date ?d.
FILTER (?d = "2012").
```

Listing 2. A SPARQL query

SPARQL also provides solution modifiers, which make it possible to modify the result set by applying classical operators like ORDER BY, DISTINCT, LIMIT, PROJECTION, or OFFSET.
In the following section, we introduce the notion of fuzzy graph patterns, which is a fuzzy extension of the graph pattern notion introduced in [16] and [4]. A fuzzy graph pattern allows to express fuzzy preferences on the entities of an F-RDF graph (through fuzzy conditions) and on the structure of the graph (through fuzzy regular expressions).

III. Fuzzy graph patterns

The introduced foundations rely on the SPARQL graph pattern syntax and semantics introduced in [16] and [4], which present SPARQL graph patterns in a more traditional algebraic formalism than the official syntax does [25]. It considers the following binary operators: AND (SPARQL \texttt{AND}) , UNION (SPARQL \texttt{UNION}) , OPT (SPARQL \texttt{OPTIONAL}) and FILTER (SPARQL \texttt{FILTER}). We redefine the associated syntax and semantics in order to introduce fuzzy conditions and fuzzy regular expressions expressed over the F-RDF data model of Definition 1.

Syntax of fuzzy graph patterns

For the following, we assume the existence of an infinite set \( V \) of variables such that \( V \cap (U \cup L) = \emptyset \). By convention, we prefix the elements of \( V \) by a question mark symbol. Before giving the formal definition of a fuzzy graph pattern, we introduce the notion of a fuzzy regular expression.

Definition 2 (Fuzzy regular expression): The set \( F \) of fuzzy regular expressions, defined over the set \( U \) of URIs, is recursively defined by:

- \( \epsilon \) is a fuzzy regular expression of \( F \) denoting the empty pattern;
- \( u \in U \) and \( \cdot \) are a fuzzy regular expressions of \( F \);
- if \( A \in F \) and \( B \in F \) then \( A|B, A, B, A^*, A_{\text{cond}} \) are fuzzy regular expressions of \( F \).

The character \( \cdot \) denotes any element of \( U \), \( A|B \) denotes alternative expressions, \( AB \) denotes the concatenation of expressions, \( A^* \) stands for the classical repetition of an expression (the Kleene closure), \( A_{\text{cond}} \) denotes paths satisfying the pattern \( A \) with a condition \( \text{cond} \) where \( \text{cond} \) is a Boolean combination of atomic formulas of the form: \( \text{sprop} \text{\texttt{IS}} \text{\texttt{FILTER}} \) where \( \text{sprop} \) is a structural property of the path defined by the expression and \( \text{FILTER} \) denotes a predefined or user-defined fuzzy term like short (see figure 3). In the following, we limit the path structural properties to \( \text{ST} \) and \( \text{distance} \) (see Section II). We denote by \( A^+ \) the classical shortcut for \( AA^* \).

Figure 3. Representation of the fuzzy term short

Definition 3 (fuzzy graph pattern): Fuzzy graph patterns are recursively defined by:

- A fuzzy triple from \((U \cup V) \times (U \cup F \cup V) \times (U \cup L \cup V)\) is a fuzzy graph pattern.
- If \( P_1 \) and \( P_2 \) are fuzzy graph patterns then \((P_1 \text{ AND } P_2), (P_1 \text{ UNION } P_2)\) and \((P_1 \text{ OPT } P_2)\) are fuzzy graph patterns.
- If \( P \) is a fuzzy graph pattern and \( C \) is a fuzzy condition then \((P \text{ FILTER } C)\) is a fuzzy graph pattern. A fuzzy condition is a logical combination of fuzzy terms defined by:
  - if \( \{x, y\} \subseteq V \) and \( c \in (U \cup L) \), then \( \text{bound}(\{x\}) \), \( \{x \theta c \text{ and } \{x \theta y \} \) are fuzzy conditions, where \( \theta \) is a fuzzy or crisp comparator,
  - if \( \{x\} \subseteq V \) and \( \text{FILTER} \) is a fuzzy term then, \( \{x\} \text{ FILTER} \) is a fuzzy condition,
  - if \( C_1 \) and \( C_2 \) are fuzzy conditions then \((\neg C_1)\) and \((C_1 \circ C_2)\) where \( \circ \) is a fuzzy connective are fuzzy conditions. Fuzzy connectives include of course fuzzy conjunction \( \wedge \) (resp. disjunction \( \lor \), usually interpreted by the triangular norm minimum (resp. maximum), but also many other operators that may be used for expressing different kinds of trade-offs, such as the weighted conjunction and disjunction [8], mean operators, fuzzy quantifiers [9], or the non-commutative connectives described in [6].

Given a pattern \( P \), \( \text{var}(P) \) denotes the set of variables occurring in \( P \).

Semantics of fuzzy graph patterns

Intuitively, given an F-RDF data graph \( G \), the semantics of a fuzzy graph pattern \( P \) defines a set of mappings, where each mapping (from \( \text{var}(P) \) to URIs and literals of \( G \)) maps the pattern to an isomorphic subgraph of \( G \). For introducing such a concept, the notion of satisfaction of a fuzzy regular expression must first be defined.

Definition 4 (Fuzzy regular expression matching of a path): Let \( G = (T, \zeta) \) be an F-RDF graph and \( exp \) be a fuzzy regular expression. Let \( p = ((s_1, p_1, o_1), \ldots, (s_n, p_n, o_n)) \subseteq G \) be a path of \( G \). \( p \) satisfies \( exp \) with a satisfaction degree of \( \text{sat}(exp)(p) \) is defined as follows, according to the form of \( exp \) (in the following, \( f, f_1 \) and \( f_2 \) are fuzzy regular expressions):

- \( \text{exp} \) is of the form \( c \). If \( p \) is empty then \( \text{sat}(exp)(p) = 1 \) else \( \text{sat}(exp)(p) = 0 \).
- \( \text{exp} \) is of the form \( u \in U \) (resp. \( \cdot \) ) If \( p_1 \) is \( u \) (resp. any \( u \in U \)) then \( \text{sat}(exp)(p) = \zeta((s_1, p_1, o_1)) \) else 0.
- \( \text{exp} \) is of the form \( f_1 f_2 \). Let \( P \) be the set of all pairs of paths \((p_1, p_2)\) s.t. \( p \) is of the form \( p_1p_2 \). One has \( \text{sat}(exp)(p) = \max_p \min_p \text{sat}(f_1)(p_1), \text{sat}(f_2)(p_2)) \).
- \( \text{exp} \) is of the form \( f_1 \cup f_2 \). One has \( \text{sat}(exp)(p) = \max_p \text{sat}(f_1)(p) \text{sat}(f_2)(p) \).
- \( \text{exp} \) is of the form \( f^* \). If \( p \) is the empty path then \( \text{sat}(exp)(p) = 1 \). Otherwise, we denote by \( P \) the set of all tuples of paths \((p_1, \ldots, p_n)\) \((n > 0)\) s.t. \( p \) is of the form \( p_1 \cdots p_n \). One has \( \text{sat}(exp)(p) = \max_p \min_p \text{sat}(p_1), \ldots, \text{sat}(p_n)) \).
- \( \text{exp} \) is of the form \( f^{\text{cond}} \) where \( \text{cond} \) where \( \text{cond} \) is a fuzzy condition. \( \text{sat}(exp)(p) = \min_p \text{sat}(f)(p) \text{ sat}(\text{cond})(p) \) where \( \mu_{\text{cond}}(p) \) denotes the degree of satisfaction of \( \text{cond} \) by \( p \).
Again, not satisfying is equivalent to getting a degree of 0.

Definition 5 (Satisfaction of a fuzzy regular expression of a pair of nodes): Let \( G = (\mathcal{T}, \zeta) \) be an F-RDF graph and \( \exp \) be a fuzzy regular expression. Let \( (x, y) \) be a pair of nodes of \( G \). \((x, y)\) satisfies \( \exp \) with a satisfaction degree of \( \text{sat}_{\exp}(x, y) \) is defined by \( \max_{p \in \text{Paths}(x, y)} \text{sat}_{\exp}(p) \). Note that only cycle-free paths need to be considered in order to compute the satisfaction degree.

Example 3 (Satisfaction of a fuzzy regular expression): Let us consider the following fuzzy regular expressions, for which we give the semantics and their satisfaction according to \( G_{MB} \). Note that the paths represented in Figure 4 are some cycle-free paths among many others from the graph \( G_{MB} \).

Expression \( f_1 = \text{(friend+).creator} \) is a fuzzy regular expression. A pair of nodes \((x, y)\) satisfies \( f_1 \) if \( x \) has a “friend-linked” artist (an artist connected to \( x \) with a path made of \text{friend} edges), that created the album \( y \). All of the pairs of nodes (EnriqueI, Justified), (Shakira, Butterfly), (Beyonce, Euphoria), (MariahC, Euphoria) and (Shakira, Euphoria), illustrated in Figure 4, satisfy \( f_1 \) with the satisfaction degree \( \text{sat}_{f_1}(\text{EnriqueI, Justified}) = 0.4 \), \( \text{sat}_{f_1}(\text{Shakira, Butterfly}) = 0.7 \), \( \text{sat}_{f_1}(\text{Beyonce, Euphoria}) = 0.3 \), and \( \text{sat}_{f_1}(\text{Shakira, Euphoria}) = 0.5 \) respectively.

Expression \( f_2 = \text{(friend+)distance is short}.creator \) is a fuzzy regular expression. A pair of nodes \((x, z)\) satisfies \( f_2 \) if \( x \) has a “close” friend artist \( y \) that created an album \( z \), “close” meaning that \( x \) is connected to \( y \) by a short path made of friend edges (see Figure 3). According to Figure 4, the pairs of nodes (EnriqueI, Justified), (Shakira, Butterfly) and (Shakira, Euphoria) are the only ones that match \( f_2 \) as \( \mu_{\text{short}}(3.5) = 0.75 \) (where length of pair (EnriqueI, Justified) = 1/0.4 + 1 = 3.5), \( \mu_{\text{short}}(2.4) = 1 \) (where length of pair (Shakira, Butterfly) = 1/0.7 + 1 = 2.4), \( \mu_{\text{short}}(7.7) = 0 \) (where length of pair (Beyonce, Euphoria) = 1/0.6 + 1/0.2 + 1 = 7.7), \( \mu_{\text{short}}(6.33) = 0 \) (where length of pair (MariahC, Euphoria) = 1/0.3 + 1/0.5 + 1 = 6.33) and \( \mu_{\text{short}}(3) = 1 \) (where length of pair (Shakira, Euphoria) = 1/0.5 + 1 = 3) with satisfaction degree of \( \text{sat}_{f_2}(\text{EnriqueI, Justified}) = 0.4 \), \( \text{sat}_{f_2}(\text{Shakira, Butterfly}) = 0.7 \) and \( \text{sat}_{f_2}(\text{Shakira, Euphoria}) = 0.5 \) respectively.

Figure 4. Some paths from \( G_{MB} \).

Let us now come to the definition of a mapping. A mapping is a pair \( (m, d) \) where \( m : \mathcal{V} \rightarrow (\mathcal{U} \times \mathcal{L}) \) and \( d \in [0, 1] \). Intuitively, \( m \) maps the variables of a fuzzy graph pattern into a subgraph (“answer”) of the F-RDF data graph and \( d \) denotes the satisfaction degree associated with the mapping (the more satisfactory the subgraph, the higher the satisfaction degree). \( m(t) \), where \( t \) is a triple pattern, denotes the triple obtained by replacing each variable \( x \) of \( t \) by \( m(x) \). The domain of a mapping \( m \) denoted by \( \text{dom}(m) \) is the subset of \( \mathcal{V} \) for which \( m \) is defined. Two mappings \( m_1 \) and \( m_2 \) are compatible iff for all \( v \in \text{dom}(m_1) \cap \text{dom}(m_2), \) one has \( m_1(v) = m_2(v) \).

Intuitively, \( m_1 \) and \( m_2 \) are compatible if \( m_1 \) can be extended with \( m_2 \) to obtain a new mapping \( m_1 \oplus m_2 \) and vice versa.

Let \( M_1 \) and \( M_2 \) be two sets of mappings. We define the join, union, difference and left outer-join of \( M_1 \) with \( M_2 \) as:

\[
M_1 \times M_2 = \{(m_1 \oplus m_2, \min(d_1, d_2)) \mid (m_1, d_1) \in M_1 \text{ and } (m_2, d_2) \in M_2 \text{ and } m_1 \text{ and } m_2 \text{ are compatible}\}
\]

The operation \( M_1 \times M_2 \) denotes the set of new mappings that result from extending mappings in \( M_1 \) with their compatible mappings in \( M_2 \).

\[
M_1 \cup M_2 = \{(m, d) \mid (m, d) \in M_1 \text{ and } m \notin \text{support}(M_2)\} \cup \{(m, d) \mid (m, d) \in M_2 \text{ and } m \notin \text{support}(M_1)\} \cup \{(m, \max(d_1, d_2)) \mid (m, d_1) \in M_1 \text{ and } (m, d_2) \in M_2\}
\]

Here, \( \cup \) corresponds to the classical set-theoretic union.

\[
M_1 \setminus M_2 = \{m_1 \mid (m_1, d_1) \in M_1 \text{ and } \forall (m_2, d_2) \in M_2, m_1 \text{ and } m_2 \text{ are not compatible}\}
\]

\[
M_1 \bowtie M_2 = (M_1 \times M_2) \cup (M_1 \setminus M_2)
\]

A mapping \( m \) is in \( M_1 \bowtie M_2 \) if it is the extension of a mapping of \( M_1 \) with a compatible mapping of \( M_2 \), or if it is in \( M_1 \) and cannot be extended with any mapping of \( M_2 \).

Definition 6 (Mapping satisfying a fuzzy condition): Let \( m \) be a mapping and \( C \) be a fuzzy condition. Then \( m \) satisfies the fuzzy condition \( C \) with a satisfaction degree defined as follows, according to the form of \( C \):

- \( C \) is of the form bound(?x): if \( ?x \in \text{dom}(m) \) then \( m \) satisfies the condition \( C \) with a degree of 1, else 0.
- \( C \) is of the form \(?x \in c\): if \( ?x \in \text{dom}(m) \) then \( m \) satisfies the condition \( C \) with a degree of \( \mu_f(m(?x), c) \), else 0.
- \( C \) is of the form \(?x \in d\): if \( ?x \in \text{dom}(m) \) and \( ?y \in \text{dom}(m) \), then \( m \) satisfies the condition \( C \) with a degree of \( \mu_f(m(?x), m(?y)) \), else 0.
- \( C \) is of the form \(?x \in Fterm\): if \( ?x \in \text{dom}(m) \) then \( m \) satisfies the condition \( C \) to the degree \( \mu_{\text{Fterm}}(m(?x)) \) (which can be 0).
- \( C \) is of the form \(-C_1 \) or \( C_1 \circ C_2 \) where \( \circ \) is a fuzzy connective: we use the usual interpretation of the fuzzy operator involved (complementation to 1 for the negation, minimum for the conjunction, maximum for the disjunction, etc (see e.g. [9]).

Definition 7 (Evaluation (interpretation) of a fuzzy graph pattern): The evaluation of a fuzzy graph pattern \( P \) over an F-RDF graph, denoted by \( [P]_G \), is recursively defined by:

\[
[P]_G = \begin{cases} \{(m, 1) \mid \text{dom}(m) = v(t) \text{ and } m(t) \in G\} \end{cases}
\]
• if $P$ is of the form of a fuzzy triple graph pattern $t \in (\mathcal{U} \cup \mathcal{V}) \times \mathcal{F} \times (\mathcal{U} \cup \mathcal{L} \times \mathcal{V})$ denoted by $(t, \exp, ?y)$ (where variables occur as subject and object) then $[P]_G = \{(m, d) | \text{dom}(m) = \{t, ?y\} \text{ and } m(\exp) \text{ satisfies } \exp \text{ with a satisfaction degree } d(\exp, t, ?y)\}$. The case for which the subject (resp. the object) of $t$ is a constant of $\mathcal{U}$ (resp. $\mathcal{U} \cup \mathcal{L}$) is trivially induced from this definition.

- if $P$ is of the form $P_1$ AND $P_2$ then $[P]_G = [P_1]_G \times [P_2]_G$.
- if $P$ is of the form $P_1$ OPT $P_2$ then $[P]_G = [P_1]_G \triangleright [P_2]_G$.
- if $P$ is of the form $P_1$ UNION $P_2$ then $[P]_G = [P_1]_G \cup [P_2]_G$.
- if $P$ is of the form $P_1$ FILTER $C$ then $[P]_G = \{(m, d) | m \in [P]_G \text{ and } m \text{ satisfies } C \text{ to the degree of } d\}$.

Intuitively, expressions $(P_1$ AND $P_2)$, $(P_1$ UNION $P_2)$, $(P_1$ OPT $P_2)$, and $(P_1$ FILTER $C)$ refer to conjunction graph patterns, union graph patterns, optional graph patterns, and filter graph patterns respectively. Optional graph pattern allows a partial match for the query (i.e., the query tries to match a graph patterns and does not omit solution where some part of the optional patterns does not satisfy this query).

Note that a crisp graph pattern is a special case of fuzzy graph pattern where no fuzzy term or condition occurs.

Example 4 (Evaluation of a fuzzy graph pattern): Let us consider $P_{rec_{low}}$ the fuzzy graph pattern defined by $(\text{Art1}, \text{friend} \cdot \text{distance} \cdot \text{short} \cdot \text{creator}, \text{Alb})$ AND $(\text{Art1}, \text{recommends}, \text{Alb})$ AND $(\text{Art1}, \text{low})$ FILTER $(\text{r} \text{ rating})$, for which Figure 6 is a graphical representation.

Intuitively, $P_{rec_{low}}$ retrieves the list of artists $(\text{Art1})$ in $G_{MB}$ s.t. $(\text{Art1})$ recommends a low rated album $(\text{Alb})$ created by another artist who is a close friend of them $(\text{Art1})$.

Figure 7 gives the set of subgraphs of $G_{MB}$ satisfying the pattern $P_{rec_{low}}$. The matching value of Art1 is either Shakira or Enriquei who match the pattern $P_{rec_{low}}$ (i.e. they are the only artists that have liked a low rated album created by another artist among their close friends). Note that $(\text{friend} \cdot \text{distance} \cdot \text{short} \cdot \text{creator})$ is the fuzzy regular expression $f_2$ of Example 3 with $sat_{f_2}(\text{Enriquei}, \text{Justified}) = 0.4$, $sat_{f_2}(\text{Shakira}, \text{Butterfly}) = 0.7$ and $sat_{f_2}(\text{Shakira}, \text{Euphoria}) = 0.5$ and we consider $\mu_{low}(4) = 0.66$, $\mu_{low}(6) = 0.33$ and $\mu_{low\text{-rating}}(9) = 0$ defined in Figure 5.

Then, the evaluation of the pattern $P_{rec_{low}}$ over the RDF graph $G_{MB}$ includes two mappings with their respective satisfaction degrees and is represented as follows:

$$[P_{rec_{low}}]_{G_{MB}} = \{(\text{Art1} \rightarrow \text{Enriquei}, \text{Alb} \rightarrow \text{Justified}, \text{r} \rightarrow 6), (\text{Art1} \rightarrow \text{Shakira}, \text{Alb} \rightarrow \text{Butterfly}, \text{r} \rightarrow 4), 0.66\}.$$

Figure 6. Graphical representation of pattern $P_{rec_{low}}$

Figure 7. Subgraphs satisfying $P_{rec_{low}}$

IV. ABOUT THE FURQL QUERY LANGUAGE AND IMPLEMENTATION ISSUES

We now introduce FURQL (Fuzzy RDF Query Language), which consists in extending SPARQL graph patterns by fuzzy graph patterns. We can only outline the syntax of the language here, due to space limitation.

Syntactically the extension naturally extends the SPARQL one, by allowing the occurrence of fuzzy graph patterns in the WHERE clause and the occurrence of fuzzy conditions in the FILTER clause. A fuzzy regular expression is close to a property path, as defined in SPARQL 1.1 [10], and involves a fuzzy structural property (e.g. distance and strength over fuzzy graphs). Semantically, the extension relies on the semantics defined in Section III.

Example 5: The FURQL query of Listing 3 retrieves artists that recommend low-rated albums by close friends (see Pattern $P_{rec_{low}}$ of Example 4), and performs an alpha-cut on the answers (only those having a satisfaction degree greater or equal to 0.4 are kept). The cut clause is of course optional.

```sql
SELECT ?art1 WHERE {
    { ?art1 (friend+ | distance is short) ?art2.
      ?art2 creator ?alb.
      ?alb rating ?r.
      ?art1 recommends ?alb. }
    FILTER (?r is low).
} cut 0.4

Listing 3. A FURQL query containing $P_{rec_{low}}$
```

The result of this query over $G_{MB}$ is the singleton \{\text{Enriquei}\} which is $m(\text{Art1})$ in the mapping \{\text{Art1} \rightarrow \text{Enriquei}, \text{alb} \rightarrow \text{Justified}, \text{r} \rightarrow 6\}, i.e., the only mapping of $[P_{rec_{low}}]_{G_{MB}}$ having a satisfaction degree greater or equal to 0.4 (see Example 4). ⊙

We now discuss implementation issues related to our proposal. Two aspects have to be considered: i) the storage of fuzzy RDF graphs, and ii) the evaluation of FURQL queries.
The first point does not raise any serious problem since one may use the reification mechanism that makes it possible to attach fuzzy degrees to triples, as proposed in [22].

Concerning the evaluation of FURQL queries, two architectures may be thought of. A first solution consists in implementing a specific query evaluation engine inside the data management system. Figure 8.(a) is an illustration of this architecture. The advantage of this solution is that optimization techniques implemented directly in the query engine should make the system very efficient for query processing. An important downside is that the implementation effort is substantial, but the strongest objection for this solution is that the evaluation of a FURQL query in a distributed architecture would imply having available a FURQL query evaluator at each SPARQL endpoint, which is not realistic at the time being. An alternative, more realistic architecture consists in adding a FURQL software layer over a standard — and possibly distant — classical SPARQL engine (endpoint). This layer basically consists in a query compiler producing a (crisp) SPARQL query for retrieving the appropriate data, a (query-dependent) function for computing the satisfaction degrees, and a (query-dependent) function for ranking the answers (and filtering them in case an $\alpha$-cut has to be returned). Figure 8.(b) is an illustration of this architecture. Deriving a SPARQL query from a FURQL query should rely on the derivation principle proposed in [5] in a relational database context and applied in [13] to “simple” fuzzy SPARQL queries (i.e., queries that do not involve any structural conditions). It remains to be studied how the different types of structural conditions allowed in FURQL can be derived into crisp conditions in the syntax of SPARQL 1.1., but note that it is always possible to derive them into true (which amounts to discarding them from the crisp SPARQL query) and to take them into account only in the “fuzzy treatment” module (cf. Figure 8.(b)) if no better derivation can be found. This solution may look extreme at first sight but it could be in fact reasonable if the non-structural part (i.e., the “value-based” part) of the global fuzzy condition in the query is selective enough for avoiding a plethoric set of answers to assess.

V. RELATED WORK

Related work concerns two categories of approaches: those that extend SPARQL with navigational capabilities, and those that extend it with fuzzy querying functionalities. To the best of our knowledge, there does not exist any work in the literature that does both.

In the first category, some new query languages [11], [3], [17], [1] have been designed to overcome the limited functionalities of the standard query language SPARQL in terms of path extractions queries (generally of unknown length) within RDF databases. These languages are more expressive and extend SPARQL by allowing querying RDF data through graph path patterns by using regular expressions. Motivated by this concern, [11] and [3] introduce respectively SPARQLeR and SPARQL2L, two extensions of the classical query language SPARQL with path patterns that represent (undirected and directed) paths between nodes in the RDF graph. They both extend the SPARQL graph patterns with path variables (i.e., path triple patterns) in the predicate position. This makes it possible to express more complex queries such as subgraph extraction queries (e.g., how are A, B, C and D related?), reachability queries (e.g., does there exist a path from A to F?), etc. More recently, [1] proposed an extension of SPARQL which captures more expressive graph patterns, called PSPARQL (Path SPARQL). This query language is a combination of SPARQL and graph query languages (namely G, G+ and Graphlog). The authors propose a syntactic and semantic redefinition of the RDF model making it possible to express regular expression patterns instead of predicates in the RDF triple. This new model is called Path RDF (PRDF) which is an extension of a GRDF graph (i.e., it permits the expression of variables as predicates, as well as literals as subjects) with regular expression patterns constructed over the set of urirefs and the set of variables in the predicate position. Then, they develop the query language PSPARQL, which extends SPARQL with PRDF graph patterns (i.e., graph patterns with (involving) regular expression) able to find nonsimple and arbitrary length path queries. In [2], the same authors extend (P)SPARQL by adding the ability to express complex constraints over regular expressions during path search. Another important related work — that we chose as one of the bases of our own approach as it is very well formalized — is [17] where the authors introduce a new query language named nSPARQL, which makes it possible
to express complex navigation statements in an RDF graph. With nSPARQL, one may navigate through a ground RDF graph using the notion of nested regular expressions in the predicate position.

As for the second category — that gathers approaches that deal with the fuzzy querying issue —, different researchers have extended SPARQL so as to make it more flexible. A variety of proposals aimed to introduce preferences into SPARQL queries can be found in the literature (see the survey paper [18]). These approaches may be classified into two categories: i) quantitative ones (fuzzy-set-based and top-k queries, and ii) qualitative ones that extend the classical Skyline approach. To the best of our knowledge, the existing fuzzy-set-based approaches, namely [7], [26], [13], make it possible to express preferences on the value of the nodes but not on the structure of the RDF graph, contrary to ours. This is also the case, by the way, of the other flexible querying approaches, either qualitative or quantitative, see [18].

Let us mention, however, that the expression of fuzzy preferences involving both a value-based and a structural aspect was first tackled in [19] in a more general context than that of RDF databases, namely that of graph databases. The present work can thus be seen as a (nontrivial) adaptation of the concepts from [19] to the RDF/SPARQL setting.

VI. CONCLUSION

In this paper, we have defined a fuzzy extension of SPARQL that goes beyond the previous proposals in terms of expressiveness inasmuch as it makes it possible i) to deal with fuzzy RDF data, and ii) to express fuzzy structural conditions beside more classical fuzzy conditions on the values of the nodes present in the graph. The language, called FURQL, is based on the notion of fuzzy graph pattern which extends Boolean graph patterns introduced by several authors in a crisp querying context. Perspectives for future work are manifold. First, it is of course necessary to deal more in-depth with implementation issues (which could only be briefly discussed here) and to implement a prototype illustrating the power of the approach on real-world RDF databases. Secondly, different possibilities exist for extending the language with more sophisticated fuzzy conditions. One may think for instance of conditions involving fuzzy quantifiers, and of fuzzy predicates related to some structural properties of the nodes (centrality, in-degree, out-degree, etc.). It would also be worth investigating the way this framework could be applied to the management of quality-related metadata (about freshness, reliability and so on) which are in general of a fuzzy nature.

Acknowledgement: This work has been partially funded by the French DGE (Direction Générale des Entreprises) under the project ODIN (Open Data INelligence).

REFERENCES