A Flexible XPath-Based Query Language Implemented with Fuzzy Logic Programming*

Jesús M. Almendros-Jiménez[1], Alejandro Luna[2], and Ginés Moreno[2]

[1] Dep. of Languages and Computation, University of Almería, Spain
jalmen@ual.es
[2] Dep. of Computing Systems, University of Castilla-La Mancha, Spain
{Gines.Moreno,Alejandro.Luna}@uclm.es

Abstract. In this paper we present an extension of the XPath query language for the handling of flexible queries. In order to provide ranked answers, our approach proposes fuzzy variants of *and*, *or* and *avg* operators for XPath conditions, as well as two structural constraints, called *down* and *deep*, for which a certain degree of relevance is associated. Our proposal has been implemented with a fuzzy logic language to take profit of the clear sinergies between both target and source fuzzy languages.

1 Introduction

The *XPath* language [3] has been proposed as a standard for XML querying and it is based on the description of the path in the XML tree to be retrieved. XPath allows to specify the name of nodes (i.e., tags) and attributes to be present in the XML tree together with boolean conditions about the content of nodes and attributes. XPath querying mechanism is based on a boolean logic: the nodes retrieved from an XPath expression are those matching the path of the XML tree. Therefore, the user should know the *XML schema* in order to specify queries. However, even when the XML schema exists, it can not be available for users. Moreover, XML documents with the same XML schema can be very different in structure. Let us suppose the case of XML documents containing the curriculum vitae of a certain group of persons. Although they can share the same schema, each one can decide to include studies, jobs, training, etc. organized in several ways: by year, by relevance, and with different nesting degree.

Therefore, in the context of semi-structured databases, the need for *flexible query languages* arises, in which the user can formulate queries without taking into account a rigid schema database. In addition, they should be equipped with a mechanism for obtaining a certain *ranked list* of answers. The ranking of answers can provide *satisfaction degree* depending on several factors. In a structural XPath-based query, the main criteria to provide a certain degree of

* This work has been partially supported by the EU, under FEDER, and the Spanish Science and Innovation Ministry (MICINN) under grants TIN 2008-06622-C03-03, TIN 2007-65749 and TIN2011-25846, as well as by the Castilla-La Mancha Administration under grant PII1I09-0117-4481.

N. Bassiliades et al. (Eds.): RuleML 2011 - Europe, LNCS 6826, pp. 186–193, 2011.

satisfaction depends on the *hierarchical deepness* and *document order*. Therefore the query language should provide mechanisms for giving *priority* to answers when they occur in different parts of the document.

In this paper we present an extension of the XPath query language for the handling of flexible queries. Our approach proposes two structural constraints called *down* and *deep* for which a certain degree of relevance can be associated. In such a way that *down* provides a ranked set of answers depending on the path is found from "top to down" in the XML document, and *deep* provides a set of answers depending on the path is found from "left to right" in the XML document. Both structural constraints can be combined. In addition, we provide fuzzy operators *and*, *or* and *avg* for XPath conditions. In this way, users can express the priority they give to answers. Such fuzzy operators can be combined to provide ranked answers. Our approach has been implemented by means of multi-adjoint logic programming and the FLOPER tool.

The need for providing flexibility to XPath has recently motivated the investigation of extensions of the XPath language. The most relevant ones are [4,5] in which authors introduce in XPath flexible matching by means of fuzzy constraints called *close* and *similar* for node content, together with *below* and *near* for path structure. In addition, they have studied *deep-similar* notion for tree matching. In order to provide ranked answers they adopt a *Fuzzy set theory*-based approach in which each answer has an associated numeric value (the membership degree). The numeric value represents the *Retrieval Status Value (RSV)* of the associated item. In the work of [7], they propose a satisfaction degree for XPath expressions based on associating a degree of importance to XPath nodes, and they study how to compute the best k answers. In both cases, authors allow the user to specify in the query the degree in which the answers will be penalized. On the other hand, in [6], they have studied how to relax XPath queries by means of rewriting in order to improve information retrieval in the presence of heterogeneous data resources.

Our work is similar to the proposed by [4,5]. The *below* operator of [4,5] is equivalent to our proposed *down*: both extract elements that are direct descendants of the current node, and the penalization is proportional to the distance. The *near* operator of [4,5], which is defined as a generalization of *below*, ranks answers depending of the distance to the required node, in any XPath axis. Our proposed *deep* ranks answers depending of the distance to the current node, but the nodes considered can be direct and non direct descendants. Therefore our proposed *deep* combined with *down* is a particular case of *near*. However, our aim is to extend the number of constraints and fuzzy operators of our approach thanks to the expressivity power of our framework based on fuzzy logic programming. The so-called *multi-adjoint logic programming* approach, MALP in brief [9], is an extension of logic programming for covering with fuzzy logic. Such framework provides theoretical basis for defining flexibility to XPath in many directions. In addition, the framework provides mechanism for customizing ranked answers as assigning priority of elements with independence when they occur.

With respect to *similar* and *close* operators proposed in [4,5], our framework lacks on similarity relations, rather than it focus on structural (i.e. path-based) flexibility. With regard to tree matching, the operator *deep-similar* defined in [4,5] can be simulated by means of *deep* and *down* operators. We believe that we could also work in the future in adapting our framework for working with degree of importance to XPath nodes in the line of [7], and relaxing XPath expressions by rewriting in the line [6]. In both cases, our framework could provide ranked answers w.r.t. the degree of importance, and degree of matching. Our proposal makes use of the multi-adjoint logic programming framework for defining new fuzzy operators for XPath: *and*, *or* and *avg*. Such operators are used in XPath conditions on nodes and attribute values. They provide fuzzy combinations for ranking answers.

Finally, let us remark that our work is an extension of previous works about the implementation of XPath by means of logic programming [2], which has been extended to XQuery in [1]. The proposed extension follows the same encoding proposed in [1] in which a predicate called *xpath* is defined by means of Prolog rules, which basically traverse the Prolog representation of the XML tree by means of a Prolog list. In order to implement the flexible extension of XPath by means of the «*Fuzzy LOgic Programming Environment for Research*» FLOPER (which is devoted to the management of MALP programs [10,11]), we proceed similarly to the Prolog implementation of XPath, but proposing a new (fuzzy) predicate called *fuzzyXPath* implemented in MALP. The new query language returns a set of ranked answers each one with an associated RSV. Such RSV is computed by easily using MALP rules (thus exploiting the correspondences between the languages *for-being* and *to-be* implemented), where the notion of *RSV* is modeled inside a multi-adjoint lattice, and usual fuzzy connectives of the MALP language act as ideal resources to represent new flexible XPath operators.

The structure of the paper is as follows. Whereas in Section 2 we present our fuzzy extension of XPath, Section 3 is devoted to describe the main elements of the implementation of XPath in MALP and FLOPER. Finally, Section 4 concludes by also planning future work.

2 Flexible XPath

Our flexible XPath is defined by means of the following rules:

```
    xpath := [deepdown]path
     path := literal | text() | node | @att |
             node/path | node//path
     node := QName | QName[cond]
     cond := path op path
 deepdown := DEEP=degree,DOWN=degree
       op := > | = | < | and | or | avg
```

```
<bib>
   <book year="2001" price="45.95">
      <title>Don Quijote de la Mancha</title>
      <author>Miguel de Cervantes Saavedra</author>
      <publications> <book year="1997" price="35.99">
                     <title>La Galatea</title>
                     <author>Miguel de Cervantes Saavedra</author>
                     <publications>
                              <book year="1994" price="25.99">
                              <title>Los trabajos de Persiles y Segismunda</title>
                              <author>Miguel de Cervantes Saavedra</author></book>
                     </publications></book>
      </publications></book>
   <book year="1999" price="25.65">
      <title>La Celestina</title>
      <author>Fernando de Rojas</author></book>
   <book year="2005" price="29.95">
      <title>Hamlet</title>
      <author>William Shakespeare</author>
      <publications>
         <book year="2000" price="22.5">
            <title>Romeo y Julieta</title>
            <author>William Shakespeare</author></book>
      </publications></book>
   <book year="2007" price="22.95">
      <title>Las ferias de Madrid</title>
      <author>Felix Lope de Vega y Carpio</author>
      <publications>
         <book year="1996" price="27.5">
            <title>El remedio en la desdicha</title>
            <author>Felix Lope de Vega y Carpio</author> </book>
         <book year="1998" price="12.5">
            <title>La Dragontea</title>
            <author>Felix Lope de Vega y Carpio</author></book>
      </publications></book>
</bib>
```

Fig. 1. Input XML document in our examples

Basically, our proposal extends XPath as follows:

- A given XPath expression can be adorned with «[DEEP = r_1, DOWN = r_2]» which means that the *deepness* of elements is penalized by r_1 and that the *order* of elements is penalized by r_2, and such penalization is proportional to the distance. In particular, «[DEEP = 1, DOWN = r_2]» can be used for penalizing only w.r.t. document order. $DEEP$ works for //, and $DOWN$ works for / and //.
- Moreover, the classical *and* and *or* connectives admit here a fuzzy behavior based on fuzzy logic, i.e., assuming two given RSV's r_1 and r_2, operator *and* is defined as $r_3 = r_1 * r_2$ and operator *or* returns $r_3 = r_1 + r_2 - (r_1 * r_2)$. In addition, the *avg* operator is defined as $r_3 = (r_1 + r_2)/2$.

In general, an extended XPath expression defines, w.r.t. a XML document, a sequence of subtrees of the XML document where each subtree has an associated RSV. XPath conditions, which are defined as fuzzy operators applied to XPath expressions, compute a new RSV from the RSVs of the involved XPath expressions, which at the same time, provides a RSV to the node. In order to illustrate

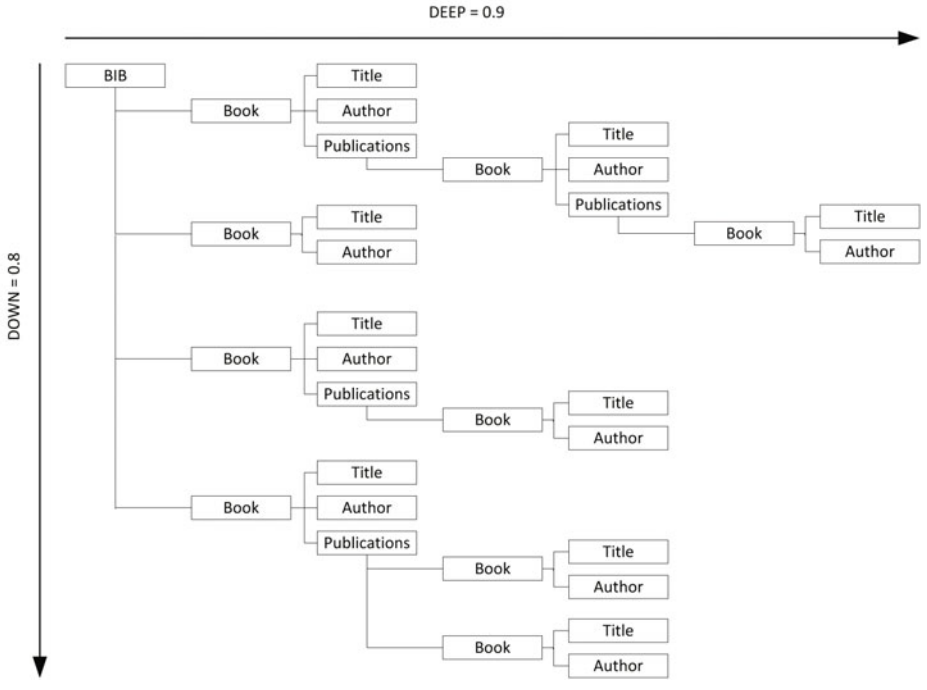

Fig. 2. XML skeleton represented as a tree

these explanations, let us see some examples of our proposed fuzzy version of XPath according to the XML document shown in Figure 1, whose *skeleton* is depicted in Figure 2.

Document	**RSV computation**
<result> <title **rsv**="0.81">Don Quijote de la Mancha</title> <title **rsv**="0.6561">La Galatea</title> <title **rsv**="0.531441">Los trabajos de Persiles y ...</title> <title **rsv**="0.648">La Celestina</title> <title **rsv**="0.5184">Hamlet</title> <title **rsv**="0.419904">Romeo y Julieta</title> <title **rsv**="0.41472">Las ferias de Madrid</title> <title **rsv**="0.3359232">El remedio en la desdicha</title> <title **rsv**="0.26873856">La Dragontea</title> </result>	$0.81 = 0.9^2$ $0.6561 = 0.9^4$ $0.531441 = 0.9^6$ $0.648 = 0.9^2 * 0.8$ $0.5184 = 0.9^2 * 0.8^2$ $0.419904 = 0.9^4 * 0.8^2$ $0.41472 = 0.9^2 * 0.8^3$ $0.3359232 = 0.9^4 * 0.8^3$ $0.26873856 = 0.9^4 * 0.8^4$

Fig. 3. Output of a query using $DEEP/DOWN$

Example 1. Suppose the XPath query: « [DEEP=0.9,DOWN=0.8]//title », that requests *title*'s penalizing the occurrences from the document root by a proportion of 0.9 and 0.8 by nesting and ordering, respectively, and for which we obtain the file listed in Figure 3. In such document we have included as attribute of each subtree, its corresponding RSV. The highest RSVs correspond the main *book*'s of the document, and the lowest RSVs represent the *book*'s occurring in nested positions (those annotated as related *publication*'s).

Document	RSV computation
<result>	
<book **rsv**="0.5" ...> <title>Don Quijote ...</title> ...</book>	$0.5 = (0 + 1)/2$
<book **rsv**="1.0"...><title>La Celestina</title> ...</book>	$1 = (1 + 1)/2$
<book **rsv**="1.0" ...><title>Hamlet</title> ...</book>	$1 = (1 + 1)/2$
<book **rsv**="0.5" ...><title>Las ferias de Madrid</title> ...</book>	$0.5 = (1 + 0)/2$
</result>	

Fig. 4. Output of a query using *AVG*

Document	RSV computation
<result>	
<title **rsv**="0.3645">La Galatea</title>	$0.3645 = 0.9^3 * 1/2$
<title **rsv**="0.295245">Los trabajos de Persiles y... </title>	$0.295245 = 0.9^5 * 1/2$
<title **rsv**="0.72">La Celestina</title>	$0.72 = 0.9 * 0.8 * 1$
<title **rsv**="0.288">Hamlet</title>	$0.288 = 0.9 * 0.8^2 * 1/2$
<title **rsv**="0.2304">Las ferias de Madrid</title>	$0.2304 = 0.9 * 0.8^3 * 1/2$
<title **rsv**="0.2985984">El remedio en la desdicha</title>	$0.2985984 = 0.9^3 * 0.8^4 * 1$
<title **rsv**="0.11943936">La Dragontea</title>	$0.11943936 = 0.9^3 * 0.8^5 * 1/2$
</result>	

Fig. 5. Output of a query using all operators

Example 2. Figure 4 shows the answer associated to the XPath expression: « /bib/book[@price<30 avg @year<2006] ». Here we show that books satisfying a *price* under 30 and a *year* before 2006 have the highest RSV.

Example 3. Finally, combining all operators «[DEEP=0.9,DOWN=0.8] //book [(@price>25 and @price<30) avg (@year<2000 or @year>2006)]/title», the RSV values are more scattered, as shown in Figure 5.

3 Some Implementation Hints Using MALP

In this section we assume familiarity with logic programming and its most popular language Prolog [8], for which MALP [9] (*Multi-Adjoint Logic Programming*[1]) allows a wide repertoire of *fuzzy connectives* connecting atoms in the bodies of clauses.

Although the core of our application is written with (fuzzy) MALP rules, our implementation is based on the following items:

(1) We have reused/adapted several modules of our previous Prolog-based implementation of (crisp) XPath described in [1,2].
(2) We have used the SWI-Prolog library for loading XML files, in order to represent a XML document by means of a Prolog term[2].
(3) The parser of XPath has been extended to recognize the new keywords *deep, down, avg*, etc... with their proper arguments.

[1] See also [10,11] and visit `http://dectau.uclm.es/floper` for downloading our prototype system FLOPER.
[2] The notion of *term* (i.e., data structure) is just the same in MALP and Prolog.

```
[element(bib,[],
    [element(book,[year=2001,price=45.95],
             [element(title,[],[Don Quijote de la Mancha]),
             element(author,[],[Miguel de Cervantes Saavedra]),
             element(publications,[],
                          [element(book,[year=1997,price=35.99],
                                   [element(title,[],[La Galatea]),
                                   element(author,[],[Miguel de Cervantes Saavedra]),
                                element(publications,[],...])...]),])])
```

Fig. 6. A data-term representing a XML document

(4) Each tag is represented as a data-term of the form: `element(Tag, Attributes, Subelements)`, where `Tag` is the name of the XML tag, `Attributes` is a Prolog list containing the attributes, and `Subelements` is a Prolog list containing the subelements (i.e. subtrees) of the tag. For instance, the document of Figure 1 is represented in SWI-Prolog like in Figure 6. Loading of documents is achieved by the predicate `load_xml(+File,-Term)` and writing by the predicate `write_xml(+File,+Term)`.

(5) A predicate called `fuzzyXPath` where `fuzzyXPath(+ListXPath,+Tree,+Deep,+Down)` receives four arguments: (1) `ListXPath` is the Prolog representation of an XPath expression; (2) `Tree` is the term representing an input XML document and (3) `Deep/Down` which have the obvious meaning.

(6) The evaluation of the query generates a *truth value* which has the form of a tree, called *tv tree*. For instance, the query shown in Example 1, generated the one illustrated in Figure 7. The main power of a fuzzy logic programming language like MALP w.r.t. Prolog, is that instead of answering questions with a simple *true/false* way, solutions are reported in a much more tinged, documented form. Basically, the `fuzzyXPath` predicate traverses the Prolog tree representing a XML document annotating into the *tv tree* the corresponding *deep/down* values according to the movements performed in the horizontal and vertical axis, respectively. In addition, the *tv tree* is annotated with the values of *and*, *or* and *avg* operators in each node.

(7) Finally, the *tv tree* is used for computing the output of the query, by multiplying the recorded values. A predicate called `tv_to_elem` has been implemented to output the answer in a pretty way.

```
tv(0.9,[[],
    tv(0.9,[element(title,[],[Don Quijote de la Mancha]),[],
    tv(1,[[],[],
    tv(1,[[],
       tv(0.9,[[],
          tv(0.9,[element(title,[],[La Galatea]),[],
          tv(1,[[],[],
          tv(1,[[],
             tv(0.9,[[],
                tv(0.9,[element(title,[],[Los trabajos de Persiles..]),...]),
    tv(0.8,[[],
      tv(0.9,[element(title,[],[La Celestina]),[],[]]),...
```

Fig. 7. Example of a *MALP* output

More details about our implementation of the flexible version of XPath reported in this paper, are available on: `http://dectau.uclm.es/fuzzyXPath/`

4 Conclusions and Future Work

In this paper we have enriched XPath with new constructs (both structural -*deep* and *down*- and constraints -*avg* and fuzzy versions of classical *or*/*and* operators-) in order to flexibly query XML documents. This paper represents the first real-world application developed with the fuzzy logic language MALP, by showing its capabilities for easily modeling scenarios where concepts somehow based on fuzzy logic play a crucial role. We think that this research line promises fruitful developments in the near future by reinforcing the power of fuzzy XPath commands, extensions to cope with XQuery and the semantic web, etc.

References

1. Almendros-Jiménez, J.M.: An Encoding of XQuery in Prolog. In: Bellahsène, Z., Hunt, E., Rys, M., Unland, R. (eds.) XSym 2009. LNCS, vol. 5679, pp. 145–155. Springer, Heidelberg (2009)
2. Almendros-Jiménez, J.M., Becerra-Terón, A., Enciso-Baños, F.J.: Querying XML documents in logic programming. TPLP 8(3), 323–361 (2008)
3. Berglund, A., Boag, S., Chamberlin, D., Fernandez, M.F., Kay, M., Robie, J., Siméon, J.: XML path language (XPath) 2.0. W3C (2007)
4. Campi, A., Damiani, E., Guinea, S., Marrara, S., Pasi, G., Spoletini, P.: A fuzzy extension of the XPath query language. Journal of Intelligent Information Systems 33(3), 285–305 (2009)
5. Damiani, E., Marrara, S., Pasi, G.: FuzzyXPath: Using fuzzy logic an IR features to approximately query XML documents. In: Melin, P., Castillo, O., Aguilar, L.T., Kacprzyk, J., Pedrycz, W. (eds.) IFSA 2007. LNCS (LNAI), vol. 4529, pp. 199–208. Springer, Heidelberg (2007)
6. Fazzinga, B., Flesca, S., Furfaro, F.: On the expressiveness of generalization rules for XPath query relaxation. In: Proceedings of the Fourteenth International Database Engineering & Applications Symposium, pp. 157–168. ACM, New York (2010)
7. Fazzinga, B., Flesca, S., Pugliese, A.: Top-k Answers to Fuzzy XPath Queries. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2009. LNCS, vol. 5690, pp. 822–829. Springer, Heidelberg (2009)
8. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Berlin (1987)
9. Medina, J., Ojeda-Aciego, M., Vojtáš, P.: Similarity-based Unification: a multi-adjoint approach. Fuzzy Sets and Systems 146, 43–62 (2004)
10. Morcillo, P.J., Moreno, G.: Programming with Fuzzy Logic Rules by using the FLOPER Tool. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) RuleML 2008. LNCS, vol. 5321, pp. 119–126. Springer, Heidelberg (2008)
11. Morcillo, P.J., Moreno, G., Penabad, J., Vázquez, C.: A Practical Management of Fuzzy Truth Degrees using FLOPER. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 20–34. Springer, Heidelberg (2010)