An XQuery Specification for Requests with Preferences on XML Databases

Maurice Tchoupé Tchendji[1] and Patrik Joslin Kenfack[2](✉)

[1] Department of Mathematics and Computer Science, University of Dschang, Dschang, Cameroon
maurice.tchoupe@univ-dschang.org

[2] Department of Computer Science and Automated Systems, Belgorod State Technological University, Belgorod, Russia
kenfackjoslin@gmail.com

Abstract. An exact query is a query in which the user specifies precisely what to retrieve from a database (XML or relational database). For these queries only data that strictly respect all user's conditions is returned. XML documents are generally semi-structured. Due to the non-existence or lack of knowledge of the model of the document being queried, when exact queries are used, there is a high risk of obtaining an empty result (in the case of too specific queries) or too large (in the case of too vague queries). In contrast to exact queries, requests with preferences aim to return only the most relevant results in order to avoid empty or too important results as much as possible. To achieve this goal, requests with preferences generally consist of two parts: the first part is used to express strict constraints and the second part to express preferences or wishes. The satisfaction of both parts increases the relevance of the corresponding results. This paper presents *XQuery preference*, an extension of the XQuery language, that allows to express requests with preferences relating to both the values and the structure of an XML document. A representation model of such requests based on the Generalized Tree Pattern (GTP) model is also proposed in order to allow an evaluation of these requests through a tree pattern matching process. Integration of the proposed language in open source implementations of XQuery like BaseX, Berkeley DB XML, eXist-db, Galax and much more, will allow users to get much more relevant responses to their concerns.

Keywords: Semi-structured documents · XQuery · XQuery preference · XML · Open source

1 Introduction

One of the consequences of the proliferation of online information today is data diversity. XML is widely used as a core technology for knowledge management within companies and the dissemination of data on the Web (such as product catalogues), as it allows semi-structured data to be organized and manipulated.

V. Ivanov et al. (Eds.): OSS 2020, IFIP AICT 582, pp. 120–130, 2020.
https://doi.org/10.1007/978-3-030-47240-5_12

A collection of XML documents is considered to be a forest of trees with labelled nodes. To manipulate the data stored in XML and extract the relevant information in terms of structure and/or content, many query languages have been proposed such as XPath [6] and XQuery [7]. Indeed, these query languages take into account both the content and the structure of underlying documents, as it can completely change their relevance and adequacy with regard to the needs expressed by the user. However, it is important to note that in order to query a document using these languages, the user must a priori know its structure. This requirement is difficult to meet in an open environment such as the Web, where document structures are not always available. Thus, due to the non-existence or lack of knowledge of the model of the document being queried, the documents are queried almost blindly. Query writers in this context do so according to an imaginary document structure they believe to be that of the document. Such queries would generally return either no results (cases of too specific queries having little or no match with the content or structure of the document in question) or, in the extreme, too many results (case of too general queries intensively using wildcards).

The problems of absence or very large number of results of a query have also emerged within the classical database community. *Requests with preference* aims to be a solution to this problem. Intuitively, a request with preferences specifies the user's wishes and consists of two parts specifying on the first one, the mandatory requirements called *constraints* and on the second part, the optional requirements called *wishes or preferences.* However, a result of a request with preferences must necessarily satisfy the first part and possibly the second; if there is at least one answer satisfying the first and second part of the query, only the answers satisfying both parts must be returned as a result.

Some specific languages (generally extensions of SQL or XPath) have been proposed for the formulation of queries with preferences: *SQLf* [2], *Preference SQL* [10], *Preferences Queries* [5], ... for relational databases (RBD), *XPref* [1], *Preference XPATH* [9], *PrefSXPath* [13], ... for XML databases (XML BDs). Generally, proposed extensions of XPath language for importing the concept of preferences are only interested in either value based preferences [1,9] or structure based preferences [13]. Our approach takes into account these two types of preferences in order to better satisfy user requirements.

The rest of this paper is organized as follows: Sect. 2 presents some concepts related to XML documents, queries languages and open source implementations, we detail our specifications by presenting in Sect. 3, a grammar for the language *XQuery preference* and the representation model. And we finally end with a conclusion and future work.

2 XML Document and Queries Languages

An XML document is commonly considered as a tree where nodes are the elements or attribute names, and edges represent the child node's membership of the parent node and where leaves are the contents of the elements or attribute values. An XML database is a forest of XML document trees. The Fig. 2 illustrates an example of a tree representation of an XML document (Fig. 1).

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
  <person id="122">
   <profile>
    <name>Joel Dongmo</name>
    <email>joeldongmo@gmail.com</email>
   </profile>
   <experience>3</experience>
   <diploma>MSc</diploma>
   <skills>
    <java>3</java>
    <spring-mvc>4</spring-mvc>
    <symfony>4</symfony>
   </skills>
  </person>
</persons>
```

Fig. 1. XML document.

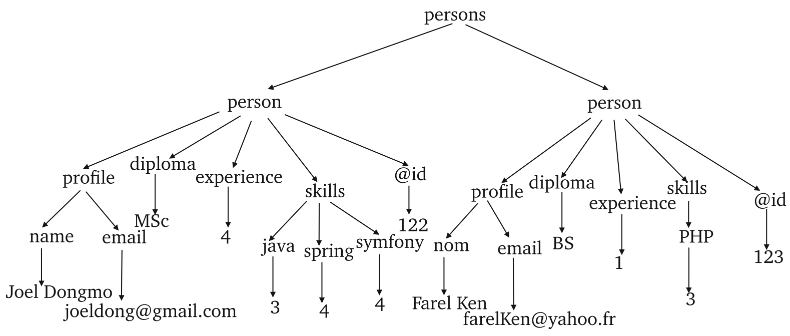

Fig. 2. Tree representation of an XML document.

The use of XML documents requires the ability to extract information and reformulate it for applications. Thus, there are many languages to retrieve information from a semi-structured document. Here we present the most popular: XPath and XQuery.

2.1 XPath

XPath allows to designate one or more nodes in an XML document, using path expressions. Thus, an XPath expression is a sequence of steps. $[/]step_1/step_2/.../step_n$. An XPath step consists of an axis, a filter and a predicate (optional): $axe :: filtre[predicat]$ The axis indicates a search direction. The most used axes are parent-child (represented by A/B) and descendant axis ($A//B$). The filter selects a node type. For example the expression A/B returns all elements B children of an element A. Predicates select content.

2.2 XQuery

XQuery [7] is the query language recommended by the W3C to extract information from many types of XML data sources. XQuery is the XML equivalent of

SQL language, for retrieving data contained in relational databases and inherits the properties of several other languages. From XPath it uses the path expression syntax for addressing elements in XML documents. From SQL it takes up the idea of a series of clauses based on keywords that provide a model for data restructuring (the SQL SELECT-FROM-WHERE model). XQuery queries have several expression forms, the most famous is the FLWOR form. The acronym FLWOR comes from the reserved words of the language which make help to define the main clauses of this type of expression: **F**or - **L**et - **W**here - **O**rder By - **R**eturn.

Each clause in a FLWOR expression plays a particular role in the query and some of these clauses are optional. Thus, a FLWOR instruction consists of the following parts:

- **For:** iteration on an XML document part list
- **Let:** allows the assignment of values to a variable
- **Order by:** sorting results
- **Where:** restriction clause (constraints)
- **Return:** form of the expression to be returned

An example of this type of request is given in Fig. 3. This query select the email addresses and skills of people who have a Java skill level above 3 and have more than two years of experience.

```
FOR $p IN document("candidat.xml")//persons, $c IN $p/skills
WHERE $c/java >= 3 AND $p/experience > 2
RETURN <result> {$p//profile/email} {$c}</result>
```

Fig. 3. Example of an XQuery query

2.3 Open Source Implementations of XQuery

There are many open source implementations of XQuery. We present here a non-exhaustive list of them:

- BaseX is a very light-weight, high-performance and scalable XML Database engine and XPath/XQuery 3.0 Processor, including full support for the W3C Update and Full Text extensions all developed in Java. It comes with interactive user interfaces (desktop, web-based) that give users great insight into their data [8].
- eXist-db is a high-performance open source native XML database - a NoSQL document database and application platform built entirely around XML technologies. A Browser-based IDE allows managing and editing all artifacts belonging to an application. Syntax-coloring, code-completion and error-checking help to get it right. Being a complete solution, eXist-db tightly integrates with XForms for complex form development [12].

- Galax is an open-source implementation of XQuery, the W3C XML Query Language. It includes several advanced extensions for XML updates, scripting, and distributed programming. Implemented in O'Caml, Galax comes with a state of the art compiler and optimizer. Most of Galax's architecture is formally documented, making it ideal for users interested in teaching XQuery, in building new language extensions, or developing new optimizations [11].
- Oracle Berkeley DB XML is an open source, embeddable XML database with XQuery-based access to documents stored in containers and indexed based on their content. Implemented in C, Oracle Berkeley DB XML is built on top of Oracle Berkeley DB and inherits its rich features and attributes.

XML queries (XPath or XQuery) to be evaluated on XML documents (trees), need to be represented in a model (a tree representation) in order to facilitate their evaluation. Therefore, evaluating the query is equivalent to apply the corresponding model to the XML tree trough a *tree pattern matching* process.

2.4 Generalized Tree Pattern (GTP)

The concept of the generalized tree model (GTP) is introduced in [4] and allows to express more precisely the semantics of XQuery. The arcs of a GTP may be Parent-Child (PC), Ancestor-Descendant (AD) or optional. They are indicated by solid edges, double solid edges and dotted edges, respectively. A mandatory arc links an sub-expression corresponding to clauses *FOR* and *WHERE* with the rest of the query. An optional arc links an subexpression corresponding to clauses *LET* and *RETURN* with the rest of the query.

Definition 1. A generalized tree pattern is a couple $G = (T, F)$ where T is a tree and F a Boolean formula such as.

- Each node in the tree T is labeled with different variables and has a group number.
- To each arc of T is associated a pair of labels $< x, m >$, where $x \in \{PC, AD\}$ specifies the axis (parent-child and ancestor-descendant, respectively) and $m \in \{mandatory, optional\}$ specifies the arc's status.
- F is a Boolean combination of predicates applicable to nodes.

Zhimin Chen et al. [4] also propose an algrithm for translating an XQuery expression[1] in GTP. The request is put in a canonical form and is then parsed clause by clause while the GTP is progressively built up to the last clause, we invite you to read [4] for more details. The GTP is intended to be mapped (*Pattern matching*) to the XML tree.

Definition 2. A *Pattern Match* of a GTP $G = (G, F)$ in a tree collection C is a subtree h partial: $h : G \rightarrow C$ such that:

[1] The GTP model deals with a very significant subset of the XQuery language and supports nesting, quantizer, grouping and aggregation.

– h contains at least group 0 of G.
– h preserves the relational structure of G. This means that whenever h is defined on two nodes u, v and there is a PC arc (respectively AD) (u, v) in G, then h(v) is a son (respectively a descendant) of h(u).
– h satisfies the Boolean formula F of G (Fig. 4).

```
FOR $p IN document("auction.xml")//person, $l In $p/profile
WHERE $l/age > 25 AND $p//state != 'MI'
RETURN<result> {$p//watches/watch} {$l/interest}</result>
```

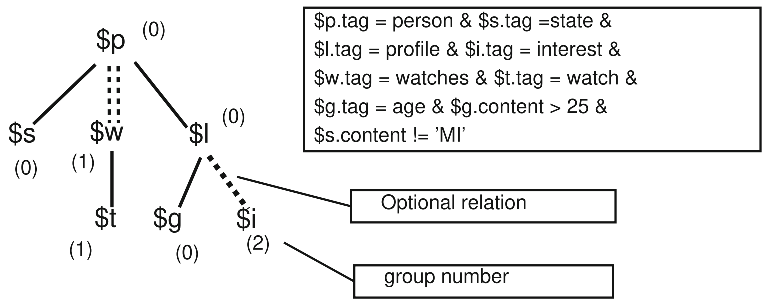
$p (0)
$s (0)
$w (1)
$l (0)
$t (1)
$g (0)
$i (2)
$p.tag = person & $s.tag =state &
$l.tag = profile & $i.tag = interest &
$w.tag = watches & $t.tag = watch &
$g.tag = age & $g.content > 25 &
$s.content != 'MI'
Optional relation
group number

Fig. 4. Example of an XQuery expression and corresponding GTP query.

3 XQuery Preference: A Language for Expressing XQuery Requests with Preference

In order to take into account the two types of preferences, namely on values based and structure based, we propose an extension of XQuery at two levels: an extension of XPath language for the integration of structural preferences, through the use of the operator "!" introduced in [13] and the addition of a new clause (the **Pref** clause) for the expression of preferences on content.

3.1 Grammar of the Language *XQuery Preference*

Based on the grammar given in [4] to describe a significant subset of the XQuery language, we consider the following grammar for XQuery preference expressions (Fig. 5).

FLWR ::= (ForClause | LetClause)+ WhereClause **PrefClause** ReturnClause.
ForClause ::= FOR $\$f_{v1}$IN E_1 , ..., $\$f_{vn}$ IN E_n.
LetClause ::= LET $\$l_{v1}$:= E_1 , ..., $\$l_{vn}$:= E_n.
WhereClause ::= WHERE $\varphi(E_1, ..., E_n)$.
PrefClause ::= Pref φ**(**E_1**, ...,** E_n**)**
ReturnClause ::= RETURN $E_1 ... E_n$.
E_i ::= FLWR | XPATH.

Fig. 5. Syntax of XQuery preference expressions

For simplicity, we make other assumptions about our grammar as follows.

– The atomic predicates allowed in the boolean formula φ are the integrated relop predicates ($<, \leq, >, \geq, =, \neq$) or the integrated predicate $empty(FLWR)$. The operand of a relop predicate can be one of the following: constant c, XPath expression XPE, or $agg(XPE)$, where agg is one of the integrated aggregate functions, namely, avg, $count$, min, max, or sum [4].
– The XPath expressions used are expressions extended by the operator "!" except for the **Pref** clause.

3.2 Expression of Preferences on Values and Structure

XPath language is a component of XQuery and is used to extract nodes from the XML document. This language therefore allows you to browse the structure of the document. We use the notation "!" (a unary operator) introduced by Tchoupé et al. [13] for expressing preferences in an XPath (sub)path. For example, in the path $Q_1 = /a/b!/c$, the subpaths $/a$ and $/c$ represent the *constraints*, while $/b$ represents a *preference*. Q_1 is interpreted as a request returning all occurrences c_i of the node c such that the path from the root to c_i must have an occurrence of a and possibly an occurrence of b. The occurrences c_i candidates to be included in the solution are the sub-trees of the form $/a_i/b_i/c_i$ or $/a_i/c_i$. For simplification purposes, we assume that operator "!" can only be applied on a single node and not on an XPath subpath for example $(a/b/c)!$. In other words, this operator does not take into account parentheses.

We introduce a new clause in the XQuery language: the **Pref** clause. A Boolean clause to define preferences (wishes) on results. Just like the clause *Where*, *Pref* will define the constraints on the values but with the only difference that, it will not accept an embedded XQuery expression.

In this way, requests for *XQuery preference* language are composed of two parts: one to express mandatory constraints (the *Where* clause), and the other to express preferences or wishes (the *Pref* clause). The language *XQuery preference* therefore follows the bipolar model. Thus, the answers satisfying constraints and wishes are returned in priority to the user. But if such answers do not exist then results that satisfy only the constraints are delivered. The example below shows an example of an XQuery request with preferences on values and structure.

Example 1. ***XQuery query with preference on values and structure:*** Let's consider a query in witch, a user is looking for a candidate with Java skills and at least a diploma level above 3. But would like this person to have at least two years of experience, and skills with the framework Sping-boot would be a plus. The user also wants the returnees to have filled in the information on their profiles.

In this example, the condition on Java skills and diploma level above 3 is a constraint while the condition on the number of years of experience and Spring-Boot skills is a wish whose satisfaction increase relevance of the associated result.

The Fig. 6 illustrates how this request can be expressed in the language *XQuery preference*.

For this request, people with only Java skills and a diploma level above 3 represent potential responses but are dominated by candidates who have in addition, an experience of more than 2 years and Spring-Boot skills. However, if no answer respects all the constraints of the **Pref** clause, only those that best integrate the user's preferences are returned: They are called *undominated solutions*. The operator "!" present in the XPath expression $\$p/profile!//name$ specifies that the *person* elements with a *profile* sub-element are preferred. For now, we have imposed certain restrictions on *XQuery preference* expressions. For example, we require that XPath paths defined in the clause *Pref* do not include a preference element for the simple reason that all these paths are already preferential since they appear in the pref clause.

```
For $p in document('candidate.xml')//persons
Where $p/skills/Java > 1 AND $p/diploma > 3
Pref $p/experience > 2 AND $p/skills/Spring-Boot > 1
Return <result> {$p/profile!//name} </result>
```

Fig. 6. An example of XQuery request with preference on structure and values

In the Sect. 2.4 we presented a model for XPath and XQuery queries. Indeed, these model considerably reduce the complexity of evaluating requests. In the following section, in the same vein we propose a query model for *XQuery preference* requests based on the GTP model of Chen Zhimin et al.

3.3 GTP Request with Preference

As previously stated, GTP model is used to represent exact XQuery queries. We adapt this model as follows for the representation of XQuery requests with preferences:

- to the GTP tree we add a new type of arc called *preference arcs*. A preference arc can be of the type AD (Ancestral-descending), PC (Parent-child) or PP (Preference Path).
- the Boolean formula F verifying the predicates applicable to nodes is divided into two groups of formulas S and P:
  - group S defines the mandatory constraints.
  - and group P the preferential constraints (wishes).
- A set of nodes forms a group if they are connected to each other by non-optional or preferential relationships.

The Fig. 7 illustrates an example of a GTP request with preference (see Fig. 8), distinguishing between the different types of arcs to take preferences into account:

- The arcs *preferences of type ancestor-descendant (APN)*. This type of arc allows to represent a structural preference between an element and its descendant for example $a//b$!
- The arcs *of type parent-child preferences (PPN)*. For the representation of a structural preference between an element and its child for example a/b!
- Finally, arcs of the type *preference path (PP)* for the representation of preferences on values. This type of arc will be generated for XPath expressions found in the Pref clause.

The formula group S defines constraints on mandatory nodes while the group P defines constraints on preference nodes. All the nodes of the tree are part of the same group because they are all connected to each other by optional arcs or preference. More formally, a GTP with preference is defined as follows:

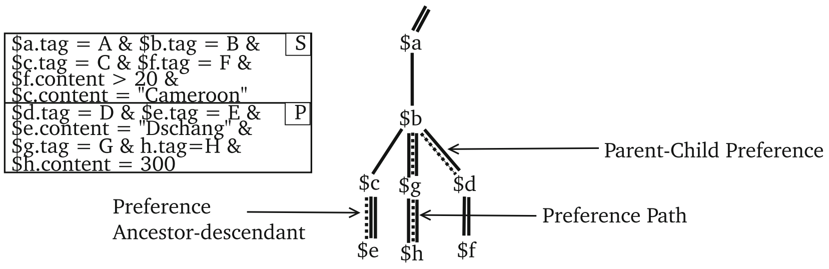

Fig. 7. Example of representation of a GTP request with preference

Definition 3. A generalized tree pattern with preference is a couple $G = (T, F)$ where T is a tree and F a Boolean formula such as:

- Each node in the tree T is labeled with a different variable and has a group number.
- to each arc of T is associated a tag e, where $e \in \{PC, AD, OP, PN, APN, PP\}$
  - **PC, AD:** specify the parent-child and ancestor-descendant axis
  - **PN, APN:** specify respectively the parent-child preference and ancestor-descendant preference axes.
  - **OP, PP:** specify the optional axes and preference path respectively.
- F is a group of formula S and P specifying respectively a Boolean combination of predicates applicable to non-preferential nodes and to *preferential nodes.*

Definition 4. A node is said to be a preference if it is located at target of a preference arc.

```
FOR $p IN document("doc.xml")//A, $b IN $a/B
WHERE $b/D!/F >= 20 AND $b/C = "Cameroon"
PREF $b/G/H = 300 AND $b/C//E = "Dschang"
RETURN <result> {$b/C//E!}</result>
```

Fig. 8. XQuery preference request corresponding to the GTP with preference of the Fig. 7

The model thus presented integrates the different elements for the representation of preferences in the GTPs. The GTP with preference is obtained by a *parsing* of the query *XQuery preference* as in [4]. The process for parsing a request is as follows: We have a global analysis environment ENV to manage information collected from the request analysis i.e The association variable name-nodes. The query is parsed clause by clause. We also use a help function $buildTPQ(xp)$ where xp is an extended XPath expression[2], which constructs part of GTP with preference from xp. This function is described as follows:

- Each time that xp starts with the built-in document function, a new GTP is added to ENV.
- If xp begins with a variable, the node associated with this variable is searched for in the tree and the new part resulting from xp starts from it.
- When xp contains the operator "!", The arcs associated with the nodes considered are of PN or APN type, depending on the axis (/ or // respectively) and the constraints linked to these nodes are placed in group P of the GTP formula.

During the analysis of a Pref clause, the arcs created are of type PP and the constraints linked to these nodes are placed in group P.

4 Conclusion

In this paper, we have made two major proposals: *XQuery preference* an extension of the XQuery language for the expression of requests with *preferences* which can relate to both the structure and the content; the GTP model with preference an extended version of the GTP model for the representation of XQuery preference queries; The examples given show that the proposed language is quite expressive and thus makes it possible to meet a variety of needs. We have carried out a study for the evaluation of GTPs requests with preference by an adaptation of the algorithm $Twig^2Stack$ [3] for the evaluation of exact GTPs requests. Due to the limitation of the number of pages in this paper, we could not describe it here, but it is accessible via the open repository on Github[3]. The results obtained

[2] An extended XPath expression is an expression containing the notation $ and the operator "!".

[3] https://github.com/patrikken/PrefTwig2Stack.

are very interesting and show that we can extend the existing algorithms with a very small loss in performance but an improvement in terms of quality of the results delivered to users.

Furthermore, this work serves as the foundation for a much larger body of work, which could be the subject of further study. Among which, integration and experimentation of XQuery preference in the presented open source sources implementations, an in-depth study of the nested XQuery preference queries and the search for the best algorithms for evaluating queries with preferences.

References

1. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: XPref: a preference language for P3P. Comput. Netw. **48**, 809–827 (2005)
2. Bosc, P., Pivert, O.: SQLf query functionality on top of a regular relational database management system. In: Pons, O., Vila, M.A., Kacprzyk, J. (eds.) Knowledge Management in Fuzzy Databases. Studies in Fuzziness and Soft Computing, vol. 39, pp. 171–190. Physica-Verlag HD, Heidelberg (2000). https://doi.org/10.1007/978-3-7908-1865-9_11
3. Chen, S., Li, H., Tatemura, J., Hsiung, W., Agrawal, D., Candan, K.: Twig$_2$stack: bottom-up processing of generalized-tree-pattern queries over XML documents. In: Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB 2006), pp. 283–294 (2006)
4. Chen, Z., Jagadish, H.V., Lakshmanan, L.V.S., Paparizos, S.: From tree patterns to generalized tree patterns: on efficient evaluation of XQuery. In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2003), vol. 29, pp. 237–248. VLDB Endowment (2003). http://dl.acm.org/citation.cfm?id=1315451.1315473
5. Chomicki, J.: Preference formulas in relational queries. ACM Trans. Database Syst. **28**(4), 427–466 (2003). https://doi.org/10.1145/958942.958946
6. Consortium, W.: XML path language (XPath) 2.0 (2006). http://www.w3.org/TR/XPath20
7. Consortium, W.: XQuery 1.0: an XML query language, October 2004. (w3C Working Draft)
8. Database, T.U.K., Group's, I.S.: Implements the XQuery update facility; full-text support (since 2006). http://www.basex.org
9. Kießling, W., Hafenrichter, B., Fischer, S., Holland, S.: Preference XPATH: a query language for e-commerce. In: Buhl, H.U., Huther, A., Reitwiesner, B. (eds.) Information Age Economy, pp. 427–440. Physica-Verlag HD, Heidelberg (2001). https://doi.org/10.1007/978-3-642-57547-1_37
10. Kießling, W., Endres, M., Wenzel, F.: The preference SQL system - an overview. IEEE Data Eng. Bull. **34**, 11–18 (2011)
11. Mary Fernàndez, J.S.: Galax: an XQuery implementation, 1 October 2009. http://galax.sourceforge.net
12. Meier, W.: The high-performance native XML database engine (since 2000). http://exist-db.org
13. Tchoupe Tchendji, M., Nguefack, B.: XPath bipolar queries and evaluation. In: Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées, Special issue CARI 2016, vol. 27, October 2017