Making XPath Reach for the Web-Wide Links

Lule Ahmedi*
Institut für Informatik, Universität Freiburg
Georges-Köhler-Allee, Geb. 51
79110 Freiburg, Germany
ahmedi@informatik.uni-freiburg.de

ABSTRACT

Opting for link semantics in XML is almost like hyperlinks for HTML documents. XLink describes a standard way to add hyperlinks to an XML document. The current XPath technologies are restricted to follow ID/IDREF(S) links for intra-document navigation only. We investigate the Lightweight Directory Access Protocol (LDAP) that offers a rich collection of primitives to express links among distributed data collections in the network, and facilities to follow links when searching. In virtue of querying the underlying LDAP model by referrals, we developed an extended XPath processor that is capable of addressing links of any type (IDREF(S), XLink) embedded in XML data on the Web. Links may be inter-document, or even traverse across different local or remote servers. We describe the internal LDAP data representation and query model used by the processor for the storage and querying in XPath of XML documents based on links, and provide examples to illustrate them. We complement the discussion with experimental analyses that prove the efficiency of our query evaluation techniques. The latter confirms the relevance of our approach for applications that need to interact with interlinked XML document networks in an XPath-like fashion.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*data directory*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*information networks*

General Terms

Management, Documentation

Keywords

XML hyperlinks, XPath, LDAP, query processing

*The work of this author is partially supported by the Deutsche Forschungsgemeinschaft (DFG), Aktenzeichen La 598/4-1.

SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA

1. INTRODUCTION

XML as a de facto standard for information interchange on the Web is supposed to support links between related resources. Opting for link semantics in XML is almost like hyperlinks for HTML documents. The W3C XML Linking Language (XLink) [2] describes a standard way to add hyperlinks to an XML document. It uses XML syntax from the xlink: namespace to create more sophisticated links as the simple hyperlinks of today's HTML, i.e. database-oriented rather than browser-centric links. From a database perspective, the Web can be seen as a distributed database where links pointing from one object to another can carry out sophisticated tasks for users when querying. As regards the aspect of handling links when querying, the Web technology presents yet an evolving infrastructure. So far, the interaction between linked XML data and XPath-based technologies (XPath [8], XQuery [4], XSLT [1], etc.) is limited to following the ID/IDREF(S) links for intra-document navigation only [8]. A number of XLink systems dedicated to browsing, not to querying, are in works [20].

At the same time, since the conception of the LDAP protocol version 3 in 1997 [21], the use of lightweight directories to store a variety of information has been steadily gaining momentum. Many universities and research centers use LDAP servers [11] to manage data about their members, organizations, networks, etc., and several companies offer LDAP support even in their Internet browsers.

The similarities between LDAP and the XML tree model, e.g. DOM [19], has led to the novel idea of processing XML data in LDAP [13]. The referral mechanism provided by LDAP and the linking concept of XLink present another set of commonalities between LDAP and the XML family of standards. This prompted us to consider deploying LDAP's facility of following referrals when querying interlinked XML documents in XPath. In particular, the flexible behaviour of LDAP towards networked data promises an efficient query processing of links that may relate remote data regardless of the level of their distribution.

Hence, in virtue of querying the underlying LDAP model by referrals instead of using the XPath model which, as noted, suppresses XLink elements, we developed an extended XPath processor capable to address link annotations of any type (IDREF(S), XLink) in XML data. That is, links may be inter-document, or even traverse across different local or remote servers in the network.

Section 2 gives an overview of the notions from the XML family of standards and LDAP used throughout the paper. Section 3 provides a simple motivating example. In Sect. 4,

the architecture of our extended XPath processor is outlined, leaving the details about its internal data model of storing XML documents and their links for Sect. 5. The query evaluation strategy based on links is elaborated in Sect. 6, for the case of chains of links as well. Section 7 presents the experimental data. Finally, Sect. 8 considers related work and then concludes.

2. PRELIMINARIES

2.1 XML-Related Notions

Example 1 *Consider the following excerpt of XML data and its DTD taken from the* MONDIAL *database [14] for illustrations. The example presents a DTD for* country *and* city *elements and an according instance.*

```
<!ELEMENT countries (country+)>
<!ELEMENT country (name,population?,area,city+,...)>
<!ATTLIST country   car_code ID #REQUIRED
                    capital IDREF #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT population (#PCDATA)>
<!ATTLIST population year CDATA #IMPLIED>
<!ELEMENT area (#PCDATA)>
<!ELEMENT city (name,population*)>
<!ATTLIST city id ID #REQUIRED>

<countries>..
  <country car_code="D" capital="cty-D-Berlin">
    <name>Germany</name>
    <area>356910</area>
    <city id="cty-D-Berlin">
      <name>Berlin</name>
      <population year="95">3472009</population></city>
    <city id="cty-D-Hamburg">
      <name>Hamburg</name>
      <population year="95">1705872</population></city>..
  </country></countries>
```

□

XPath [8] is a W3C recommendation for addressing node sets of an XML document. Most XML query languages (XQuery [4], XSLT [1], etc.) are built on top of it. The core XPath includes facilities for navigating through the linked structure of *an XML document*. It provides [8] the id() function (or fn:id() in [12]) for resolving intra-document references declared by ID/IDREF(S) attributes.

Example 2 *The following XPath query is used throughout the paper for illustrating dereferencing*

```
id(//countries/country[@car_code = "D"]/@capital).
```

It selects the capital of the country whose car code is "D". □

XLink [2] defines the XML Linking Language, which allows XLink elements to be inserted into XML documents in order to add and describe rich link semantics between resources. An XLink element specifies the resource fragment to be located in the XML document through the href attribute. The value of href must be a URI reference as defined in [6], i.e. uri-reference = url#fragment where url = scheme://host:port/path. The fragment identifier is specified by using XPointer [3] which, instead of pointing to entire resources on the Web, provides specialized extended XPath constructs for addressing a node or selection within resources. To our conditions, fragment is an XPath expression, i.e. uri-reference = url#xpath-expr. Apart from href, XLink's namespace provides some more global attributes for use on elements declared as links: type (which can be simple, locator, extended, or arc), from, to, actuate, show, etc. Later, Example 7 illustrates how XLink elements are declared/specified in a DTD/XML document.

2.2 LDAP Overview

An LDAP [21,10] server can be viewed as a semistructured database system, specialized for the operations performed on the Internet, that is, simple and fast remote read operations that occur much more frequently than (local) writes.

The LDAP **data model** consists of two parts: The *directory schema* defines a finite set of classes, their organization in the hierarchy, together with their content, attributes and datatypes. The *directory instance* contains a set of entries organized in a forest, where each entry has a non-empty set of multi-valued attribute-value pairs, and belongs to at least one class (the objectClass attribute).

The **naming model** defines how to arrange entries into an instance hierarchy based on their names. Giving a unique name to any entry in the hierarchy allows to refer to any entry unambiguously. The unique name of an LDAP entry is its *distinguished name* (dn). It is formed by enumerating all the individual names of the parent entries on the path to the root of the hierarchy. E.g., reading the entry's dn cn=capital,cn=country,cn=mondial.xml, one can trace from the entry capital itself through the country back to the root cn=mondial.xml of the tree.

The **functional model** determines the search operations to act in a similar way a database query does. It is defined as a combination of the following components: The *base* denotes the dn of the entry in the hierarchy where the search should start. The *scope* can be base, if the search is to be restricted to just the first node, onelevel, if only the first level of nodes is to be searched, or subtree, if all nodes under the base should be considered by the filter expression. The *filter expression* is a boolean combination of atomic filters of the form $(a\ op\ v)$, where a is an attribute name, op is a comparison operator, and v is an attribute value.

Example 3 *The LDAP search request of the form*

```
Q = (cn=countries,cn=mondial.xml ? subtree ?
     (&(name=provinces)(linkType=ID*)) )
```

*returns all nodes from the subtree (*scope*) rooted at the node whose (*base*) dn is cn=countries,cn=mondial.xml that satisfy the last argument, i.e. the given* filter. □

2.2.1 Referral Mechanism.

There are two kinds of referral mechanisms in LDAP we make use of in our framework, *aliases* and *smart referrals*.

An *alias* is an entry that contains the dn of another entry in the same server. If the search result contains an alias, that entry is *replaced* by the entry the alias points to through its dn, i.e., by its aliased entry.

Example 4 *Consider the following LDAP entry which is an alias (objectClass) and represents the XML attribute @capital of Germany provided in Example 1*

```
                dn: cn=capital,cn=countryGermany,
                    cn=countries,cn=mondial.xml
              name: capital
             value: Berlin
         objectClass: alias
aliasedObjectName: cn=cityBerlin,cn=countryGermany,
                    cn=countries,cn=mondial.xml
```

It states among others that, iff appearing in the answer set, it is to be replaced by its aliased entry, i.e., by the entry whose dn is (see aliasedObjectName)

cn=cityBerlin,cn=countryGermany,cn=countries,cn=mondial.xml. □

Example 5 *The query of Example 2 in LDAP*

Q = (cn=capital,cn=countryGermany,cn=countries,cn=mondial.xml
? subtree ?)

addresses the alias capital of Germany shown above. Hence, the aliased entry

dn: cn=cityBerlin,cn=countryGermany,cn=countries,
cn=mondial.xml
name: city
:

that is pointed to by the attribute aliasedObjectName in capital is returned. □

A *smart referral* is a generalization of an alias. It is an entry in the directory instance that refers to one or more entries in the same *or another* LDAP server. References (ref) are LDAP uniform resource locators [9], i.e. ldap-url = ldap-host/ldap-query, where ldap-host = ldap://host : port, and ldap-query is an LDAP search request of the form base??scope?filter to be sent to the contacted host.

Example 6 *The following reference points to nodes that satisfy a filter indep_date > 1990 in a tree rooted at the node cn=mondial.xml and located at ServerA.LGAccess.org*

ldap://ServerA.LGAccess.org/cn=mondial.xml??sub?
(indep_date>1990). □

Regarding querying, smart referrals are handled in the same way as aliases.

In XML terms, aliases are local, similar to the ID/IDREF mechanism, whereas smart referrals are global and resemble to the XLink/XPointer linking mechanism. The use of aliases and smart referrals for the the purpose of querying linked XML documents *in XPath* is investigated in the subsequent sections.

3. SIMPLE MOTIVATION

Example 7 *Consider Example 1 in its "distributed" version, extracted from the distributed* MONDIAL *database [14]. Here, data about countries and cities are stored in separate files, and links among them as provided by XLink are added.*

```
<!– from the file "mondial-countries.dtd" –>
<!ELEMENT countries (country+)>

<!ELEMENT country (name,capital,cities,...)>
<!ATTLIST country car_code ID #IMPLIED>

<!ELEMENT name (#PCDATA)>

<!– "capital" and "cities" are defined as XLink elements here –>
<!ELEMENT capital EMPTY>
<!ATTLIST capital xlink:type (simple|extended|locator|arc)
                    #FIXED "simple"
                  xlink:href CDATA #IMPLIED>

<!ELEMENT cities EMPTY>
<!ATTLIST cities  xlink:type (simple|extended|locator|arc)
                    #FIXED "simple"
                  xlink:href CDATA #IMPLIED>

<!– from the file "mondial-cities.dtd" –>
<!ELEMENT cities (city+)>
<!ELEMENT city (name,population*)>
<!ELEMENT name (#PCDATA)>

<!ELEMENT population (#PCDATA)>
<!ATTLIST population year CDATA #IMPLIED>
```

```
<!– from the file "mondial-countries.xml" –>
<countries>..
    <country car_code="D">
        <name>Germany</name>
        <capital href="cities-D.xml#//city[name='Berlin']" />
        <cities href="cities-D.xml#//city"/></country>
</countries>

<!– from the file "cities-D.xml" –>
<cities>..
    <city>
        <name>Berlin</name>
        <population year="95">3472009</population></city>
    <city>
        <name>Hamburg</name>
        <population year="95">1705872</population></city>
</cities>
```
□

There is no syntactical construct in XPath to express the query of Example 2 in a distributed version. Asking to query distributed XML data by links goes beyond the facilities of the current XPath query concept.

4. ARCHITECTURE AT A GLANCE

An LDAP server [17, 5] is able to efficiently evaluate distributed LDAP queries in the network by using referrals. We continued the work done by [13] on processing XML data in LDAP and empowered the existing modules there, *xml2ldap* and *xpath2ldap*, to model XML links and handle them in XPath (Fig. 1). We refer with *xml2ldap-link* and *xpath2ldap-link* to the extended modules in *links2ldap*.

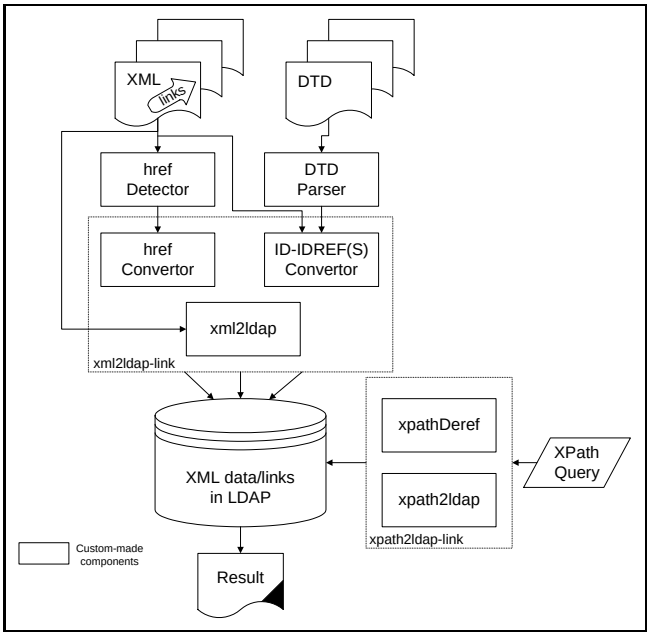


Figure 1: The links2ldap architecture

XML data and their DTDs are incorporated into the system such that attributes which identify elements as links, i.e. IDREF(S) and href, are detected (*DTD Parser*, *href Detector*). Then, the conversion modules synthesize LDAP instances for the given XML data (*xml2ldap*) and their links (*ID/IDREF(S) Convertor* and *href Convertor*). Now, each time an XPath query is received, if it contains a dereferencing directive, the *xpathDeref* module is activated to carry out the link-related processing, while the rest is handled as usual by *xpath2ldap*. The answer is generated by retrieving the data pointed to by links, if any.

5. MODELING XML LINKS IN LDAP

In the context of modeling XML links in LDAP, we review the structure of the XMLAttribute class (and its superclass) from a generic LDAP schema that was defined in [13] to model XML, as well as the content of the alias and referral classes of the core LDAP schema that will be referred to in the following sections. The new attribute linkType serves to distinguish between different types of link nodes.

```
XMLNode OBJECT-CLASS ::= {
  SUBCLASS OF {top}
  MUST CONTAIN {objectClass,name}
  TYPE objectClass OBJECT-CLASS
  TYPE name STRING }

XMLAttribute OBJECT-CLASS ::= {
  SUBCLASS OF {XMLNode}
  MUST CONTAIN {value}
   MAY CONTAIN {linkType}
  TYPE value STRING
  TYPE linkType {IDREF,IDREFS,XLink} }
```

Following the XML2LDAP translation algorithm given in [13], the attribute @capital of Example 1 is treated as any other attribute in XML (the IDREF declaration is ignored), and its LDAP homologue is created as a simple node capital of type XMLAttribute with the value cty-D-Berlin and arranged directly beneath (a child node of) the country node of Germany in the hierarchy:

```
          dn: cn=capital,cn=countryGermany,cn=countries,
              cn=mondial.xml
objectClass: XMLAttribute
       name: capital
      value: cty-D-Berlin
```

We treat next, for each type of link separately, the way we extend the LDAP model to capture link semantics involved in XML documents. The entire process is summarized in the XMLink-to-LDAP algorithm that is sketched in Fig. 2.

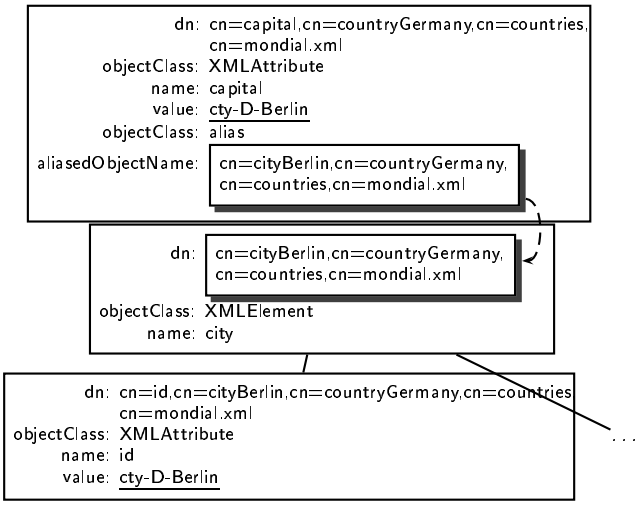
5.1 IDREF attributes as internal single-target links

The property of IDREF being an intra-document and single-target link is similar to the modeling capabilities of *aliases* in LDAP. Therefore, in contrast to other attributes in XML, we treat IDREFs specially and assign them an alias semantics during the XML to LDAP transformation. This means (Fig. 2), an LDAP node of types (classes) XMLAttribute and alias is created for each IDREF, sharing attributes of both of them: (1) the attribute objectClass which is set to XMLAttribute, the linkType=IDREF attribute, as well as attributes name=*name* and value=*value*, where *name* and *value* denote the name, respectively the value of the IDREF attribute in the XML document, and (2) the attribute objectClass which is set to alias, as well as the aliasedObjectName attribute.

```
alias OBJECT-CLASS ::= {
   MUST CONTAIN {aliasedObjectName}
   TYPE aliasedObjectName DN SINGLE-VALUE }
```

The DN given to aliasedObjectName is the dn value of the XMLElement node that is the parent of the node which represents the ID attribute where the given IDREF points to, i.e. the parent of the LDAP ID node with the same value for the value attribute as the IDREF node. This value is found by the GetELEMENTbyID function shown in Fig. 3. Note that instantiating an entry as alias (objectClass: alias) in LDAP is similar to declaring an attribute as IDREF in a DTD.

Example 8 (IDREF Attribute in LDAP) *The IDREF attribute* capital *of Germany and the ID attribute* id *of Berlin of Example 1 including link looks like:*



□

5.2 IDREFS attributes as internal multi-target links

IDREFS attributes can point to multiple elements in the document as they are allowed to enumerate multiple ID values in a token list, e.g. $ID_1 \sqcup ID_2 \sqcup \cdots \sqcup ID_k$, (k>1). This multiple-end referencing goes beyond the referencing scope of aliases; recall from Sect. 2.2 that the LDAP naming model is restricted to allow at most one aliasedObjectName per alias. In the naming model of LDAP, *smart referrals* provide the functionality for pointing to one or more entries in

Algorithm (XMLink2LDAP(X, D))
Let X be an XML document, and D its DTD
While there is an element $elem$ from X at the input
 /* XML2LDAP creates a new node e of type XMLElement */
 e = XMLElement($elem$)
 for each attribute $attr(elem, name, value)$ of $elem$
  /* XML2LDAP creates an XMLAttribute node a, child of e */
  a = XMLAttribute($e, name, value$)
  /* check the type of $attr$ in D and mark it resp. in LDAP */
  if isIDREF($D, elem, attr$)
   $a.linkType = IDREF$
  else if isIDREFS($D, elem, attr$)
   $a.linkType = IDREFS$
  else if isXLinkHREF($attr$) // if it is an href attribute
   $a.linkType = XLink$
   /* handle the XLink a nodes (those with the href attr.) */
   $ldap - uri$ = XML_URI2LDAP($uri - reference$)
   /* make a an instance of referral */
   a = LDAPReferral($a, ldap - uri$)
/* Handle the IDREF and the IDREFS a nodes */
While there is $a(linkType, IDREF)$ or $a(linkType, IDREFS)$
 $a.value$ is a string of the form $ID_1 \sqcup ID_2 \sqcup \cdots \sqcup ID_i \sqcup \cdots \sqcup ID_k$
 $i = 1$
 Let X_L denote the LDAP representation of X so far
 if ($k == 1$) { // the (IDREF or IDREFS) check
  e_{ID_1} = GetELEMENTbyID(X_L, D, ID_1)
  /* make a an instance of alias */
  a = LDAPAlias($a, dn(e_{ID_1})$)
 } else do {
  e_{ID_i} = GetELEMENTbyID(X_L, D, ID_i)
  /* make a an instance of referral */
  a = LDAPReferral($a, dn(e_{ID_i})$)
  i++
 } while ($i <= k$)

□

Figure 2: The XMLink-to-LDAP algorithm

Algorithm (GetELEMENTbyID(X_L, D, ID))
/* evaluate an LDAP query that filters X_L */
a_{ID} = LDAP(X_L, "(value = ID)")
/* check if a_{ID} (as $attr_{ID}$ in XML) is declared as ID attr. in D */
if isID($D, NULL, attr_{ID}$)
return e_{ID} = Parent(a_{ID}) // retrieve its parent node □

Figure 3: The GetELEMENTbyID function

the LDAP hierarchy. Thus (see the pseudocode in Fig. 2), for every IDREFS attribute we create an entry (1) of type XMLAttribute by adding the attribute-value pairs objectClass=XMLAttribute and linkType=IDREFS, and attributes name and value, and (2) of type referral, inserting objectClass=referral, and as much ref attributes as is the number (k) of space-delimited ID tokens listed in the IDREFS attribute. For each ID value ID_i (i=1,k), the corresponding ref attribute is assigned the dn of the XMLElement node that is the parent of the LDAP node representing the ID attribute whose value is equal to ID_i (GetELEMENTbyID).

```
referral OBJECT-CLASS ::= {
   MUST CONTAIN {ref}
   TYPE ref DN MULTIPLE-VALUE }
```

5.3 href links as external multi-target links

Instead of modeling each type of XLink link separately in conjunction with a set of its dedicated attributes, we take a rather simple approach and model them all in a unique form in LDAP: We model the common href attribute(s).

Different from ID/IDREFS links that are intra-document (local), href attributes may define external links, i.e. links to nodes that are located at different local or remote servers in the network, as illustrated in Example 7. *Smart referrals* used to model IDREF(S) attributes, due to their ability to refer to entries in another LDAP server from the one the actual LDAP resource resides, they are suitable for modeling href attributes too. The target XML entries are, hereby, also supossed to have been stored in an LDAP system. For each href attribute (Fig. 2), an entry of types XMLAttribute and referral is created. The ref attribute value for this entry is generated by the XML_URI2LDAP function (cf. Fig. 2) which takes the $uri - reference$ value of the href attribute as argument, and returns the corresponding $ldap - uri$ at the output.

LDAP URL values of the form $ldap - host/ldap - query$ as described in Sect. 2.2 would suffice to model href's $uri - reference$ values. But, the current LDAP systems support only a restricted form of URIs, namely $ldap - host/dn$ (hereafter $ldap - url$), akin to URL references in HTML hyperlinks [7]. That is why XML_URI2LDAP has been adapted to return not an LDAP URL as introduced in Sect. 2.2, but an $ldap - uri$ that can be handled in our system. Therefore, we distinguish between two modeling cases in XML_URI2LDAP.

5.3.1 Supported LDAP URLs

XLink references of the form url#id('...') are mapped into $ldap - url = ldap - host/dn$, which are (ref attributes) supported in LDAP.

Example 9 *A simple XLink element* capital *(similar to* capital *of Example 7) and the corresponding LDAP model for its* href *attribute are shown below.*

```
<capital href="http://serverB/fileB#id('Berlin')">
                                .. </capital>
```

```
          dn: cn=capitalHREF,cn=capital,...,cn=fileA
 objectClass: XMLAttribute
        name: href
       value: http://serverB/fileB#id('Berlin')
 objectClass: referral
         ref: ldap://serverB/dn(Berlin)
```

where dn(Berlin) *is the node in the LDAP database which corresponds to the XML document* fileB *at the remote host machine* serverB *and represents the XML element node whose ID attribute is equal to* 'Berlin'. □

5.3.2 Combining LDAP URLs with XPath

XLink expressions of the form $url\#xpath{-}expr$ are mapped into extended $ldap - url$ expressions $ldap - uri = ldap - url\#xpath - expr$:

- The url part $scheme://host:port/path$ is mapped into ldap://$host:port/dn$; for $path$, a corresponding LDAP DN is generated, e.g., /mondial/cities.xml is mapped into cn=cities.xml,cn=mondial.
- The fragment identifier $xpath - expr$ is left unchanged to be interpreted by the *xpath2ldap* module.

Example 10 *An XLink locator element and the corresponding LDAP representation for its* remote href *attribute are shown below.*

```
<capital xtype="locator" href=
 "http://serverB/fileB#/gs/country/capital[../@code='D']"
> .. </capital>
```

```
          dn: cn=capitalHREF,cn=capital,...,cn=fileA
 objectClass: XMLAttribute
        name: href
       value: http://serverB/fileB#/gs/country/
              capital[../@code='D']
 objectClass: referral
         ref: ldap://serverB/dn(fileB)#/gs/country/
              capital[../@code='D']
```
□

Remark. The from and to attributes within the XLink elements of type arc fall into the ID/IDREF(S) category of links, from both, modeling and querying point of view in our system, as well as the ID attributes of the locator XLink-s.

6. PROCESSING XML LINKS IN LDAP

Having constituted the model of representing XML links in LDAP, we are now ready to tackle the problem of evaluating those links in XPath.

6.1 Dereferencing ID/IDREF(S) links

For resolving this kind of links, we accommodate the concept for an explicit dereference expression that is already in use by XPath and enable the id() function there to operate with LDAP-encoded ID/IDREF(S) links, giving thereby the same result set as if applying the traditional XPath processor over the same set of data and links in XML.

The evaluation of the id function (upon LDAP data) is done by the XPathDEREF algorithm (Fig. 4) as follows:

1. Use XPath2LDAP to evaluate the XPath query in id.
2. For each node obtained in step 1, we call them *start nodes*, perform the LDAP dereference operation:
   - Follow the located alias/referral node (the start node) using the LDAPDeref function which includes contacting the required host server, scanning the LDAP-encoded XML document located there, and, selecting the referred nodes pointed to

by a given start node. This is achieved by issuing an LDAP query with the dn of the start node as base DN, the scope set to base, and with no filter expression (NULL).

- Incrementally store the result into the set of, we call them *end nodes*, until there is no more start node from step 1.

Example 11 *Consider again Example 8. Applying the* XPathDEREF *algorithm for the non-distributed query in Example 2*

id(//countries/country[@car_code = "D"]/@capital)

results in the following steps: (1) the expression within the id() *function is first evaluated, (2) its result, i.e. the LDAP node* capital *with value* cty-D-Berlin*, which we refer to as a* start node*, is taken by the* xpathDeref *module and dereferenced by searching for the alias object with base set to the dn value of* start node*, giving its aliased object, i.e. the node* city *for Berlin as a final result.* □

6.2 Dereferencing XLink links

For expressing dereferencing, again the id syntax is used to serve all types of XLink links in a common way, their href attributes, by simply putting the @href attribute as argument to id. The behavior attributes like xlink:show and xlink:actuate are dedicated to browsing, thus it makes no sense to consider them when evaluating the XPath queries. For addressing other XLink metadata attributes like role, arcrole, title, etc., the XPath expressions as for any other XML attribute should be used. What is meant by "dereferencing XLink elements in XPath"? The semantics as known from resolving IDREF(S) references apply.

Algorithm (XPathDEREF(Q_x))
Let Q_x = id(arg) be an XPath dereference query
Let $endNodes = \{\}$ be the result of dereferencing
/* Distinguish between two cases of the arg argument */
If arg is a string $ID_1 \_ ID_2 \_ \cdots \_ ID_i \_ \cdots \_ ID_k$
 /* select XMLElement nodes by the ID val. of their child nodes */
 for each $ID_i \in arg$ from $i = 1, k$
  e_{ID_i} = GetELEMENTbyID(X_L, D, ID_i)
  $e_{ID} = e_{ID} \cup e_{ID_i}$
 return e_{ID}
If arg is another XPath query expression Q_x'
 /* evaluate Q_x' and store the result nodes into $startNodes$ */
 $startNodes$ = XPath2LDAP(Q_x')
 /* Now, $startNodes$ is a set $\{sN_1, sN_2, \cdots, sN_i, \cdots, sN_k\}$ */
 while there is an alias/referral node $sN_i \in startNodes$
  /* dereference sN_i and add the result into $endNodes$ */
  $endNodes_i$ = LDAPDeref(sN_i, base, NULL)
  if (($sN_i.linkType ==$ IDREF)
  OR ($sN_i.linkType ==$ IDREFS))
   $endNodes = endNodes \cup endNodes_i$
  else // sN_i is an XLink node: $sN_i.linkType = XLink$
   /* retrieve all ref attributes of the current sN_i */
   retrieve $sN_i.ref_i$ // it's a set $\{ref_{i1}, \cdots, ref_{ij}, \cdots, ref_{im}\}$
   /* $endNodes_i$ is a set $\{eN_{i1}, eN_{i2}, \cdots, eN_{ij}, \cdots, eN_{im}\}$ */
   for each (ref_{ij}, eN_{ij}) pair of sN_i from $j = 1, m$
   where $eN_{ij} = endNode(ref_{ij})$
    if $sN_i.ref_{ij}.xpath - expr$ // exists a URI fragment
     /* if ref is remote, establish the required connection */
     if $sN_i.ref_{ij}.ldap - host$ // contains a URI scheme
      connect to $host : port$ specified in $sN_i.ref_{ij}.ldap - host$
     /* initialize the query context to the current end node */
     $C_{ij} = eN_{ij}$
     /* finally, call XPath2LDAP to eval. the XPath expr. */
     $endNodes_{ij}$ =
      XPath2LDAP($C_{ij}, sN_j.ref_{ij}.xpath - expr$)
   $endNodes_i = endNodes_i \cup endNodes_{ij}$
  $endNodes = endNodes \cup endNodes_i$
 return $endNodes$

Figure 4: The XPathDEREF algorithm □

Definition 1 *Dereferencing an XLink element in XPath means, the argument of* id() *which is an XPath query is evaluated, and its value, that is an* uri-reference *(or several of them, one for each* @href *attribute) is further processed to end up with a set of nodes that are pointed to by this* uri-reference *expression.* □

In our LDAP system, the dereferencing includes (cf. Fig. 4):

Find XLink Nodes: Evaluate the XPath query expression that is argument of id against the LDAP database for the given XML data/links by using the XPath2LDAP algorithm [13]. The result is a set of smart referral nodes we refer to as *start nodes*. For each start node, the values of their ref attributes are retrieved. Given all smart referrals in our system have a ref attribute with a valid $uri - reference$ value, i.e. $ldap - url\#xpath - expr$, every returned start node is further handled by the *xpathDeref* module as follows.

Processing ldap-url: (i.e. ldap://*host:port/dn*) After the dereferencing option is activated, the LDAPDeref function dereferences the start node found in the previous step. If the retrieved ref attributes of that start node contain an $xpath - expr$ as a suffix, the corresponding referred nodes, we call them *end nodes*, serve as a new context when evaluating the given $xpath - expr$, as described next. Otherwise, these (end) nodes are actually the final result.

Processing xpath-expr: The XPath expression that comes after the "#" sign is ignored by the standard LDAP server. XPath2LDAP is invoked to evaluate it with a context node initialized to the node found in the previous step. A corresponding LDAP server is contacted previous to the XPath2LDAP invocation if the prefix of $xpath - expr$ – the $ldap - url$ expression – contains a host/port information telling that the LDAP database that is subject of querying is located elsewere from the current host, e.g. in a remote machine. If $xpath - expr$ is itself an XPath expression with id() function, the *xpathDeref* module is invoked (recursively) to interpret the located links instead of *xpath2ldap*.

Example 12 *Consider the distributed version of the query in Example 11*

id(/countries.xml//country[@car_code = "D"]/capital/@href).

If applied to the data of Example 7 in their LDAP representation, it proceeds in the following manner: (1) the XPath expression within id() *is first processed by the* xpath2ldap *module resulting in a smart referral node from the LDAP hierarchy whose* value *attribute is equal to the string* "cities-D.xml#//city[name='Berlin']" *which is the value of the* @href *attribute assigned to capital of Germany in the XML document of Example 7; retrieve the value of the* ref *attribute from the LDAP smart referral node; (2) the LDAP dereferencing is performed next to retrieve the nodes referred to by the node found in step 1; the result is the* dn:cn=cities-D.xml *node; (3) according to the retrieved value for the* ref *attribute* cn=cities-D.xml#//city[name='Berlin'], *a new query* //city[name='Berlin'] *with* dn:cn=cities-D.xml *found in step 2 as a new context is evaluated by* xpath2ldap*, yielding the* city *element of Berlin as a final result node.* □

Table 1: Run-time analyses for *xml2ldap-link*

XML Document	Entries	IDREF/IDREFS/href Attrs.	*xml2ldap*	*xml2ldap-link*
test	156	35/13/1	0.780s	3.246s
organizations	1068	0/0/133(simple)	10.965s	17.162s
countries	10057	0/0/1381(simple)	49.260s	170.080s
memberships	41002	7888(arc)/0/389(extn.,loct.)	368.716	455.334s

Table 2: Distribution overheads of *xpath2ldap-link*
Q = id(id(id(/countries.xml/countries/country[car_code='F']/capital/@href)/province/@href)/cities/city/@href)/name

Data Dissemination					(Result)
mirkwood:389	mirkwood:1025	no *ref*	1st *ref*	2nd *ref*	3rd *ref*
countries,cities,provinces	-	1.387s	1.543s (l)	1.642s (l)	2.023s (l)
countries	cities,provinces	1.247s	1.375s (r)	1.479s (l)	1.826s (l)
countries,cities	provinces	1.311s	1.348s (l)	1.620s (r)	2.021s (r)
countries,provinces	cities	1.272s	1.429s (r)	1.556s (r)	1.940s (r)

6.2.1 Chains of Links

An link element may itself point further to some other set of links in the same or another server. In that case, for each link among the set of result nodes, the dereferencing in our extended XPath processor *links2ldap* continues until no more link entries retain in the final answer set, given there is no cyclic reference. Moreover, *links2ldap* is able to handle arbitrary composed chains of links no matter of the types or order in which different types occur in the chain. For example, first the dereferencing of an IDREF attribute may proceed, then of an XLink, then again of an IDREFS link, etc.

7. EXPERIMENTAL RESULTS

We have carried out a series of experiments that show the capability of our system (1) to model distributed XML documents, and (2) to evaluate distributed XPath queries. As experimental data we used the distributed XML MONDIAL database [14] (about countries, cities, provinces, organizations and their members, etc.) whose total size is 110430 entities, 12899 of which are XLink elements.

7.1 Data Set Generation

Table 1 shows the performance results for (1) the *xml2ldap* module [13] that ignores all types of link semantics attached in XML data, and (2) our *xml2ldap-link* module that captures links of any type during the XML to LDAP transformation. It proves that our algorithm is *scalable* with respect to the number of links.

7.2 Distributed Queries

The aim of the following experiments on *xpath2ldap-link* is to detect the influence of the distribution level of links in the network to the query evaluation cost. Best scenario therefore is as follows (cf. Tab. 2):

- change the location of documents from mirkwood:389 to mirkwood:1025, and accordingly, the specification of links from local (l) to remote (r),
- choose a query with several ids, each of them traversing inter-document links (Fig. 5); it searches for cities in the same province as the capital of France.

1st *ref*(countries→cities):
  cities-F.xml#cities/city[name='Paris']
2nd*ref*(cities→provinces):
  provs-F.xml#provinces/province[name='Ile de France']
3rd *ref* (provinces→cities):
  cities-F.xml#cities-F.xml#cities/city[name='Paris'];
  cities-F.xml#cities-F.xml#cities/city[name='Boul. Billanc.']

Figure 5: Traverse files following href-s in the query

Table 2 shows that remote links does not add significant costs compared to local links. The gain in performance due to the partitioning of data (smaller search space) "balances" the loss in time needed to contact/scan remote data.

8. RELATED WORK AND CONCLUSION

The current XPath technology on top of which XQuery is built excludes linking semantics as defined by XLink and is restricted to follow ID/IDREF(S) links only [18]. A number of XLink systems dedicated to browsing, not to querying, are available [20]. Concerning research activities, the LinXIS project [16, 15] addresses the above deficiencies by taking a similar approach to ours. But, as opposed to LinXIS which defines an XML-to-XML transformation, we use a relatively simple LDAP model at the internal level that resembles that of XML links, and offers, by design, additional benefits derived by the *distributed* nature of the LDAP directory services.

8.1 Conclusion

We presented an approach of providing enhanced query facilities to the current XPath in relation with the link semantics embedded in XML data on the Web. The novelty thereby lies in the ability to handle links as defined by XLink apart from the ID/IDREF(S) links. Links may even traverse across different local or remote servers in our framework. The idea of deploying the rich LDAP repository of links therefore has proven to be feasible, referring to experiments. The increasing amount of data in a new form of the Web, which is semistructured and linked up to be easily accessible in a database-like way from anywhere in the network, is demanding for what our approach is designated. XML data are assumed to be "webbed" (contain hyperlinks) in order to coexist as an ultimate data format in a World Wide Web.

9. REFERENCES

[1] XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 1999. http://www.w3.org/TR/xslt.

[2] XML Linking Language (XLink) Version 1.0. W3C Recommendation, June 2001. http://www.w3.org/TR/xlink/.

[3] XML Pointer Language (XPointer). W3C Working Draft, August 2002. http://www.w3.org/TR/xptr/.

[4] XQuery 1.0: An XML Query Language. W3C Working Draft, November 2003. http://www.w3.org/XML/Query/.

[5] S. Amer-Yahia, D. Srivastava, and D. Suciu. Distributed query evaluation in LDAP directories. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(4), July/August 2003. IEEE Computer Society Press.

[6] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 2396: Uniform Resource Identifiers (URI): Generic syntax, Aug. 1998. Status: Draft Standard.

[7] T. Berners-Lee, L. Masinter, and M. McCahill. RFC 1738: Uniform resource locators (URL), Dec. 1994. Status: Proposed Standard.

[8] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999. http://www.w3c.org/TR/xpath.

[9] T. Howes and M. Smith. RFC 2255: The LDAP URL format. ftp://ftp.isi.edu/in-notes/rfc2255.txt, Dec. 1997. Status: Proposed Standard.

[10] T. A. Howes, M. C. Smith, and G. S. Good. *Understanding and deploying LDAP directory services.* Macmillan Technical Publishing, 1999.

[11] Innosoft International Inc. LDAP world implementation survey. http://www.innosoft.com/ldap_survey/lisurvey.html, February 1997.

[12] A. Malhorta, J. Melton, and N. Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Working Draft, November 2003. http://www.w3.org/TR/xpath-functions/.

[13] P. J. Marrón and G. Lausen. On processing XML in LDAP. In *Proceedings of the 27th Intl. Conference on Very Large Data Bases (VLDB), September, 2001, Roma, Italy*, pages 601–610, Los Altos, USA, 2001. Morgan Kaufmann Publishers.

[14] W. May. Mondial database. http://www.informatik.uni-freiburg.de/~may/Mondial, 2000.

[15] W. May. Linking the semantic web with existing sources. In *DEXA Workshop on Web Semantics (WebS)*, pages 93–97, Aix-en-Provence, France, September 2002. IEEE Computer Society Press.

[16] W. May. Querying linked XML document networks in the Web. In *The 11th Intl. World Wide Web Conference (WWW)*, Honolulu, USA, May 2002.

[17] OpenLDAP Group. OpenLDAP server. LDAP implementation available at http://www.openldap.org/.

[18] D. Veillard. Libxml - The Gnome/W3C XML library. http://www.xmlsoft.org.

[19] W3C DOM Working Group. Document object model specification. http://www.w3.org/DOM/, December 1997.

[20] W3C Implementation Chart. XML Linking implementations survey. http://www.w3.org/XML/2000/09/LinkingImplementations.html.

[21] M. Wahl, T. Howes, and S. Kille. RFC 2251: Lightweight Directory Access Protocol (v3), Dec. 1997.