A Logical, Transparent Model for Querying Linked XML Documents

Wolfgang May
Institut für Informatik*
Universität Göttingen, Germany
may@informatik.uni-goettingen.de

Dimitrio Malheiro
Institut für Informatik
Universität Freiburg, Germany
malheiro@informatik.uni-freiburg.de

Abstract: The W3C XML Linking Language (XLink) provides a powerful means for interlinking XML documents all over the world. While the effects when browsing through linked XML documents are well-defined, there is not yet any proposal how to handle interlinked XML documents that make use of the XLink language from the database point of view, i.e., considering the data model and navigation/querying aspects. From the database (and in general, querying) point of view, elements with linking semantics can be seen as *virtual* XML subtrees, i.e., *XML views*. Compared with classical databases, i.e., SQL and relational data, the situation of having links *inside* the data is new. We define a *logical, transparent* data model for linked documents. Queries are then formulated in standard XPath against the logical model. We propose additional attributes using the dbxlink (database-xlink) namespace for specifying the mapping from XLinks to the logical model.

1 Introduction

XML data instances are structured as trees, consisting of elements and attributes. The data is *self-describing*, i.e., each data item consists of a name and data contents (cf. the excerpt of the MONDIAL XML database [May01b] given in Figure 1 that will be used for illustrations throughout the paper).

XPath [XPa01] is the common language for addressing node sets in XML documents; we assume that the reader is familiar with XPath. It provides the base for several languages in the XML world, e.g., the query language XQuery, and for XLink. The core XML/XPath concept already provides unidirectional intra-document references by `ID/IDREF` attributes.

Example 1 *Consider the query "search all names (abbreviations) of organizations such that the headquarter city of the organization is also the capital of one of its member countries".*

The query is expressed in XPath as

```
//organization[@headq⇒city = members/@country⇒country/@capital⇒city]/@abbrev .
```

XML data is not required to be self-contained on an individual server, but may include *links* to XML data on other servers. With XLink, the targets of the links are given in XPath-like syntax within the XML data. When querying such distributed XML data, the query language must support following the links, and it must be clear what the *logical* schema of the accessible data as a whole is. This aspect, i.e., extracting a query expression from the intermediate answer that must be evaluated to continue the query evaluation, did not occur before in databases: SQL databases do not contain queries in their data fields.

*On leave from Universität Freiburg.

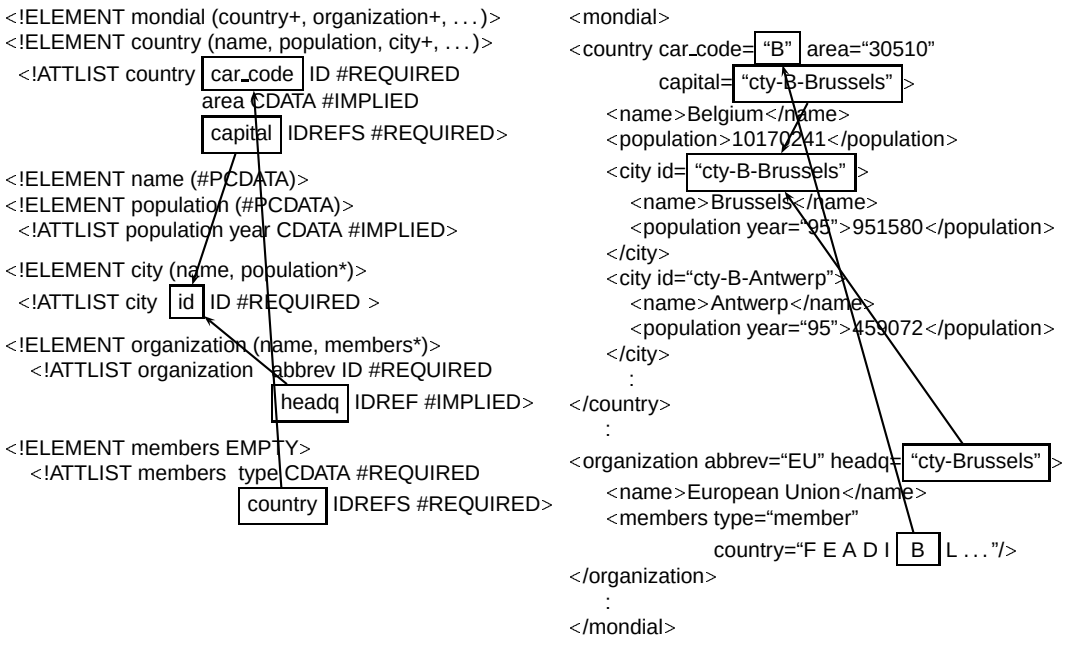


Figure 1: Excerpt of the MONDIAL XML database [May01b]

2 Linked XML Documents

XPointer and XLink specify how to *express* inter-document links in XML. XPointer [XPt00] is a specialized extension of XPath for selecting parts of XML documents – which are not necessarily sets of nodes. The XPointer concept combines the URL document addressing mechanism with an extension of the XPath mechanism for addressing fragments of the document. XPointer "hyperlink" addresses are of the form *url#ext-xpath-expr* . For this work, we restrict ourselves to standard XPath expressions as pointers, i.e., our XPointers are of the form *url#xpath-expr*. E.g., the following XPointer addresses the *country* element that has a *car_code* attribute with value "B" in the document with the url `www.ourserver.de/Mondial/mondial.xml`:

www.ourserver.de/Mondial/mondial.xml#descendant::country[@car_code="B"]

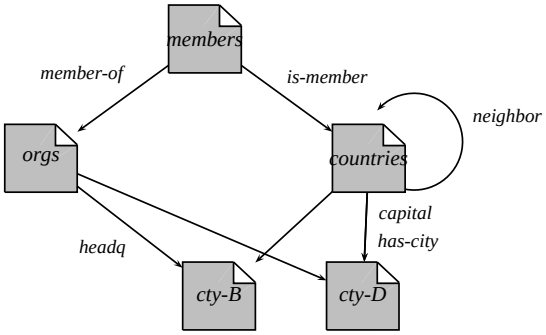
The XML linking semantics is specified in the *XML Linking Language (XLink)* [XLi00] by providing special tags in the xlink: namespace that tell an application that an element is equipped with link semantics. Arbitrary elements can be declared to have link semantics by equipping them with an xlink:type attribute and suitable additional attributes and subelements from the xlink: namespace. The xlink:type attribute selects between basic types of links: *simple links* extend the semantics known from <A href="...">. Their xlink:href attribute selects a target of the individual link instance, allowing for addressing nodes inside the target document by an XPointer. *Extended links* allow for grouping of targets, and also for specifying relationships *between* such targets. So far, XLink provides just a *syntactic representation* of references.

The additional xlink: attributes xlink:actuate and xlink:show specify the *behavior* of a link, i.e., its activating event and the triggered action. This behavior is tailored to the use of links when *browsing*, it does not cover the requirements of querying XML instances.

XLink does not provide any information about the *data model* or how queries are stated: there is not yet an official proposal (i) how to add link semantics to the actual data model, e.g., the DOM or the XML Query Data Model [XMQ01a], and (ii) how to *handle* links in queries and applications (which in part depends on the data model, but orthogonally, evaluation strategies have to be defined). In this paper, we focus on the *data modeling* aspect – which is then the base for formulating queries. We investigated the evaluation aspects of distributed queries in [May02b].

Example 2 *In the following, we illustrate the use of the different types of links by a "distributed" version of* MONDIAL *where all countries, all cities of a country, all organizations, and all memberships are stored in separate files.*

- *countries.xml (all countries)*
- *cities-car-code.xml (the cities for each country)*
- *organizations.xml (all organizations)*
- *memberships.xml (relates countries and organizations)*



Example 3 (Cities) *The cities-country.xml documents are very simple. Note that cities even do not have an ID; we assume that their name inside a country is unique. Below, the DTD and an excerpt of cities-B.xml is given:*

```
<!ELEMENT cities (city+)>
<!ELEMENT city (name, population*)>

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

```
<cities>
 <city> <name>Brussels</name>
    <population year="95">951580</population>
 </city>
 <city> <name>Antwerp</name>
    <population year="95">459072</population>
 </city>
    :
</cities>
```

Simple Links. A simple link is similar to the HTML <A href="..."> construct. It contains only a single pointer, but note that this pointer can address one or more elements.

Example 4 (Countries and Cities) *The country data is stored in countries.xml. A country has a capital and several cities. The capital is referenced by a simple link. The cities are also referenced by a simple link that addresses a set of nodes.*

```
<!ELEMENT countries (country+)>
<!ELEMENT country (. . . , capital, cities, . . . )>
```

```
   <!ATTLIST country  car_code ID #REQUIRED>
<!ELEMENT capital EMPTY>
   <!ATTLIST capital xlink:type (simple|extended|locator|arc)  #FIXED “simple”
                     xlink:href  CDATA #REQUIRED>
<!ELEMENT cities EMPTY>
   <!ATTLIST cities xlink:type (simple|extended|locator|arc)  #FIXED “simple”
                    xlink:href  CDATA #REQUIRED>

<countries>
   <country car_code=“B”> <name>Belgium</name>
     <capital href=“file:cities-B.xml#//city[name='Brussels']”/>
     <cities href=“file:cities-B.xml#//city”/>
     :
   </country>
     :
</countries>
```

Example 5 (Headquarters of Organizations) *The file organizations.xml in the distributed version does not contain information about memberships. Thus, only the @headq attribute of organizations is replaced by a headq subelement which is a simple link:*

```
<!ELEMENT organizations (organization+)>
<!ELEMENT organization (name, headq)>
   <!ATTLIST organization  abbrev ID #REQUIRED>
<!ELEMENT headq EMPTY>
   <!ATTLIST headq xlink:type (simple|extended|locator|arc)  #FIXED “simple”
                   xlink:href  CDATA #REQUIRED>

<organizations>
   <organization abbrev=“EU”> <name>European Union</name>
     <headq xlink:href=“file:cities-B.xml#//city[name='Brussels']”/>
   </organization>
     :
</organizations>
```

Additionally, there are *inline extended links*, and *out-of-line* extended links. The latter allow to create references not only inside documents, but also to create XML instances that consist *only* of links between other documents (e.g., memberships of countries in organizations).

3 Querying along Links

Each link can be seen as a view definition – possibly recursively containing further links; in this case, the view may be even infinite (due to cyclic references). Whereas in SQL, a view or a database link appears as a table or a database schema that easily fits with the language syntax and semantics, links as *tree view definitions* embedded into the data itself need some special handling.

We propose a *logical* data model where the link elements are regarded to be *transparent*:

the linked XML sources are mapped to a logical model that consists of a single XML tree. This logical model can then be processed with standard XPath, XQuery, or XSLT. This *logical* data model silently replaces link elements of the types xlink:simple and xlink:locator by the result sets of their XPointers, and elements of the types xlink:extended and xlink:arc are assigned with a (re)structuring semantics. Thus, the logical, transparent model is already a kind of a view of the data. The view is generated from the input documents only by restructuring the tree at the XLink elements. Thus, it does not require any separate query. Figure 2 illustrates the general intuition of replacing references by tree views.

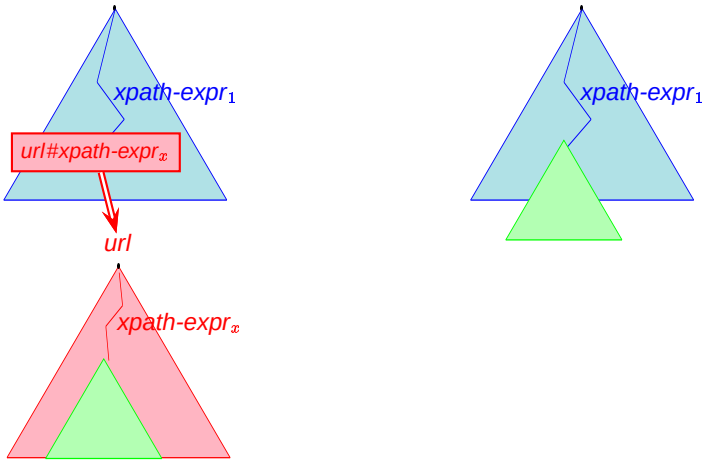


Figure 2: Extended XML Data Model with XLink Elements

The *external*, *logical* schema is induced in a well-defined way (that is described in more detail below) by the structure of the entry document, and by the structures of the linked documents. This *external schema* in turn induces the possible queries against the entry data source. Evaluation of these queries maps them back to the underlying documents, evaluating parts of the queries against the linked sources.

Example 6 (Motivation) *A simple, generic transparent model is obtained by always* replacing *the XLink elements by the XML contents that is referenced by them (details are described in the rest of the paper). In such a model, the sample query reads as*

```
(**)   document("memberships.xml")//membership
           [organization/headq = country/capital]/organization/@abbrev
```

The mapping between distributed, linked XML data, and a single XML instance finds applications in both directions:

- mapping a set of distributed, linked documents into a single, *logical* XML instance that is then queried in XPath, and
- a given XML instance can be distributed over several instances that are connected by

XLinks *without* changing its *logical model* – i.e., all queries yield the same answer against the original instance as against the logical model of the distributed database.

There is no *generic* intuitive transparent model. We propose a language extension to XLink that uses attributes – in the same way as e.g. xlink:show – to specify for each link element how it should be mapped to a transparent model.

4 Transparent Links: Modeling Switches

Depending on the link type, there are alternatives for mapping it to the logical model:

Simple Links. For simple links, the link consists of the link element – providing a name, and possibly attributes and element contents – and an XPointer:

- the *XLink element itself* can either be (i) kept, or (ii) dropped, or (iii) dropped and its attributes are replicated into the (resulting) subelements, or (iv) transformed into a (reference) attribute (its attributes are replicated into the (resulting) subelements).
- the result of *evaluating* an XLink element or a reference can be inserted as an element, or for each element of the result, its *attributes* and *contents* can be inserted an existing "element hull" that is provided by the surrounding XLink element.

Inline Extended Links. For inline extended links, "the link" consists of a grouping structure and a sequence of locators. Both can be handled separately in the above way.

Out-of-Line Links and Arcs. An extended, out-of-line link element is a collection of (i) locators and (ii) arcs. The surrounding link itself can be kept or dropped, and either

- the locator elements themselves can be ignored in the transparent model and the xlink:from and xlink:to attributes of the arcs are materialized as subelements, or
- the locators are kept (and handled with the same alternatives as simple links) and the arc is translated into reference attributes to the locators.
- It is possible to introduce new names for the xlink:from and xlink:to roles of arcs.

4.1 Specification in the dbxlink Namespace

We use the dbxlink:transparent attribute (denoting the database aspect of XLink) for specifying how the respective link elements are treated in the logical model (a formal characterization and further examples can be found in [May02a]):

For all XLink elements:

- keep-element: the XLink element itself is kept (without the XLink attributes), and contents and/or attributes are inserted.
- drop-element: the XLink element is dropped, i.e., replaced by the results of evaluating its attributes and contents.
- keep-attributes: the XLink element itself is dropped, the non-XLink attributes are kept

and added to each referenced element.

For simple links and locators (i.e., those elements that have an href attribute):

- insert-elements inserts the whole referenced element(s),
- insert-contents inserts the contents and attributes of the referenced element(s) into the surrounding element,
- insert-nothing does nothing: when the locator is actually only used for arcs, it should not be considered itself in the transparent model.
- make-attribute: the XLink element itself is dropped, instead a reference attribute is added to the surrounding element that yields the referenced elements (which are added to the logical instance "somewhere"). The non-XLink attributes of the link element are added to the referenced element(s).

Each arc contains a specification how to handle the from-locator and the to-locator: the dbxlink:transparent attribute can contain the values

- keep-from, drop-from, keep-from-attributes or make-from-attribute,
- keep-to, drop-to, keep-to-attributes or make-to-attribute,
- from-elements or from-contents, to-elements or to-contents,
- optionally, dbxlink:from-role and dbxlink:to-role specify what names are used for the from and to references.

Default Values. In general, data sources are given on the Web without dbxlink attributes. For that case, we propose the following default setting for dbxlink:transparent that keeps the names and possible additional attributes of the navigation elements and fills them with the contents of the referenced elements..

- keep-element for simple links, extended links, and arcs,
- insert-contents for simple links,
- keep-from/keep-to and from-contents/to-contents for arcs,
- drop-element and insert-elements for locators in inline extended links,
- drop-element and insert-nothing for locators in out-of-line extended links since they are just auxiliary.

4.2 Examples

Single-Target Simple Links. If a single target element is linked, it should often either appear instead of the link, or its contents should be integrated into the link element.

Example 7 (Headquarters of Organizations) *Consider again Example 5.*

The default settings for dbxlink:transparent (i.e., "keep-element insert-contents") map the contents and attributes of the referenced city element into the headq link element. The corresponding excerpt of the logical transparent instance of organizations.xml looks as follows, using the contents of the city element that represents Brussels in cities-B.xml:

```
<organizations>
  <organization abbrev="EU"> <name>European Union</name>
    <headq id="cty-B-Brussels">
      <name>Brussels</name>
      <population year="95">951580</population>
    </headq>
  </organization>
    :
</organizations>
```

Then the name of the headquarter city of an organization can be selected by

document("organizations.xml")/organization[@abbrev="EU"]/headq/name .

Example 8 (Headquarters of Organizations – Alternative Modeling) *Consider again Examples 5 and 7. Another modeling alternative is to resolve the headq link element into a* reference attribute*. Then, the source document looks as follows, containing the dbxlink:transparent specification:*

```
<organizations>
  <organization abbrev="EU"> <name>European Union</name>
    <headq dbxlink:transparent="make-attribute"
           href="file:cities-B.xml#//city[name='Brussels']"/>
  </organization>
    :
</organizations>
```

The logical instance then looks as follows (referenced elements are added "somewhere"):

```
<organizations>
  <organization abbrev="EU" headq="localcopyofbrussels01">
    <name>European Union</name>
  </organization>
    :
  <!-- here, the "imported" elements are stored -->
    <city id="localcopyofbrussels01"> <name>Brussels</name> ... </city>
    :
</organizations>
```

Then, the query can be stated in the same way as for the non-distributed document (Fig. 1):

document("organizations.xml")/organization[@abbrev="EU"]/@headq⇒city/name

Example 9 *Similar considerations as above for the headquarter cities of organizations hold for the capital reference in Example 4: the capital reference can either be mapped to a subelement or to an attribute. Then, the queries read as*

document("countries.xml")/country[@car_code="B"]/capital/name *or*
document("countries.xml")/country[@car_code="B"]/@capital⇒city/name *respectively*.

Further examples -including arcs- can be found in [May02a]).

5 Conclusion

We have discussed a *logical model* for linked XML documents that makes the links transparent. Queries are then stated against the logical schema without bothering the user about the distributed nature of the data, and how to handle the links. The intended logical schema can be specified by the owner of the XML documents by appropriately setting the attributes for the XLink elements in the dbxlink namespace. Mainly, we see two scenarios where the logical model is relevant:

- Querying: Given autonomous sources that are interlinked by XLink, the above defines a logical model for stating querying against such sources (in general, when no dbxlink attributes are given, the defaults specify the modeling).
- Database redesign: Given a single XML instance (according to a given, public schema) that should be split/distributed over several instances, the references between its parts are usually expressed by XLink (cf. Ex. 2). The attributes in the dbxlink namespace can then be used for retaining the original *logical model* and *external schema* wrt. the user – i.e., all queries can remain unchanged.

Recall that the transparent model is just a *logical, virtual* model. The decision whether it is materialized depends on the evaluation strategies (see [May02b]).

Materialization. For validating the above definition of the transparent model, an XSLT script [Mal02] has been created according to the recursive definition for the generation of the logical instance. Starting with a given XML instance, it processes the XML tree recursively. For each element that is equipped with XLink functionality (i.e., an xlink:type attribute), an appropriate template is applied that transforms the link element. In case of simple links and locators, the xlink:href attribute is evaluated, and the result list is processed recursively and the result is included into the result document. For arcs, the from and to attributes are evaluated, the corresponding locators are processed, and the results are again included into the result. In case of infinite, cyclic logical instances, the script returns a warning when a link/element pair is processed that is already on the current path and stops the recursion.

Note that materializing the transparent instance is only an intermediate step to define a *logical* instance (with a logical schema) as a base for querying.

Query Evaluation. Based on the promising results, an extension of the LOPiX system [May01a] is under work for investigating the handling of links in an XPath-based environment. An implementation in a standard XML database is planned. In these cases, the logical instance will not be materialized, but the evaluation of the query is split at the links, subqueries/views are evaluated wrt. the referenced documents, and the answers are then recombined. There are several possibilities, concerning *when* and *where* the views defined by links are evaluated, and what results may be cached. These issues are discussed in [May02a, May02b], also proposing the use of additional attributes in the dbxlink namespace for specifying evaluation strategies.

Related Work. The XML query languages XML-QL [DFF+99] and XQuery [XQu01] (and related approaches) allow to express "distributed" queries (e.g., for information inte-

gration) as *joins* of several queries to different sources. But, although the *W3C XML Query Requirements* [XMQ01b] explicitly state that *"3.4.12: Queries MUST be able to traverse intra- and inter-document references"*, neither XPath nor XQuery (and also not the earlier XML-QL) support navigation along XLink references. There is not yet any other work on querying linked XML instances.

XML Linking and Style [Wal01] is concerned with styling linked XML documents via XSL stylesheets. It proposes to add attributes to the xsl namespace that define – refining the behavior specified by the xlink:show attribute – how the *styled result* is embedded into the presentation of the current document. The main difference between *XML Linking and Style* and our approach is that the former operates on the representational, browsing level, where our approach operates on the data model level, defining a "database instance" that can then be queried.

Theoretical aspects of distributed query evaluation for semistructured data are discussed in [Suc02]. The paper does not consider the details how to resolve the links into a logical instance and schema, but focuses on the algorithms and distributed evaluation techniques for queries that use such distributed data.

Bibliography

[DFF+99] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XML-QL: A Query Language for XML. In *8th. WWW Conference*. W3C, 1999. World Wide Web Consortium Technical Report, `www.w3.org/TR/NOTE-xml-ql`.

[Mal02] Dimitrio Malheiro. Generating a Transparent Instance from Linked XML Documents (XSLT script). Available at `www.informatik.uni-freiburg.de/~may/LinXIS/`.

[May01a] Wolfgang May. LoPiX: A System for XML Data Integration and Manipulation. In *Intl. Conf. on Very Large Data Bases (VLDB), Demonstration Track*, 2001.

[May01b] Wolfgang May. The MONDIAL Database, 2001. `http://www.informatik.uni-freiburg.de/~may/Mondial/`.

[May02a] Wolfgang May. Considerations on Linked XML Document Networks in the Web. Available at `www.informatik.uni-freiburg.de/~may/LinXIS/`, 2002.

[May02b] Wolfgang May. Querying Linked XML Document Networks in the Web. In *11th. WWW Conf.*, 2002. Available at `http://www2002.org/CDROM/alternate/166/`.

[Suc02] Dan Suciu. Distributed Query Evaluation on Semistructured Data. *ACM Transactions on Database Systems (TODS)*, 27(1):1–62, 2002.

[Wal01] N. Walsh (ed.). XML Linking and Style. W3C Note `http://www.w3.org/TR/xml-link-style`, 2001.

[XLi00] XML Linking Language (XLink). `http://www.w3.org/TR/xlink`, 2000.

[XMQ01a] XML Query Data Model. `http://www.w3.org/TR/query-datamodel`, 2001.

[XMQ01b] XML Query Requirements. `http://www.w3.org/TR/xmlquery-req`, 2001.

[XPa01] XML Path Language (XPath) Version 1.0: 1999; version 2.0: 2001. `http://www.w3.org/TR/xpath20`, 2001.

[XPt00] XML Pointer Language (XPointer). `http://www.w3.org/TR/xptr`, 2000.

[XQu01] XQuery: A Query Language for XML. `http://www.w3.org/TR/xquery`, 2001.