Considerations on Linked XML Document Networks in the Web
DRAFT; JULY 2002

Wolfgang May

Institut für Informatik, Universität Freiburg, Germany
may@informatik.uni-freiburg.de

Abstract

The W3C XML Linking Language (XLink) provides a powerful means for interlinking XML documents all over the world. There is not yet an official proposal for the handling interlinked XML documents using 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 as specified by XLink can be seen as embedded views in an XML instance. We investigate a model where interlinked documents are regarded 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, raising new aspects for the data model and for query languages: for querying such documents, strategies how to handle links are required. We define a *transparent* data model for linked documents. The links are augmented with additional information that define an integrated view of an XML instance. Queries are then posed against the transparent model. Several strategies are presented how to handle such subtrees, concerning the timepoint when the link is evaluated, and the evaluation and caching strategies. The evaluation strategies are influenced by capabilities of the linked XML servers.

In our approach, the modeling and the evaluation strategies are specified by additional attributes in the dbxlink (database-xlink) namespace. So far, the approach is independent from the actual querying language. The approach is under implementation as an extension of LoPiX [May01b], a web-aware system for XML data manipulation.

1 Introduction

XML has been designed and accepted as *the* framework for semi-structured data where it plays the same role as the relational model for classical databases. As a data format tailored to the use in the Web, XML data is not required to be self-contained on an individual server, but may include *links* to XML data on other servers. The remote servers may be *autonomous*, i.e., they are not aware of being referenced by other documents. 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.

In general, links specified by using XLink express whatever relationships between resources. We see our work from the database point of view, where information is distributed over several XML instances. We investigate both the *data modeling* aspect and then the aspects of actually *querying* that data. Although, many of the considerations carry over to the document area, and to resources described by XML in general. In our context, a link element in an XML document is primarily an embedded view definition, given as a query against another document where suitable information can be found. Thus, this view has to be evaluated when the link element is actually traversed in a query. Working with such documents that contain links to remote sources raises several questions:

- it is desirable that links are transparent, i.e., queries behave as if the target of the link is declaratively accessible via the link element.

- the size of the complete linked documents may be tremendous (since they may again contain links to further documents), even if perhaps only a small excerpt is needed for an application.
- if a link element is found when parsing an XML instance, is it preferable to "evaluate" it first, "materializing" the answer instead of retaining the link, or is it preferable to regard the link as a *virtual* tree fragment that is queried on-demand? In both cases, the practicability heavily depends on the services provided by the linked, remote data source. In the virtual case, additional strategies may be used based on metadata information of the remote source to check if the source is relevant for answering the actual query.

In the presence of internet-wide distributed documents provided by servers that implement different access policies and server-side/client-side querying mechanisms, efficient evaluation of queries requires a well-defined and well-specified cooperation between the involved components.

The paper is organized as follows: Since we do not assume that everybody is familiar with the XLink specification, we first review the semantics of links as given in [XLi00] in Section 2. Section 3 describes the individual types of XLink elements, also introducing a running example. Section 4 analyzes the correspondences between links and the view concept and investigates navigation along links from the point of view of the data model. The distribution aspect of evaluating queries on the local server or on remote servers, and considerations on prefetching and caching of data are introduced in Section 5. We systematically identify characteristic parameters for specifying the handling of links and relate them to the service functionality provided by remote sources in Section 6. An experimental environment is described in Section 7. Section 8 concludes the paper.

2 XML Notions

2.1 Querying XML

Example 1 *Consider the following excerpt of the* MONDIAL *database [May01c] for illustrations.*

```
<!ELEMENT mondial (country+, organization+, ...)>
<!ELEMENT country (name, population, city+, ...)>
  <!ATTLIST country car_code ID #REQUIRED
                    area CDATA #IMPLIED
                    capital IDREFS #REQUIRED>

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

<!ELEMENT city (name, population*)>
  <!ATTLIST city   id ID #REQUIRED >

<!ELEMENT organization (name, established?, members*)>
  <!ATTLIST organization   abbrev ID #REQUIRED    seat IDREF #IMPLIED>

<!ELEMENT established (#PCDATA)>
<!ELEMENT members EMPTY>
  <!ATTLIST members   type CDATA #REQUIRED   country IDREFS #REQUIRED>

<country car_code="B" area="30510" capital="cty-B-Brussels">
  <name>Belgium</name>
  <population>10170241</population>
  <city id="cty-B-Brussels">
      <name>Brussels</name>
      <population year="95">951580</population>
  </city>
  <city id="cty-B-Antwerp">
      <name>Antwerp</name>
      <population year="95">459072</population>
```

```
  </city>
      ⋮
</country>
<organization abbrev="EU" seat="cty-Brussels">
    <name>European Union</name>
    <established>07 02 1992</established>
    <members type="member" country="GR F E A D I B L NL DK SF S IRL P GB"/>
    <members type="membership applicant" country="AL CZ H SK LV LT PL BG RO EW M CY"/>
</organization>
```

XPath [XPa99] 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 most XML querying languages that extend it with their special constructs (e.g., functional style commands and variables in XSLT [XSL99], and SQL/OQL style (e.g., joins) in Quilt/XQuery [CRF00, XQu01]). The core XML/XPath concept already provides unidirectional intra-document references by `ID/IDREF/IDREFS` attributes.

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

> *"Select all names of cities that are seats of an organization and the capital of one of its members."*

XPath uses the id(...) function for dereferencing: the expression

```
id(//organization[id(./@seat) = id(id(./members/@country)/@capital)]/@seat)/name
```

yields the names of the above-mentioned cities in the above MONDIAL *database.*

XQuery [XQu01] (which is based on Quilt [CRF00]) is the W3C proposal for querying XML documents. It embeds XPath syntax into higher-level constructs (similar to SQL/OQL). Although, already the dereferencing functionality of XPath is augmented in Quilt/XQuery with a "⇒"-operator for explicit dereferencing. An additional *name test* (added by XQuery wrt. Quilt) specifies the expected name of the target element.

Example 3 *In XQuery/XPath-2.0, the above expression is written as*

```
//organization[@seat⇒city = members/@country⇒country/@capital⇒city]/@seat⇒city/name .
```

In this paper, we use implicit dereferencing (as introduced for XPathLog/LoPiX in [May01b, May01d].

Example 4 *In XPathLog, the above expression is written as*

```
//organization[@seat = members/@country/@capital]/@seat/name .
```

Example 5 *The complete query given in Example 2 can be stated in XQuery as follows (returning the organization's abbreviation and the city's name). We assume that the city's name is unique to have a simpler criterion later when using distributed sources:*

```
FOR $org IN //organization,
      $abbrev IN $org/@abbrev,
      $seatname IN $org/@seat/name,
      $capname IN $org/members/@country/@capital/name
WHERE $capname = $seatname
RETURN <result org = $abbrev city = $seatname/>
```

*Here the **FOR** clause defines variables that are bound to the elements of the result set of XPath expressions, yielding a list of variable bindings. The **WHERE** clause – similar to SQL/OQL – states additional conditions which tuples of variable bindings are used for generating a result element for each variable binding in the **RETURN** clause.*

2.2 XPointer and XLink

2.2.1 XPointer

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* . Thus, everybody can define links from his XML documents to any other XML source. 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"]

2.2.2 XML Linking Language (XLink)

XPointers are used in the *XML Linking Language (XLink)* [XLi00] for expressing links between XML documents. 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. These specify the *behavior* of a link, i.e., its activating event and the triggered action. Whereas the HTML behavior is simply "browse this document", the XLink language allows for more sophisticated activities (e.g., not to delay the behavior until the link is activated by a user, but evaluate it immediately when the document is parsed). When links are used for *querying* XML document networks, this functionality has to be extended (as we will do in the subsequent sections). First, we give an overview of the notions defined by the XLink language and relate them to modeling issues in Section 3:
The xlink:type attribute selects between four basic types of linking semantics:

- simple: roughly, the semantics known from <A href="...">,
- extended: similar, allowing for multiple targets, or also specifying relationships *between* such targets,
- locator: elements with an xlink:type="locator" attribute are used for specifying targets that are then used by extended link elements,
- arc: elements with an xlink:type="arc" attribute are used for defining bidirectional links between a set of locator elements.

In most cases, the information that some element type is equipped with link semantics is given in the DTD. Here, attributes from the xlink: namespace are declared and some of them are assigned with a fixed value:

```
<!ELEMENT linkelement (contentsmodel)>
   <!-- Dependent on the xlink:type, some conditions on contentsmodel apply. -->
<!ATTLIST linkelement
        xlink:type      (simple|extended|locator|arc) #FIXED "..."
        xlink:href      CDATA #REQUIRED
        xlink:actuate   (auto|user)   #IMPLIED
        xlink:show      (new|parsed|replace)  #IMPLIED
        xlink:role      CDATA   #IMPLIED
        xlink:title     CDATA   #IMPLIED  >
```

(*linkelement* is any element type of the application to be augmented with link semantics)

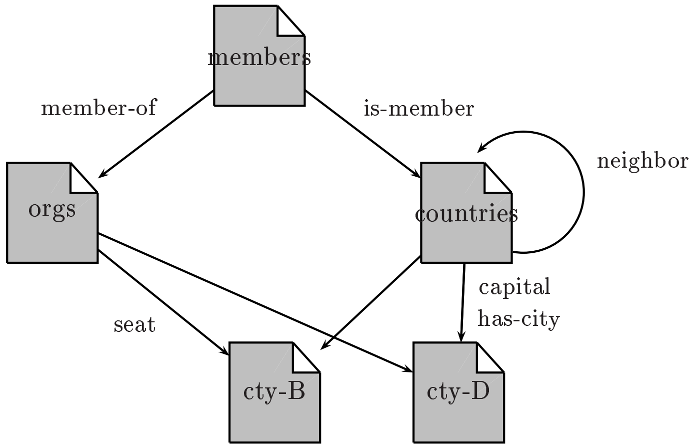
The xlink: attributes have the following semantics:

- xlink:href: selects a target of the individual instance (it "is" the link).
- xlink:actuate: defines the event on which the link is activated (when browsing).
- xlink:show: specifies what happens if the link is activated (when browsing).
- xlink:role and xlink:title: application-specific.

3 Modeling Relationships in XLink

In the following, we illustrate the use of the different types of links. Among other small examples, a "distributed" version of Mondial (see Example 1 for the non-distributed version) is described where all countries, all cities of a country, all organizations, and all memberships are stored in separate files:

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



Example 6 (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>
```

3.1 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 7 (Seats of Organizations)
The organizations file in the distributed version does not contain information about memberships. Thus, only the @seat *attribute of organizations is replaced by a* seat *subelement which is a simple link:*

```
<!ELEMENT organizations (organization+)>

<!ELEMENT organization (name, established?, seat)>
    <!ATTLIST organization  abbrev ID #REQUIRED>

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

An excerpt of the XML document mondial-organizations.xml *looks as follows:*

```
<organizations>
    <organization abbrev="EU">
      <name>European Union</name>
      <established>07 02 1992</established>
      <seat href="file:cities-B.xml#//city[name='Brussels']"/>
    </organization>
      ⋮
</organizations>
```

Modeling aspects are investigated later in Examples 16 and 17.

In the above example, the simple links are used for referencing a single node. The next example shows how a set of nodes can be addressed by a simple link - provided that the nodes can be addressed *by a single pointer*:

Example 8 (Countries and Cities) *A country has a capital and several cities. Similar to the above* seat *link, there is a single capital. The cities are referenced by a simple link that addresses a set of nodes (the modeling aspect is investigated later in Example 22):*

```
<!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>
```

An excerpt of the XML document mondial-countries.xml *looks as follows:*

```
<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>
```

Simple links can also be used for “attributed relationships”. Attributes of the link element describe the relationship that is represented by the link.

Example 9 (Neighbor Countries) *The neighbor relationship between countries has a* borderlength *attribute.*

```
<!ELEMENT countries (..., country*, ...)>

<!ELEMENT country (..., neighbor*, ...)>
   <!ATTLIST country  car_code ID #REQUIRED>

<!ELEMENT neighbor EMPTY>
   <!ATTLIST neighbor xlink:type (simple|extended|locator|arc)  #FIXED “simple”
                      xlink:href  CDATA #REQUIRED
                      borderlength CDATA #REQUIRED>
```

An excerpt of the country element for Belgium *looks as follows:*

```
<countries>
   <country car_code=“B”>
     ⋮
     <neighbor href=“file:countries#//country[@car_code=’F’]”  borderlength=“620”/>
     <neighbor href=“file:countries#//country[@car_code=’L’]”  borderlength=“148”/>
     ⋮
   </country>
     ⋮
</countries>
```

Again, the modeling aspect is investigated later in Example 19.

Multi-Target Attributed Simple Links. Attributed multi-target relationships can be expressed in this way only if the attributes are the same for all targets. From the modeling point of view, this is only a good choice if the attribute should be associated with the whole *set* of references (keep in mind that the whole element acts as a link).

Example 10 (Attributed Multi-Target Link) *Consider a database* customers.xml *that contains data about customers, e.g. birthday, address, sales p.a., etc. The company wants to associate them to different consultants, according to their sales and address:*

```
<customerrelationships>
  <group consultant=“John” xlink:type=“simple”
        xlink:href=“file:customers.xml#//customer[address/country=’Germany’ and sales>1000000]”/>
  <group consultant=“Bob” xlink:type=“simple”
        xlink:href=“file:customers.xml#//customer[address/country=’Austria’ and sales>1000000]”/>
  <group consultant=“Bill” xlink:type=“simple”
        xlink:href=“file:customers.xml#//customer[address/country=’Germany’ and sales<1000000]”/>
  ⋮
</customerrelationships>
```

The situation will be considered again in Examples 23.

Recall that we have now represented simple single-target relationships, attributed single-target relationships, and multi-target relationships that can be expressed by a single XPointer.
In the next section, grouping references are shown.

3.2 Inline Extended Links

Extended links do not have an href attribute, but instead they contain one or more locator elements, each of which is a reference of its own. Note that the *link* is the combination of the outer element and the locators. Thus, the *set* of locators should act *together*.

The effect of inline extended links is easier to illustrate from the point of view of browsing. Consider an XLink where the pointer is given by a pure url. Then, the behavior of this single-target simple link when browsing is that the link is activated and e.g. the target url is opened in a separate window. If the pointer is of the form *url#//xpath-expr*, i.e., the result is an arbitrary XML tree (or a sequence of trees), it can e.g. be embedded into presentation of the original source.

Consider now an inline extended link that contains several locators, each of them pointing to a url. By activating the *complete* link element, *all* locators are activated, and e.g., open their target urls in separate windows.

Example 11 (Inline Extended Links; Browsing) *Consider the following excerpt of a Web page:*

```
<B>Cities</B>
<UL>
   <LI xlink:type=“simple” xlink:href=“www.berlin.de”>Berlin </LI>
   <LI xlink:type=“simple” xlink:href=“www.muenchen.de”>Muenchen </LI>
   <LI xlink:type=“simple” xlink:href=“www.freiburg.de”>Freiburg </LI>
   ⋮
</UL>
```

where the HTML <LI> elements are extended with simple link behavior: when clicking on a city, its homepage opens in a separate window. In contrast, in the slightly changed page

```
<B>Cities</B>
<UL xlink:type=“extended” >
   <LI xlink:type=“locator” xlink:href=“www.berlin.de”>Berlin</LI>
   <LI xlink:type=“locator” xlink:href=“www.muenchen.de”>Muenchen</LI>
   <LI xlink:type=“locator” xlink:href=“www.freiburg.de”>Freiburg</LI>
   ⋮
</UL>
```

where the <UL> element is an extended *link, and the <LI> elements act as locators, a click on the* whole *<UL> element (that is the link) opens* all *city homepages in separate windows.*

In Section 4.5.2 we will show that from the database point of view, extended links with a sequence of locators do not add expressiveness compared with the use of a sequence of simple links.

Nevertheless, from the *modeling* point of view, the "grouping" mechanism provided by extended links allows for more intuitive modeling.

Example 12 (Inline Extended Links) *Consider again Example 10. A more flexible composition of the customer groups can be described by inline extended links, where each group is an extended link and the customers are referenced by auxiliary locators (each locator either addressing one or more customers):*

```
<customerrelationships>
  <group consultant=“John” xlink:type=“extended”>
    <aux xlink:type=“locator”
         xlink:href=“file:customers.xml#//customer[address/country=’Germany’ and sales>1000000]”/>
    <aux xlink:type=“locator”
         xlink:href=“file:customers.xml#//customer[address/country=’Austria’ and sales>1000000]”/>
  </group>
  ⋮
```

</customerrelationships>

In contrast to the before example, the auxiliary aux *locator elements allow for a more flexible definition; on the other hand, they lead to a more complex structure and less intuitive queries. This example is again considered in Example 24.*

3.3 Out-of-Line-Links

Out-of-line-links allow to create references not only inside documents, but also to create XML instances that consist *only* of links between other documents. This can be seen as a “personalized overlay” on arbitrary existing (autonomous) sources. Here, the link is not a directed link to a node, but a relationship (or a set of relationships) between a set of nodes.

Out-of-line-links consist of locator subelements that specify *potential* endpoints of links, and arc elements that specify the relationships between these endpoints:

```
<!ELEMENT linkelement (...|locatorelement|arcelement)*>
<!ATTLIST linkelement as above >

<!ELEMENT locatorelement as above >
<!ATTLIST locatorelement as above, including type, href, id: ID >

<!ELEMENT arcelement (contentsmodel)>
<!ATTLIST arcelement  xlink:type   (simple|extended|locator|arc)  #FIXED “arc”
                      xlink:from   IDREF  #REQUIRED
                      xlink:to     IDREF  #REQUIRED  >
```

Example 13 *For the distributed* Mondial *database, the memberships of countries in organizations are stored in mondial-memberships.xml:*

```
<!ELEMENT memberships (country*,organization*,membership*)>
<!ATTLIST memberships xlink:type (simple|extended|locator|arc)  #FIXED “extended”  >

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

<!ELEMENT organization EMPTY>
<!ATTLIST organization xlink:type (simple|extended|locator|arc)   #FIXED “locator”
                       xlink:href  CDATA  #REQUIRED
                       id          ID #REQUIRED  >

<!ELEMENT membership EMPTY>
<!ATTLIST membership xlink:type    (simple|extended|locator|arc)   #FIXED “arc”
                     xlink:from    IDREF  #REQUIRED
                     xlink:to      IDREF  #REQUIRED
                     membership_type CDATA  #REQUIRED  >
```

The file mondial-memberships.xml consists now of only one memberships *element that contains locators to countries (in countries.xml and organizations (in organizations.xml), and arcs that relate countries to organizations:*

```
<memberships>
  <country id=“B” xlink:href=“.../countries.xml#//country[@car_code=’B’]”/>
  <country id=“D” xlink:href=“.../countries.xml#//country[@car_code=’D’]”/>

  <organization id=“org-EU” xlink:href=“.../organizations.xml#//organization[@abbrev=’EU’]”/>
  <organization id=“org-UN” xlink:href=“.../organizations.xml#//organization[@abbrev=’UN’]”/>
```

```
    <membership xlink:from=“B” xlink:to=“org-EU” membership_type=“member”/>
    <membership xlink:from=“B” xlink:to=“org-UN” membership_type=“member”/>
</memberships>
```

3.4 Expressing Derived Properties by XLink Elements

In addition to links with other elements, *derived properties* can be defined as *views* over the same document.

Example 14 (Derived Properties) *Each country has additional elements* density *and* urban_pop *that yield the population density and the total number of urban population. The element for Belgium looks as follows:*

```
<country car_code=“B” area=“30510” ...”>
  <name>Belgium</name>
  <population>10170241</population>
  <density xlink:type=“simple” xlink:href=“sibling::population/text() div ../@area”/>
  <urban_pop xlink:type=“simple” xlink:href=“sum(sibling::city/population/text())”/>
  <city id=“cty-B-Brussels”>
      <name>Brussels</name>
      <population year=“95”>951580</population>
  </city>
      ⋮
</country>
```

Since each of the country *elements contains these subelements, and they are the same for each element (although naturally computing different values), they can be predefined by the DTD:*

```
<!ELEMENT country (name, population, density, urban_pop, city+, ...)>
  ⋮
<!ELEMENT density EMPTY>
  <!ATTLIST density xlink:type (...) #FIXED “simple”
                     xlink:href CDATA #FIXED “sibling::population/text() div ../@area”>
<!ELEMENT urban_pop EMPTY>
  <!ATTLIST urban_pop xlink:type (...) #FIXED “simple”
                       xlink:href CDATA #FIXED “sum(sibling::city/population/text())”>
```

This example is discussed again in Example 21.

Summary. Above, we have described different types of links from the syntactic point of view and we have given intuitive examples for their use. In the subsequent sections, the data modeling and evaluation aspects of linked documents are investigated: from the modeling point of view, each linked element induces an external DTD or XML Schema structure, consisting of the structure of the entry document, and of the structures of the linked documents. This external schema in turn induces the possible queries against the entry data source. Then, the evaluation of these queries maps them back to the underlying documents, evaluating parts of the queries against the linked sources.

4 Data Models and Queries for Linked Documents

XPointer and XLink specify how to *express* inter-document links in XML. There is not yet an official proposal

- how to add link semantics to the actual data model, e.g., the DOM or the XML Query Data Model [XMQ01], and
- 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 section, we describe two alternative extensions of the data model with link elements. Evaluation issues, e.g., when and where links are evaluated, are then investigated in Sections 5 – 6.

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 *embedded views* need one of the following semantic extensions:

- extending the abstract data model with a linking construct. In this case, an additional syntactic construct in form of an explicit navigation operator is required.
- the abstract data model makes the links transparent, i.e., the embedded views implicitly become subtrees that can be queried by standard XPath expressions.

4.1 An Explicit Navigation Operator

The query in Example 2 uses the XQuery [XQu01] syntax with the dereferencing operator for `IDREF(S)`, "$\Rightarrow$ *nodetest*". Similarly, we define the dereferencing operator "$\rightsquigarrow$ *nodetest*" for following XPointers in xlink:href attributes for explicit *inter-document navigation*.

Example 15 *Consider again the query given in Example 2: The entry point for the query is the document mondial-memberships.xml, iterating over all membership elements (e.g., ("B", "EU")),*

- *following the link to the corresponding organization in mondial-organizations.xml, searching for the <organization id="org-EU"> element, which contains a seat subelement (which is a simple link). The target of the link is a city element in mondial-cities-B.xml whose name subelement has to be retrieved.*
- *following the link to the corresponding country in mondial-countries.xml, searching for the <country car_code="B"> element, which contains another (simple) xlink subelement capital. The target of the link is the city element that represents Brussels in mondial-cities-B.xml; again, the name subelement has to be retrieved and compared with the result from the first search.*

For illustrating in which document the current navigation step is applied, the namespaces ms: (in mondial-memberships.xml), org: (in mondial-organizations.xml), ctry: (in mondial-countries.xml), and cty: (in mondial-city-XXX.xml) are used:

```
FOR $ms IN document("mondial-memberships.xml")/ms:memberships/ms:membership,
    $org IN $ms/@xlink:to⇒ms:organization/@xlink:href⇝org:organization,
    $abbrev IN $org/@org:abbrev,
    $seatname IN $org/org:seat/@xlink:href⇝cty:city/cty:name,
    $capname IN $ms/@xlink:from⇒ms:country/@xlink:href⇝ctry:country/
                ctry:capital/@xlink:href⇝cty:city/cty:name
WHERE $capname = $seatname
RETURN <result org = $abbrev city = $seatname/>
```

The use of an explicit navigation operator requires non-semantic navigation steps (e.g., along the xlink:from attribute). Additionally, it is not stable against changes in the representation of links, including splitting an XML source into interlinked sources. Thus, a transparent modeling where the XLink elements are present only on the physical level, but queries navigate in a *logical* model along semantic notions is desirable.

4.2 Transparent Links: Modeling Switches

In the above example, an explicit operator for dereferencing links has been introduced. Another possibility is to regard link elements to be *transparent*, i.e., the (abstract) data model should silently replace 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 of the virtual XML instance. With this, the 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 1 illustrates the general intuition of replacing references by tree views.

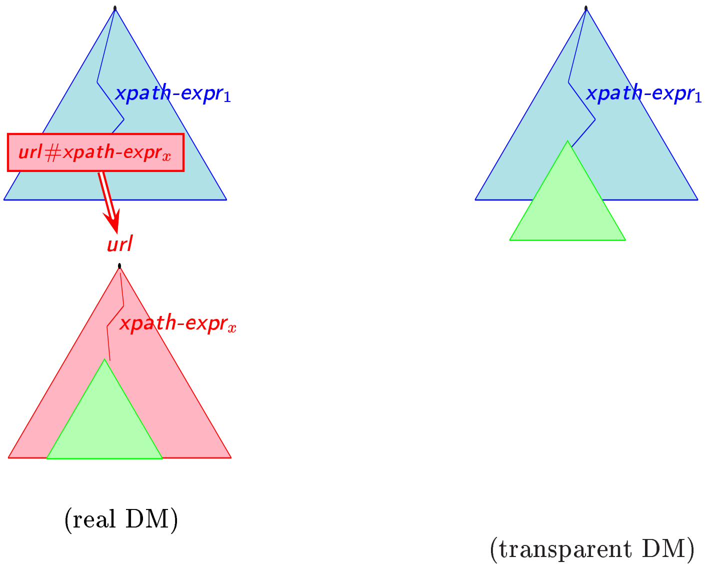


Figure 1: Extended XML Data Model with XLink Elements

Our investigations have shown that dependent on the intended modeling, there is no *generic* intuitive transparent model for XLink elements. Instead, we propose to use attributes – in the same way as e.g. xlink:show – to specify for each link element how it should be handled.

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
  - kept, or
  - dropped, or
  - dropped and its attributes are replicated into the (resulting) subelements, or
  - 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.

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, similar to the considerations above for single-target or multi-target simple links.

In most cases, the locator elements themselves are mainly of technical use and can be ignored in the transparent model. The arc elements then provide application-specific information. Thus, in general, the arc will be kept.

Every arc element specifies an own policy how to handle the from-locator and the to-locator in a similar way as above: the locator elements can either be copied into the arc or dropped, and the whole elements resulting from evaluating the links can be inserted, or only their contents. In these cases, the result of the from is placed before the result of the to as contents of the surrounding element. Additionally, from and to can be transformed into IDREF-reference attributes in the virtual model.

Note that keeping the locator element and inserting only the contents of the linked elements allows to use an element name (that of the locator) that is specified in the linking file instead of the original name. Even more, it is possible, to specify completely different names for the from and the to-contents (cf. Example 28).

4.3 Transparent Links: Modeling Switches

We use the dbxlink namespace (denoting the database aspect of XLink) for additional attributes for XLinks. The attribute dbxlink:transparent (NMTOKENS-valued) specifies how the respective link elements are treated in the transparent model:
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 subelement (in the virtual model).

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 (note that the referenced elements must be added to the virtual instance). The non-XLink attributes of the link element are added to the referenced element(s) (in the virtual model).

For extended links, insert-elements and insert-contents analogously specify how the locator subelements are handled. In case of multi-target references of simple links or locators (cf. Example 22) and for extended links, how multiple results are modeled in combination with keep-element:

- collect specifies that in case that the element addresses multiple elements, these should be collected immediately;
- distribute specifies that each referenced element is handled separately (cf. Examples 23 and 24.

For arc elements, there are the same choices as above. Additionally, 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 (cf. Example 28).

Note that when an element is kept, also the attributes and contents of the original link element (in case it has non-xlink contents) are kept.

Default Values. For default, we propose the following 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, inline extended links, out-of-line extended links and arcs,
- insert-contents for simple links,
- keep-from/keep-to for arcs,
- from-contents/to-contents for simple links and for arcs,
- insert-elements for out-of-line extended links and 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,
- distribute for simple links and locators.

Note that for simple links and locators where

- the link element has the same element name as the referenced elements, and
- the link element does not have non-xlink attributes and contents,

"keep-element insert-contents" and "drop-element insert-elements" are equivalent.
We will show in Section 4.5 that this default yields the intended, "natural" meaning in most of the introductory examples given in Section 3.

4.4 Definition of the Virtual Instance

The virtual instance is now defined by structural induction. We define a mapping ϕ that maps collections of element and attribute nodes or an XPointer to collections of attribute nodes and element nodes. Let μ be an element of type *elem_name* with contents $(\mu_1, \ldots, \mu_n)$.

- Elements that have no xlink:type attribute, are unchanged on their upper level: the attributes of $\phi(\mu)$ are the attributes of μ and the contents of $\phi(\mu)$ is $(\phi(\mu_1, \ldots, \mu_n))$.
- for a sequence $\mu_1, \ldots, \mu_n$ of elements, $\phi(\mu_1, \ldots, \mu_n)$ consists of all attribute nodes in $\phi(\mu_1) \cup \ldots \cup \phi(\mu_n)$ and the concatenation of all element nodes in $\phi(\mu_1) \cup \ldots \cup \phi(\mu_n)$.
- for an element μ, a name $name$ and a sequence $(\mu_1, \ldots, \mu_n)$ of elements, the auxiliary function (with side effect) mkattr$(\mu, name, (\mu_1, \ldots, \mu_n))$ does the following:
  - if $(\mu_1, \ldots, \mu_n)$ contains only element nodes, then
    - each top-level element in $(\mu_1, \ldots, \mu_n)$ is equipped with an ID attribute, and the non-xlink attributes and non-xlink subelements of μ are added to it, and the resulting element is added somewhere in the virtual instance.
    - return an attribute node of type IDREFS with name $name$ that contains the IDREFs to all top-level elements in $(\mu_1, \ldots, \mu_n)$.
  - if $(\mu_1, \ldots, \mu_n)$ contains a single attribute node with value x, or just a value x, then return an attribute node with name $name$ and value x.
  - otherwise not allowed.
- if μ/@xlink:type is either "simple" or "locator"
  - for μ/@xlink:href="*xpath-expr*", ϕ(*xpath-expr*) is the result of evaluating *xpath-expr*,
  - if μ/@dbxlink:transparent contains "keep-element distribute" then $\phi(\mu)$ is a list of elements of the same type as μ such that the ith element
    - contains the non-xlink attributes of μ, and
    - in case that "insert-elements" is specified,
      - if ϕ(*xpath-expr*) is a list of elements (including the case of text elements), the ith element of ϕ(*xpath-expr*) is added,
      - if ϕ(*xpath-expr*) contains a single attribute node, this attribute node is added.

    * in case that "insert-contents" is specified, the attributes and element contents of the ith element of $\phi(\textit{xpath-expr})$ are added.
    * $\phi(\mu_1, \ldots, \mu_n)$ is added.
  - if μ/@dbxlink:transparent contains "keep-element collect" then $\phi(\mu)$ is an element of the same type as μ that
    * contains the non-xlink attributes of μ, and
    * in case that "insert-elements" is specified,
      · if $\phi(\textit{xpath-expr})$ is a list of elements (including the case of text elements), then $\phi(\textit{xpath-expr})$ is added,
      · if $\phi(\textit{xpath-expr})$ contains a single attribute node, this attribute node is added.
    * in case that "insert-contents" is specified, all attributes and element contents of the elements in $\phi(\textit{xpath-expr})$ are added.
    * $\phi(\mu_1, \ldots, \mu_n)$ is added.
  - if μ/@dbxlink:transparent contains "drop-element" then $\phi(\mu)$ is generated as follows:
    * if "insert-elements" is specified,
      · if $\phi(\textit{xpath-expr})$ is a list of elements (including the case of text elements), then $\phi(\mu) = \phi(\textit{xpath-expr}) \cup \phi(\mu_1, \ldots, \mu_n)$.
      □ if $\phi(\textit{xpath-expr})$ is a not a list of elements, "drop-element" is not allowed.
    □ "insert-contents" is not allowed.
  - if μ/@dbxlink:transparent contains "keep-attributes"
    * if $\phi(\textit{xpath-expr})$ is a list of elements (including the case of text elements), then $\phi(\mu)$ is the concatenation of $\phi(\textit{xpath-expr})$ where each element is extended with the non-xlink attributes of μ, and $\phi(\mu_1, \ldots, \mu_n)$.
    □ if $\phi(\textit{xpath-expr})$ contains text elements and μ contains non-xlink attributes, then "drop-element" is not allowed.
    □ if $\phi(\textit{xpath-expr})$ is a not a list of elements, "drop-element" is not allowed.
    □ "insert-contents" is not allowed.
  - if μ/@dbxlink:transparent contains "make-attribute" then $\phi(\mu) = \mathsf{mkattr}(\mu, \mathsf{name}(\mu), (\nu_1, \ldots, \nu_k))$ where $(\nu_1, \ldots, \nu_k)$ is obtained as follows:
    * if $\phi(\textit{xpath-expr})$ contains only element nodes, then k is the number of elements in $\phi(\textit{xpath-expr})$ and ν_i is the ith element in $\phi(\textit{xpath-expr})$.
    * if $\phi(\textit{xpath-expr})$ contains a single attribute node or value a, then $k = 1$ and $\nu_1 = a$.
    * if the result of xlink:href="*xpath-expr*" contains a set of attribute nodes of type IDREF with the same name, then $k = 1$ and ν_1 is an attribute node of type IDREF the same name and whose value consists of all these IDREFs.
    □ only allowed if one of the above rules is applicable.
    □ "insert-contents" is not allowed.
- if μ/@xlink:type="extended", then the same as above, but
  - use $\phi(\ldots)$ of its locator and arc contents instead of $\phi(\textit{xpath-expr})$, and
  - use $\phi(\ldots)$ of its non-locator-and-arc contents instead of $\phi(\mu_1, \ldots, \mu_n)$.
- if μ/@xlink:type="arc", then
  - "drop-element" and "keep-attributes" are not recommended (although they are defined),
  - $\phi(\text{from})$ is defined as follows:
    * let *element* the locator element with the id specified by μ/@from. Then, let *element'* be the same element where the dbxlink:transparent attribute is adjusted according to the specification given by μ/@dbxlink:transparent (i.e., concerning keep-from/drop-from/... and from-elements/from-contents/...),

* let $temp := \phi($*element'*$)$,
* if μ/@dbxlink:transparent contains "make-from-attribute", then $temp :=$ mkattr$(\mu, name(temp), temp)$,
* in case that μ/@dbxlink:from-name is set,
  - in case that "make-from-attribute" is set, rename the returned attribute node to that name,
  - otherwise rename all top-level elements in *temp* to that name.
* let ϕ(from) := *temp*.

- ϕ(to) is defined analogously.
- if μ/@dbxlink:transparent contains "keep-element", then $\phi(\mu)$ is an element of the same type as μ that contains the non-xlink attributes of μ and the union of ϕ(from) and ϕ(to) and $\phi(\mu_1, \ldots, \mu_n)$,
- if μ/@dbxlink:transparent contains "drop-element", then $\phi(\mu)$ is the union of ϕ(from) and ϕ(to) and $\phi(\mu_1, \ldots, \mu_n)$.
- if μ/@dbxlink:transparent contains "keep-attributes" then $\phi(\mu)$ is the union of ϕ(from) and ϕ(to) where the attributes of μ are added to each top-level element. Not allowed if there is non-xlink element contents.
- if μ/@dbxlink:transparent contains "make-attribute", then generate *temp* as an element of a new type whose attributes and contents is the union of ϕ(from) and ϕ(to). Then, $\phi(\mu) =$ mkattr$(\mu,$ name$(\mu), temp)$.

Remark 1 In case that an XPointer references a single element or attribute node (or computers a single value), the definition is much easier since ϕ(*xpath-expr*) contains only a single node.

4.5 Examples

In this section, the transparent data model is illustrated by considering some of the aforementioned examples in more details and pointing out special aspects of the modeling. We show the intuitiveness of the chosen default values and give also examples where there are intended deviations from the defaults.

4.5.1 Simple Links

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 16 (Seats of Organizations) *Consider again Example 7*

```
<organizations>
  <organization abbrev="EU">
    <name>European Union</name>
    <established>07 02 1992</established>
    <seat href="file:cities-B.xml#//city[name='Brussels']"/>
  </organization>
    ⋮
</organizations>
```

The default settings for dbxlink:transparent (i.e., "keep-element insert-contents") map the contents and attributes of the referenced city element into the seat link element. The corresponding excerpt of the virtual transparent instance of mondial-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>
        <established>07 02 1992</established>
        <seat id=“cty-B-Brussels”>
            <name>Brussels</name>
            <population year=“95”>951580</population>
        </seat>
    </organization>
        ⋮
</organizations>
```

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

```
document(“mondial-organizations.xml”)/organization[@abbrev=“EU”]/seat/name .
```

Example 17 (Seats of Organizations) *Consider again Examples 7 and 16. Another modeling alternative is to resolve the* seat *link element into a* reference attribute*.*

```
<organizations>
    <organization abbrev=“EU”>
      <name>European Union</name>
      <seat dbxlink:transparent=“make-attribute”
            href=“file:cities-B.xml#//city[name=’Brussels’]”/>
    </organization>
      ⋮
</organizations>
```

The virtual instance then looks as follows (note that the referenced elements are placed “somewhere”):

```
<organizations>
    <organization abbrev=“EU” seat=“localcopyofbrussels01”>
        <name>European Union</name>
    </organization>
        ⋮
    <!-- here, the “imported” elements are stored -->
    <onlyusedbyreferenceattributes>
        <city id=“localcopyofbrussels01”>
            <name>Brussels</name>
            <population year=“95”>951580</population>
        </city>
    </onlyusedbyreferenceattributes>
        ⋮
</organizations>
```

Then, the query can be stated as

```
document(“mondial-organizations.xml”)/organization[@abbrev=“EU”]/@seat/name
```

in the same way as for the non-distributed document.

Example 18 *Similar considerations as above for the seats of organizations hold for the* capital *reference in Example 8: the* capital *reference can either be mapped to a subelement or to an attribute. Then, the queries read as*

```
document(“mondial-countries.xml”)/country[@car_code=“B”]/capital/name
```

or

```
document("mondial-countries.xml")/country[@car_code="B"]/@capital/name
```

respectively.

Attributed Single-Target Simple Links.

Example 19
Consider again Example 9 where the neighbor *link element contains additional information about the borderlength between two countries.*

```
<countries>
  <country car_code="B">
    ⋮
    <neighbor href="file:countries#//country[@car_code='F']" borderlength="620"/>
    <neighbor href="file:countries#//country[@car_code='L']" borderlength="148"/>
    ⋮
  </country>
    ⋮
</countries>
```

According to the default setting of dbxlink:transparent *(i.e.,* "keep-element insert-contents"*) to keep the link element and to insert the contents of the referenced element, the* neighbor *elements are kept, together with the* borderlength *attribute, and filled with the attributes and contents of the referenced country elements. The country element for Belgium in the virtual instance looks as follows:*

```
<country car_code="B" ...>
  <name>Belgium</name>
  <population>10170241</population>
  ⋮
  <neighbor code="F" borderlength="620"/>
    <name>France</name>
    <population>...</population>
    ⋮
  </neighbor>
  ⋮
</country>
```

Note that the virtual instance is infinite since the element that represents France again contains a neighbor *subelement that represents Belgium etc ...*
Then, the borderlength of Belgium with France can be selected by

```
document("mondial-countries.xml")/country[@car_code="B"]
                                   /neighbor[name="France"]/@borderlength
```

Attributed simple links can also be transformed into attributes. Then, the referenced element is added to the virtual instance, and is augmented with the attributes of the link element.

Example 20 *Consider again Examples 9 and 19, now with the following setting in the original* countries.xml *document.*

```
<countries>
  <country car_code="B">
```

```
    ⋮
    <neighbor dbxlink:transparent="make-attribute"
              href="file:countries#//country[@car_code='F']" borderlength="620"/>
    <neighbor dbxlink:transparent="make-attribute"
              href="file:countries#//country[@car_code='L']" borderlength="148"/>
    ⋮
  </country>
    ⋮
</countries>
```

The neighbor element itself is mapped into an attribute (here, all neighbors are collected in an IDREFS attribute). The referenced country *elements are augmented with a* borderlength *attribute. The corresponding excerpt of the virtual instance looks as follows:*

```
<country car_code="B" neighbor="local-B-F local-B-L ..." >
  <name>Belgium</name>
  <population>10170241</population>
  ⋮
  ⋮
</country>
  <!-- here, the "imported" elements are stored -->
  ⋮
<onlyusedbyreferenceattributes>
  <neighbor code="F" id="local-B-F"
            borderlength="620" neighbor="local-F-E local-F-B .../>
     <name>France</name>
     <population>...</population>
     ⋮
</neighbor>
</onlyusedbyreferenceattributes>
  ⋮
```

Here, the virtual instance contains an additional element for each neighborship (e.g., for Belgium with France, for Belgium with Luxembourg etc) - their neighbor *attributes reference again these elements, thus, the virtual instance is finite.*
Then, the borderlength of Belgium with France can be selected by

```
document("mondial-countries.xml")/country[@car_code="B"]
                                  /@neighbor[name="France"]/@borderlength
```

Links that compute a derived Property.

Example 21 (Derived Properties) *Consider again Example 14 where the XPointer expression of a simple link actually computes a single value. By defining*

- *"keep-element insert-contents" for the population density, whose value is inserted as text contents into the (former) link element, and*
- *"make-attribute" for the urban population that is added as an attribute to the* country *element,*

the result looks as follows:

```
<country car_code="B" area="30510" urban_pop="2439050" >
  <name>Belgium</name>
```

```
  <population>10170241</population>
  <density>333.33</density>
      ⋮
</country>
```

Multi-Target Simple Links. In case of multi-target simple links, the link element can either be kept – providing a grouping structure – or it can be omitted and the referenced elements take its position. The next example shows that in case of a *multi-target simple link*, it is preferable not to use the default setting.

Example 22 *Consider again Example 8. The* cities *link element is skipped, and each of the referenced elements is inserted. The excerpt of the XML document* mondial-countries.xml *for Belgium looks as follows:*

```
<countries>
    <country car_code=“B”>
      <name>Belgium</name>
      <capital href=“file:cities-B.xml#//city[name=’Brussels’]”/>
      <cities dbxlink:transparent=“drop_element insert_elements”
              href=“file:cities-B.xml#//city”/>
      ⋮
    </country>
      ⋮
</countries>
```

It defines the virtual transparent instance as

```
<country car_code=“B”>
    <name>Belgium</name>
    <capital >
        <!-- attributes and contents of document(“cities-B.xml”)//city/[name=’Brussels’]” -->
    </capital >
    <city id=“cty-B-Brussels”>
      <name>Brussels</name>
      <population year=“95”>951580</population>
    </city>
    <city>
      <name>Antwerp</name>
      <population year=“95”>459072</population>
    </city>
    ⋮
</country>
```

The transparent model allows for addressing all names of cities by

```
document(“mondial-countries.xml”)/country[@car_code=“B”]/city/name .
```

If the multi-target simple link has attributes, it must be decided whether it is intended to have a *grouping* semantics, or if the attributes are intended to belong to *each* of the referenced elements.

Attributed Multi-Target Simple Links.

Example 23 (Attributed Multi-Target Link) *Consider again Example 10 for illustrating several alternatives.*

a) *keep the group element, and extend it by the sequence of customer elements that are referenced by multi-target XPointer:*

```
<customerrelationships>
  <group consultant=“John” xlink:type=“simple”
      dbxlink:transparent=“keep-element insert-elements collect”
      xlink:href=“file:customers.xml#//customer[address/country='Germany' and sales>1000000]”/>
  ⋮
</customerrelationships>
```

Here, the virtual instance looks as follows:

```
<customerrelationships>
  <group consultant=“John”>
    <customer> ... </customer>
    <customer> ... </customer>
      ⋮
  </group>
  <group consultant=“Bob”>
    <customer> ... </customer>
    <customer> ... </customer>
      ⋮
  </group>
  ⋮
</customerrelationships>
```

b) *drop the grouping element and move the attributes to each of the referenced elements: the consultant attribute is replicated in every customer element (using “distribute”):*

```
<customerrelationships>
  <group consultant=“John” xlink:type=“simple”
      dbxlink:transparent=“keep-attributes insert-elements distribute”
      xlink:href=“file:customers.xml#//customer[address/country='Germany' and sales>1000000]”/>
  ⋮
</customerrelationships>
```

Here, the virtual instance looks as follows:

```
<customerrelationships>
  <customer consultant=“John”> ... </customer>
  <customer consultant=“John”> ... </customer>
    ⋮
  <customer consultant=“Bob”> ... </customer>
  <customer consultant=“Bob”> ... </customer>
    ⋮
  ⋮
</customerrelationships>
```

4.5.2 Extended Links

For extended links, “the link” consists of a grouping structure and a sequence of locators. Both can be kept or dropped.

Example 24 (Inline Extended Links) *Consider again Example 12. There are the same alternatives as in Example 23.*

a) Keep the extended link elements, remove the locators and replace them by their result sets:
   - keep-element *for the* group *elements, and*
   - drop-element *and* insert-elements *for the* aux *locators.*

   The result is then the same virtual model as above in Example 23(a).

b) move the attributes of the group *element downwards to extend the referenced elements – note that also the locators are dropped:*
   - keep-attributes *and* distribute *for the* group *elements (shifts the attributes inside the referenced subelements), and*
   - drop-element *and* insert-elements *for the locators.*

   The result is then the same virtual model as above in Example 23(b).

In Example 12, there is no reason to equip the group element with XLink functionality – it would have been sufficient to define the aux locators as simple links. Cases (b) above showed how to consider the "grouping" extended link element to be just auxiliary in the sense that it contains attributes which are intended to be added to each referenced element (obviously, this is a misuse of the "link" idea).

In contrast to the case that has been described in Example 11 when a user is *browsing* through an extended link by activating the whole link in a set-like way, there is no corresponding semantics for querying. Thus, an extended link with several locators can always be replaced with a non-link grouping element that contains simple links.

On the other hand, a group of links can be associated with a *logical* semantics – i.e., conjunctive or disjunctive.

Alternative Extended Links. The disjunctive interpretation of extended XLink elements can be applied for *alternative* sources; both in case of redundancy, and in case that there are overlapping sources that possibly provide the information: the first locator that adresses a non-empty result is regarded as the "contents" of the extended link. By further specifying the term "the first" in the before sentence, several semantics can be defined:

- choice: evaluate all locators in parallel. If any evaluation returns an answer, use it and discard the other computations.
- priority: evaluate the locators in the given order. The first one that is accessible and returns an answer is used.

Conjunctive Extended Links. In the conjunctive interpretation, each locator specifies a set of elements, and the semantics of the link is the computed by an operator over all partial results:

- intersection: all elements that are contained in each of the results of the locators (wrt. identity or deep equality)
- set union: all elements that are contained in at least one of the results of the locators (wrt. identity or deep equality)
- multiset union: concatenation of all result trees. Note that multiset union is equivalent to the use of a non-link element with simple link subelements.

For extended links with no arcs, the modeling switches in dbxlink:transparent can include the following values:

- choice, priority, intersect, setunion, multisetunion are allowed.

Out-of-line Extended Links.

Example 25 (Out-of-Line Links, Default) *Consider again Example 13. Here, the default setting specifies*

- keep-element *for the surrounding (root)* memberships *element*

- *keep-element, keep-from, from-contents and keep-to, to-contents for the membership arc elements: the arc elements are kept and the xlink:from attribute is replaced by the referenced country subelement (combining the locator element with the referenced country element). Analogously, the xlink:to attribute is replaced with an organization subelement.*
- *drop-element insert-nothing for the country and organization locator elements (dropping them completely).*

```
<memberships>
  <membership>
    <!-- originally: xlink:from="B" xlink:to="org-EU" +
              together with types/element names of the referenced locator elements -->
    <country id="B">
       <!-- with attributes and contents of
            document(".../countries.xml")//country[@car_code='B'] -->
    </country>
    <organization id="org-EU">
       <!-- with attributes and contents of
            document(".../organizations.xml")//organization[@abbrev='EU'] -->
    </organization>
  </membership>
    ⋮
</memberships>
```

Additionally, as shown in Example 16, in mondial-organizations.xml, the <seat xlink:type="simple"> subelements are resolved by the corresponding document("mondial-city-XXX.xml")/city elements; analogously for the capital subelements in mondial-countries.xml.
Then, the query from Examples 2 and 15 reads in XQuery syntax in the virtual transparent model as

```
FOR $ms IN document("mondial-memberships.xml")/ms:memberships/ms:membership,
    $org IN $ms/ms:organization,      <!-- dereferencing the arc element -->
    $abbrev IN $org/@org:abbrev,
    $seatname IN $org/org:seat/cty:name,      <!-- dereferencing the simple link -->
    $capname IN $ms/ms:country/ctry:capital/cty:name   <!-- dereferencing the simple link -->
WHERE $capname = $seatname
RETURN <result org = $abbrev city = $seatname/>
```

Analogously to Example 16, there is also the possibility to replace the from and to attributes by reference attributes (that point to copies of the respective referenced elements in the virtual instance).

Example 26 (Out-of-Line Links, from and to as Attributes) *Consider again Example 13. In contrast to Example 25, we present a modeling of the from and to attributes of the arcs by attributes:*

- *keep-element for the surrounding (root) memberships element*
- *keep-element, from-attribute and to-attribute for the membership arc elements: the arc elements are kept and the xlink:from attribute is replaced by a reference to the elements that result from evaluating the locators. Analogously for the xlink:to attribute.*
- *keep-element insert-contents for the country and organization locator elements (which are then used as targets of the reference attributes in the arcs.*

In the resulting virtual instance, each arc contains an attribute country and an attribute organization that point to the respective elements. Note that the locators have been kept and resolved and provide the targets for the reference attributes. Note also that the attribute names coincide with the names of the locator elements:

```
<memberships>
  <country id="B">
    <!-- with attributes and contents of
         document(".../countries.xml")//country[@car_code='B'] -->
  <country id="D"'>
    <!-- with attributes and contents of
         document(".../countries.xml")//country[@car_code='D'] -->
  <organization id="org-EU">
    <!-- with attributes and contents of
         document(".../organizations.xml")//organization[@abbrev='EU'] -->
  <organization id="org-UN">
    <!-- with attributes and contents of
         document(".../organizations.xml")//organization[@abbrev='UN'] -->
  <membership country="B" organization="org-EU" membership_type="member"/>
  <membership country="B" organization="org-UN" membership_type="member"/>
</memberships>
```

Then, the same query reads as

```
FOR $ms IN document("mondial-memberships.xml")//membership
WHERE $ms/@organization/seat/name = $ms/@country/capital/name
RETURN <result org = $ms/@organization/@abbrev city = $ms/@organization/seat/name>
```

Next, we give an example that shows that the default setting – that inserts the contents of the referenced element into the locator – is not the preferable way if

- the locator element describes an object of its own,
- and then references another element as a property.

Example 27 *Consider an event consulting company that organizes events with artists. Such events take places in halls or stadiums that are located in cities. They combine an XML instance* artists.xml *that contains artists and groups with their names, adresses etc. with the* cities.xml *file.*

```
<!ELEMENT schedule (artist*,place*,event*)>
<!ATTLIST schedule xlink:type (simple|extended|locator|arc) #FIXED "extended" >

<!ELEMENT artist EMPTY>
<!ATTLIST artist xlink:type (simple|extended|locator|arc)  #FIXED "locator"
                 xlink:href CDATA #REQUIRED
                 id         ID #REQUIRED >

<!ELEMENT place EMPTY>
<!ATTLIST place xlink:type (simple|extended|locator|arc)  #FIXED "locator"
                xlink:href CDATA #REQUIRED
                name       CDATA #REQUIRED
                id         ID #REQUIRED >

<!ELEMENT event EMPTY>
<!ATTLIST event xlink:type   (simple|extended|locator|arc)  #FIXED "arc"
                date         CDATA #REQUIRED
                xlink:from   IDREF #REQUIRED
                xlink:to     IDREF #REQUIRED >
```

The file schedule.xml *consists now of only one* schedule *element:*

```
<schedule>
  <artist id="cking" xlink:href="artists.xml#//artist[name='Carole King']"/>
  <artist id="jmj" xlink:href="artists.xml#//artist[name='Jean Michel Jarre']"/>

  <place id="carnegie" name="Carnegie Hall"
```

```
        xlink:href="…/cities-USA.xml#//city[name='New York']"/>
    <place id="pyramids" name="Pyramids"
        xlink:href="…/cities-ET.xml#//city[name='El Giza']"/>

    <event date="18-06-1971" xlink:from="cking"  xlink:to="carnegie"/>
    <event date="31-12-1999" xlink:from="jmj"  xlink:to="pyramids"/>
</schedule>
```

i.e., to keep the name of the place *locator element, and its attribute and then insert the referenced city element into it.*

- *extended links: use the default* keep-element
- *locators: use the default* drop-element insert-nothing
- *arcs:* keep-element*,* keep-from*,* from-contents *and* keep-to*,* to-elements *for the* event *arc elements: the arc elements are kept. The* xlink:from *attribute is replaced by the referenced* artist *subelement (combining the locator element with the referenced element in* artists.xml*). The* xlink:to *attribute is replaced with a* place *subelement (taking the element hull – the name and the* name *attribute – from the locator) that* contains *a* city *element taken from* cities.xml*.*

The virtual *entry for an event looks as follows:*

```
<event date="18-06-1971"
    <artist id="cking">
        <!-- attributes and contents of document("artists.xml")//artist[name='Carole King'] -->
    </artist>
    <place id="carnegie" name="Carnegie Hall"                    <!-- the former locator -->
        <city ... >                                         <!-- the referenced element -->
            <!-- attributes and contents of document("cities-USA.xml")//city[name='New York'] -->
        </city>
    </place>
</event>
```

The population of cities where Carole King gives a concert can then be selected by

```
//event[artist/name="Carole King"]/place/city/population
```

There are also cases where the attributes and contents of the locator element should be combined directly with the attributes and contents of the referenced element; similar to Example 19.

So far, either the name of the locator or of the referenced element was used for the subelement in the virtual model. Sometimes, arcs describe different kind of relationships such that the referenced objects have different *roles* in different arcs. Then, it is favorable to allow that the arcs specify the names of the subelemements or attributes according to their roles.

Example 28 *Consider a database* people.xml *that contains data about famous historical people, e.g. birthday, profession, their works etc. Another database* relations.xml *consisting of out-of-line links provides their relationships, e.g., parent/children, teacher/student etc.*

Here, the elements in people.xml *are all of type* person*. The locators in* relations.xml *must also have a general name, since a person X can be mentioned simultaneously as a son of Y and the teacher of W (we also use* person *as locator name). The arcs then describe the individual relationships and assign* roles *to the referenced locators/elements:*

```
<!ELEMENT relationships (person*,descent*,education*)>
<!ATTLIST relationships xlink:type (simple|extended|locator|arc) #FIXED "extended" >

<!ELEMENT person EMPTY>
<!ATTLIST person xlink:type (simple|extended|locator|arc)  #FIXED "locator"
                 xlink:href  CDATA  #REQUIRED
                 id          ID #REQUIRED  >
```

```
<!ELEMENT descent EMPTY>
<!ATTLIST descent xlink:type    (simple|extended|locator|arc)   #FIXED "arc"
                  xlink:from    IDREF  #REQUIRED
                  xlink:to      IDREF  #REQUIRED
                  dbxlink:transparent #FIXED "keep-element insert-elements
                                  drop-from from-elements drop-to to-elements"
                  dbxlink:from-role #FIXED "father"
                  dbxlink:to-role #FIXED "son"  >

<!ELEMENT education EMPTY>
<!ATTLIST education xlink:type    (simple|extended|locator|arc)   #FIXED "arc"
                    xlink:from    IDREF  #REQUIRED
                    xlink:to      IDREF  #REQUIRED
                    dbxlink:transparent #FIXED "keep-element insert-elements
                                    drop-from from-elements drop-to to-elements"
                    dbxlink:from-role #FIXED "teacher"
                    dbxlink:to-role #FIXED "student"  >
```

Consider the following sample instance:

```
<relationships xlink:type="extended" >
   <person xlink:type="locator" xlink:href="persons.xml#//person[name='X']" id="X"/>
   <person xlink:type="locator" xlink:href="persons.xml#//person[name='Y']" id="Y"/>
   <person xlink:type="locator" xlink:href="persons.xml#//person[name='Z']" id="W"/>

   <descent xlink:type="arc" xlink:from="X" xlink:to="Y"/>
   <education xlink:type="arc" xlink:from="Y" xlink:to="W"/>
   ⋮
</relationships>
```

*The locators are dropped (due to the default specification of "*drop-element insert-nothing*"). The arc elements are kept, the* from *and* to *attributes are replaced by inserting the referenced elements (dropping the additional element hull provided by the locators) – renamed according to the given values of* from-role *and* to-role*. An excerpt of the intended virtual instance then looks as follows:*

```
<relationships>
   <descent>
      <father id="X" >
         <!-- attributes and contents of document("persons.xml")//person[name='X']" -->
      </father>
      <son id="Y" >
         <!-- attributes and contents of document("persons.xml")//person[name='Y']" -->
      </son>
   </descent>

   <education>
      <teacher id="Y" >
         <!-- attributes and contents of document("persons.xml")//person[name='Y']" -->
      </teacher>
      <student id="W" >
         <!-- attributes and contents of document("persons.xml")//person[name='W']" -->
      </student>
   </education>
```

```
      ⋮
</relationships>
```

4.6 XLinks and Data Integration [Sketch]

Extended links – as a sequence of locators – can also be used for collecting the contents of a large element from several sources. Here, we use an example that extends our MONDIAL world: some other database does not only describe cities as above, but adds hotel information from some other sources:

```
<country car_code="B">
  <capital href="file:cities-B.xml#//city[name='Brussels']"/>
  <city>
    <fromMondial href="file:cities-B.xml#//city[name='Brussels']"/>
    <fromHotels href="file:hotels#//hotel[city='Brussels']"/>
  </city>
</country>
```

Here, the transparent model should contain one city element per city (thus, the extended link element provides the element name) whose contents is (i) the *contents* of the city element in the MONDIAL source, and (ii) all hotel elements of the hotels source.
A possible XQuery statement selects all hotel names in Brussels:

```
FOR $c IN document("mondial-countries")/city[name="Brussels"],
     $hotelname in $c/hotel/name
RETURN <result name = $hotelname/>
```

The above example shows that there is close relationship a between data integration using the global-as-view approach, and the use of XLink elements. We will show later that appropriate specifications of XLink elements can be used for defining an XML instance wrt. an external schema.

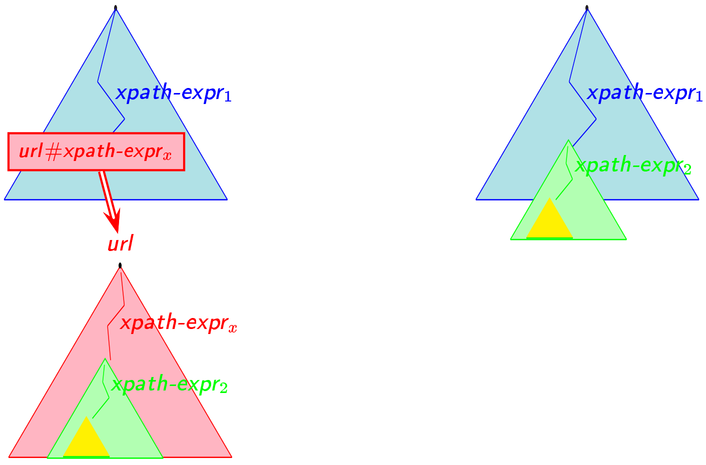
5 View Evaluation and XML Information Server Cooperation

For evaluating XPath expressions that navigate through link elements, there are several possibilities, concerning the issues, *when* and *where* the views defined by links are evaluated. Thus, when links are involved in queries, several queries/views have to be evaluated, and possibly combined (cf. Figure 2):

- the link *url#xpath-expr*$_x$ defines views on different levels:
  - wrt. the source document, the whole link is a view (in the sense of a virtual tree extension),
  - and wrt. the referenced url *url*, the XPointer portion *xpath-expr*$_x$ defines a view (in the sense that it is freqently queried).
- in general, a query does not need the whole result of the link, but in course contains a query *against* it. Thus, the query which is stated against the source document can be decomposed as *xpath-expr*$_1$/*xpath-expr*$_2$ where the result set of *xpath-expr*$_1$ contains the link element and *xpath-expr*$_2$ is a "subquery" which is evaluated against the view which is defined by the link.

Thus, for the actual data exchange, several possibilities of materializing one of the views emerge:

- transfer the whole contents of *url* and evaluate *xpath-expr*$_x$/*xpath-expr*$_2$.
- submit the query *xpath-expr*$_x$ to the server at *url* who materializes and transfers the result. Then, evaluate *xpath-expr*$_2$ against it.
- submit the query *xpath-expr*$_x$/*xpath-expr*$_2$ to the server at *url* (see Section 5.2). In this case, the views defined by the link remain completely virtual also during evaluation.



(real DM)

(transparent DM)

Figure 2: Queries over XLink Elements

When the target of the link resides on a remote server, the actually applicable strategies depend on the provided services (i.e., whether the whole XML document is accessible, and/or what queries can be answered), cf. Section 6.2.

We first sketch two potential evaluation strategies, and then identify and investigate systematically the parameters for specifying the behavior of link evaluation.

5.1 Local Evaluation

Current XML querying tools are restricted to evaluate XPath expressions wrt. the DOM-resident XML tree (which may contain several documents). When a query is parsed, the documents occurring in expressions of the form document(*url*) are added to the DOM to be evaluated. When handling links, the XPointers giving the urls and XPath expressions to be evaluated are not contained in the query, but in the data, thus they are only known at runtime. The XQuery pattern below describes how an actual query *xpath-expr*$_1$/*xpath-expr*$_2$ where the result set of *xpath-expr*$_1$ consists of link elements can be evaluated in Dynamic SQL or JDBC style:

```
<!-- evaluate   "FOR X IN xpath-expr1/xpath-expr2"   -->
FOR $xlink IN xpath-expr1
LET $xpointer = $xlink/@xlink:href,
    $url = string-before("#",$xpointer),
    $path = string-after("#",$xpointer),
    $view = document($url)/$path    <!-- XPath expression over variables -->
FOR $x IN $view/xpath-expr2
    ⋮
```

Such queries are not supported by the current querying languages. Using XQueryX, the above behavior can be implemented in XSLT [XSL99].

5.2 Remote Evaluation: Keeping the Views Virtual

When evaluating *xpath-expr*$_1$/*xpath-expr*$_2$ as above, for a link

```
<foo xlink:type="simple" href="url#xpath-exprx" .../>
```

belonging to the result set of *xpath-expr*$_1$, neither the document at *url* nor the result set of *url*#*xpath-expr*$_x$ are actually needed, but only used for evaluating *xpath-expr*$_2$ over the result set. Thus, instead the rewritten query

xpath-expr$_x$/*xpath-expr*$_2$

can be evaluated against *url* to answer the original query. In general, the result will be much smaller than when evaluating *xpath-expr*$_x$ – and the computation is done at the server which is potentially optimized wrt. its database.

Example 29 *In our example, the source* mondial-memberships.xml *contains locator elements like*

<country id="B" xlink:href=".../countries.xml#//country[@car_code='B']" />

that are used when answering the query

(∗) $capname IN $ms/@xlink:from⇒ms:country/@xlink:href⇝ctry:country/
ctry:capital/@xlink:href⇝cty:city/cty:name,

for the above XPointer, the whole ctry:country *element with* car_code *"B" is returned, and then queried by*

ctry:capital/@xlink:href⇝cty:city/cty:name .

Instead, countries.xml *can be queried for*

//ctry:country[@ctry:car_code='B']/ctry:capital/@xlink:href⇝cty:city/cty:name

which will in turn query cities-B.xml *with*

//cty:city[cty:name="Brussels"]/cty:name

*which just returns "*Brussels*" which is also the answer to (∗).*

XML Information Server Cooperation. As already stated, the applicability of both strategies depends on the services provided by the remote data source, cf. Section 6.2. As a straightforward strategy, the client will first try to send the rewritten query *xpath-expr*$_x$/*xpath-expr*$_2$ via HTTP to the server at *url* where it will be answered if the server is an XML database system. If the query is not answered, the client will try to access the XML document, parse it into its own database, and evaluate the query by itself. More involved strategies that are tailored to the needs of a given application are described in the following.

Next, we investigate the "when" aspect, i.e., if the result sets of link elements should be prefetched when parsing an XML instance, or if links are evaluated on demand.

5.3 Materializing and Prefetching Results

Similar to relational views, strategies for saving computational effort when answering queries apply to the views induced by links. Note that here, not only computational expenses (either at the local or the remote server) may be saved, but also internet communication can be reduced.

Since the required links are already known when parsing an XML instance, already at this stage, the whole virtual tree can be materialized locally: when the view is "defined" by parsing an XML document, its extension can be computed (either at the local or the remote server) and stored. Then, queries can be evaluated based on the precomputed view. Note that even the whole remote XML instance can be stored locally, which may reduce the internet communication in case that there are several links that use the same url. In both cases, no internet access is required at query answering time. On the other hand, the actual evaluation of queries wrt. the materialized link views must then be done locally wrt. the cached data.

Another possibility is to cache the answer when a query against the remote source via a link is evaluated for the first time. Again, either the view defined by the link (i.e., the result of *xpath-expr*$_x$) can be materialized as a subtree (and used later also for other queries), or the answer to a given query (i.e., *xpath-expr*$_x$/*xpath-expr*$_2$) is stored, keeping the link subtrees virtual.

Similar to relational databases, if an answer or view is cached, the consequence is that subsequent answers use the stored state instead of the current state of the remote source. On the other hand, this guarantees that all queries are evaluated wrt. the same state of the remote source.

Summarizing, when querying links, several parameters should be specifiable:

- when is the link evaluated (at parse time or at query answering time),
- where does the computation take place (locally or at the remote server), and
- which (intermediate) results should be stored and reused.

Furthermore, there are interferences between the above options. In the subsequent section, we propose an extension to the xlink: namespace that allows for specifying the behavior of links wrt. querying. For these extensions, we use again the dbxlink namespace (denoting the database aspect of XLink).

6 Specification Parameters for Evaluating Links

XLink [XLi00] defines several attributes for link elements that specify the *behavior* of the link element, i.e., *when* it should become "activated" and *what* happens then. In the current version, this behavior is tailored to the use of links when browsing, it does not cover the requirements of querying XML instances. As defined above, there are several aspects for specifying the behavior of links wrt. *querying*:

Activating Event: The activating event of links is considered in XLink with the dbxlink:actuate attribute: the value auto states that the link is activated when it is parsed, whereas user states that it is activated by the user (HTML: clicking).

In our context, auto means that the XPointer is evaluated when the node containing it is parsed whereas user denotes that it is evaluated when it is used by a query.

We add an dbxlink:actuate="noaction" alternative that does nothing: the unresolved link remains – as a kind of view *definition* – in the answer tree, but without the dbxlink:actuate="noaction" attribute. The intention here is that the application/user who stated the query should decide (and possibly pay) by himself when to finally resolve the link. Especially, information servers will use this alternative (thus, returning unresolved links to be processed by the client).

Evaluation: As described above, for the evaluation – both the evaluation of the link when parsing, or the evaluation of the link together with the remaining query on demand – there are several possibilities:

- dbxlink:eval="local" means that the whole target document is requested and *xpath-expr*$_x$ (when parsing) or *xpath-expr*$_x$/*xpath-expr*$_2$ (when answering a query) is evaluated locally.
- dbxlink:eval="distributed" states that *xpath-expr*$_x$ is submitted to the remote server, and the result (which is the view represented by the link) is transferred. When a query is evaluated, the remaining *xpath-expr*$_2$ is evaluated locally.
- Finally, dbxlink:eval="remote" means that *xpath-expr*$_x$ (when parsing) or *xpath-expr*$_x$/*xpath-expr*$_2$ (when evaluating a query) is submitted to the remote server for evaluation, and only the final result is transferred.
- dbxlink:eval is `NMTOKENS`, thus, combinations like dbxlink:eval="remote local" mean that first, it is tried to make the remote server answer the complete query. If this fails, the document is requested and the evaluation is done locally.

Answer Caching: When computing the result or accessing the target server (for the complete document), the obtained data may be cached:

- dbxlink:cache="complete" parses the *whole* target *document* (useful in case that there are many links to the document, such that it makes sense to regard it as a part of the database) and stores it (centrally) in the local XML database. When later a query traverses the link, it has only to evaluate *xpath-expr*$_x$ on the already parsed document.
- dbxlink:cache="transparent" replaces the link by the result set of document(*url*)/*xpath-expr*$_x$ (i.e., materializing the answer as in Example 25).
- dbxlink:cache="answer" stores the answer to a query document(*url*)/*xpath-expr*$_x$/*xpath*$_2$, against a remote source and keeps it somewhere in the database (useful if the same link or

query occurs several times).

- default is dbxlink:cache="none".
- dbxlink:cache is NMTOKENS, thus, combinations like dbxlink:cache="complete answer" mean that the whole document is cached (in case there are many links to it) and also the answers to individual queries are cached.

Note that the caching specifications strongly depend on the evaluation specifications: only such information can be cached that is actually communicated to the server; on the other hand, caching at rough granularity makes only sense if the information is actually used by subsequent query evaluations:

- dbxlink:cache="complete" makes only sense for dbxlink:eval="local",
- dbxlink:cache="transparent" (materializing the link view) makes sense for dbxlink:eval="local" and dbxlink:eval="distributed".

6.1 Runtime Behavior

The actual evaluation of links happens either when parsing a document into the local XML database, or when answering queries.

Parsing. When parsing a document the links with dbxlink:actuate="auto" are evaluated and cached: If dbxlink:cache="complete", the whole remote document is requested and parsed; if dbxlink:cache="transparent", the view defined by the individual link is materialized as a subtree. In the latter case, the value of dbxlink:eval specifies where the actual computation of document(*url*)/*xpath-expr*$_x$ is carried out. Note that links specified as dbxlink:cache="answer" cannot be precomputed at parse time, since there is not yet any query.

In the *abstract* data model, the resulting subtrees are replaced for the link elements, but the actual implementation may store them separately.

Evaluating Queries. When the database is actually used for evaluating queries, the answers that have been cached either from parsing the document, or from previous queries are reused. Specifying dbxlink:cache="none" disables caching for a query, e.g., when the remote data sources may change and the current state is required. In other cases, either (i) the link is evaluated against the cached document (for dbxlink:cache="complete"), or (ii) the materialized subtree that replaced the link is used (for dbxlink:cache="transparent"), or (iii) the cached answer from a previous evaluation of the same query is used. In all cases, the cached data may also be stored from another query that uses the same remote data source, link, or the same query against the link.

If there is not yet a precomputed answer (including the case dbxlink:cache="none"), the evaluation strategy given by dbxlink:eval specifies how the answer is computed, and according to dbxlink:cache, the intermediate or final results are cached.

Possible Combinations. The table given in Figure 3 shows which actions are necessary when a link is traversed the first time, and all subsequent times for all possible combinations of specification parameters.
The following strategies are the cornerstones of the spectrum:

- The strategy (dbxlink:actuate="auto" dbxlink:cache="transparent answer") guarantees the best performance for the query answering tasks: all link views are precomputed as transparent subtrees after parsing the document, and later, also all answers are added to the cache.
- The strategies (dbxlink:actuate="user" dbxlink:cache="none" dbxlink:eval="remote") and (dbxlink:actuate="user" dbxlink:cache="answer" dbxlink:eval="remote") are the "leanest" ones: only the necessary computations are done (by the remote server). The first variant does not cache anything (minimizing space, and always using the current state of the remote data source), the second one caches the answers (minimizing communication).
- On the other hand, specifying (dbxlink:actuate="auto" dbxlink:cache="complete transparent" dbxlink:eval="local") makes the system as independent as possible from the remote source,

which is especially useful if transient sources are used, or if the internet connection is not persistent.

	dbxlink:actuate="auto"								
(dbxlink:eval × dbxlink:cache)	lc	lt	la	ln	dt	da	dn	ra	rn
use cached document	33	–	–	–	–	–	–	–	–
access remote document	–	–	9-	99	–	–	–	–	–
add document to cache	–	–	–	–	–	–	–	–	–
use transparent subtree	–	11	–	–	11	–	–	–	–
compute link view remote	–	–	–	–	–	1-	11	1-	11
transfer link view	–	–	–	–	–	4-	44	–	–
compute link view locally	33	–	3-	33	–	–	–	–	–
replace link by subtree	–	–	–	–	–	–	–	–	–
use cached answer	–	–	-1	–	–	-1	–	-1	–
compute answer remote	–	–	–	–	–	–	–	1-	11
transfer answer	–	–	–	–	–	–	–	1-	11
compute answer locally	22	22	2-	22	22	2-	22	–	–
add answer to cache	–	–	1-	–	–	1-	–	1-	–

	dbxlink:actuate="user"								
(dbxlink:eval × dbxlink:cache)	lc	lt	la	ln	dt	da	dn	ra	rn
use cached document	-3	–	–	–	–	–	–	–	–
access remote document	9-	9-	9-	99	–	–	–	–	–
add document to cache	3-	–	–	–	–	–	–	–	–
use transparent subtree	–	-1	–	–	-1	–	–	–	–
compute link view remote	–	–	–	–	1-	1-	11	1-	11
transfer link view	–	–	–	–	4-	4-	44	–	–
compute link view locally	33	3-	3-	33	–	–	–	–	–
replace link by subtree	–	1-	–	–	1-	–	–	–	–
use cached answer	–	–	-1	–	–	-1	–	-1	–
compute answer remote	–	–	–	–	–	–	–	1-	11
transfer answer	–	–	–	–	–	–	–	1-	11
compute answer locally	22	22	2-	22	22	2-	22	–	–
add answer to cache	–	–	1-	–	–	1-	–	1-	–

1st entry: first time - : no action required
2nd entry: all subsequent times n: action required (estimated cost: n)
Note that multigranularity caching (e.g., dbxlink:cache="lcta") adds additional variants.
Also, previous operations may have cached useful information (e.g., pages)

Figure 3: Required Actions when Traversing a Link

Using dbxlink:cache="complete transparent" provides a double caching which is useful when the target document is referenced by different XPointers, together covering large parts of the target document, and each of the XPointers occurs in several links.

Note that the actual runtime behavior can also be influenced by different specifications for different links in a document, e.g., when another link already accessed the complete document or another link has already precomputed the answer.

6.2 Constraints

In practice, the functionality which is provided by the remote servers constrains the possible alternatives:

"Lazy" Servers. If only a plain XML document (e.g., as ASCII) is provided, the client that initiated the query has to access the document and evaluate the XPointer expression by itself (dbxlink:eval="local"). Then, the client has to decide if he accepts the link strategies defined in the document.

"Hiding" Servers. If the referenced document is located on a server that does not publish the whole document, but only answers queries, dbxlink:eval="local" (and also dbxlink:cache="complete") is not applicable.

Note that even servers that answer only a restricted set of XPath queries are possible. In this case, it may be necessary to *rewrite* queries according to the provided views.

Costs. Additionally, for the user of information services, the costs associated with an information source are relevant:

- dbxlink:eval="local" (especially, without dbxlink:cache="complete") requires transferring the whole XML document may be expensive when the costs depend on the size of data,
- Any strategy without dbxlink:cache="complete" may be expensive when the costs depend on the number of queries, especially, if no caching at all is used.
- dbxlink:eval="remote" is expensive wrt. the remote computation time, especially, if the answers are not cached.
- Obviously, dbxlink:cache="none" is potentially expensive in all cases: with dbxlink:eval="local", the transferred volume is excessive (always, the whole XML documents are transferred), and with dbxlink:eval="remote", the remote computation time can be considerable.

So far, the strategies have been investigated from the point of view of the "client".

6.3 Information Providers

For an information provider, dbxlink:actuate="noaction" assures that he will not have any effort (neither computational resources, nor costs) for resolving the links in the documents he provides. Obviously, then, the provider is not able to answer XPath queries, not only because of his decision not to have any expenses wrt. computational resources, but also since the links in the documents are opaque for him. The provider must deny any access with dbxlink:eval="remote" and dbxlink:eval="distributed" (hopefully, the client specified a fallback to dbxlink:eval="local" for links pointing to this server).

Thus, clients may only access the whole XML document and to provide an overriding specification for dbxlink:actuate="noaction" how to handle links (pointing to third party documents) in this document.

6.4 Data Model

Another question is, whether virtual subelements via links should be considered as descendants. The idea of the transparent data model requires this feature: Regarding the views defined by links as subtrees that replace the original link element, the data model, i.e., the definition of the tree, must be "continued" with these subtrees. Then, queries are robust against some restructuring of information servers. On the other hand, the set of descendants may become arbitrary large. Additionally, navigation using // has to resolve (transitively) all links. Thus, it is desirable that the behavior of a link in this aspect can be defined (cf. Section 6.4).

Descendants strategy: XLink does not address this issue. We propose an additional attribute dbxlink:descendants that can be used to specify if the link should be transparent wrt. the descendant axis:

- the link itself may specify if it wants to be transparent wrt. the descendants relation ("yes"/"no"),
- the application may specify which elements in virtual subtrees resulting from transparent links are regarded as descendants: all, none, sameserver (those resulting from links where the target url is on the same server as the source url), or predefined (use the strategy given by the link).

When a link is specified with dbxlink:descendants="yes", and dbxlink:descendants="all" is specified by the application, the link must be considered whenever it is involved in a descendant navigation step. Thus, it is often preferable to materialize it, or to use additional metadata information to check if it may yield relevant answers (see Section 6.5).

Similar considerations are concerned with navgation along other axes: when a referenced element is integrated into the virtual model, what node belong to its parent/ancestor and sibling axes? This issue is currently still open. Probably there will be additional extensions to the dbxlink namespace for covering this aspect.

Referential Integrity. Another problem – which is in some sense concerned with the data model – is the maintenance of referential integrity in an interlinked network of sources: links are expected to point to one or more elements that have a certain connection to the parents of the link elements (e.g., the seats of organizations). Here, the $\mathit{xpath\text{-}expr}_x$ query part of the link has to use values for identifying these elements. In the relational model, this is done using *keys*, leading to the well-known referential integrity maintenance issues. Since XML Schema also provides a notion of keys and key references, these are natural candidates for use in link references. Thus, it is likely that similar problems will arise.

6.5 Optimizations and Perspectives

Resource Descriptions. If the provider of a referenced XML document also provides a *resource description*, it is often even possible to decide a priori *whether* the actual query may be successful.

Especially, when links are regarded as transparent wrt. the descendant axis, for expressions of the form .../*/nodetest*, the simple information whether the linked document can contain elements that satisfy the *nodetest* can save the expense of querying the linked data source.

Example 30 *The XPath expression document("mondial-countries")//inflation (which selects all inflation elements by following the links to all documents described above) does not need to follow the links to the city documents since these do not contain any inflation element.*

With a more detailed metadata description, e.g., in *RDF (Resource Description Format)*, the evaluation of many queries can be optimized.

XML-Aware Web Caches. In the context of frequently used remote information sources, the current "local" caching may create redundant cached copies of views. An obvious optimiziation is to use caches at the remote servers that store the results of frequent queries.

Another possibility is to use XML-aware caches technology in the Web which allows to store (url,query,answer) tuples that occur frequently. In contrast to the above alternative, this can – at least in a restricted way – be implemented by the user, e.g., in a proxy server for his company. This approach is e.g. investigated in [LM01].

7 LoPiX

Currently, the LoPiX system [May01b, May01a] is used for investigating the handling of links in an XPath-based environment. LoPiX is an environment for using XPathLog (an extension of XPath, see [May01d]) as a language for XML data integration. LoPiX (cf. Figure 4) consists of a central XPathLog evaluation engine, a storage module that implements the internal data model, an Internet access module, and an optional XML/XPath interface for electronic data interchange with external XML-based systems. For the present paper, the data-driven Web access functionality and its transparent resolving of links is relevant.

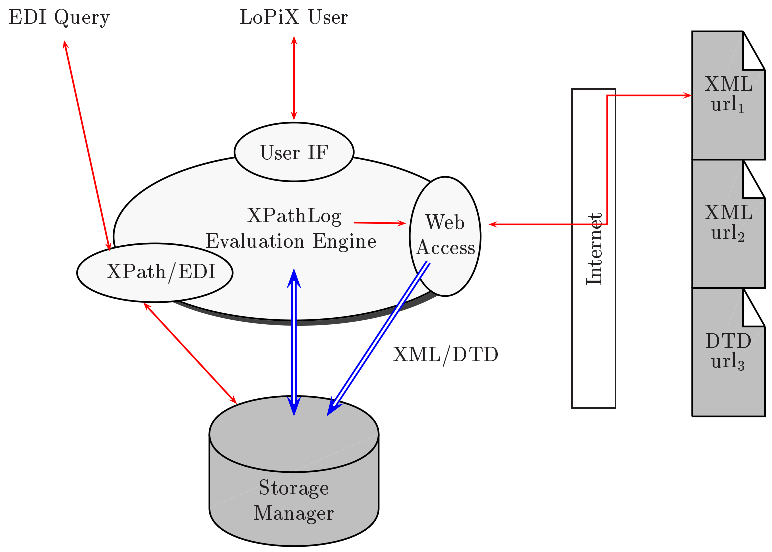


Figure 4: LoPiX architecture

The internal database can be extended by new XML documents depending on information and links (i.e., pure urls or links) found in already known documents: Whenever for some url *url*, the built-in active method *url*.parse@(xml) is called, the document located at *url* is automatically loaded as an XML tree. All link elements with dbxlink:actuate="auto" are preprocessed according to the values of dbxlink:cache and dbxlink:eval.

When a navigation step traverses a link element such that dbxlink:actuate="user", the link is evaluated according to the values of dbxlink:cache and dbxlink:eval. Thus, when querying linked XML sources, the actual query syntax is independent from the values of dbxlink:actuate, dbxlink:cache, and dbxlink:eval since these specifications are completely handled by the evaluation component.

8 Conclusion

Related Work. In contrast to, e.g., the relational model or previous semistructured data models, XML and XLink provide a construct for seamless integration of *view definitions* into the data via *links*. This semantics raises new aspects for querying languages. Nevertheless, the most prominent ones, i.e., XML-QL [DFF+99], and Quilt/XQuery [CRF00, XQu01], do not support links as defined by XLink. XSLT [XSL99] allows for handling links by special patterns (to be specified manually).

[(ed01] investigates aspects of how to process XLink elements in stylesheets. The stylesheets also define an XML instance similar to our *transparent, logical model*. Their solution is not to add attributes to the XLinks, but to add style elements to XSL for handling links in a generic way. They do not consider traversal of links for querying.

Conclusion. We investigated possible semantics and strategies for handling them in a declarative query language. We have shown that the problem divides into two aspects:

- Data model: there is a straightforward interpretation that leads to an explicit dereferencing operator. We proposed a "transparent" semantics for XLink elements, similar to views. The transparent data model maps linked documents to a single, virtual XML instance that can be queried by standard XPath/XQuery.

- Evaluation strategies: there are several strategies for evaluating queries over linked, distributed XML sources. The applicability and preformance depends heavily on the functionality of the sources, and also on the characteristics of the queries itself – whether there are related, frequent queries etc.

  Especially when information sources are used that can answer XPath queries on their contents, on-demand resolving of links can be used to minimize the answer and to exploit the server-side optimization in an optimal way.

Both aspects of querying distributed XML documents on the Web by declarative languages require a more detailed specification of xlink-related attributes. We have defined attributes concerning the data model, evaluation time and place, and result caching that could be added favorably to the XLink language.

In addition to a proposal how to handle links in XML querying frameworks, we have identified the following areas for further investigations: (i) XML/XPath query rewriting, (ii) XML-aware Web caches, and (iii) intra-document and intra-source referential integrity maintenance.

References

[CRF00] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *WebDB 2000*, pp. 53–62, 2000.

[DFF+99] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. In *8th. WWW Conference*. W3C, 1999. World Wide Web Consortium Technical Report, NOTE-xml-ql-19980819, `www.w3.org/TR/NOTE-xml-ql`.

[(ed01] N. W. (ed.). XML Linking and Style. W3C Note `http://www.w3.org/TR/xml-link-style`, 2001.

[LM01] G. Lausen and P. J. Marròn. On processing XML in LDAP. In *Intl. Conference on Very Large Data Bases (VLDB)*, 2001.

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

[May01b] W. May. The LoPiX System, 2001. `http://www.informatik.uni-freiburg.de/~may/lopix/`.

[May01c] W. May. The Mondial Database, 2001. `http://www.informatik.uni-freiburg.de/~may/Mondial/`.

[May01d] W. May. XPath-Logic and XPathLog: A Logic-Based Approach for Declarative XML Data Manipulation. Technical report, Habilitation Thesis, Universität Freiburg, Institut für Informatik, 2001. Available from `http://www.informatik.uni-freiburg.de/~may/lopix/`.

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

[XMQ01] XML Query Data Model. `http://www.w3.org/TR/query-datamodel`, 2001.

[XPa99] XML Path Language (XPath) Version 1.0: 1999. `http://www.w3.org/TR/xpath`, 1999.

[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.

[XSL99] XSL Transformations (XSLT). `http://www.w3.org/TR/xslt`, 1999.