Querying along XLinks in XPath/XQuery: Situation, Applications, Perspectives

Erik Behrends Oliver Fritzen Wolfgang May

Institut für Informatik, Universität Göttingen, Germany
{behrends|fritzen|may}@informatik.uni-goettingen.de

Abstract. This paper summarizes the situation about using XLink for connecting XML instances. We discuss applications where XLink functionality can be useful, and derive requirements how the basic XLink technology should be supported in these scenarios. We compare several proposals dealing with interlinked XML data with our dbxlink approach which is a minimal extension to XLink and XPath, and we show how its semantics can be added to arbitrary XPath-based query engines.

1 Introduction

We start the presentation with a short introduction to XLink and the current, unsatisfying state of the art of dealing with XLinks in XQuery. Section 2 then points out tasks where XLinks can be applied successfully and show what features would be nice to be supported. We describe our dbxlink proposal and correlate it with existing proposals in Section 3. We give a high-level description here since the formal details of dbxlink can be found in [BFM06], and then report our experiences with extending a "common" XML database system with this functionality in Section 4 before closing with some concluding remarks.

XML and XLink. XML has been designed and accepted as *the* framework for semi-structured data. XML data is not required to be self-contained on an individual server, but may include *links* to XML data on other servers. Such references inside XML data can be expressed by the *XML Linking Language (XLink)* [XLi01,XLi06]. XLink provides special tags in the xlink namespace that tell an application that an element is equipped with link semantics. The well-known HTML <a href="..."> construct is a simple XLink element whose href attribute references a document. XLink defines general functionality of such references: (i) *arbitrary* elements can be distinguished as *XLink* elements, (ii) the allowed values of the href attribute are enhanced for addressing XML data, and (iii) the behavior of the link can further be specified.

The essential step in (ii) is to allow to specify the remote resource not only as usual in HTML by a URL optionally extended with an *anchor* (e.g., `http://www.example.org#foo`), but, suitable for the XML data model, for addressing *nodes*. The extended addressing functionality is provided by the XPointer Framework [XPt03] and the XPointer addressing scheme [XPt02].

XPointer in turn is based on XPath expressions: An XPointer expression of the form *url*#xpointer(*xpointer-expr*) (where the syntax of *xpointer-expr* is a slight extension of XPath) identifies a fragment inside the XML document located at *url*. E.g., the following XPointer (see also Figure 1 and Example 1 below)

```
http://www.foo.de/countries.xml#xpointer(//country[@car_code="D"])
```

addresses the node that represents Germany in http://www.foo.de/countries.xml.

XLink defines several types of links, i.e., *simple links* that provide *referencing* functionality similar to the HTML <a> element, and *extended links* that allow for *connecting* sources by *arcs*. In this paper, the focus of our interest is on *simple links*, where one XLink element with an XPath expression in place of the *xpointer-expr* references one or more nodes from a remote document. A simple link has the following form:

```
<qname xlink:type="simple" xlink:href="xpointer" further-xlink-attributes>
  content
</qname>
```

Example 1 (Simple XLinks) *The document countries.xml in Figure 1 contains basic data about countries, and for each country, cities-XX.xml (where XX is the country's car code) contains information about the cities in this country.*

Query Support for XLink References. How can the instance be queried – e.g., for finding out how many inhabitants the capital of Belgium has? Although the W3C's *XML Query (XQuery) Requirements* [XMQ05, Sec. 3.3.4/3.4.12 (References)] explicitly state that

> *"the XML Query Data Model MUST include support for references, including both references within an XML document and references from one XML document to another"*,

and XLink is a well-established W3C Recommendation, neither XPath nor XQuery support navigation along XLink references. While for intra-document references, the id(...) function does this task, and the doc(...) function allows for accessing remote documents, there is not yet complete support for XPointer in XPath/XQuery: users can select the pointer with

```
for $pointer in
doc("http://.../countries.xml")//country[@car_code="B"]/capital/@xlink:href
```

but XQuery cannot be told to resolve it.

The crucial point of handling XLink references is the evaluation of a *data item* (i.e., the value of the href attribute) as a query. This is currently not possible in XPath/XQuery, neither in the base language, nor by the functions and operators given in *XQuery 1.0 and XPath 2.0 Functions and Operators* [XPQ05].

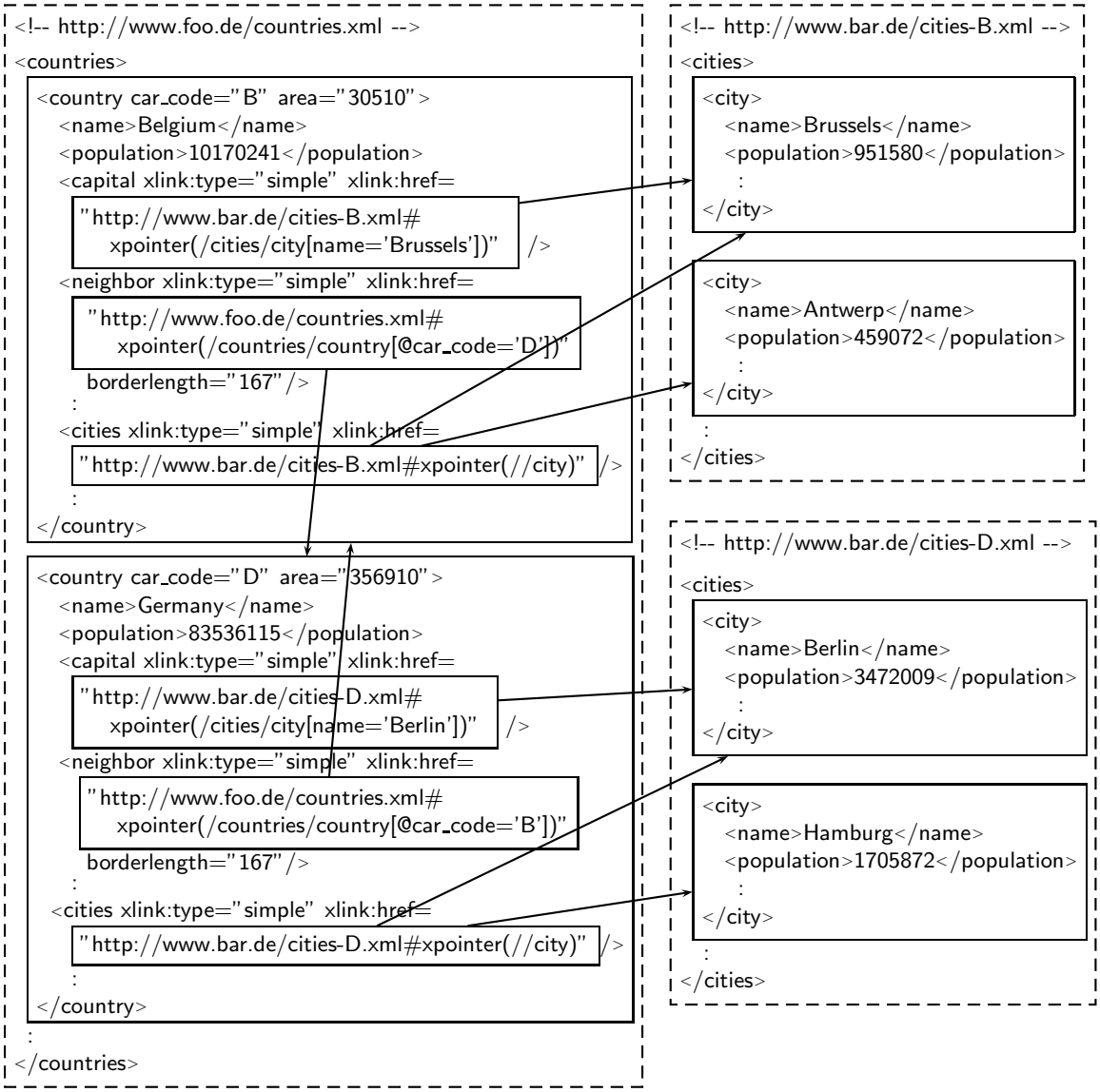

Fig. 1. Excerpt of the Distributed MONDIAL XML Database [May01b]

Simple XPointers. Simple XPointers actually consisting of an fn:id() function application of the form *url*#xpointer(id(*string*)) (equivalent to the "shorthand pointers" like http://.../country.xml#D in [XPt03]) can be resolved by combining the doc() and id() functions. In [LS04, Section 7.4.2], a solution by an XQuery user-defined function is given which is restricted to such simple XPointers:

```
declare namespace fu = "http://www.example.org/functions";
declare function fu:follow-xlink($href as xs:string) as item()*
{  let $docValue := fn:substring-before($href,"#")
   let $x := fn:substring-after($href,"#xpointer(id('")
   let $idValue := fn:substring-before($x,"')")
   return fn:doc($docValue)/fn:id($idValue)  };
```

XPath Expressions in XPointers. In the general case, instead of fn:id($idValue), any XPointer expression must be allowed:

```
declare function fu:follow-xlink($href as xs:string) as item()*
{  let $docValue := fn:substring-before($href,"#")
   let $path := fn:substring-before(fn:substring-after($href,"#xpointer("),")")
   return fn:doc($docValue)/ $path  };
```

Such functionality must evaluate a dynamically constructed XPath expression. This is not yet available in XQuery (and can also not be programmed by the current *XQuery 1.0 and XPath 2.0 Functions and Operators* [XPQ05]; note that the function given in [LS04, Section 7.4.2] explicitly returns a message "XPointer Syntax nicht unterstützt/XPointer syntax not supported" in this case).

The required functionality is available in Saxon [saxon] as an *extension function* saxon:evaluate(*string*) where the above function can be expressed as

```
declare function fu:follow-xlink($href as xs:string) as item()*
{  let $docValue := fn:substring-before($href,"#")
   let $path := fn:replace($href, "^.*#xpointer.(.*).$","$1")
   return fn:doc($docValue)/saxon:evaluate($path)  };
```

[RBHS04] proposes another XQuery extension as "execute at *url* xquery {*xquery*}". Then, queries use –similar to the id function– an *explicit* dereferencing, e.g.

```
doc("http://.../countries.xml")//country[@car_code="B"]/
                        capital/ fu:follow-xlink(@xlink:href) /population .
```

With respect to the applications where XLink references are used, we argue that *implicit* dereferencing is preferable, seeing XLink elements as embedded views, like doc("http://.../countries.xml")//country[@car_code="B"]/capital/population.

2 Applications

Data Integration. An integrated view over distributed, autonomous data can be defined according to a given target DTD or XML Schema. In this case, the integration approach is realized by the *Global as View (GAV)* [Len02] approach, i.e., queries are answered by *view unfolding* which in this case amounts to evaluating the XPointer and integrating its result into the surrounding structure. By this, also calls to Web Services can be integrated via XLink.

Example 2 *Consider a similar structure as in Example 1, but instead of the parts of the distributed* MONDIAL *database, not the "own" city data is referenced, but remote, autonomous city data residing at* http://www.geohive.com/*. Here, referencing remote data (e.g.,* http://www.geohive.com/cy/c_de.xml *for German cities) guarantees that in case that this data is updated, subsequent queries always return the most up-to-date results.*

Data Integration Process. Not only the final result of an integrated view can be expressed by XLink references: we have shown in [May05] how to carry out the integration process by partial materialization of an integrated XML instance. Nodes that are only *referenced* from the so far integrated fragment are integrated by suitable XLinks. When the integration process proceeds, the materialized fragment is minimally extended just by the structure that is generated by the integration, still referencing as much as possible the remote data-carrying nodes. Queries are actually evaluated against (i) the partially materialized integrated database, and (ii) remaining parts that reside in the original sources. This has the advantage that in case that remote data is modified, any query against the integrated model uses the up-to-date modified data.

Data Reorganization and Splitting. When XML documents grow, it is sometimes preferable or necessary to split them over several documents or even servers. In this case, the original schema should be kept, seeing the integrated document as a GAV view over the –now distributed– data. Then, the same queries that were stated against the original instance can also be used against the split-up instance.

Requirements. The above list shows that XLinks can be applied as a basic mechanism for *syntactical representation* of references in several scenarios. This basic mechanism has to be equipped with a *semantics* that supports the application and defines how actually to deal with the references:

1. modeling: how to integrate the referenced nodes with the referencing document in a logical model,
2. querying: how to express and evaluate queries against this model.

It is preferable that the result of (1) is a standard XML document according to a given target DTD or XML Schema. Then, the semantics of (2) is obvious since common XML query concepts (i.e., XPath, XQuery, XSLT) can be immediately used *without* the need for explicit dereferencing.

In contrast, the proposals described in Section 1 are directly based on the original XML structure (the *XML Infoset* [XML99]) and do not use any logical model of the XLink elements. All of them require that the query expressions include *explicit* dereferencing operations. The use of an explicit navigation operator requires non-semantic navigation steps along the xlink:href attribute. The above applications for **data integration** and **splitting** with obtaining/retaining an original DTD or XML schema are not possible with them.

Thus, a transparent modeling as an XML-to-XML transformation where the XLink elements are present only on the syntactical level, but queries navigate in a *logical* model along semantic notions is desirable.

3 Proposed Solutions

Several solutions have been proposed up to now that deal with distributed and/or linked XML documents. We start with our dbxlink approach as described in detail in [BFM06], which keeps close to XML and XPath, and then discuss other approaches that cover similar topics and can be used for such issues.

3.1 The DBXLink Approach

In [BFM06], we presented the dbxlink approach for handling distributed XML data where XLink elements are extended with attributes in the dbxlink namespace that specify the *modeling*, *evaluation strategies*, and *caching of remote query results* of the links. Here, links are *transparent*, i.e., we define a logical, transparent model for mapping distributed, XLinked XML documents virtually to an integrated XML instance: The XLink elements are seen as view definitions that integrate the referenced data within the referencing XML instance where the XLink element contains the following attributes (see [BFM06] for details):

- specification of the referenced nodes by xlink:href,
- how they are mapped into the surrounding instance by dbxlink:transparent,
- when (at parsing time or at query answering time) the XPointer is actually evaluated by dbxlink:actuate,
- where (query shipping, data shipping or hybrid shipping) the evaluation takes place by dbxlink:eval, and
- whether intermediate results are cached by dbxlink:cache.

This virtual instance can then be processed by standard languages like XPath, XQuery, or XSLT. The variety of modeling variants (e.g., replacing the link element by the referenced nodes, or keeping the link element and inserting the contents of the referenced nodes into it, or attaching the referenced nodes as reference attributes to the element that surrounds the XLink element) is formally discussed in [BFM06]. There, also implementation aspects and pitfalls (ancestor axis, cycles) are discussed.
Concerning the above scenarios, this modeling flexibility allows

- to define an integrated view over remote data sources according to a given target DTD or XML Schema, and
- splitting an existing XML instance at arbitrary edges (i.e., subelement edges and also reference attributes) while keeping the original DTD or XML Schema.

Since the logical model is an XML instance, XPath, XQuery and XSLT can be applied to it as usual. dbxlink allows for controlling when, and whether, the virtual instance is actually materialized; usually, it is not materialized, but queries are just evaluated against the logical model via appropriate algorithms.

XPath vs. XQuery and XSLT. Note that the actual work is only concerned with extending XPath for smoothly dereferencing of XLinks according to the logical model: The *addressing* of nodes is done completely within XPath. Thus, extending the XPath module of an XQuery and/or XSLT system makes this functionality also available for XQuery and XSLT. The approach has been implemented as an extension to the eXist [exi] XML database system, an open-source implementation of the common languages of the XML area, supporting XPath/XQuery as query languages.

We will discuss the general applicability of this approach in Section 4.

3.2 Comparison with Related Approaches

XLink for Browsing. Up to now, the XLink approach is primarily interpreted for browsing, as it is mirrored by the W3C XLink Recommendation [XLi01] where several attributes for link elements are defined that specify the *behavior* of the link element during browsing. The show='embed' behavior of XLink can be seen as one special case of the above approach, specified by transparent="drop-element insert-nodes" replacing the XLink by the referenced contents. In this case, also a logical model is defined that is directly materialized and presented as XHTML.

XInclude. A restricted approach for distributed documents is proposed with XInclude [XIn04]: the <xi:include href="*uri*" xpointer="*xpointer*"> element provides also a *uri* and an *xpointer*. XInclude defines a fixed XML-to-XML transformation where the xi:include elements are replaced by the corresponding included items. In fact, this model generalizes XLink's browsing behavior for show='embed', replacing the XLink by the referenced contents. The specification of XInclude also corresponds to the dbxlink specification transparent="drop-element insert-nodes" and actuate="parse", i.e., the target is included when the document is loaded/parsed, materializing the model completely.

General Investigations on Distributed Semistructured Data. In [Suc02], distributed query evaluation for general semistructured data graphs is investigated. Queries are split into *decomposed queries*, then, their parts are evaluated independently at each site, and the result fragments are reassembled. The logical modeling of [Suc02] is similar to XInclude. In [BG03], distribution of XML repositories is investigated, focussing on index structures for answering queries.

Other approaches to distributed XML data apply a schema-based distribution. There are no explicit references in the data, but schema components are associated with databases and identified by their types and key attributes. During query answering the different databases are queried (here, a database dictionary is needed which tells where the data to certain schema components can be found) and the fragments are put together in the answer.

Example 3 (Schema-based Distribution) *Consider a similar structure as in Example 1. For a schema-based distribution, all country data is still in one database, but also all city data is together in one database.*

In contrast, by using XLinks, the city data for each country can reside in an individual database, or even on different hosts. Explicit references in the data here allow for full flexibility without need for a central database dictionary.

Active XML. A general approach for integrating intensional data generated by Web Services into XML documents is proposed by *Active XML* [ABM+02]. With this technology, calls to Web Services are embedded into XML documents by <axml:call> elements.

Active XML on the one hand and XInclude or dbxlink on the other hand differ significantly wrt. generality (Active XML) and specialization (XInclude and dbxlink) and in the degree of integration with the database functionality. While XInclude and dbxlink are incremental extensions to the existing concepts of XLink and XPointer, targeting to provide a transparent data model and support XPath/XQuery for them from the database point of view, Active XML is a generic extension of functionality towards Web Services. Nevertheless, dbxlink and Active XML can be used to implement each other: on one hand, an Active XML service that implements the dbxlink modeling and takes a dbxlink-extended XLink element as input could return the appropriate XML fragment. On the other hand, XLink elements with dbxlink evaluation that refer to Web Services can be used for implementing functionality like Active XML, providing higher modeling flexibility (which is the main focus of the approach), but less operational alternatives (i.e., no active functionality).

4 Enabling XPath/XQuery Engines for Handling XLinks

Between XInclude on the one side and Active XML on the other side, the dbxlink approach is *specialized* to XLink, and provides functionality that we think is necessary and sufficient for using XLink for references between XML instances, and for querying these. In this section, we discuss how query engines have to be modified in order to handle queries on distributed, interlinked XML instances according to the dbxlink model. Recall that the queries are stated wrt. the DTD of the integrated GAV view and must be evaluated based on the original documents.

Naive Approach. An intuitive, naive approach to achieve this would consist of two steps. First, materialize the whole virtual instance induced by all interlinked XML instances wrt. the contained XLinks and their dbxlink directives. Then, tell the query engine to evaluate the given XPath expression on this new instance. This approach is not suitable for two reasons. In case of many distributed documents, it might be time-consuming to fetch all partaking XML documents and to compute the virtual instance, and usually, not the whole instance is needed to answer the given query. Even worse, the materialized view might contain cycles and thus the straightforward materialization process might not terminate.

4.1 Dynamic Query Evaluation

Given an XPath expression, we assume that it has the following form

$\mathsf{doc}(url)/\mathsf{step}_1/\mathsf{step}_2/\ldots/\mathsf{step}_n$.

In our distributed setting, we require XPath queries to start with the doc() function for specifying an XML document which shall serve as a starting point for evaluating the query. The query itself consists of *n location steps*.

There are several possibilities how XPath query engines evaluate XPath queries. We discuss different approaches and show how the navigation across XLinks can be integrated accordingly.

Stepwise Result Set Evaluation. The most common and intuitive method (which is also induced by the semantics definition by the W3C in [XPa99] or other sources, e.g., [Wad99]) for evaluating XPath queries is to subsequently apply all location steps. In every step, the set of nodes selected by the previous step is called the current *context*; in the first step, the document node is the initial context. Then, for each node of the current context, the current step is evaluated, selecting a sequence of matching items, i.e. attribute or element nodes, or atomic values that form the context for recursively applying the next step. Note that not complete intermediate results are materialized, but only local contexts on the way to the next step. Most XPath engines like Saxon, Xalan, and eXist, the native XML database system we chose for an implementation, use this strategy.

Extension of the Stepwise Evaluation. In order to implicitly replace all relevant link elements during navigation in an XML tree, thus making the navigation transparent, all subelements of every node belonging to the *context* have to be analyzed: any XLink subelement of the current context node can potentially be replaced by one or more nodes that are relevant for the next step. Thus, a kind of lookahead evaluation in order to make the required nodes available for the next step has been implemented, temporarily materializing fragments of the virtual instance on-demand.

4.2 Example Evaluation

In the general case, the navigation across XLinks takes place as follows. Consider an expression $\mathit{xpath\text{-}expr}_1/\mathit{xpath\text{-}expr}_2$, where a result node of $\mathit{xpath\text{-}expr}_1$ contains a simple XLink element with an XPointer $\mathit{url}\#\mathsf{xpointer}(\mathit{xpath\text{-}expr}_x)$. For the most "intuitive" case, assume that the remote server is capable of answering XPath queries. The query $\mathit{xpath\text{-}expr}_x$ is thus submitted to the server at *url* that transfers the result which is then mapped into the current context. Then, the local query evaluation continues with (the first step of) $\mathit{xpath\text{-}expr}_2$.

Consider again the example "capital" query whose evaluation is illustrated in Figure 2: /countries/country[@car_code="B"]/id(@capital)/population (note that we chose the modeling dbxlink:transparent="make-attribute insert-nodes" which turns the capital into a reference attribute to adhere to a "given" target DTD). In the distributed MONDIAL database (cf. Figure 1), after evaluating $\mathit{xpath\text{-}expr}_1$:= /countries/country[@car_code="B"], the capital XLink subelement is a child node

of the context element that represents *Belgium* and it has to be expanded. The rest of the query is then $xpath\text{-}expr_2$:= id(@capital)/population, and $xpath\text{-}expr_x$:= "http://dbis05/cities-B.xml#xpointer(//city[name='Brussels'])" is the XPointer expression.

As illustrated in Figure 2, $xpath\text{-}expr_x$ is sent to the remote server which returns the city node for Brussels. The screenshot in Figure 3 illustrates the communication between two servers when resolving the XPointer in the capital XLink subelement of Belgium traced by the *Apache Axis TCPMonitor*. On the left hand side, the corresponding GET request for http://dbis05/cities-B.xml//city[name="Brussels"] from the country server (ap34) to the city server (dbis05) can be seen, whose result, i.e., the XML fragment representing Brussels, is shown on the right hand side.

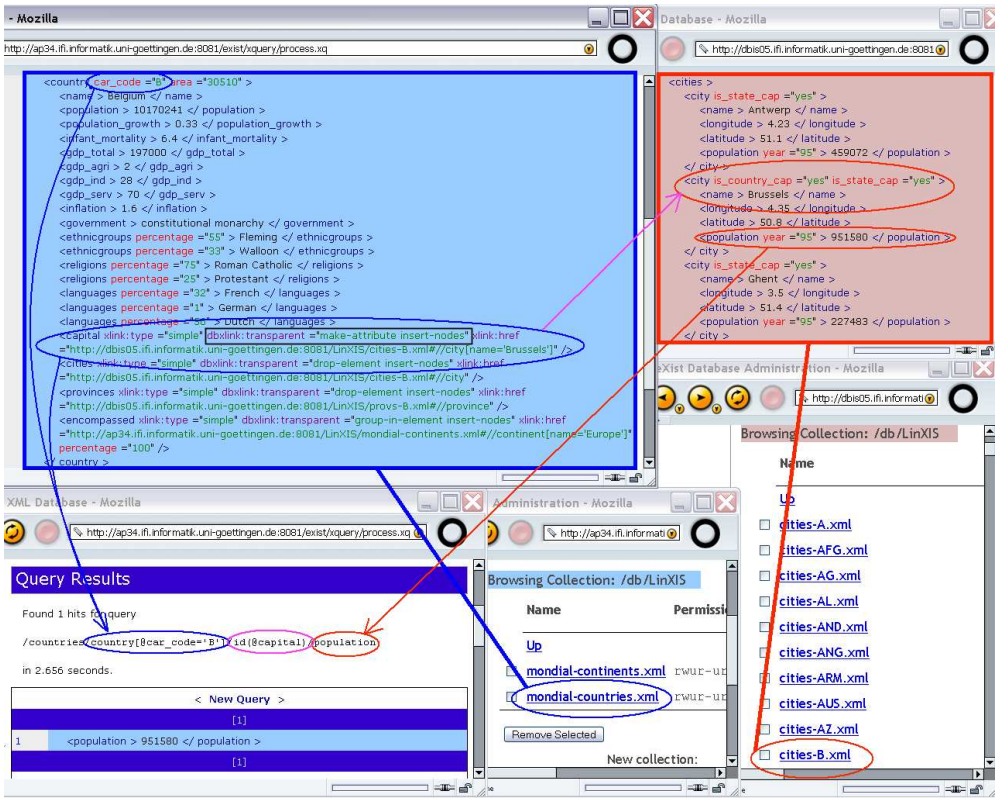

Fig. 2. Querying the Distributed MONDIAL Database

Once the local server has received the XML data for Brussels, it maps it into a reference attribute of its parent element *Belgium* (as required by the modeling dbxlink:transparent="make-attribute insert-nodes"): The new, local *Brussels*

node is extended with a (local) ID attribute with value *brus-id*. Additionally, an IDREF attribute node capital="*brus-id*" is added to the *Belgium* element in the currently materialized context. Then, the remaining part of the original query, $xpath\text{-}expr_2$ = id(@capital)/population is evaluated locally (using the new IDREF/ID attributes to navigate from the *Belgium* element to *Brussels*).

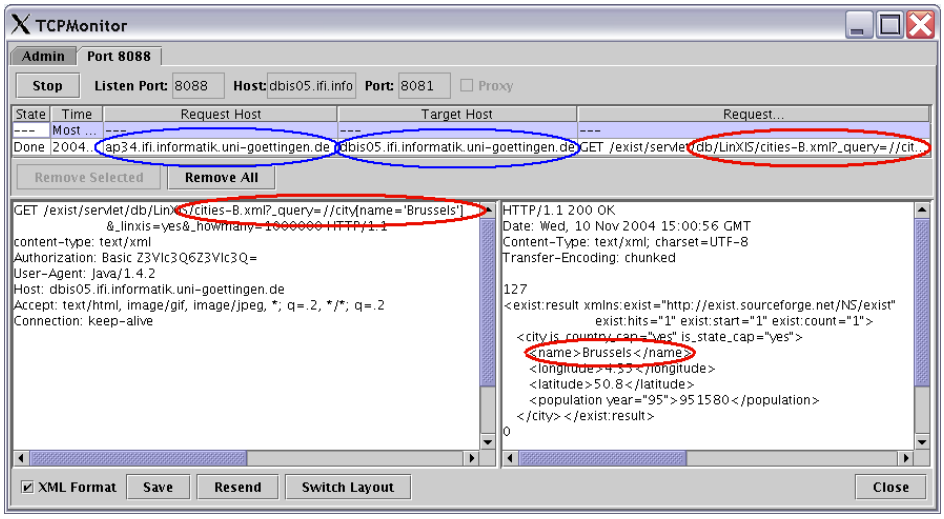

Fig. 3. Communication: Answer Shipping

4.3 Extending Alternative Evaluation Strategies

Iterator-based Evaluation. Relational database systems usually do not materialize intermediate results except when needed for aggregations; instead combinations of iterators are used that implement the algebra tree. Here, the lookahead evaluation can be covered inside the iterators that smoothly return the actual sequence of children or attributes in the logical model. Such an evaluation has been used in first experiments when extending the LoPiX system [May01a].

[GKP05] present an algorithm that reduces the worst-case complexity of XPath from exponential (in the size of the query) to polynomial time. For this, the proposed algorithm uses a kind of *tabling* via dynamic programming where earlier results are stored for later lookup. These *context-value-tables* contain combinations of contexts (given as node, position in the context and size of the context), expressions and the resulting node set. This approach can easily be combined with the caching of the results of XPointers in the dbxlink approach.

Non-XPath-based Query Languages. Since the core of the dbxlink approach is only concerned with defining an XML-to-XML mapping, its usage is not restricted to XPath-based, or navigation-based at all, environments. For example,

the Xcerpt language [BS02] uses positional *query terms* that are *matched* against an underlying XML instance via *unification simulation*. Since in fact individual bisimulation paths are again navigational, integrating the expansion of XLinks into these navigation steps (using the same basic functionality and mappings as in our eXist reference implementation) would provide the required functionality.

Distributed Query Evaluation. In the above example, the actual evaluation of the XPointer took place at the referenced host and evaluating the remaining query locally (hybrid shipping). Other evaluation strategies allow to fetch the whole referenced document (data shipping) and evaluate the remaining query locally, or to rewrite the remaining query with the XPointer and evaluate both remote (query shipping). Intermediate results can be cached. The above functionality has been implemented in the eXist-based system.

Approaches that focus on distributed XML query evaluation in general like [Suc02,BG03] are orthogonal to ours (where the focus is on the modeling and handling of the interplay of links seen as views) and could probably be applied for a more efficient implementation.

5 Conclusion and Perspectives

We discussed the situation of employing the XLink mechanism for expressing references between XML instances. We have shown how the support for querying along XLinks given by the dbxlink approach can be integrated into XPath/XQuery evaluation algorithms and engines, providing a proof-of-concept implementation. The more elaborate and efficient handling of distributed queries poses a lot of questions that call for combinations with results of other work.

Projecting XML Fragments. For reducing the amount of data transmitted from one server to another, the techniques of projecting XML documents proposed in [MS03] can be applied. Given the remaining part $xpath\text{-}expr_2$, the referenced XML fragment can be reduced significantly to the projection relevant wrt. the query before transmitting it.

XPointer Containment. When an XML document containing XLinks is parsed and stored, the static set of links can be detected. XPath query containment algorithms as suggested e.g. in [MS04] can be used for the corresponding XPointer expressions. Then, assuming hybrid shipping and caching, queries that are subsumed by other links that are already cached, can be answered using the cached knowledge.

Further relevant work that might be worthwhile to be incorporated into our framework comprise parallel evaluation of remote queries, refined caching strategies, optimization strategies for local evaluation of XPath queries and stream processing of the results of XPointers, as well as strategies based on metadata, schema reasoning, and path indexes for finding out which XLinks will contribute

to the result of a given query. In a global scale, such strategies require a sophisticated P2P-based infrastructure with appropriate communication. Hence more specialized research results, some of which are mentioned above, can be applied.
Acknowledgements. This work is supported by the German Research Foundation (DFG) under grant no. MA 2539 within the LinXIS project.

References

[ABM+02] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: Peer-to-Peer Data and Web Services Integration. In VLDB, 2002.
[BFM06] E. Behrends, O. Fritzen, and W. May. Handling Interlinked XML Instances on the Web. In EDBT, Springer LNCS 3986, pp. 792–810, 2006.
[BG03] J.-M. Bremer and M. Gertz. On Distributing XML Repositories. In *WebDB*, pp. 73–78, 2003.
[BS02] F. Bry and S. Schaffert. Towards a declarative query and transformation language for XML and semistructured data: Simulation Unification. In ICLP, Springer LNCS 2401, pp. 255–270, 2002.
[exi] eXist: an Open Source Native XML Database. `http://exist-db.org/`.
[GKP05] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. *ACM Transactions on Database Systems (TODS)*, 30(2), 2005.
[Len02] M. Lenzerini. Data integration: a theoretical perspective. In *ACM Intl. Conference on Management of Data (SIGMOD)*, pp. 233–246, 2002.
[LS04] W. Lehner and H. Schöning. *XQuery.* dpunkt, 2004.
[May01a] W. May. LoPiX: A System for XML Data Integration and Manipulation. In *Intl. Conf. on Very Large Data Bases (VLDB)*, 2001.
[May01b] W. May. The MONDIAL Database, 2001. `http://dbis.informatik.uni-goettingen.de/Mondial/`.
[May05] W. May. Logic-based XML data integration: A semi-materializing approach. *Journal of Applied Logic*, (3):271–307, 2005.
[MS03] A. Marian and J. Siméon. Projecting XML Documents. In VLDB, 2003.
[MS04] G. Miklau and D. Suciu. Containment and Equivalence for a Fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.
[RBHS04] C. Re, J. Brinkley, K. Hinshaw, and D. Suciu. Distributed XQuery. In *Workshop on Information Integration on the Web (IIWEB)*, 2004.
[saxon] M. Kay. SAXON: The XSLT and XQuery processor. `http://saxon.sf.net/`.
[Suc02] D. Suciu. Distributed Query Evaluation on Semistructured Data. *ACM Transactions on Database Systems (TODS)*, 27(1):1–62, 2002.
[Wad99] P. Wadler. Two semantics for XPath. 1999. `http://www.cs.bell-labs.com/who/wadler/topics/xml.html`.
[XIn04] XML Inclusions (XInclude). `http://www.w3.org/TR/xinclude/`, 2004.
[XLi01] XML Linking Language (XLink). `http://www.w3.org/TR/xlink`, 2001.
[XLi06] XML Linking Language (XLink) Version 1.1. `http://www.w3.org/TR/xlink11`, 2006.
[XML99] XML Information Set. `http://www.w3.org/TR/XML-infoset`, 1999.
[XMQ05] XML Query Requirements. `http://www.w3.org/TR/xmlquery-req`, 2005.
[XPa99] XML Path Language (XPath) Version 1.0: 1999. `http://www.w3.org/TR/xpath`, 1999.
[XPQ05] XQuery 1.0 and XPath 2.0 Functions and Operators. `http://www.w3.org/TR/xquery-operators`, 2005.
[XPt02] XPointer xpointer() Scheme. `http://www.w3.org/TR/xptr-xpointer`, 2002.
[XPt03] XPointer Framework. `http://www.w3.org/TR/xptr-framework`, 2003.