Query Reformulation for the XML standards XPath, XQuery and XSLT

Sven Groppe, Stefan Böttcher

University of Paderborn
Faculty 5 (Computer Science, Electrical Engineering & Mathematics)
Fürstenallee 11,
D-33102 Paderborn , Germany
sg@uni-paderborn.de
stb@uni-paderborn.de

Abstract: Whenever transformation of data is used to bridge the gap of different data formats, and a query is given in the destination format, *query reformulation* can speed up the transformation of data. We achieve this speed-up in transformation when only the required data segment, described by the computed reformulated query, is transformed. Whenever the required section of data is not too large, query reformulation allows transformation *on demand*, even when the input data is large. Among other things, using query reformulation avoids problems of replication (especially synchronization problems), saves time and memory space for transformation and, in distributed scenarios, reduces the size of transmitted documents and transmission time. Whereas query reformulation is used in traditional relational databases, there has not been any complete query reformulation approach applicable to all of the most widely used XML standards of the W3C, most notably XPath, XQuery and XSLT. Within this paper, we outline a global approach to implement query reformulation for each of the XML standards XPath, XQuery and XSLT.

1 Introduction

The W3C developed XML as a standard for exchanging data in the web and introduced languages for querying XML documents such as XPath [W3C99] and XQuery [W3C03]. The result of evaluating an XPath expression is a set of nodes. In comparison, XQuery expressions embed XPath expressions. Additionally, XQuery expressions contain a return statement, which specifies the result in the form of an XML fragment. Therefore, XQuery can be used for both the querying of XML documents and transformation of XML data. For the purpose of transforming XML documents, the W3C also developed XSLT [W3C01]. XSLT stylesheets contain XPath expressions for read operations on the input XML document. Furthermore, XSLT can also be used for querying XML documents. Within this paper, we outline an approach to implement query reformulation for the XML standards XPath, XQuery and XSLT within a single system as shown in Figure 1.

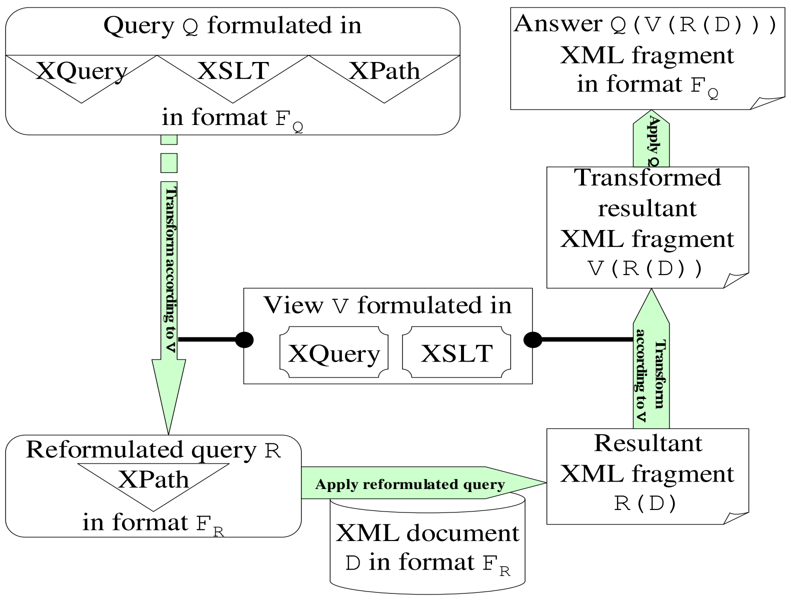


Figure 1: Overview of complete system

Whenever an XML document D is given in one XML format F_R, but the data is needed in a different format F_Q, we have to transform at least parts of D into the format F_Q. The time needed for transformation, and the necessary memory and disk space increase with the size of D. Therefore, on demand transformations of complete documents are not applicable to larger XML documents D. Very often however, applications only use a small fragment of the transformed XML document. Whenever a query Q, which is formulated in the XML format F_Q, describes the required small XML fragment, the technique of *query transformation* allows us to determine a query R that is formulated in terms of XML format F_R (see Figure 1). We can apply this query R in order to filter the input XML document and get a small remaining input XML fragment called *resultant XML fragment* R(D). This *resultant XML fragment* R(D) is defined to contain all the nodes and all their ancestors up to the root of the original XML document D, which contribute to the successful evaluation of the query Q. The transformed version of the resultant XML fragment R(D) contains all the information that is required in order to answer the query Q correctly. The transformation of the resultant XML fragment is much faster than the transformation of the entire XML document. As a conclusion, our technique allows on demand transformation of fragments of XML documents.

A technique corresponding to query transformation called *query reformulation* is used in traditional database technology within different scenarios, as for example within the following three scenarios. Firstly, query reformulation is used within data integration, where schema F_Q is the *global schema* and schema F_R is one of several *local schemas*. Secondly, query reformulation is used within schema evolution, where schema F_R is the *old schema* and schema F_Q is the *new schema*. Thirdly, query reformulation is used within bilateral situations, where two applications exchange data.

In database theory, the problem of *query reformulation* is commonly defined as follows (e.g. [DT03]): Given two schemas F_R and F_Q and a correspondence V which maps data from F_R to F_Q, find a query R formulated in terms of schema F_R which is equivalent to a given query Q formulated in terms of schema F_Q modulo the correspondence V.

Within our approach, the correspondence V can be an XSLT stylesheet or an XQuery expression, which we call *view definition* in the following. Furthermore, we allow the combination of both types of view definitions with queries Q that can be formulated in XPath, in XQuery or in XSLT.

Section 2 presents the complete query reformulation framework, whereas Section 3 discusses the relation to other contributions.

2 Query Reformulation Framework

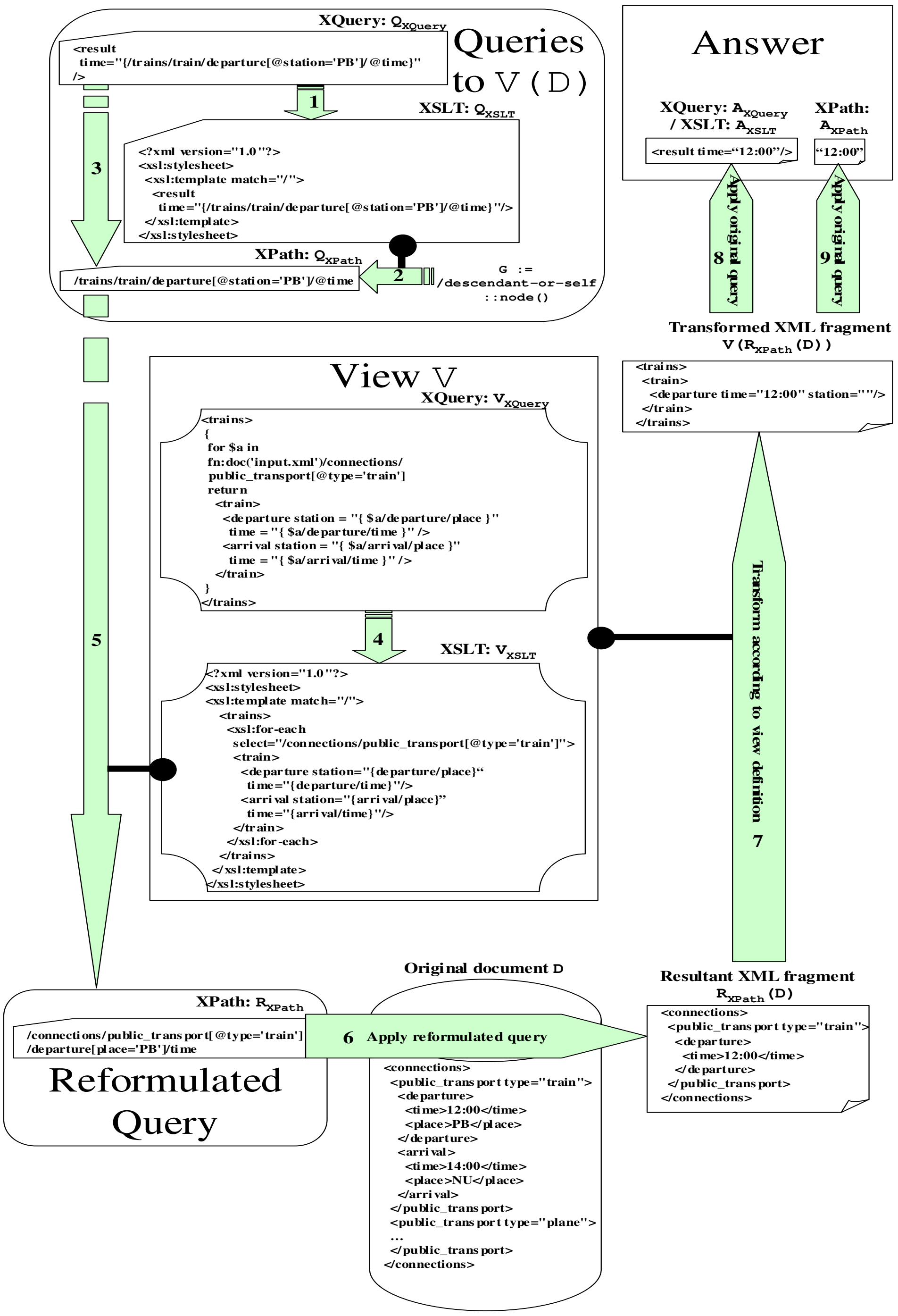
Figure 2 gives an overview of the query reformulation framework, whereas Section 2.1 to Section 2.5 present in detail, how every combination of the query languages XPath, XQuery or XSLT can be reformulated according to an XQuery or XSLT view definition. Note that our approach to query reformulation over views includes 6 combinations of views and query languages (see Figure 1), thereby going far beyond previous partial results, which are included into our approach wherever possible.

We adapt the definition of query reformulation to XPath, XQuery and XSLT in such a way that we can use standard XPath evaluators, standard XQuery engines and standard XSLT processors wherever possible, and call it *query transformation*.

In the following, we use the notation R(D) for the query result of applying the query R to the data D, and V(D) for the transformation of the data D (which can again be a *resultant XML fragment* of a query) according to V.

Problem definition: The *algorithmic problem of query transformation* is to determine the XPath expression R according to a given query Q formulated in XPath, XQuery or XSLT and a view definition V formulated in XQuery or XSLT such that R meets the following conditions: The resultant XML fragment of R(D) has to be as small as possible, but must also guarantee the equivalence of Q(V(R(D))) and Q(V(D)), i.e. that Q(V(R(D))) returns the same result as Q(V(D)) for every XML document D.

The following discussion refers to the transformation steps 1 to 9, which are summarized in Figure 2.



```
<result
 time="{/trains/train/departure[@station='PB']/@time}"
/>
```

```
<?xml version="1.0"?>
<xsl:stylesheet>
 <xsl:template match="/">
  <result
   time="{/trains/train/departure[@station='PB']/@time}"/>
 </xsl:template>
</xsl:stylesheet>
```

```
/trains/train/departure[@station='PB']/@time
```

```
<result time="12:00"/>
```

```
"12:00"
```

```
<trains>
 <train>
  <departure time="12:00" station=""/>
 </train>
</trains>
```

```
<trains>
 {
 for $a in
 fn:doc('input.xml')/connections/
 public_transport[@type='train']
 return
  <train>
   <departure station = "{ $a/departure/place }"
    time = "{ $a/departure/time }" />
   <arrival station = "{ $a/arrival/place }"
    time = "{ $a/arrival/time }" />
  </train>
 }
</trains>
```

```
<?xml version="1.0"?>
<xsl:stylesheet>
<xsl:template match="/">
  <trains>
   <xsl:for-each
    select="/connections/public_transport[@type='train']">
    <train>
     <departure station="{departure/place}"
      time="{departure/time}"/>
     <arrival station="{arrival/place}"
      time="{arrival/time}"/>
    </train>
   </xsl:for-each>
  </trains>
 </xsl:template>
</xsl:stylesheet>
```

```
/connections/public_transport[@type='train']
/departure[place='PB']/time
```

```
<connections>
 <public_transport type="train">
  <departure>
   <time>12:00</time>
   <place>PB</place>
  </departure>
  <arrival>
   <time>14:00</time>
   <place>NU</place>
  </arrival>
 </public_transport>
 <public_transport type="plane">
 ...
 </public_transport>
</connections>
```

```
<connections>
 <public_transport type="train">
  <departure>
   <time>12:00</time>
  </departure>
 </public_transport>
</connections>
```

Figure 2: Example of queries, views and XML data

2.1 XPath Reformulation according to XSLT Stylesheets (Steps 5, 6, 7 and 9)

The system presented in Figure 2 extends our previous approach of XPath reformulation according to XSLT stylesheets by XQuery views, by XQuery queries and XSLT queries. An example of our approach of [GBB04], [Gr04a] and [Gr04b] contains transformation steps 5, 6, 7 and 9, which transform an XPath query Q_{XPath} according to a view V_{XSLT} into an XPath query R_{XPath}, and further with an input XML document D into the sufficient source XML fragment R_{XPath}(D), the transformed XML fragment V(R_{XPath}(D)), and the answer A_{XPath} to the query Q_{XPath}.

The XSLT stylesheet V_{XSLT} defines the transformation of public transportation information given in XML document D into a model of train connections. Assume that we want to know all the times when trains leave Paderborn. For this purpose, we formulate an XPath query

```
QXPath = /trains/train/departure[@station=„PB“]/@time
```

to be applied to the transformed XML document. It is sufficient to transform only a *resultant XML fragment* R_{XPath}(D) for answering Q_{XPath}, where R_{XPath} is a query computed by our query transformation algorithm [GBB04] in step 5. Within our query transformation algorithm, a modified XPath evaluator searches for paths of those XSLT nodes in the XSLT stylesheet V_{XSLT}, which generate output, which is relevant to answer the query Q_{XPath}, and returns the successfully searched paths of V_{XSLT}. Within a second step, our query transformation algorithm computes the transformed query R_{XPath} by combining the input path expressions of the successfully searched paths of V_{XSLT}. We refer to [GBB04] for a more detailed discussion. In the example of Figure 2, we get

```
RXPath = /connections/public_transport[@type='train']
         /departure[place='PB']/time
```

Notice that standard XPath evaluators only return a query result as a node set, not as a resultant XML fragment. However, this *resultant XML fragment* R_{XPath}(D) is defined to contain all the nodes and all their ancestors up to the root of the original XML document D, which contribute to the successful evaluation of the query Q_{XPath}.[1] We retrieve the resultant XML fragment of the query R_{XPath}, i.e. R_{XPath}(D), in step 6. Within step 7, a standard XSLT processor transforms R_{XPath}(D) into V_{XSLT}(R_{XPath}(D)), i.e. a transformed XML fragment according to the XSLT stylesheet V_{XSLT}. In order to finally retrieve the result set, as the last step, the original query Q_{XPath} is applied to the transformed XML fragment in step 9, and we retrieve the answer A_{XPath}, i.e. "12:00".

[1] The computed resultant XML fragment R_{XPath}(D) is minimal in the following sense: R_{XPath}(D) does not contain other XML nodes of the input XML document D except those, which will be successfully visited by the internal XPath evaluator of the XSLT processor, when the XSLT processor executes that section of an XSLT stylesheet, which generates output that is relevant to answer the query Q_{XPath} (see the query transformation algorithm in [GBB04]).

[GBB04] contains an experimental evaluation, which we summarize in this paragraph: The measurement system is an Intel Pentium 4 with 512 Megabyte DDR-RAM, Windows XP, Java VM build version 1.4.2, Xerces2 Java parser 2.5.0 and Xalan-Java version 2.5.1. In the case of querying for single entries, our approach was 2 times faster compared to transforming the whole input XML document at a document size of 200 Kilobytes, 3 times faster at 500 Kilobytes and up to 40 times faster at 17 Megabytes. Furthermore, we varied the selectivity of the transformed query in order to determine the limit of the selectivity, when our approach is faster. The measurements show that the limit increases with the document size: Our approach is faster for queries with selectivity less than 30 % at 3.5 Megabyte, and is faster for queries with selectivity less than 53.3 % at 7 Megabyte. Therefore, our approach is scalable, i.e. our approach performs increasingly better the larger the XML documents are compared to transforming the whole input XML document.

2.1.1 Optimized subsets of XPath and XSLT

At first, we describe the supported subsets of the most widely used XML standards for each transformation step. Based on this, Section 2.5 summarizes the supported subsets of the complete system and the bottlenecks.

Transformation step 5 of Figure 2, i.e. the XPath query reformulation based on XSLT stylesheets, supports a subset of XPath queries, which we call S_{XPath} and which is a superset of Core XPath [GKP03].

Definition of the subset S_{XPath} of XPath: The syntax of the subset S_{XPath} of XPath is defined by the following grammar

```
locpath  ::= '/' locpath | locpath '/' locpath | locpath '|' locpath | locstep.
locstep  ::= axis '::' ntest '[' bexpr ']' … '[' bexpr ']'.
bexpr    ::= bexpr 'and' bexpr | bexpr 'or' bexpr | 'not(' bexpr ')' |
             locpath | (locpath | const) relop (locpath | const).
axis     ::= 'self' | 'child' | 'parent' | 'descendant' | 'descendant-or-self' |
             'ancestor' | 'ancestor-or-self' | 'following' | 'following-sibling' |
             'preceding' | 'preceding-sibling'.
ntest    ::= 'node()' | 'text()' | '*' | name.
relop    ::= '=' | '!=' | '<' | '<=' | '>' | '>='.
```

where "locpath" is the start production, "axis" denotes the axis relations, "ntest" denotes node tests and "**name**" denotes tags labeling document nodes. "const" represents a string or number constant.

Furthermore, we restrict the XSLT stylesheet V_{XSLT} to use only the subset S_{XSLT} of XSLT, which we define in the following.

Definition of the subset S_{XSLT} of XSLT: The XSLT stylesheets of S_{XSLT} are restricted to the following nodes:

- `<xsl:stylesheet>`,

- `<xsl:template match=M name=N>`,
- `<xsl:element name=N>`,
- `<xsl:attribute name=N>`,
- `<xsl:apply-templates select=I>`,
- `<xsl:text>`,
- `<xsl:value-of select=I>`,
- `<xsl:for-each select=I>`,
- `<xsl:call-template name=N>`,
- `<xsl:attribute-set name=N>`,
- `<xsl:if test=T>`,
- `<xsl:choose>`,
- `<xsl:when test=T>`,
- `<xsl:otherwise>`,
- `<xsl:processing-instruction>`,
- `<xsl:comment>`,
- `<xsl:sort>`,
- `<xsl:copy-of select=I>`,
- `<xsl:variable name=N>`,
- `<xsl:with-param name=N>` and
- `<xsl:param name=N>`,

where `I` and `M` contain an XPath expression without function calls, `T` is a boolean expression and `N` is a string constant.

Whenever attribute values are generated by the XSLT stylesheet, we assume that each value is generated by one XSLT node (i.e. `<xsl:text>` or `<xsl:value-of select=I>`).

2.2 Collecting XPath Expressions from an XQuery Expression (Step 3, 5, 6, 7, 8)

Within this section, we present how the approach described in Section 2.1 can be extended in such a way that we can use an XQuery expression to describe the query. As before, an XSLT stylesheet is used to define the view. We adapt the contribution of [MS03] for this purpose.

[MS03] projects XML documents to the necessary input nodes according to an XQuery expression. It furthermore contains a static XPath analysis which retrieves the XPath expressions that describe the necessary input nodes of an XML document.

Figure 2 shows how the steps 3, 5, 6, 7 and 8 evaluate a given query Q_{XQuery} in order to retrieve the answer A_{XQuery}.

Step 3 can use the contribution presented in [MS03] in order to retrieve the *used XPath expression* $\mathtt{Q}_{\mathtt{XPath}}$ from the XQuery query $\mathtt{Q}_{\mathtt{XQuery}}$ such that $\mathtt{Q}_{\mathtt{XPath}}$ describes the necessary input nodes of the XML document. From now on, we use our query reformulation algorithm already described in Section 2.1: Within step 5, we reformulate the XPath expression $\mathtt{Q}_{\mathtt{XPath}}$ (which is the result of step 3) according to the XSLT stylesheet $\mathtt{V}_{\mathtt{XSLT}}$, into the reformulated query $\mathtt{R}_{\mathtt{XPath}}$. The reformulated XPath expression $\mathtt{R}_{\mathtt{XPath}}$ describes a sufficiently smaller resultant XML fragment $\mathtt{R}_{\mathtt{XPath}}\mathtt{(D)}$, which we retrieve in step 6. After transforming $\mathtt{R}_{\mathtt{XPath}}\mathtt{(D)}$ into the transformed XML fragment $\mathtt{V(R}_{\mathtt{XPath}}\mathtt{(D))}$ in step 7, we finally apply the XQuery query $\mathtt{Q}_{\mathtt{XQuery}}$ to $\mathtt{V(R}_{\mathtt{XPath}}\mathtt{(D))}$ in step 8 in order to retrieve the final result $\mathtt{A}_{\mathtt{XQuery}}$.

2.2.1 Optimized subsets of XQuery and XSLT

The XQuery expression $\mathtt{Q}_{\mathtt{XQuery}}$, which can be used within step 3, is restricted to XQuery expressions without function definitions. Furthermore, the embedded XPath expressions in $\mathtt{Q}_{\mathtt{XQuery}}$ are restricted to the `child`, `self`, `descendant`, `descendant-or-self` and `attribute` axes each of which can be combined with the node name test, or the node tests `node()` or `text()`.

2.3 XSLT Reformulation according to XSLT Stylesheets (Steps 2, 5, 6, 7 and 8)

XSLT is specified for the transformation of XML documents, but XSLT can also be used as a query language.

Figure 2 gives an example of our approach to query reformulation of XSLT queries, i.e. it shows how the steps 2, 5, 6, 7 and 8 transform a given query $\mathtt{Q}_{\mathtt{XSLT}}$ into the answer $\mathtt{A}_{\mathtt{XSLT}}$, where the steps 5, 6 and 7 are performed as before.

In the following, we show how to determine all the necessary and sufficient input nodes, which are required by the XSLT processor to answer the XSLT query $\mathtt{Q}_{\mathtt{XSLT}}$ applied to the view `V`. The answer $\mathtt{A}_{\mathtt{XSLT}}$ of step 8, is the output of the XSLT stylesheet $\mathtt{Q}_{\mathtt{XSLT}}$ applied to the transformed XML fragment $\mathtt{V(R}_{\mathtt{XPath}}\mathtt{(D))}$. As a matter of fact, the required part of $\mathtt{A}_{\mathtt{XSLT}}$ is the complete $\mathtt{A}_{\mathtt{XSLT}}$, which can be described by the most general XPath query `G:=/descendand-or-self::node()`. Therefore in step 2, we compute the XPath expression $\mathtt{Q}_{\mathtt{XPath}}$ that describes the sufficient input nodes of the query $\mathtt{Q}_{\mathtt{XSLT}}$ by applying the query reformulation algorithm as described in Section 2.1 to the most general XPath query `G` and to the XSLT stylesheet $\mathtt{Q}_{\mathtt{XSLT}}$.

Steps 5, 6, and 7 are then performed as before. Step 8 is different to the approach in Section 2.1, because we now retrieve the answer $\mathtt{A}_{\mathtt{XSLT}}$, i.e. `<result time="12:00">`, by processing the XSLT stylesheet $\mathtt{Q}_{\mathtt{XSLT}}$ instead of applying the query $\mathtt{Q}_{\mathtt{XPath}}$ to $\mathtt{V(R}_{\mathtt{XPath}}\mathtt{(D))}$ in step 9.

2.3.1 Optimized subset of XSLT

The input of our transformation step 5 is the output Q_{XPath} of the query reformulation step 2 according to the XSLT stylesheet Q_{XSLT}. Therefore, Q_{XPath} is restricted to the subset S_{XPath} described in Section 2.1.1. As Q_{XPath} is computed from the XPath expressions within the XSLT stylesheet Q_{XSLT}, the XPath expressions within Q_{XSLT} are restricted to the same subset S_{XPath} as the query Q_{XPath}. Furthermore, the XSLT stylesheets Q_{XSLT} and V_{XSLT} are restricted to S_{XSLT}, which is described in Section 2.1.1.

2.4 XQuery Transformation to XSLT Stylesheets (Steps 1, 2, 4, 5, 6, 7 and 8)

The approach presented in [LPS01] can be used for transforming a query Q_{XQuery} into Q_{XSLT} (see step 1 in Figure 2), and for transforming a view definition V_{XQuery} into V_{XSLT} (see step 4). However, within Figure 3, we show an optimized example avoiding unnecessary transformation steps compatible to the approach in [LPS01].[2]

Step 1 together with following step 2, is an alternative approach to step 3 described in Section 2.2.

Using this approach and the approaches of Section 2.1 and Section 2.3, we can support query reformulation for all combinations of XPath, XQuery and XSLT as query languages, and XQuery and XSLT as view languages.

2.4.1 Optimized subsets of XQuery and XSLT

The approach of transforming XQuery into XSLT, as outlined in [LPS01], supports all XQuery expressions of the subset S_{XQuery}, which is defined as follows:

Definition of the subset S_{XQuery} of XQuery: The subset S_{XQuery} of XQuery, contains all XQuery expressions, which do not use runtime type information of XQuery.

As our step 1 and step 4 use the contribution of [LPS01], Q_{XQuery} and V_{XQuery} are restricted to S_{XQuery}.

Furthermore, the XPath expressions within Q_{XQuery} are restricted to the subset S_{XPath} of XPath expressions described in Section 2.1.1 for the following reasons. Step 1 generates the XSLT stylesheet Q_{XSLT} from the XQuery expression Q_{XQuery}. Thereby, the embedded XPath expressions in Q_{XQuery} are transferred to Q_{XSLT}. Step 2 processes Q_{XSLT} further to Q_{XPath}, which is composed of the embedded XPath expressions of Q_{XSLT}. As step 5 requires the restrictions described in Section 2.1.1, Q_{XPath} and the XPath expressions in both Q_{XSLT} and Q_{XQuery} are restricted to S_{XPath}.

[2] We can optimize the example, because the node identity operator of XQuery is not used. If the node identity operator of XQuery is used, there are further transforming steps necessary, as described in [16].

The XPath expressions within view V_{XQuery} are not restricted, as we do not apply the query reformulation algorithm to the reformulated queries R_{XPath} again.

2.5 Subsets optimized by the complete system

Summarizing all, the considered subsets of the complete systems are:

1) The subset S_{XPath}, as described in Section 2.1.1, for the XPath expressions Q_{XPath} and the XPath expressions embedded within the XQuery expression Q_{XQuery} and within the XSLT stylesheet Q_{XSLT}.

2) The subset S_{XQuery}, as described in Section 2.4.1, for the XQuery expressions Q_{XQuery} and V_{XQuery}.

3) The subset S_{XSLT}, as described in Section 2.1.1, for the XSLT styleheets Q_{XSLT} and V_{XSLT}.

3 Further Related Work

For the transformation of XML queries into queries based upon other data storage formats, at least two major research directions can be distinguished. Firstly, the mapping of XML queries to object oriented or relational databases (e.g. [BBB00], [DT03]), and secondly, the transformation of XML queries or XML documents into other XML queries or XML documents (e.g. [Ab99]). We follow the second approach; however, we focus on XSL and XQuery for the transformation of both XML data and queries, where the queries can be formulated in XPath, XQuery or XSLT.

Within related contributions to schema integration, two approaches to data and query translation can be distinguished. While the majority of contributions (e.g. [ACM97], [Cl98]) map the data to a unique representation, we follow [CG00] and map the queries to those domains where the data resides.

The contribution in [CVV01] presents query reformulation according to path-to-path mappings. We go beyond this, as we support each of the current XML standards XQuery, XPath and XSLT. [Mo02] describes how XSL processing can be incorporated into database engines, but focuses on efficient XSL processing. The complexity of XPath query evaluation on XML documents is examined in [GKP03]. In comparison, we use an evaluation based on output nodes of an XSLT stylesheet and consider query transformation. Altinel and Franklin present in [AF00], an algorithm, which filters XML documents according to a given query and analyzes the performance, but the algorithm does not contain query transformation.

We presented approaches to reformulate XPath queries according to XSLT stylesheets ([GBB04], [Gr04b]). Within this paper, we present how to combine our approaches with other approaches in order to support query reformulation not only for XPath, but also for XQuery and XSLT queries. Furthermore, we extend query reformulation to view definitions formulated in XSLT or in XQuery.

[MS03] projects XML documents to a sufficient XML fragment before processing XQuery queries. [MS03] contains a static path analysis of XQuery queries, which computes a set of projection paths formulated in XPath. Our approach is compatible to [MS03], but additionally provides query reformulation through views in XSLT or XQuery.

[Fe02] and [Sh01] present systems, which support to answer XQuery queries on XQuery views, where the XML data is stored in relational databases. In comparison, within our approach, the view can be formulated in XQuery or XSLT and the query in XPath, XQuery or XSLT. Furthermore, our approach stores the data in XML and does not store the XML data in relational databases.

In comparison to all other approaches, some of which are partial results of our result, our approach is the first, which integrates the two different view definition languages of XSLT and XQuery for XML views, with the possibility to express queries in XPath, XSLT or XQuery.

4 Summary and Conclusions

In summary, we have presented how query reformulation can be applied to XPath, XQuery or XSLT queries. The view definition can be either formulated in XQuery or XSLT. Within our approach, we support the most widely used XML document query standards and transformation standards of the W3C. Our implementation of the XSLT transformation part, i.e. query reformulation of XPath queries using V_{XSLT}, has been completed and shows that the document size and overall speed can be reduced for a large set of queries (see [GBB04], [Gr04a]). Future work will involve the integration of the contributions of [LPS01] and [MS03] in order to evaluate the performance of the proposed integrated query reformulation system. Another big challenge is to investigate how to extend our approach in such a way that XML data updates can be efficiently propagated through an XML view formulated in XQuery or XSLT.

References

[Ab99] Abiteboul, S.: On views and XML. In *PODS*, pages 1-9, 1999.

[ACM97] Abiteboul, S., Cluet, S., and Milo, T.: Correspondence and translation for heterogeneous data. In *Proc. of the 6th ICDT,* 1997.

[AF00] Altinel, M., and Franklin, M. J.: Efficient Filtering of XML documents for Selective Dissemination of Information. In *Proceedings of 26th International Conference on Very Large Databases,* Cairo, Egypt, 2000.

[BBB00] R. Bourret, R., Bornhövd, C., and Buchmann, A.P.: A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases. *2nd Int. Workshop on Advanced Issues of EC and Web-based Information Systems (WECWIS)*, San Jose, California, 2000.

[CG00] Chang, C.-C. K., and Garcia-Molina, H.: Approximate Query Translation Across Heterogeneous Information Sources. *VLDB 2000, 2000.*

[CLL02] Chen, Y. B., Ling, T. W., and Lee, M. L.: Designing Valid XML Views, *ER 2002, LNCS 2503*, pp. 463-477, 2002.

[Cl98] Cluet, S., Delobel, C., Simon, J., and Smaga, K.: Your mediators need data conversion! In *Proc. of the 1998 ACM SIGMOD Conf., 1998.*

[CVV01] Cluet, S., Veltri, P., and Vodislav, D.: Views in a Large Scale XML Repository. In *Proceedings of the 27th VLDB Conference*, Roma, Italy, 2001.

[DT03] Deutsch, A., and Tannen, V.: Reformulation of XML Queries and Constraints, In *ICDT 2003*, LNCS 2572, pp. 225-241, 2003.

[Fe02] Fernández, M., Kadiyska, Y., Suciu, D., Morishima, A. and Wang Chiew Tan: SilkRoute: A Framework for Publishing Relational Data in XML, ACM Transactions on Database Systems, Vol. 27, No. 4, December 2002, pages 438-493.

[GKP03] Gottlob, G., Koch, C., and Pichler, R.: The Complexity of XPath Query Evaluation, In *Proceedings of the 22th ACM SIGMOD-SIGACT-SIGART symposium of Principles of database systems (PODS 2003)*, San Diego, California, USA, 2003.

[GB03] Groppe, S., and Böttcher, S.: Querying transformed XML documents: Determining a sufficient fragment of the original document. *3. International Workshop Web Databases (WebDB)*, Berlin, Germany, 2003.

[GBB04] Groppe, S., Böttcher, S., and Birkenheuer, G.: Efficient Querying of transformed XML documents, submitted to *6th International Conference of Enterprise Information Systems (ICEIS 2004)*, Porto, Portugal, 2004.

[Gr04a] Groppe, S., Böttcher, S., Heckel, R., and Birkenheuer, G.: XPath query reformulation based on XSLT stylesheets. *Technical Report*, University of Paderborn, Paderborn, Germany, 2004.

[Gr04b] Groppe, S., Böttcher, S., Heckel, R., and Birkenheuer, G.: Using XSLT Stylesheets to Transform XPath Queries. Eighth East-European Conference on Advances in Databases and Information Systems (ADBIS 2004), Budapest, Hungary, September 2004.

[LPS01] Lechner, S., Preuner, G., and Schrefl, M.: Translating XQuery into XSLT, In *ER 2001 Workshops*, Yokohama, Japan, 2001.

[MS03] Marian, A., and Siméon, J.: Projecting XML Documents. In *Proceedings of the 29th VLDB Conference*, Berlin, Germany, 2003.

[Mo02] Moerkotte, G.: Incorporating XSL Processing Into Database Engines. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[Sh01] Shanmugasundaram, J., Kiernan, J., Shekita, E., Fan, C., and Funderburk, J.: Querying XML Views of Relational Data, In Proceedings of the 27th VLDB Conference, Roma, Italy, 2001.

[W3C01] W3C: Extensible Stylesheet Language (XSL). http://www.w3.org/Style/XSL/, 2001.

[W3C99] W3C: XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath/, 1999.

[W3C03] W3C: XQuery 1.0: An XML Query Language, W3C Working Draft, http://www.w3.org/TR/xquery/, 2003.