XPath Query Transformation based on XSLT Stylesheets

Sven Groppe
University of Paderborn, Faculty 5
Fürstenallee 11
D-33102 Paderborn , Germany
+49 5251 606067
sg@uni-paderborn.de

Stefan Böttcher
University of Paderborn, Faculty 5
Fürstenallee 11
D-33102 Paderborn , Germany
+49 5251 606662
stb@uni-paderborn.de

ABSTRACT

Whenever XML data must be shared by heterogeneous applications, transformations between different application-specific XML formats are necessary. The state-of-the-art method transforms entire XML documents from one application format into another e.g. by using an XSLT stylesheet, so that each application can work locally on its preferred format. In our approach, we use an XSLT stylesheet in order to transform a given XPath query such that we retrieve and transform only that part of the XML document which is sufficient to answer the given query. Among other things, our approach avoids problems of replication, saves processing time and in distributed scenarios, transportation costs.

Categories and Subject Descriptors

H2.4 [**Database Management**]: Systems – *Query Processing*

General Terms

Algorithms, Languages

Keywords

XPath, XSLT, query transformation, query rewriting.

1. INTRODUCTION

1.1 Problem definition and motivation

Our work is motivated by the development of an XML database system which seamlessly incorporates XSLT processing. We assume that data is stored in XML format 1 but can be transformed on demand by an XSLT stylesheet S into data in XML format 2. In our approach, a given query `XP2` in XML format 2 describes the needed transformed data of a given operation. Our contribution involves translating an XPath query `XP2` in XML format 2 using a new query transformation algorithm into a query `XP1` on the original XML data `D` (i.e. the data in format 1) which is based on an XSLT stylesheet `S`. Applying `XP1` selects a fragment `F1` (i.e. `F1=XP1(D)`) from the database which is smaller in comparison to the entire XML document `D`. Only this XML fragment `F1` is then transformed by the XSLT processor (i.e. `S(XP1(D))`) and at last queried according to `XP2` (i.e. `XP2(S(XP1(D)))`). Our approach may be of considerable advantage when compared to the process of transforming the document via XSLT (i.e. `S(D)`) and applying the query `XP2` afterwards (i.e. `XP2(S(D))`).

The *algorithmic problem* is to determine `XP1` according to a given XPath query `XP2` and an XSLT stylesheet `S` as restrictive as possible but to guarantee the equivalence of `XP2(S(XP1(D)))` and `XP2(S(D))`, i.e. `XP2(S(XP1(D)))` returns the same result as `XP2(S(D))` for every XML document `D`.

For example, see Figure 1: The XSLT stylesheet `S` transforms the hierarchy of a company (XML document `D`) into a flat model, i.e. the transformed XML document `S(D)`. In order to answer an XPath query

```
XP2 = /level/worker[@family_name=„Smith"]/@*
```

on the transformed XML document `S(D)`, it is sufficient to transform only that XML fragment `F1`, which is the query result of the following query `XP1`

```
XP1 = (/employee/responsible_for)*/employee
      [@surname=„Smith"]
```

given in XML format 1, where `A*` is a short notation for an arbitrary number of paths `A`.[1]

1.2 Relation to other work and our focus

For the transformation of XML queries into queries to 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. [5]), and secondly, the transformation of XML queries or XML documents into other XML queries or XML documents (e.g. [1]). We follow the second approach; however, we focus on XSL [12] for the transformation of both, data and XPath [13] queries.

Within related contributions to schema integration two approaches to data and query translation can be distinguished. While the majority of contributions (e.g. [7], [2]) map the data to

WIDM'03, November 7-8, 2003, New Orleans, Louisiana, USA.

[1] Standard XPath evaluators do not support `A*`, but we can retrieve a superset by replacing `A*/` with `//`. Furthermore, a modified XPath evaluator has to return not only the result set of `XP1` (as standard XPath evaluators do), but a result XML fragment `F1`. This result XML fragment `F1` must contain all nodes and all their ancestors up to the root of the original XML document `D`, which contribute to the successful evaluation of the query `XP1`.

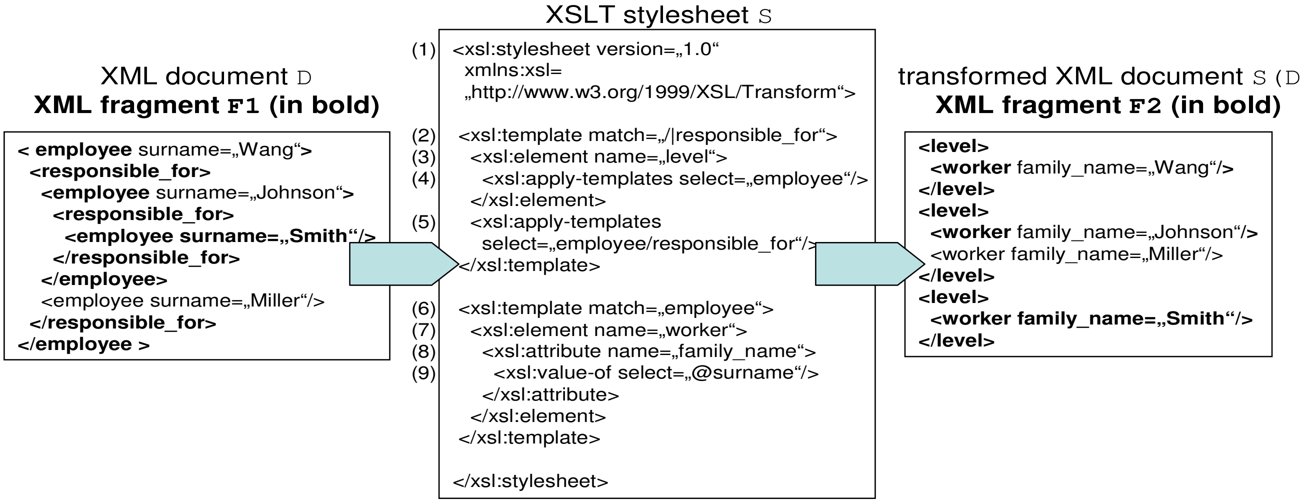

Figure 1: Example of the transformation of F1 into F2 by an XSLT stylesheet S

a unique representation, we follow [6] and map the queries to those domains where the data resides.

[8] reformulates queries according to path-to-path mappings. We go beyond this, as we use XSLT as a more powerful mapping language. [11] describes how XSL processing can be incorporated into database engines, but focuses on efficient XSL processing. [9] examines the complexity of XPath query evaluation on XML documents (instead of an evaluation based on output nodes of XSLT) and does not consider query transformation. [3] presents an algorithm to filter XML documents according to a given query and analyses the performance, but does not contain query transformation.

In contrast to all these approaches, we focus on the transformation of XPath queries according to an XSLT stylesheet.

We go beyond our contribution of [10], because we support a larger subset of XSLT (i.e. absolute paths are now allowed in select attributes of XSLT nodes) and a larger subset of XPath (i.e. predicates are now allowed) for the XPath query transformation.

1.3 Considered subsets of XPath and XSLT and parts of XPath expressions

We currently restrict XPath queries XP2, such that they conform to the following rule for AttributeQuery given in the Extended Backus Naur Form (EBNF):

```
AttributeQuery ::= LocationPath "/@*".
LocationPath   ::= Step*.
Step      ::= ("/" | "//") Name Predicate*.
Predicate ::= "[" "@" Name "=" String "]".
```

This subset of XPath allows querying for an XML fragment which can be described by succeeding elements (in an arbitrary depth), the attributes of which can be restricted to a constant value.

Similarly, we restrict XSLT, i.e., we consider the following nodes of an XSLT stylesheet:

- <xsl:stylesheet>,
- <xsl:template match=M1 name=N>,
- <xsl:element name=N>,
- <xsl:attribute name=N>,
- <xsl:apply-templates select=S1>,
- <xsl:text>,
- <xsl:value-of select=S2>,
- <xsl:for-each select=S1>,
- <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> and
- <xsl:sort>,

where S1, S2 and M1 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 this is only done in one XSLT node (i.e. <xsl:text> or <xsl:value-of select=S2>).

We define following terms for later use in Section 2:

Definition: An XPath expression s can be divided into a *relative part* rp(s) and an *absolute part* ap(s) (both of which may be empty) in such a way, that rp(s) contains a relative path expression, ap(s) contains an absolute path expression, and the union of ap(s) and rp(s) is equivalent to s.

Example: The relative part of s=(/E1|E2/E3|E4)/E5 is rp(s)=(E2/E3|E4)/E5, the absolute part is ap(s)=/E1/E5.

2. THE QUERY TRANSFORMATION ALGORITHM

Our goal is to determine a smaller, but sufficient part XP1(D) of the input XML document D, where the XSLT transformation of XP1(D), i.e. S(XP1(D)), contains all the information required to answer the query XP2 correctly, i.e. XP2(S(XP1(D))) is equivalent to XP2(S(D)).

For this reason, we firstly look at the so called *output nodes* of the XSLT stylesheet S. In the example of Figure 1, all the elements level in S(D) in the right part of Figure 1 are generated by the node (3) of S (see the middle box of Figure 1), all the elements worker are generated by node (7). These output nodes (3) and (7) of the XSLT stylesheet S are reached, after a sequence of nodes (which we call *stylesheet paths*) of the XSLT stylesheet S are executed. In the example, one stylesheet path for the nodes (3) and (7) is <(1),(2),(3),(4),(6),(7)>.

While executing these stylesheet paths, the XSLT processor also processes so called *input nodes* (e.g. node (6)) each of which selects a certain node set of the input XML document D. When considering all executed input nodes, the input nodes altogether select a whole node set of the input XML document D. In the stylesheet path above, this is the node set described using the query /employee. When considering our idea to reduce the amount of data of the input XML document, we notice that all nodes (but *not* more nodes!) of the input XML document which are selected within input nodes along the stylesheet path must be available for the execution of the stylesheet path in the same way as all nodes of the input XML document are available. If we can determine the whole node set (described using a query XP1), which is selected on *all* stylesheet paths, which generate output which is relevant to the query XP2, we can then select a smaller, but sufficient part XP1(D) of the input XML document D, where XP2(S(XP1(D))) is equivalent to XP2(S(D)).

In our approach, we search at first for stylesheet paths within the XSLT stylesheet (see Section 2.1), which generate elements, attributes and attribute values in the correct order according to the query XP2.

For each of these stylesheet paths, we determine the so called *input path expression* of the XSLT stylesheet (see Section 2.2), which summarizes the XPath expressions of input nodes along the stylesheet path. The transformed query XP1 is the disjunction of the determined input path expressions of each stylesheet path.

2.1 Output path search in the XSLT stylesheet

In order to determine the paths through an XSLT stylesheet, which may generate output that is relevant to XP2, we search for so called *successful element stylesheet paths* (and attached *attribute, filter* and *loop stylesheet paths*), i.e. paths which begin at the start node and contain so called *output nodes* (i.e. nodes <xsl:element name=E> and <xsl:attribute name=A>) of the XSLT stylesheet which may contribute answering the query XP2. The search continues from a node S1 to a node S2, if

a. S2 is a child node of S1 within the XSLT stylesheet, or
b. S1 is a node <xsl:call-template name=N> and S2 a node <xsl:template name=N> with the same N, or
c. S1 is a node with an attribute xsl:use-attribute-sets=N and S2 a node <xsl:attribute-set name=N> with the same N, or
d. S1 is <xsl:apply-templates select=s/> and S2 <xsl:template match=m> and the template of S2 can possibly be called from the selected node set s. This is the case if ap(s)|//rp(s) and ap(m)|//rp(m) are possibly not disjointed which can be checked by a fast (but incomplete) tester (e.g. the one in [4]).

For example, for XP2=/level/worker[@family_name="Smith"]/@* and the XSLT stylesheet of Figure 1, we search for the output nodes which generate the elements level (see node (3)) and then worker (see node (7)). The search can pass non-output nodes as they do not generate any output, which does not fit to XP2. The search can also pass any output nodes if we search next for an element E in arbitrary depth, i.e. for //E.

In order to store information about the search, we define a s*tylesheet path* as a list of entries of the form (N, XP2r) where N is a node in the XSLT stylesheet and XP2r is the suffix of XP2 which still has to be processed. We call the stylesheet path, which contains all the visited nodes of the path from the start node to the current node in the visited order, the *current stylesheet path* sp.

While searching for attributes (e.g., for /@* see nodes (8) and (9) in Figure 1), we branch off the successful element stylesheet path. In order to allow a sequential (but recursive) computation of the input path expressions in Section 2.2, we store paths resulting from a search for attributes separately in *attribute stylesheet paths*.

We store the filter itself and paths resulting from a search for filters in *filter stylesheet paths* (e.g., for [@family_name="Smith"] see nodes (8) and (9) in Figure 1). If the attribute value of the filter is generated by an input node <xsl:value-of select=S/>, we can transform the filter to a filter in XML format 1 within XP1 (see Section 2.2), which restricts the node set of the input XML document more exactly when we apply XP1.

If the value of the attribute of the filter is generated by an output node <xsl:text>const</xsl:text> within the XSLT stylesheet, we can currently decide without access to the XML document that a filter [@A1 = V] will always be

- true, if V is equal to const. In order to be sure, that the attribute @A1 and its value V will be nevertheless generated by the XSL processor, we store the suitable information in the set of attribute stylesheet paths.
- false, if V is not equal to const. We abort the search here.

During the search it may occur, that we revisit a node N of the XSLT stylesheet without any progress in the processing of XP2r. For example, we can visit node (1), node (2), then node (5) and the node (2) again in Figure 1. We call this a *loop* and we define a loop as follows: The loop is the current stylesheet path minus the stylesheet path of the first visit of N. In the example of Figure 1, the loop contains the nodes (5) and (2). For each loop in the stylesheet graph, we store the loop itself, the current node N and XP2r as an entry to the set of *loop stylesheet paths*, because we need to know the input which is consumed when the XSLT processor executes the nodes of a loop (see Section 2.2). In order to avoid an infinite search, we do not continue the search at the final node when the loop is detected.

2.2 Computing Input Path Expressions

(Only) when the XSLT processor tracks the successful element stylesheet paths (and its attached attribute, filter and loop stylesheet paths) computed within Section 2.1, the XSLT processor generates output which contributes to the query XP2. While tracking a successful element stylesheet path (and its attached paths), the XSLT processor can execute the so called *input nodes*

- <xsl:apply-templates select=S/>,
- <xsl:value-of select=S/>,
- <xsl:for-each select=S>,
- <xsl:if test=T> and
- <xsl:when test=T>,

where S occurs in the Boolean expression T. While executing input nodes, the XSLT processor selects a certain node set called *input node set* of the input XML document which is described using the so called *input path expression* S. The existence of the input node set of the input XML document is necessary in order to execute the successful element stylesheet path (and its attached paths) in the same way as they are executed when all nodes of the input XML document are available.

Figure 2: Computing Input Path Expressions for `XP2=/level/worker[@family_name=„Smith"]/@*`

The XSLT processor does not select the whole input node set of the input XML document immediately. In fact, the XSLT processor selects the input node set step by step in different input nodes of the XSLT stylesheet which are described by their input path expressions in the successful element stylesheet path and its attached paths. For this reason, we have to combine all these input path expressions along a successful element stylesheet path (and its attached paths).

For this purpose, we use two different variables:

The *current input path expression* (current ipe) contains the whole input path expression of the successful element stylesheet path up to (and including) the current node. The current ipe describes a superset of the node set of the XML document with which the XSLT processor processes this node while executing the successful element stylesheet path.

The *completed input path expression* (completed ipe) contains all such input path expressions, which are selected within the stylesheet path before the current node, but which will not be used further in the computation of a current ipe.

Figure 2 shows the computation of the current input path expressions and the completed input path expressions of the example of Figure 1 and a given query XP2 = /level /worker[@family_name=„Smith"]/@*.

The XSLT processor starts executing the successful element stylesheet path with the node set described by the match attribute m of the first template <xsl:template match=m> within the successful element stylesheet path. The template could match nodes of the node set rp(m) in arbitrary depth of the XML document because of built-in templates. Therefore, we initialize current ipe with ap(m)|//rp(m). For the example within Figure 2, the current ipe is initialized with /|//responsible_for. The completed ipe is initialized with the empty set.

We mainly iterate through each successful element stylesheet path and we

- compute new path expressions of the current ipe (current ipe$_{new}$) and the completed ipe (completed ipe$_{new}$) from the input path expression of the current node (ipe) and the old input path expressions of the current ipe (current ipe$_{old}$) and the completed ipe (completed ipe$_{old}$).
- recursively compute and combine current ipes and completed ipes of attached attribute stylesheet paths, filter stylesheet paths, and loop stylesheet paths. For this purpose, at first we initialize current ipe (current ipe$_{init}$) and completed ipe (completed ipe$_{init}$), then recursively compute along the attached path as before and get the current ipe (current ipe$_{path}$) and completed ipe (completed ipe$_{path}$) after the last node of the attached path. At last we compute current ipe$_{new}$ and completed ipe$_{new}$ of the node with the attached path.

Figure 3 lists the different computing steps depending on the current node and example nodes of the different computing steps within Figure 2.

The complete input path expression which is used as query XP1 on the input XML document is the union of all the completed ipes and the current ipe of the last node of each of the n successful element stylesheet paths (1..n),

XP1 = completed ipe$_1$ | current ipe$_1$ | … | completed ipe$_n$ | current ipe$_n$.

If there is no entry in the set of successful element stylesheet paths, i.e. n=0, XP1 remains empty.

In the example of Figure 2, we get

```
XP1=(/|//responsible_for)(/employee/responsible_for)*/employee[@surname="Smith"] |
(/|//responsible_for)(/employee/responsible_for)*/employee[@surname="Smith"]/@surname
```

3. SUMMARY AND CONCLUSIONS

In order to reduce data transformation and data transportation costs, we compute from a given query XP2 and a given XSLT stylesheet a transformed query XP1 which can be applied to the input XML document in order to retrieve a smaller fragment F1

Current Node	Computation of **current ipe** and **completed ipe**	Example Nodes
Non-input nodes without attached paths	current ipe$_{new}$ = current ipe$_{old}$ completed ipe$_{new}$ = completed ipe$_{old}$	(3), (6), (8)
Input node	current ipe$_{new}$ = if (rp(ipe) is empty) ap(ipe) else current ipe$_{old}$ / rp(ipe) | ap(ipe) completed ipe$_{new}$ = if(ap(ipe) is empty) completed ipe$_{old}$ else completed ipe$_{old}$ | current ipe$_{old}$	(5), (4), (9)
Attached attribute stylesheet path	current ipe$_{init}$ = current ipe$_{old}$ completed ipe$_{init}$ = completed ipe$_{old}$ current ipe$_{new}$ = current ipe$_{old}$ completed ipe$_{new}$ = current ipe$_{path}$ | completed ipe$_{path}$	(7)
Attached filter stylesheet path according to a filter [@A=const]	current ipe$_{init}$ = completed ipe$_{init}$ = completed ipe$_{old}$ current ipe$_{new}$ = current ipe$_{old}$[current ipe$_{path}$=const] completed ipe$_{new}$ = completed ipe$_{path}$	(7)
Attached loop stylesheet paths 1.. n	current ipe$_{init}$ = completed ipe$_{init}$ = current ipe$_{new}$ = (current ipe$_{old}$| ap(current ipe$_{path1}$) |…|ap(current ipe$_{pathn}$)) (/rp(current ipe$_{path1}$)|…|/rp(current ipe$_{pathn}$))* completed ipe$_{new}$ = completed ipe$_{old}$| current ipe$_{new}$ / (rp(completed ipe$_{path1}$) |…| rp(completed ipe$_{pathn}$))| (ap(completed ipe$_{path1}$)|…|ap(completed ipe$_{pathn}$)) [2]	(2)

Figure 3: Computation of current ipe and completed ipe

which contains all the relevant data. F1 can be transformed by the XSLT stylesheet into F2, from which the query XP2 selects the relevant data.

We expect our approach to queries on transformed XML data to have considerable advantages over the standard approach which transforms the entire XML document, particularly when querying for single entries, for queries on very large XML documents and for queries on remote XML data. Our approach enables the seamless incorporation of XSL processing into database management systems, which in our opinion will become increasingly important in the very near future.

For technical reasons, we restricted our presentation to a subset of XPath and a subset of XSLT. However, the approach is not limited to these subsets, and we consider it to be promising to extend it to a larger subset of XPath and XSLT.

4. ACKNOWLEDGEMENTS

This work is funded by the MEMPHIS project (IST-2000-25045).

5. REFERENCES

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

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

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

[4] S. Böttcher, and A. Türling, Checking XPath Expressions for Synchronization, Access Control and Reuse of Query Results on Mobile Clients. *Workshop: Database Mechanisms for Mobile Applications*, Karlsruhe, Germany, 2003.

[5] R. Bourret, C. Bornhövd, and A.P. Buchmann, 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.

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

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

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

[9] G. Gottlob, C. Koch, and R. Pichler, 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.

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

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

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

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

---

[2] If rp(completed ipe$_{path1}$),…,rp(completed ipe$_{pathn}$) are empty, we can optimize the equation to
completed ipe$_{new}$ = completed ipe$_{old}$ |(ap(completed ipe$_{path1}$) | … | ap(completed ipe$_{pathn}$))