XML Query Reformulation for XPath, XSLT and XQuery

Sven Groppe
(Sven.Groppe@deri.org, http://members.deri.at/~sveng/)

Tutorial at DBA 2006/Innsbruck

Structure of the Tutorial

- First 90 Minutes
  - Short Introductions
    - XML
    - XSLT including XPath
  - 3 Query Reformulation Methods including Performance Evaluation
- Second 90 Minutes
  - Intersection Test of XPath expressions
  - Reduction of Intersection Test to Satisfiability Test
  - Satisfiability Test without schema information
  - Satisfiability Test with schema information
  - Differences between XQuery and XSLT
  - 2 Caching Strategies for transformed XML data

Introductions - XML

- Extensible Markup Language (XML)
  - derived from Standard Generalized Markup Language (SGML)
  - used for
    - large-scale electronic publishing
    - exchanging a wide variety of data on the web and elsewhere
  - describes data in a tree structure consisting of nodes of type
    - document,
    - element,
    - attribute,
    - text,
    - namespace,
    - processing-instruction and
    - comment
  - Order of nodes is important (except attribute nodes)
  - Each node has an (own) identity
XML - Textual versus Graphical Representation

- Textual representation
- Graphical Representation

```
<Maps>
  <Map><title>World</title><content>... Bitmap of the World ...</content></Map>
  <Map><title>Africa</title><content>... Bitmap of whole Africa ...</content></Map>
</Maps>
```

introductions - xpath

- XPath expressions describe a node set of an XML document

```
Q_{XPath} = /Maps/Map[title='Africa']/title
```

xpath evaluation on xml document
Extensible Stylesheet Language Transformations (XSLT)

- Part of the Extensible Stylesheet Language (XSL)
- Declarative language
  - Purpose: Transformation of XML documents to
    - XML
    - HyperText Markup Language (HTML),
    - Adobe Portable Document (PDF) or
    - text documents
      by template rules

Transformation of an XSLT Stylesheet

- XSLT stylesheet:
  - XML document with root element `<xsl:stylesheet>`
  - Templates `<xsl:template match=M>` contain XPath pattern M
    - Input XSLT nodes, which select a new input node set I of the input XML document, e.g.
      - `<xsl:value-of select=I>`, which represents the content of I
      - `<xsl:apply-templates select=I>`, which calls the templates with each XML node of I
    - Output XSLT nodes, e.g.
      - `<xsl:element name=N>`, which generate an element N
      - `<xsl:attribute name=N>`, which generate an attribute N

Example of a Transformation with XSLT

XSLT Stylesheet V

- `<Maps>`
  - `<Map>`
    - `<title>World</title>`
    - `<content>`
      - `<bitmap>`
        - `… Bitmap of the world …`
      - `<content>`
    - `<Map>`
      - `<Map>`
        - `<title>Africa</title>`
        - `<content>`
          - `<area>`
            - `<label>Africa</label>`
          - `<area>`
        - `<Map>`
      - `… Bitmap of whole Africa …`
  - `<Maps>`
Motivation Query Reformulation – Possible Scenario

- Application AA
  - Company A uses format $F_1$
  - Query formulated in format $F_1$
  - Transformed document $V(D)$

- Application AB
  - Company B uses format $F_2$
  - XML Document

Different approaches for Query Reformulation (1/2)

- Query formulated in $Q_1$
- Optimize according to $Q_1$
- Transformed resultant XML fragment $V'(D)$

Different approaches for Query Reformulation (2/2)

- Reformulated query $R$ in format $F_2$
- Applied to XML document
- Resultant XML fragment $R(D)$

- View $V$
Example of a Transformation

XSLT Stylesheet V

XSLT Stylesheets

XSLT Stylesheet V

XSLT Stylesheet V

XPath on XSLT View

XPath on XSLT View

XPath on XSLT View

XPath on XSLT View
Further support

- Arbitrary mixture of relative and absolute paths within the `select` attributes => use of 2 variables instead of `ipe`

XSLT Query on XSLT View

Example of a Transformation – XSLT query on XSLT View
**Experiments: XPath query on XSLT View-DOM CloneNode**

- **IN** XML document
- **XPath** Query
- **Transform**
- **OUT**

**DOM In**
- DOM Tree
- XPath Query
- XSLT Process
- DOM Out

**DOM CloneNode**
- DOM Tree
- Clone Node
- XSLT Process
- DOM Out

**Varying the selectivity, constant file size 7 Megabyte**

- Faster for <= 30% selectivity.
Experiments – XPath query on XSLT View - SAXFilter

Xalan using SAXFilter Approach

Xalan SAXFilter: Varying the selectivity, constant file size 7 Megabyte

Faster for <= 70% selectivity.
Optimizing the XSLT view according to a query

XSLT View Optimization

1. Eliminate unnecessary XSLT instructions (no template heads)

2. Insert predicates at selected node sets
Algorithm computeFilter

(1) Algorithm computeFilter
(2) Input: N XSLT node
(3) P detected stylesheet paths
(4) initIPE XPath expression
(5) Output: f filter expression
(6) P' = \{ p' | \exists p \subseteq P \text{ such that } p' \subseteq p \} (or one of its branched paths) starting with the node N to the end of p (or one of its branched paths)
(7) for each p' \in P' do
(8) compute current ipe_{p'} of p' initialized with initIPE
(9) f = current ipe_{p'} = \text{""} \cdot current ipe_{p'}
(10) return f
Algorithm optimizeXSLTStylesheet

Input: V XSLT stylesheet
Output: V' XSLT stylesheet

1. \( \lambda = \text{evaluateXPath}(\text{XPath}, \text{root of } V, \text{null}, \text{null}, \text{null}) \)
2. For each \( n \in \lambda \) do
3. \( x = \text{getNextOutputNodes}(\text{null}, n, \text{CONTENT}) \)
4. For each \( n \in x \) do
5. \( \text{if}(n \neq \text{null} \text{ and } n \neq \langle \text{xsl:template} \rangle) \) delete \( n \) and its subtree in \( V' \)
6. \( f = \text{computeFilter}(n, p, \text{null}) \)
7. \( n = \text{createElementNode}\left(\langle \text{xsl:if test=} \text{null}\rangle, \text{null}\right) \)
8. \( \text{setLocalInputPathExpression}(n, n \text{.getLocalInputPathExpression()} + \text{null}) \)
9. \( n = \text{createElementNode}\left(\langle \text{xsl:if test=} \text{null}\rangle, \text{null}\right) \)
10. \( \text{setLocalInputPathExpression}(n, n \text{.getLocalInputPathExpression()} + \text{null}) \)
11. \( \text{return } V' \)
Excluding XSLT instructions

```
<xsl:stylesheet>
  <xsl:template match="/child::r">
    <xsl:element name="s">
      <xsl:for-each select="child::a">
        <xsl:element name="c">
          <xsl:attribute name="d">
            <xsl:value-of select="attribute::b"/>
          </xsl:attribute>
        </xsl:element>
        <xsl:element name="e">
          <xsl:attribute name="f">
            <xsl:value-of select="attribute::b"/> h times
          </xsl:attribute>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Q = /child::s/child::c[attribute::d < X]/attribute::d
XSLT instructions excluded – Xalan / DOM

XSLT instructions excluded – Xalan / SAX

Experiment with real data - DBLP

- DBLP data contains a bibliography of publications

```
xsl:stylesheet
<xsl:template match="dblp">
  <xsl:element name="result">
    <xsl:apply-templates select="child::node()"/>
  </xsl:element>
</xsl:template>
<xsl:template match="child::node()">
  <xsl:element name="entry">
    <xsl:element name="Name">
      <xsl:value-of select="child::author"/>
    </xsl:element>
    <xsl:element name="Title">
      <xsl:value-of select="child::title"/>
    </xsl:element>
    <xsl:element name="Year">
      <xsl:value-of select="child::year"/>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Q1= /child::result/child::entry[child::Year < X]
selectivity < 11.8%

Q2= /child::result/child::entry[child::Year < X]/child::*
selectivity > 11.8%
Summary I

- Transformation of only the sufficient XML fragment, if transformed XML documents are queried
- avoids problems of replication
- saves processing time (for transformation), depending of selectivity of queries and XSLT processor
  - Xalan (CloneNode <= 30%, SAXFilter <= 80%)
  - Saxon (CloneNode <= 20%, SAXFilter <= 55%)
- scalable and efficient especially when using SAXFilter
- saves transportation costs in distributed scenarios
Summary II

- Optimize XSLT View \( V \) according to XPath query \( Q_{\text{path}} \).
  - Avoid step for retrieving resultant XML fragment \( R(D) \).
  - Faster whenever many XSLT instructions can be excluded
    - DOM: queries with < 80% selectivity are faster (Xalan, Saxon)
    - SAX: queries with <= 60% selectivity are faster (Xalan)
    - queries with <= 70% selectivity are faster (Saxon)

Questions / Remarks?

- Sven Groppe, XML Query Reformulation for XPath, XSLT and XQuery, Sierke-Verlag, Göttingen, 2005. ISBN 3-933893-24-0

Absolute Part and Relative Part of an XPath Expression

- An XPath expression \( I \) can be divided into
  - an absolute part \( \text{ap}(I) \), which contains only absolute paths, and
  - a relative part \( \text{rp}(I) \), which contains only relative paths
  so that \( \text{ap}(I) \cup \text{rp}(I) \) is equivalent to \( I \)

Example:

\[
\begin{align*}
I &= (/E1 | E2/E3 | E4) /E5 \\
\text{rp}(I) &= (E2/E3 | E4) /E5 \\
\text{ap}(I) &= /E1/E5
\end{align*}
\]