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

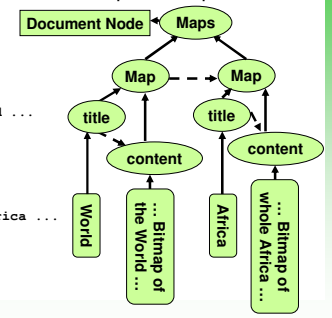
- Extensible Markup Language (XML)
  - derived from Standard Generalized Markup Language (SGML)
  - used for
    - large-scale electronic publishing
    - exchanging a wide variety of data on the web and elsewhere
  - describes data in a tree structure consisting of nodes of type
    - document,
    - 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

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

Graphical Representation



parent ← child
node ⟶ ▸ next sibling

Text Node

Element Node

Introductions - XPath

- XPath expressions describe a node set of an XML document

location steps

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

short form for child::Maps (axis: child, name test: Maps)

predicate

XPath Evaluation on XML document

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

/Maps/Map[title='Africa']/title

Extensible Stylesheet Language Transformations (XSLT)

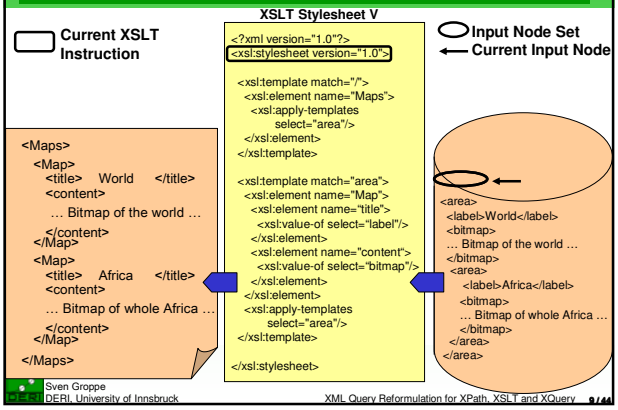
- Part of the Extensible Stylesheet Language (XSL)
- Declarative language
  - Purpose: Transformation of XML documents to
    - **XML**,
    - HyperText Markup Language (HTML),
    - Adobe Portable Document (PDF) or
    - text documents

    by template rules

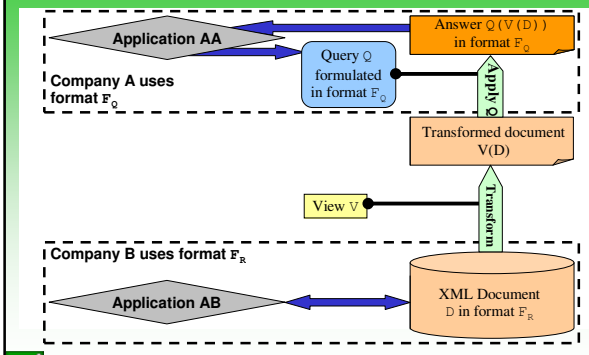
Transformation of an XSLT Stylesheet

- 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`

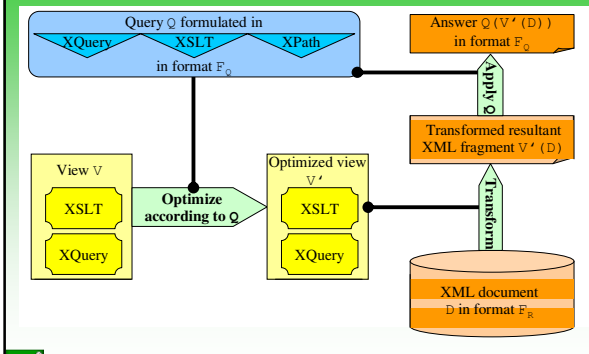
Example of a Transformation with XSLT



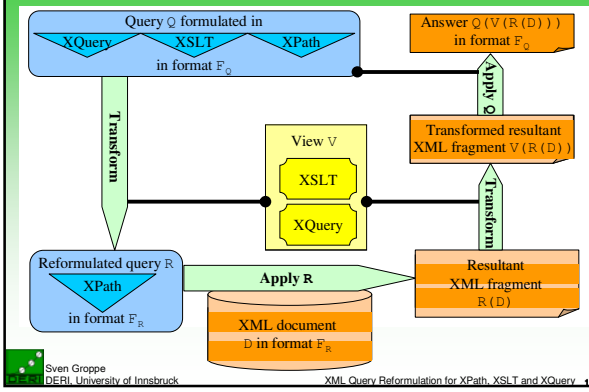
Motivation Query Reformulation – Possible Scenario



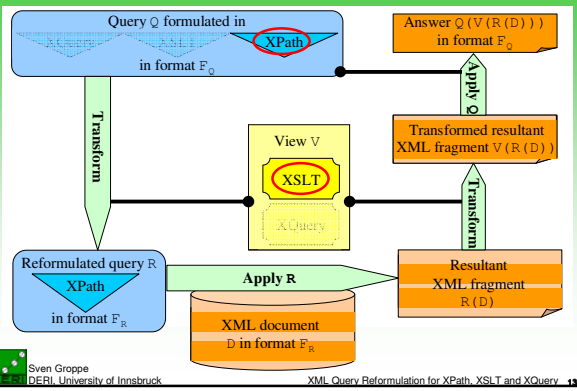
Different approaches for Query Reformulation (1/2)



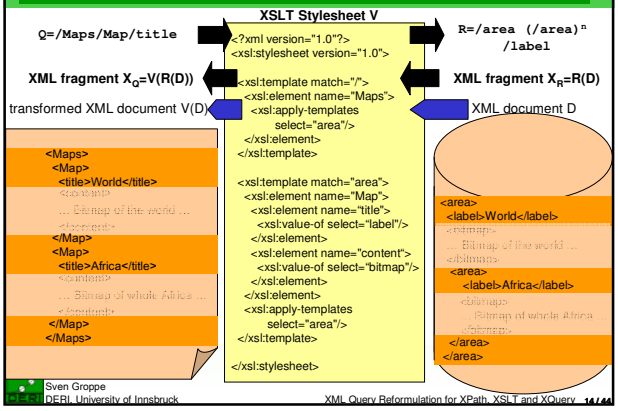
Different approaches for Query Reformulation (2/2)



XPath Query on XSLT View

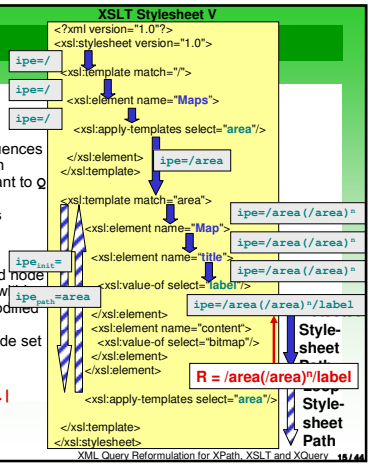


Example of a Transformation



XPath on XSLT View

- Q=/Maps/Map/title
- 1. part of the main problem:
  - Determine *all* execution sequences of the XSLT processor, which generate output that is relevant to Q
    → Modified XPath evaluator processing XSLT stylesheets instead XML documents
- 2. part of the main problem:
  - Determine the whole selected node set of the original document w… the searched paths of the modified XPath evaluator
    → Determine logically the node set (XPath expression) ⇒ R
- R = $ipe_{\text{after last node 1}}$ |…| $ipe_{\text{after last node n}}$

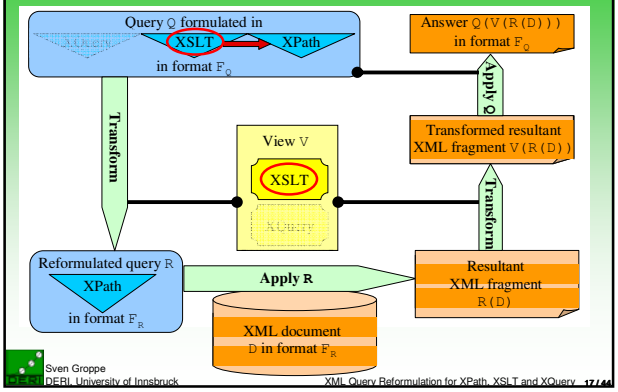


Further support

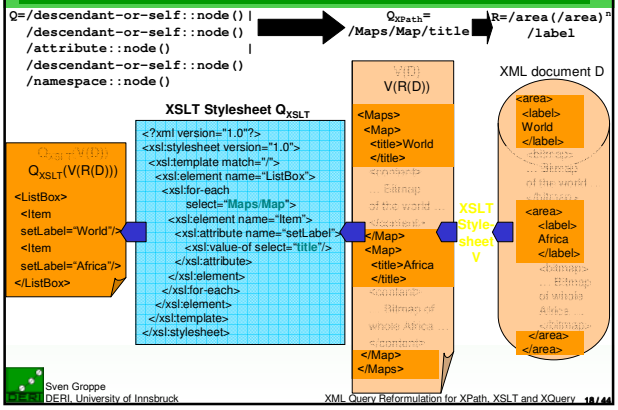
- Arbitrary mixture of relative and absolute paths within the **`select`** attributes => use of 2 variables instead of **`ipe`**

Current Node	Computation of `current ipe` and `completed ipe`
Neither input node nor test node without attached paths	$current\ ipe_{new} = current\ ipe_{old}$ $completed\ ipe_{new} = completed\ ipe_{old}$
Input node with local input path expression `I`	$current\ ipe_{new} = if(rp(I)\ is\ empty)\ ap(I)\ else\ current\ ipe_{old}\ /\ rp(I)\ |\ ap(I)$ $completed\ ipe_{new} = if(ap(I)\ is\ empty)\ completed\ ipe_{old}\ else\ completed\ ipe_{old}\ |\ current\ ipe_{old}$
Test node with Boolean test expression `T`	$current\ ipe_{new} = current\ ipe_{old}[T]$ $completed\ ipe_{new} = completed\ ipe_{old}$
Attached attached stylesheet path	$current\ ipe_{init} = current\ ipe_{old}$ $completed\ ipe_{init} = completed\ ipe_{old}$ $current\ ipe_{new} = current\ ipe_{old}$ $completed\ ipe_{new} = current\ ipe_{path}\ |\ completed\ ipe_{path}$
Attached filter stylesheet path according to a filter `[P op c]` where `P` is a path expression, `op` ∈{=, !=,>,>=,<,<=} and `c` is a constant	$current\ ipe_{init} = empty$ $completed\ ipe_{init} = completed\ ipe_{old}$ $current\ ipe_{new} = current\ ipe_{old}[current\ ipe_{path}=c]$ $completed\ ipe_{new} = completed\ ipe_{path}$
Attached loop stylesheet paths 1.. n	$current\ ipe_{init} = empty$ $completed\ ipe_{init} = empty$ $current\ ipe_{new} = (current\ ipe_{old}|ap(current\ ipe_{path1})\ |\ldots|ap(current\ ipe_{pathn}))$ $(/rp(current\ ipe_{path1})|\ldots|/rp(current\ ipe_{pathn}))^n$ $completed\ ipe_{new} = if(rp(completed\ ipe_{path1})\ \ldots=\ rp(completed\ ipe_{pathn})\ =\ empty)$ $(completed\ ipe_{old}\ |(ap(completed\ ipe_{path1})|\ldots|ap(completed\ ipe_{pathn})))$ $else\ (completed\ ipe_{old}\ |\ current\ ipe_{new}\ /(rp(completed\ ipe_{path1})\ |\ldots|$ $rp(completed\ ipe_{pathn}))|(ap(completed\ ipe_{path1})|\ldots|ap(completed\ ipe_{pathn})))$

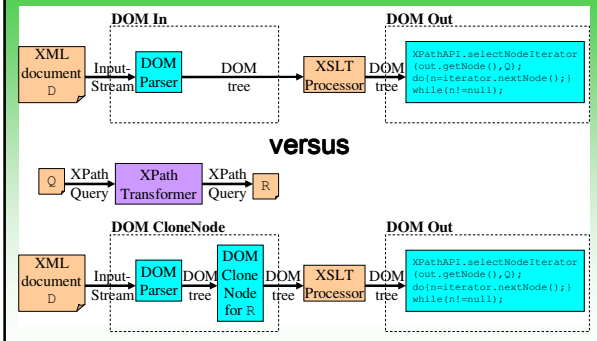
XSLT Query on XSLT View



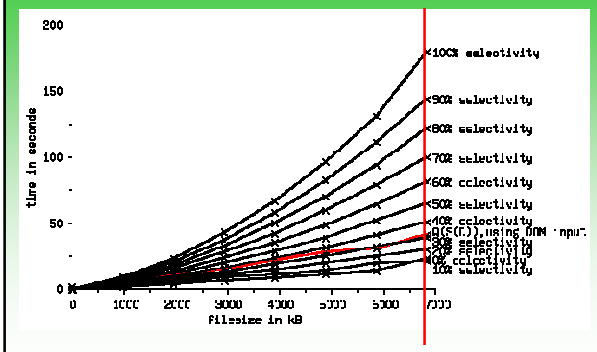
Example of a Transformation – XSLT query on XSLT View



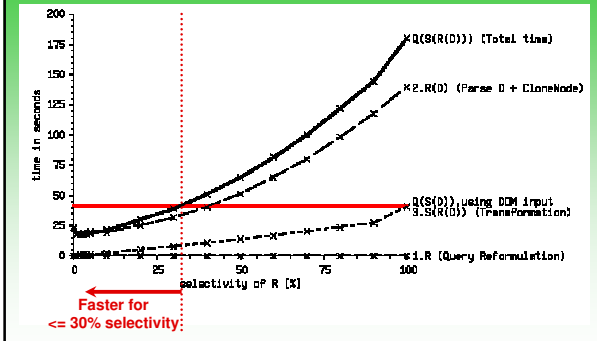
Experiments-XPath query on XSLT View-DOM CloneNode



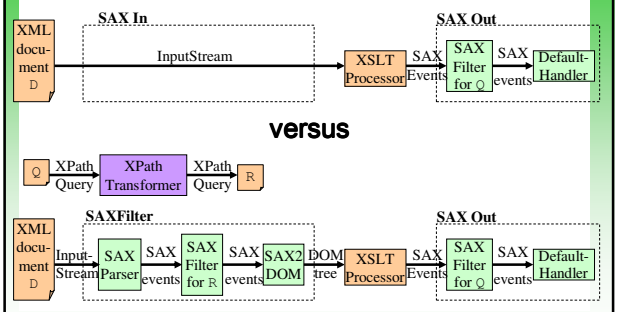
Xalan using DOM CloneNode Approach



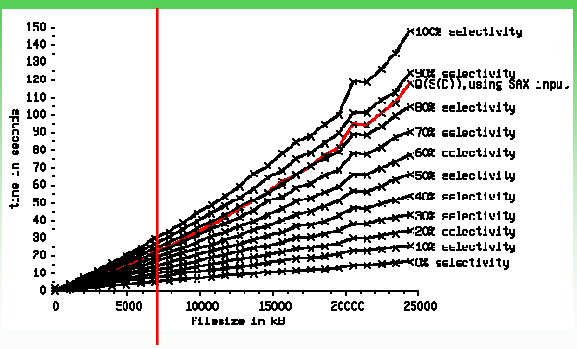
Xalan CloneNode: Varying the selectivity, constant file size 7 Megabyte



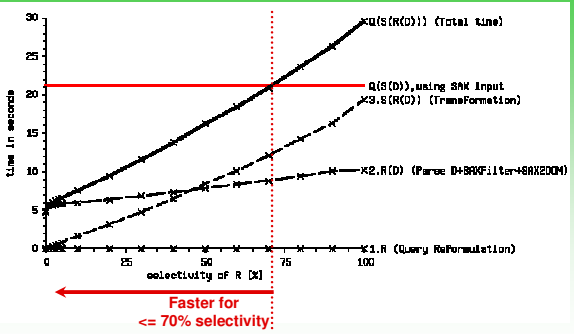
Experiments – XPath query on XSLT View - SAXFilter



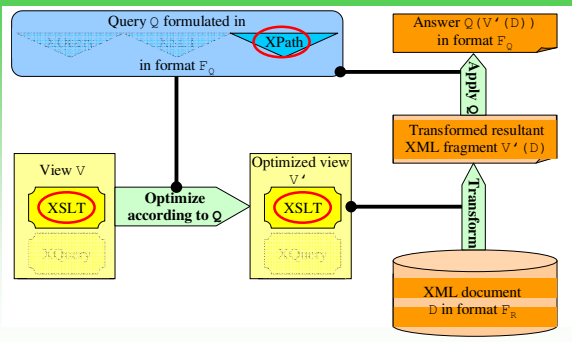
Xalan using SAXFilter Approach



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



Optimizing the XSLT view according to a query



XSLT View Optimization

- Q=/Maps/Map [title="Africa"]/title
- 1. Eliminate unnecessary XSLT instructions (no template heads)

XSLT Stylesheet V

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0">
 <xsl:template match="/">
  <xsl:element name="Maps">
   <xsl:apply-templates select="area"/>
  </xsl:element>
 </xsl:template>
 <xsl:template match="area">
  <xsl:element name="Map">
   <xsl:element name="title">
    <xsl:value-of select="label"/>
   </xsl:element>
  </xsl:element>
  <xsl:apply-templates select="area"/>
 </xsl:template>
</xsl:stylesheet>
```

="Africa"

Filter Style-sheet Path

Detected Style-sheet Path

Loop Style-sheet Path

XSLT View Optimization

- Q=/Maps/Map [title="Africa"]/title
- 1. Eliminate unnecessary XSLT instructions (no template heads)
- 2. Insert predicates at selected node sets

XSLT Stylesheet V

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0">
 <xsl:template match="/">
  <xsl:element name="Maps">
   <xsl:apply-templates select="area
   [(/area)"[label="Africa"]/label]"/>
  </xsl:element>
 </xsl:template>
 <xsl:template match="area">
  <xsl:element name="Map">
   <xsl:element name="title">
    <xsl:value-of select="label
   </xsl:element>
  </xsl:element>
  <xsl:apply-templates select="area"/>
 </xsl:template>
</xsl:stylesheet>
```

ipe=

ipe=(/area)*

ipe_init=

ipe=(/area)*[label="Africa"]

ipe=

ipe=(/area)*[label="Africa"]

ipe_path=area

ipe_path=label

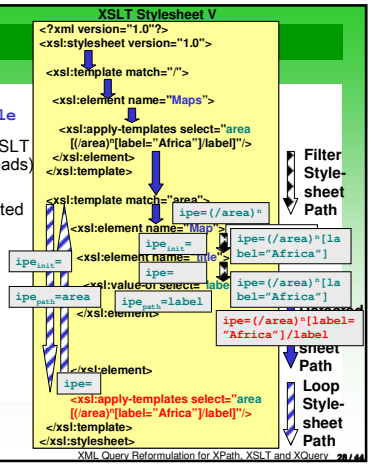
ipe=(/area)*[label="Africa"]/label

Filter Style-sheet Path

Loop Style-sheet Path

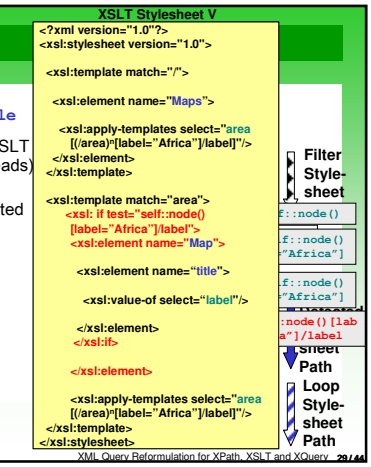
XSLT View Optimization

- Q=/Maps/Map [title="Africa"]/title
- 1. Eliminate unnecessary XSLT instructions (no template heads)
- 2. Insert predicates at selected node sets



XSLT View Optimization

- Q=/Maps/Map [title="Africa"]/title
- 1. Eliminate unnecessary XSLT instructions (no template heads)
- 2. Insert predicates at selected node sets
- 3. Insert test nodes at split XSLT nodes



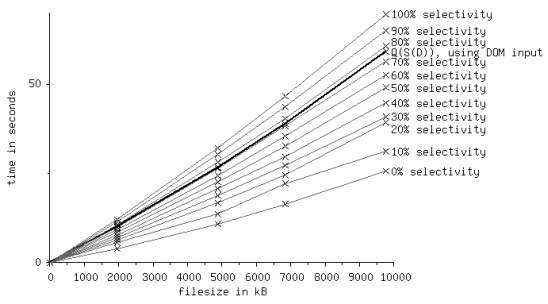
Algorithm computeFilter

```
(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' | ∃p∈P ∧ p' is subpath of 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'∈P'={p1', …, pr'} do
(8)       compute current ipe_p, of p' initialized with initIPE
(9)   f = current ipe_p1, + "|"+…+"|" + current ipe_pr,
(10)  return f
```

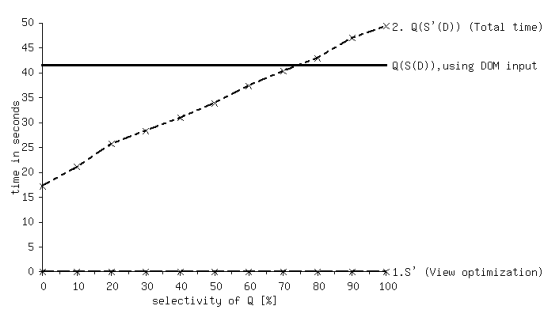
Algorithm optimizeXSLTStylesheet

```
(1) Algorithm optimizeXSLTStylesheet
(2) Input:      V   XSLT stylesheet
(3)             QXPath  XPath query
(4) Output:     V'  XSLT stylesheet

(5) l = evaluateXPath(QXPath,<(QXPath, root of V, null, null, null)>)
(6) for each p1 ∈ l do
(7)     P.addAll(getNextOutputNodes(null,p1,CONTENT))
(8) V' = V.clone()
(9) for each N ∈ V' do
(10)    if(N∉P and N≠<xsl:template>) delete N (and its subtree) in V'
(11)    else if(N is input node)
(12)        f = computeFilter(N,P,"")
(13)        N.setLocalInputPathExpression(
                                N.getLocalInputPathExpression()+"["+f+"]")
(14)    else if(N is split XSLT Node)
(15)        f = computeFilter(N,P,"self::node()")
(16)        Nn = createElementNode("<xsl:if test='" + f + "'/>")
(17)        Nn.setParentNode(N.getParentNode())
(18)        N.setParentNode(Nn)
(19)return V'
```

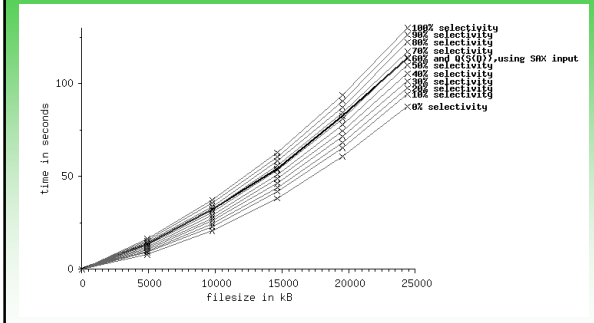

Optimizing XSLT View – Xalan / DOM



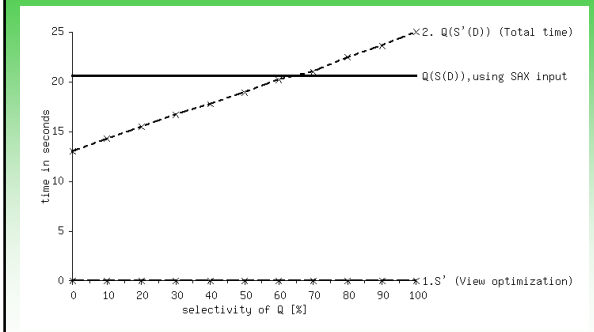
Optimizing XSLT View – Xalan / DOM



Optimizing XSLT View – Xalan / SAX



Optimizing XSLT View – Xalan / SAX



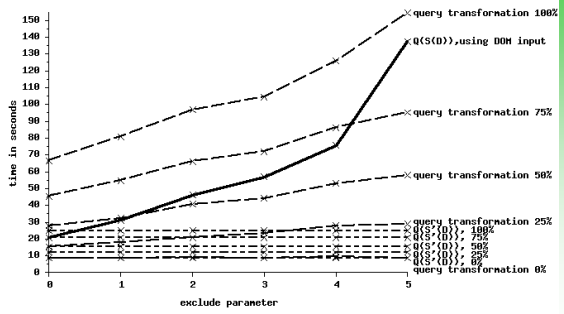
Excluding XSLT instructions

```
<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"/>
            </xsl:attribute>
        </xsl:element>

    </xsl:for-each>
  </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

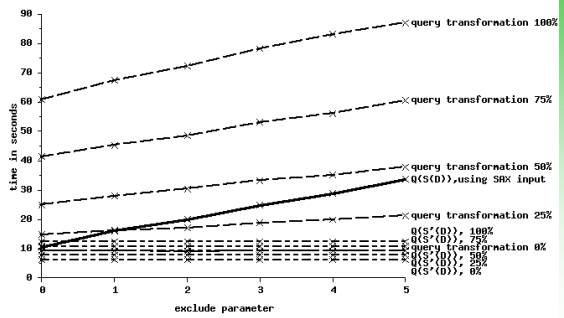
h times

Q_{XPath} = /child::s/child::c[attribute::d < X]/attribute::d

XSLT instructions excluded – Xalan / DOM



XSLT instructions excluded – Xalan / SAX

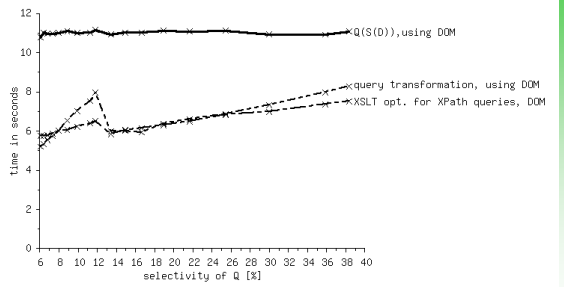


Experiment with real data - DBLP

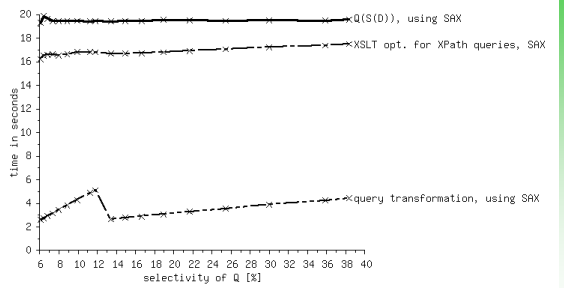
- DBLP data contains a bibliography of publications

```
<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>
</xsl:stylesheet>

Q1=/child::result/child::entry[child::Year < X]/child::Year    selectivity < 11.8 %
Q2=/child::result/child::entry[child::Year < X]/child::*       selectivity > 11.8 %
```

DBLP data – Xalan DOM



DBLP data – Xalan SAX



Summary I

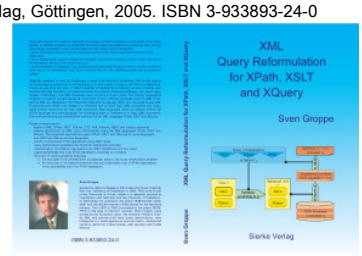
- Transformation of only the sufficient XML fragment, if transformed XML documents are queried
  - avoids problems of replication
  - saves processing time (for transformation),
    - depending of selectivity of queries and XSLT processor
      - Xalan (CloneNode <= 30%, SAXFilter <= 80%)
      - Saxon (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_{XPath}
  - 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



Sven Groppe
sven.groppe@deri.org
http://members.deri.at/~sveng/

Absolute Part and Relative Part of an XPath Expression

- An XPath expression `I` can be divided into
  - an *absolute part* `ap(I)`, which contains only absolute paths, and
  - a *relative part* `rp(I)`, which contains only relative paths

  so that `ap(I)` ∪ `rp(I)` is equivalent to `I`

- Example:

```
I     =  (/E1 | E2/E3 | E4) /E5

rp(I) =  (E2/E3 | E4) /E5
ap(I) =  /E1/E5
```