Composing XSL Transformations with XML Publishing Views

Chengkai Li
University of Illinois at Urbana-Champaign

Philip Bohannon
Lucent Technologies, Bell Labs

Henry F. Korth
Lehigh University

PPS Narayan
Lucent Technologies, Bell Labs

SIGMOD 2003
Motivation

XML: popular for data representation and exchange

- The data: stored in RDBMS
  - Vast majority of existing data stored in RDBMS
  - Efficiency, robustness of RDBMS for XML applications
  - XML Publishing Views (SilkRoute, XPERANTO)

- The query: expressed as XSLT
  - Designed for document transformation
  - Popular as XML query language

How to evaluate queries on relational data posed in XSLT?
**XML Publishing**

**view query**: specifies the mapping between relational tables and resulting XML document.

Diagram:
- View query
- Query Logic
- SQL queries
- Relational DB
- Tagger
- Publisher
- XML data
Example: tables and schema of view

<table>
<thead>
<tr>
<th><strong>METROAREA</strong></th>
<th><strong>HOTEL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>metroid</td>
<td>name</td>
</tr>
<tr>
<td>NYC</td>
<td>New York City</td>
</tr>
<tr>
<td>CHI</td>
<td>Chicago</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ROOM</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>hotel_id</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

/  

metro (name)  

hotel (name, star)  

total_room  

room (#)  

available
Example: published XML document

```
metro ("New York City")
  hotel ("Hyatt", 2)
    total_room
      room (101) room (102)
      2

metro ("Chicago")
  hotel ("Hilton", 4)
    total_room
      room (1) room (2) available
      2
```
Example of View Query

- **Relational Schema**
  - *Metroarea*(metroid, metroname)
  - *Hotel*(hotelid, hotelname, starrating, metro_id)
  - *Room*(hotel_id, room#, available)

- **Desired Hierarchical Structure of Published XML**

```xml
<metro>
  $m = \text{SELECT metroid, metroname FROM metroarea}
</metro>

<hotel>
  $h = \text{SELECT * FROM hotel WHERE metro_id = }$m.metroid AND starrating > 4
</hotel>

<total_room>
</total_room>
<room>
</room>
<available>
</available>
```
Evaluate XSLT queries on relational data?
Approach 1: Materialization

- XML parsing
- Relational engine for XML processing
- Unnecessary materialization of nodes

<table>
<thead>
<tr>
<th></th>
<th>Approach 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML parsing</td>
<td>✗</td>
</tr>
<tr>
<td>Relational engine</td>
<td>✗</td>
</tr>
<tr>
<td>of XML processing</td>
<td>✗</td>
</tr>
<tr>
<td>Unnecessary materialization of nodes</td>
<td>✗</td>
</tr>
</tbody>
</table>
Unnecessary Materializations

nodes that do not satisfy type requirement
nodes that do not satisfy selection condition
nodes not involved in output

rule 1. metro [@name="Chicago"]: output name
rule 2. hotel [@star>3]: no output
rule 3. total_room: output total number of rooms
Approach 2: View Composition

<table>
<thead>
<tr>
<th></th>
<th>Approach 1</th>
<th>Approach 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML parsing</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>relational engine for XML processing</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>unnecessary materialization of nodes</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Algorithm Overview

nodes that do not satisfy type requirements:

What type of nodes are accessed?

nodes that do not satisfy selection condition:

What are the instances of these types of nodes?

nodes not involved in output:

How do we avoid materializing uninvolved nodes?
Algorithm Overview

- **view query**
- **XSLT stylesheet**

Context Transition Graph (CTG)

Traverse View Query (TVQ)

Output Tag Tree (OTT)

- What type of nodes are accessed?
- What are the instances of these types of nodes?
- How do we avoid materializing nodes uninvolved in output?
Example of XSLT Stylesheet

R1:
<xsl:template match="/">
  <result_metro>
    <A/>
    <xsl:apply-templates select="metro/hotel/total_room"/>
  </result_metro>
</xsl:template>

R2:
<xsl:template match="total_room">
  <result_total>
    <B/>
    <xsl:apply-templates select="../../available/../../../room"/>
  </result_total>
</xsl:template>

R3:
<xsl:template match="metro/hotel/room">
  <xsl:value-of select="."/>
</xsl:template>
Template Rule

A stylesheet consists of a set of template rules.

\[
R = \langle \text{match}\_\text{pattern}(r), \text{output}(r), \text{select}\_\text{expression}(r) \rangle
\]

\[
\text{<xsl:template match="/">}
\text{<result\_metro>}
\text{<A/>}
\text{<xsl:apply-templates select="metro/hotel/total\_room"/>}
\text{</result\_metro>}
\text{</xsl:template>}
\]

match the root
generate output
process \textit{total\_room} for all hotels of all metro areas
Simplified Representation

R1:
match="/"
select="metro/hotel/total_room"

R2:
match="total_room"
select="../available/../room"

R3:
match="metro/hotel/room"
XSLT processing

R1:
match="/"
select="metro/hotel/total_room"

R2:
match="total_room"
select="../available/..//room"

R3:
match="metro/hotel/room"
Context Transition Graph (CTG)

MATCHQ: nodes   SELECTQ: edges

(/, R1)

(total_room, R2)

total_room: context node

(room, R3)

room: new context node

CTG: Which type of nodes are accessed?

Document instances of <total_room> may be matched by R2, which further selects document instances of <room>, which may be matched by R3.
Instances of accessed nodes?

(/, R1)

(total_room, R2)
$t_{\text{new}} = \ldots$

(room, R3)
$r_{\text{new}} = ?$
Traverse View Query (TVQ)

(\, R1)

(total\_room, R2)
$t\_new = \ldots$

(room, R3)
$r\_new = \text{SELECT} * \text{ FROM} \text{ room}$
WHERE hotel_id=$t\_new.$hotelid
AND EXISTS (SELECT * FROM room
WHERE hotel_id=$t\_new.$hotelid
AND available = TRUE)

TVQ: Instances of accessed nodes
TVQ: Instances of accessed nodes

R2:
match="total_room"
select="../available/../../../room"

R3:
match="metro/hotel/room"

Select-Match Tree

$\text{t}_\text{new}=\ldots$

$\text{r}_\text{new}=\text{room}$
Select-Match Tree: How does context transition happen?

(\, R1)

(totol_room,R2)
$t_{new}=\ldots$

(room,R3)
$r_{new}=?$

Select-Match Tree

<metro>

<hotel>

<total_room>  <available>

<room>
UNBIND: Select-Match Tree $\rightarrow$ tag query

$$(/, R1)$$

$$(total\_room, R2)$$
$t\_new= \ldots$

$$(room, R3)$$
$r\_new=?$

Select-Match Tree

$$<\text{metro}>$$

$$<\text{hotel}>$$

$$<\text{total\_room}>$$

$$<\text{available}>$$

$$<\text{room}>$$
UNBIND: Select-Match Tree → tag query

Select-Match Tree

<metro>

<hotel>

<total_room>

<available>

<room>

$r = SELECT * FROM room
WHERE hotel_id=$h.hotelid
UNBIND: Select-Match Tree $\rightarrow$ tag query

(\/, R1)

(tot\_room, R2)
$t\_new = \ldots$

(room, R3)
$r\_new = \text{SELECT} \ast \text{FROM room WHERE hotel\_id} = \$t\_new.hotelid$

Select-Match Tree

<metro>

<hotel>

<total\_room>

<available>

$r = \text{SELECT} \ast \text{FROM room WHERE hotel\_id} = \$h.hotelid$
UNBIND: Select-Match Tree → tag query

\[(/, \text{R1})\]

\[(\text{total\_room, R2})\]
\[\$t\_new = \ldots\]

\[(\text{room, R3})\]
\[\$r\_new = \text{SELECT} \ast \text{FROM room} \]
\[\text{WHERE hotel\_id} = \$t\_new\text{.hotelid}\]

```
<metro>
  <hotel>
    <total\_room>
      <room>
        <available>
          $a=\text{SELECT} \ast \text{FROM room}
          \text{WHERE hotel\_id} = \$h\text{.hotelid}
          \text{AND available = TRUE}
        
$\text{SELECT} \ast \text{FROM room}
\text{WHERE hotel\_id} = \$t\_new\text{.hotelid}
```

Lucent Technologies
Bell Labs Innovations
UNBIND: Select-Match Tree → tag query

(\), R1)

(total_room,R2)
$t_{\text{new}} = \ldots$

(room,R3)
$r_{\text{new}} = \text{SELECT * FROM room WHERE hotel_id} = t_{\text{new}}.\text{hotelid AND available} = \text{TRUE}$
**UNBIND: General Cases**

General Select-Match Tree with Predicates

- Unbind along the lowest common ancestor to the new context node (FROM)
- Nest of all sub-trees not on the two paths (WHERE EXISTS)
- Attribute access of all nodes (WHERE)

![Diagram](image-url)
Output Tag Tree

(root, R1)

(total_room, R2)

(room, R3)
Output Tag Tree (OTT)

(root, R1)

(total_room, R2)

(room, R3)

R1:
<xs1:template match="`/`">
 <result_metro> <A/>
  <xs1:apply-templates select="`...`"/>
</result_metro>
</xs1:template>
Output Tag Tree (OTT)

R2:
<xsl:template match="`total_room'">  
  <result_total> <B/>  
    <xsl:apply-templates select="`...'"/>  
  </result_total>  
</xsl:template>
Output Tag Tree (OTT)

(root, R1)

(total_room, R2)

(room, R3)

R3: `<xsl:template match="`metro/hotel/room'">  
    <xsl:value-of select='.'/>
  </xsl:template>`
New View Query

Forced Unbind during the generation of OTT
**XSLT_basic**

- no type coercion
- no document order
- no “//”
- no function
- no variable and parameter
- no recursion
- no predicate in expression
- no flow-control elements
  - (<xsl:if>, <xsl:for-each>, <xsl:choose>)
- no conflicting rule resolution
- select of <xsl:value-of> is “.”
Relaxing Assumptions

- recursion
- predicate in expression
- flow-control elements
  
  \(<\text{xsl:if}>, \ <\text{xsl:for-each}>, \ <\text{xsl:choose}>\)
- conflicting rule resolution
- \textit{select} of \textit{<xsl:value-of>} be other than “.” and “@attribute”
Summary

- **Problem**: Composing XSL Transformations with XML publishing views

- **Advantages compared with materialization approach**

- **Algorithm**
  - Context Transition Graph
  - Traverse View Query
  - Output Tag Tree

- **Relaxing Assumptions**
Future Work

- //: CTG graph $\rightarrow$ multigraph
- recursion
Related Work

- Translating XSLT into SQL queries: Jain et al, WWW 02

- XML publishing middleware
  - SilkRoute: Fernandez et al, WWW 00, SIGMOD 01
  - XPERANTO: Carey et al, WebDB 00 & Shanmugasundaram et al, VLDB 01

- Incorporating XSL processing into database engines: Moerkotte, VLDB 02