SilkRoute: Trading between Relations and XML

By: Mary Fernandez, Wang-chiew Tan, Dan Suciu

Presented by: Wenguo Liu

11/11/2003

Outline

- Introduction
- SilkRoute's Architecture
- Query Composition
- Discussion

Introduction: Motivation

- Need a tool for converting relational data into XML
  - XML is the standard format for data exchange on the Internet
  - Most data is stored in RDBMS or ORDBMS
- Properties of such tool
  - General – No public DTD matches exactly a proprietary relational schema
  - Dynamic – Only materialize when needed
    - Get data from execution, not from definition
  - Efficient – Utilize the mature RDBMS query processors

Introduction: Motivating Example

- Schema of supplier's relational database
  - Clothing(pid, item, category, description, price, cost)
  - SalePrice(pid, price)
  - Problems(pid, code, comments)

Introduction: Motivating Example

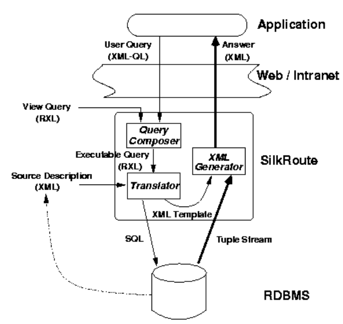
- DTD

```
<?xml encoding="US-ASCII"?>
<!ELEMENT supplier (company, product*)>
<!ELEMENT product (name, category, description, retail, sale?,
   report*)>
<!ATTLIST product ID ID>
<!ELEMENT company (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT retail (#PCDATA)>
<!ELEMENT sale (#PCDATA)>
<!ELEMENT report (#PCDATA)>
<!ATTLIST report code (size|defective|style) #REQUIRED>
```


Introduction: Motivating Example

- Supplier part
  - Convert its relational data into a valid XML view conforming to the DTD
- Reseller part
  - Supplier's relational schemas are not accessible
  - Access the data by formulating queries over the XML view defined by supplier
    - Retrieve products whose sale price is less than 50% of retail price.
    - Count the number of "defective" reports for a product.

SilkRoute's Architecture



SilkRoute's Architecture – One Sentence

- Middleware between RDBMS and an application accessing the data over the Web

SilkRoute's Architecture – Five Parts

- View Query (RXL)
  - RXL query written by database administrator
  - Defines the XML virtual view of the DB
- Application
  - Formulates an user query in XML-QL over the virtual view and sends to SilkRoute
- SilkRoute Query Composer
  - Input: RXL view query and the XML-QL user query
  - Output: A new RXL query (**Executable Query**)

SilkRoute's Architecture – Five Parts

- SilkRoute Translator
  - Input
    - Executable Query (E) from query composer
    - Description of the relational schema
  - Partition E
    - Data-extraction part (SQL queries)
    - XML-construction part (XML template)
  - Send SQL queries to RDBMS server
  - Output
    - One tuple stream per SQL query
- SilkRoute XML Generator
  - Merge tuple streams from Translator

Architecture: RXL View Query

```
from Clothing $c
where $c.category = "outerwear"
construct <product>
                <name>$c.item</name>
                <category>$c.category</category>
                        <retail>$c.price</retail>
                  </product>
```

```
<product><name>...</name><category>...</category><retail>...</retail>
</product>
<product><name>...</name><category>...</category><retail>...</retail>
</product>
...
```


RXL features: Nested Queries

```
construct <view>
{ from Clothing $c
   construct <product>
     <name>$c.item</name>
        {       from Problems $p
                where $p.pid = $c.cid
                construct
                <report>$p.comments</report>
        }
        </product>
}
</view>
```


RXL features: Skolem functions

```
from Clothing $c
construct <category ID=Cat($c.category)
   name=$c.category>
     <product>$c.item</product>
           </category>
```

```
<category><product>p1</product><product>p2</product></category>
<category><product>p3</product><product>p4</product></category>
```

If Without Skolem Term

```
<category><product>p1</product></category>
<category><product>p2</product></category>
```


RXL features: Block Structure

```
construct <view ID=View()>
{ from Clothing $c
   construct <product ID=Prod($c.item)>
              <name  ID=Name($c.item)>$c.item</name>
              <price ID=Price($c.item, $c.price)>$c.price</price>
                    </product> }
{ from Clearance $d
   where $d.disc > 50
   construct <product ID=Prod($d.prodname)>
   <name ID=Name($d.prodname)>$d.prodname</name>
    <discount
   ID=Discount($d.prodname,$d.disc)>$d.disc</discount>
              </product> }
</view>
```


RXL View Query (V : RDB -> XML)

```
construct
<supplier ID=Supp()>
   <company ID=Comp()>"Acme Clothing"</company>
   {
           from Clothing $c
            where $c.category = "outerwear"
           construct
           <product ID=Prod($c.pid)>
                       <name ID=Name($c.pid,$c.item)>$c.item</name>
                       <category ID=Cat($c.pid,$c.category)>$c.category</category>
                       <description ID=Desc($c.pid,$c.description)>$c.description</description>
                       <retail ID=Retail($c.pid,$c.price)>$c.price</retail>
                       { from SalePrice $s
                         where $s.pid = $c.pid
                         construct
                         <sale ID=Sale($c.pid,$s.pid,$s.price)>$s.price</retail>
                       }
                       { from Problems $p
                         where $p.pid = $c.pid
                         construct
                         <report code=$p.code ID=Prob($c.pid,$p.pid,$p.code,$p.comments)>
                                 $p.comments
                         </report>
                       }
           </product>
  }
</supplier>
```


Architecture: XML-QL User Query (U : XML -> XML)

```
construct
  <results> {
       where <supplier>
               <company>$company</company>
               <product>
                      <name>$name</name>
                      <retail>$retail</retail>
                      <sale>$sale</sale>
               </product>
            </supplier> in "http://acme.com/products.xml",
            $sale < 0.5 * $retail
       construct
            <result ID=Result($company)>
               <supplier>$company</supplier>
               <name>$name</name>
            </result>
  } </results>
```

Pattern

Filter

Template

Architecture: The Query Composer (C=U(V(DB)))

```
construct
  <results>
  { from Clothing $c, SalePrice $s
    where    $c.category = "outerwear",
             $c.pid = $s.pid,
             $s.price < 0.5 * $c.retail
    construct
      <result ID=Result("Acme Clothing")>
             <supplier>"Acme Clothing"</supplier>
             <name ID=Name($c.pid, $c.item)>$c.item</name>
      </result>
}
</results>
```


Architecture: Translator and XML Generator

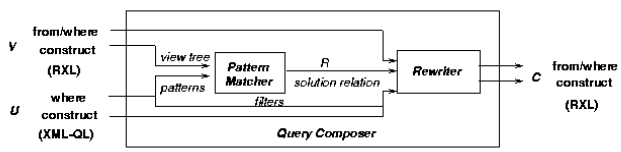
- Translator: C => SQL Query

```
select c.pid as pid, c.item as item
from Clothing c, SalePrice s
where      c.category = "outerwear",
           c.pid = s.pid,
           s.price < 0.5 * c.retail
sort by c.pid
```

- Translator: C => XML template

```
<results>
  <result ID=Result("Acme Clothing")>
    <supplier> "Acme Clothing" </supplier>
    <name ID=Name($pid, $item)>$item</name> </result>
</results>
```


Query Composition: Diagram



Query Composition: Steps

- Pattern Matching
  - Represents V by a view tree
    - View tree
      - Global template
      - A set of datalog rules
  - Evaluate U on the view tree
- Query Rewriting

Recall: V (RDB -> XML)

```
1. construct
2. <supplier ID=Supp()>
3.   <company ID=Comp()>"Acme Clothing"</company>
4.   {
…
8.        <product ID=Prod($c.pid)>
9.                  <name ID=Name($c.pid,$c.item)>$c.item</name>
10.                 <category ID=Cat($c.pid,$c.category)>$c.category</category>
…
25.       </product>
26. }
27. </supplier>
```


Step1: Pattern Matching

- Construct the view tree
  - For each Skolem function F in V
    - Rule F(x, y, …) :- body
- Examples

  <supplier ID=Supp()>

  Supp() :- true

  <company ID=Comp()>Acme Clothing</company>

  Comp() :- true

  <product ID=Prod($cpid)>

  Prod($cpid) :- Clothing($cpid, _, $category, _, _), $category = "outerwear"

Recall: U (XML -> XML)

```
1. construct
2.   <results> {
3.        where <supplier>
4.                  <company>$company</company>
5.                  <product>
6.                            <name>$name</name>
7.                            <retail>$retail</retail>
8.                            <sale>$sale</sale>
9.                  </product>
10.            </supplier> in "http://acme.com/products.xml",
11.            $sale < 0.5 * $retail
12.       construct
13.            <result ID=Result($company)>
14.                 <supplier>$company</supplier>
15.                 <name>$name</name>
16.            </result>
17.  } </results>
```

Pattern (P) [lines 3–10]

Filter (W) [line 11]

Template (T) [lines 13–16]

Step1: Pattern Matching

- Evaluate U on the view tree
  - *U*=construct **<elm>** { where *P*,*W* construct *T* } **</elm>**
  - Add new variables to P

```
<supplier ID=$t1>
  <company ID=$t2>$company</company>
     <product ID=$t3>
           <name ID=$t4>$name</name>
           <retail ID=$t5>$retail</retail>
           <sale ID=$t6>$sale</sale>
     </product>
</supplier>
```

Step1: Pattern Matching (Table R)

- Evaluate U on the view tree

Variables in U's pattern P

$t1	$t2	$company	$t3	…
Supp()	Comp()	Acme Clothing	Prod($cpid)	…

One matching of U's pattern P with V's template T

Step2: Query Rewriting

- Use the table R to construct the composed query C
- *C* = construct { **<elm>** { B_1 } … { B_k } **</elm>** }
- For each block {B_i}
  - Variable substitutions
  - Construct rule Q(S_0(x), S_0(y), …) :- …
  - Minimize Q
  - Convert Q to *from* and *where* clauses
  - Replace column variables by tuple variables

Discussion

- Recursive XML schema
  - Linear recursion of SQL sufficient?
- Composition technique
  - Evalutes the patterns on the view definition at **compile-time**
  - Filters and constructors are evaluated at **run-time**
- General techniques for RXL to efficient SQL
- Minimization of composed RXL views
- RXL or XML-QL ó Comprehension

Recall: U (XML -> XML)

```
construct
  <results> {
      where <supplier>
              <company>$company</company>
              <product>
                  <name>$name</name>
                  <retail>$retail</retail>
                  <sale>$sale</sale>
              </product>
          </supplier> in "http://acme.com/products.xml",
          $sale < 0.5 * $retail
      construct
          <result ID=Result($company)>
              <supplier>$company</supplier>
              <name>$name</name>
          </result>
  } </results>
```

Pattern (P)

Discussion

- RXL or XML-QL ó Comprehension

```
{ Result( Supplier($company), Name($name), Retail($retail), Sale($sale))
   | <supplier>
           <company>$company</company>
           <product>
                 <name>$name</name>
                 <retail>$retail</retail>
                 <sale>$sale</sale>
           </product>
     </supplier> <--
Load("http://acme.com/products.xml"),
        $sale < 0.5 * $retail
}
```