Translating Web Data

Ronald Fagin, Mauricio A. Hernandez, Lucian Popa,
IBM Almaden Research Center

Renee. J. Miller, Yannis Velegrakis (speaker)
University of Toronto
Our Motivation

- We address the problem of data translation between schemas.
  - This is an old but recurrent problem (see the old Express project from IBM – 70’s)
  - People usually write complex queries to transform data
    - Time consuming
    - Requires query experts
    - Even more when the data and schemas are XML

- Our approach emphasizes *automation* of the task of data translation:
  - Given two schemas (XML and/or relational), and a high-level specification of a mapping between schemas, we generate queries (XQuery/XSLT/SQL)
  - The user does not have to know XQuery/XSLT/SQL

- Major challenges that we address here:
  - Reason about schemas and mapping to *infer* the “right” queries
  - Guarantee that the translated data will *comply* with the target schema
Schema Mapping & Data Translation

Our approach can be applied in both target materialization and query unfolding
Mapping is not Schema Integration

**Design Problem: integrated schema**
- Designed to match sources
- Has no semantics of its own
Applications

- Data Migration
- Schema Evolution
- Data Exchange on the Web (P2P applications)
- Software engineering
Challenges in Schema Mapping

- Goal: interoperability between independent data sources
- Support Nested Structures
  - Nested Relational Model
  - Nested Constraints
- Element correspondences
  - Human friendly
  - Automatic discovery
- Capture User’s Intentions
- Preserve data meaning
  - Discover associations
  - Use constraints & schema
- Create New Target Values
- Produce Correct Grouping
- and …
<?xml version="1.0" encoding="UTF-8"?>
<statisticsDB>
  <cityStatistics>
    <city/>
    distinct {
      FOR
      $x0 IN $doc/expenseDB/grant,
      $x1 IN $doc/expenseDB/company
      WHERE
      $x1/cid/text() = $x0/cid/text()
      RETURN
      <organization>
        $x0/cid/text()  
        <oname> $x1/cname/text()  </oname>,
        distinct {
        FOR
        $x0L1 IN $doc/expenseDB/grant,
        $x1L1 IN $doc/expenseDB/company
        WHERE
        $x1L1/cid/text() = $x0L1/cid/text() AND
        $x1/cname/text() = $x1L1/cname/text() AND
        $x0/cid/text() = $x0L1/cid/text()
        RETURN
        <funding>
        <fid> "Sk35(" $x0L1/amount/text(), ", ", $x1L1/cname/text(), ", ", $x0L1/cid/text(), ")" </fid>,
        <proj> "Sk36(" $x0L1/amount/text(), ", ", $x1L1/cname/text(), ", ", $x0L1/cid/text(), ")" </proj>,
        <aid> "Sk32(" $x0L1/amount/text(), ", ", $x1L1/cname/text(), ", ", $x0L1/cid/text(), ")" </aid>
        </funding>
      </organization> },
    }
    distinct {
    FOR
    $x0 IN $doc/expenseDB/grant,
    $x1 IN $doc/expenseDB/company
    WHERE
    $x1/cid/text() = $x0/cid/text()
    RETURN
    <financial>
    <aid> "Sk32(" $x0/amount/text(), ", ", $x1/cname/text(), ", ", $x0/cid/text(), ")" </aid>,
    <amount> $x0/amount/text() </amount>
    </financial> }
  </cityStatistics>
</statisticsDB>
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/">
    <result>
      <xsl:call-template name="q0"/>
    </result>
  </xsl:template>
  <xsl:template name="q0">
    <xsl:element name="statisticsDB">
      <xsl:attribute name="isRoot">true</xsl:attribute>
      <xsl:element name="ClioSet">
        <xsl:attribute name="id">Sk_statisticsDB()</xsl:attribute>
      </xsl:element>
    </xsl:element>
    <xsl:for-each select="/expenseDB/grant">
      <xsl:variable name="x0" select="."/>
      <xsl:for-each select="/expenseDB/company">
        <xsl:variable name="x1" select="."/>
        <xsl:if test="$x1/cid=$x0/cid">
          <xsl:element name="cityStatistics">
            <xsl:attribute name="inSet">Sk_statisticsDB()</xsl:attribute>
            <xsl:element name="city"/>
            <xsl:element name="ClioSet">
              <xsl:attribute name="id">Sk_statisticsDB_0_1(Sk_statisticsDB())</xsl:attribute>
            </xsl:element>
            <xsl:element name="ClioSet">
              <xsl:attribute name="id">Sk_statisticsDB_0_2(Sk_statisticsDB())</xsl:attribute>
            </xsl:element>
          </xsl:element>
          <xsl:element name="organization">
            <xsl:attribute name="inSet">Sk_statisticsDB_0_1(Sk_statisticsDB())</xsl:attribute>
            <xsl:element name="orgid"><xsl:value-of select="$x0/cid"/></xsl:element>
            <xsl:element name="oname"><xsl:value-of select="$x1/cname"/></xsl:element>
            <xsl:element name="ClioSet">
              <xsl:attribute name="id">Sk_statisticsDB_0_1_0_2($x0/cid, $x1/cname, Sk_statisticsDB_0_1(Sk_statisticsDB())</xsl:attribute>
            </xsl:element>
          </xsl:element>
        </xsl:if>
      </xsl:for-each>
    </xsl:for-each>
    <xsl:element name="funding">
      <xsl:attribute name="inSet">Sk_statisticsDB_0_1_0_2($x0/cid, $x1/cname, Sk_statisticsDB_0_1(Sk_statisticsDB())</xsl:attribute>
      <xsl:element name="fid">
        Sk35($x0/amount, $x1/cname, $x0/cid)
      </xsl:element>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
**Mapping Algorithm**

- **Step 1.** Intra-schema associations discovery
- **Step 2.** Logical mapping generation
- **Step 3.** Query generation

Query (source):
- unnest
- join

Query (target):
- nest
- split
- create new values
**Association Discovery**

- **Step 1.** Discover intra-schema associations between schema elements
  - *relational views* that contain maximal groups of related elements
  - Each represents a different category of data that may exist in the database
Associations

- Groups of elements that are semantically associated
- *Chase* with intra-schema constraints
Logical Mapping Generation

Step 2. Logical mapping generation:
- each source association → each target association
  - based on all correspondences that are relevant
  - By construction, logical mappings preserve associations between elements
Logical Mappings

- Inter-schema constraints

\[ M_1 \]  
\[ M_2 \] 

\[ \Pi_{\text{name}} \text{expenseDB.companies} \subseteq \Pi_{\text{name}} \text{statDB.cityStat.orgs} \]
\[ \Pi_{\text{name}, \text{gid}, \text{amount}} \text{expenseDB.grants} \subseteq \Pi_{\text{name}, \text{gid}, \text{amount}} \text{statDB} \left[ \text{cityStat.orgs.org.fundings} \right] \text{cityStat.financials} \]
Query Generation

- **Step 3.** Generate queries. Each query:
  - Performs the nest and split operation into multiple target elements
  - May need to create new values (unmapped) in the target
Query Generation Issues

- Translation of the logical mappings into queries
  - flat representation of how schemas correspond
  - not all target attributes are determined by the source
  - we need to materialize the nested target

- **Skolemization** algorithm: the heart of query generation
  - Achieves a good nesting (grouping)
  - Generates new values (ids)

- Skolem functions control the creation of the unknown elements:
  - atomic values (this enforces the integrity of the target), and
  - sets (this controls how we group elements in the target)

- Skolem functions are automatically generated.
Skolemization Algorithm
Features

- Produce all the semantically meaningful queries.
  - Finds **all** the associations that exist in the schemas
  - Each one maps from a source association to a target association
  - Allows user to select a subset of them

- Target schema constraints are taken into consideration
  - To infer the user intention
  - To guarantee that the generated data satisfies the structure and the constraints of the target schema

- Target Instance is guaranteed to be in Partition Normal Form (PNF)
Grants may be associated with companies in multiple ways.

Association 1: grants ? companies join on cid = cid

Association 2: grants ? companies join on sponsor = cid

We do not make the one flavor assumption (URA)
Mapping Algorithm Implementation

● Clio System
  ❖ IBM Almaden Research Center / University of Toronto
    ✓ http://www.cs.toronto.edu/db/clio
  ❖ Advanced GUI
    ✓ Constraints
    ✓ Mappings
    ✓ Generated queries

● Experiments with real-world schemas

<table>
<thead>
<tr>
<th>Schema</th>
<th>Nesting Depth</th>
<th>Attributes</th>
<th>Constraints</th>
<th>Queries w/o constraints</th>
<th>Queries w/ constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP</td>
<td>1 / 4</td>
<td>52 / 12</td>
<td>0 / 1</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>TPC-H</td>
<td>1 / 3</td>
<td>34 / 10</td>
<td>9 / 1</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>GeneX</td>
<td>1 / 3</td>
<td>65 / 63</td>
<td>9 / 3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Mondial</td>
<td>1 / 4</td>
<td>102 / 90</td>
<td>15 / 21</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Amalgam</td>
<td>1 / 1</td>
<td>53 / 101</td>
<td>26 / 14</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
Conclusion

- **Schema mapping framework**
  - For data with schemas
  - Covers relational/XML schemas with nested constraints
  - Output: XQuery, XSLT, SQL
  - Build “data transformations” (Queries with Skolem functions)

- **Future extensions**
  - Semantics of ordering in doing mapping
  - Key constraints
  - Recursion of more than 1 level