Clio: Schema Mapping & Data Translation

University of Toronto

- Renée J. Miller, Yannis Velegrakis
- Charlotte Vilarem, Xin Wang

IBM Almaden

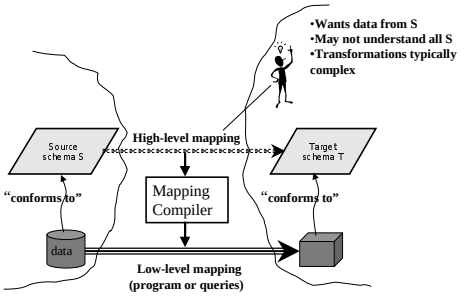
- Ron Fagin, Laura Haas
- Mauricio Hernández, Howard Ho
- Phokion Kolaitis, Felix Naumann
- Lucian Popa

Introduction

- Schema Mapping
  - Facilitate inter-operability between independent data sources
- This often requires data translation
- Challenges
  - Schemas can be arbitrarily different
  - Still, data shouldn't lose its meaning during translation
  - Performed manually: complex user queries, programs, etc.
- Goal: How to generate such a translation with a minimal amount of user specification ?

How to get translation right?

Schema Mapping

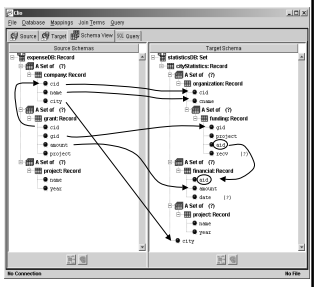


Main Issues

- Language issues:
  - Representation of the high-level mapping:
    - Human friendly
    - Tolerant of **incomplete or ambiguous** user knowledge
  - Representation of the low-level mapping:
    - Queries: XQuery, SQL
  - Representation of schemas:
    - Relational, XMLSchema, DTDs
- Semantic issue:
  - "What did the user mean?"
  - ***Infer*** a precise understanding of the high-level mapping
- Implementation issue:
  - Implement that understanding in the chosen query language

Illustration: Clio Schema Mapping

- Support Nested Structures
- Human Friendly
  - correspondences
- Capture/guide user intention
  - discover & preserve data associations
  - based on constraints & schema
- Create New Target Values
- Produce Correct Grouping
- And …



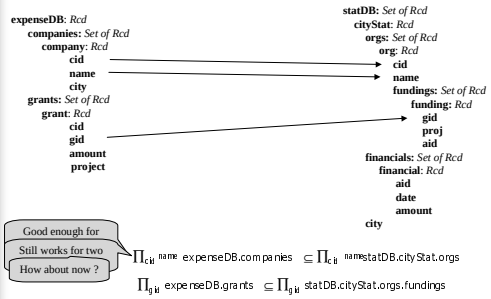
… Generate Queries (XQuery)

```
<statisticsDB>
{   FOR $x0 IN /expenseDB/grant, $x1 IN /expenseDB/project, $x2 IN /expenseDB/company
    WHERE
      $x2/cid/text() = $x0/cid/text()
      $x0/project/text() = $x1/name/text()
    RETURN
    <cityStatistics>
    {   FOR $x0L1 IN /expenseDB/grant, $x1L1 IN /expenseDB/project,  $x2L1 IN /expenseDB/company
        WHERE
          $x2L1/cid/text() = $x0L1/cid/text()
          $x0L1/project/text() = $x1L1/name/text()
          $x2/city/text() = $x2L1/city/text()
        RETURN
          <organization>
            <cid> { $x0L1/cid/text() } </cid>
            <cname>  { $x2L1/name/text() } </cname>
            {
              FOR  $x0L2 IN /expenseDB/grant, $x1L2 IN /expenseDB/project, $x2L2 IN /expenseDB/company
              WHERE
                 $x2L2/cid/text() = $x0L2/cid/text()
                 $x0L2/project/text() = $x1L2/name/text()
                 $x2L1/name/text() = $x2L2/name/text()
                 $x2L1/city/text() = $x2L2/city/text()
                 $x0L1/cid/text() = $x0L2/cid/text()
              RETURN
                 <funding>
                    ……
```

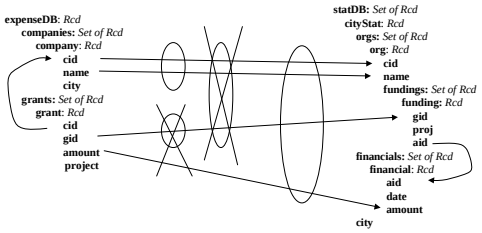
Outline

- Schema Mapping
  - Interpret high-level mapping
    - Relies on semantics of schemas
  - Generate logical (low-level) mapping
- Understand Mapping
  - Explain mappings, resolve ambiguities *using data*
- Data Translation
  - Query generation
    - Retrieve from source (join/unnest)
    - Insert into target (partition/nest)
  - Accommodate differences in data content
    - Skolemization: used to create missing data

Interpreting Arrows

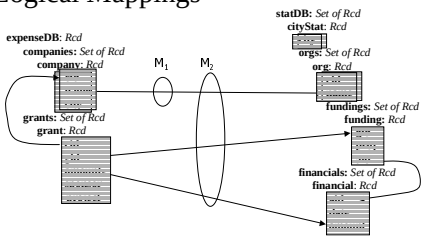


Data Associations



- We must recognize that grants are *associated* to companies
- Associations (in the source): companies, grants • companies
- Associations (in the target): **…,** statDB • orgs • fundings • financials

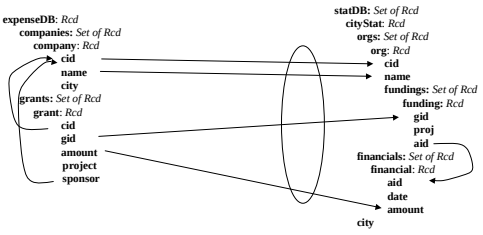
Logical Mappings



Π_{name} expenseDB.companies $\subseteq$ Π_{name} statDB.cityStat.orgs

$\Pi_{name, gid, amount}$ expenseDB.grants • expenseDB.companies $\subseteq$ $\Pi_{name, gid, amount}$ statDB [cityStat.orgs.org.fundings • cityStat.financials]

Multiple Data Associations

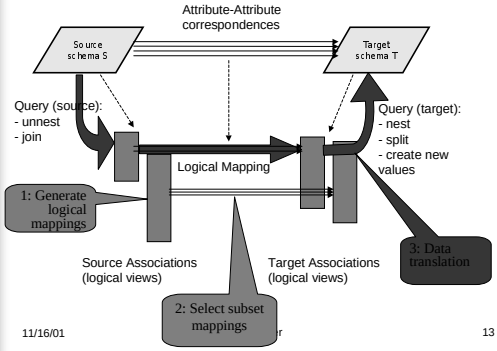


- Grants may be *associated* with companies in multiple ways
- Association 1: grants • companies join on **cid = cid**
- Association 2: grants • companies join on **sponsor = cid**
- We do not make the *one flavor assumption (URA)*

Logical Mapping

- Generate associations:
  - Modified dependency chase algorithm
  - Nested schemas (& nested referential constraints)
  - Algorithm is complete: generate all associations (in the sense of [MUV84]) for bounded deps
- Interpret arrows:
  - Over pair of a source & target association
- Logical mapping:
  - Set of (alternative) inter-schema dependencies

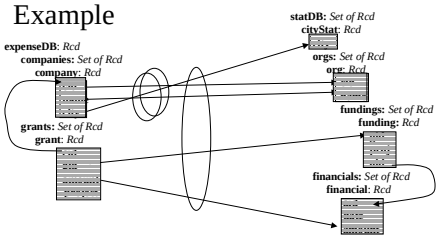
Mapping Algorithm



Outline

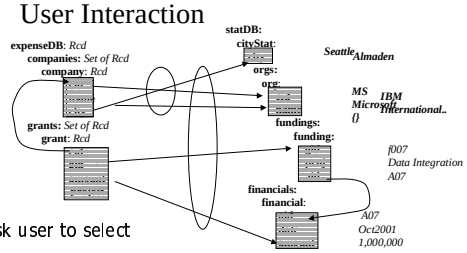
- Schema Mapping
  - Interpret high-level mapping
    - Relies on semantics of schemas
  - Generate logical (low-level) mapping
- Understand Mapping
  - Explain mappings, resolve ambiguities *using data*
- Data Translation
  - Query generation
    - Retrieve from source (join/unnest)
    - Insert into target (partition/nest)
  - Accommodate differences in data content
    - Skolemization: used to create missing data

Example



- Two alternative logical mappings
  - First includes companies without grants
  - Second includes companies with one or more grants
- User selects desired semantics (neither, one or both)

User Interaction

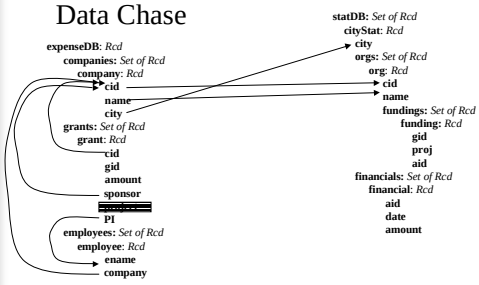


- Use data to show effect of each alternative
  - show data in target form
  - user can select or deselect examples

Data-Driven Schema Mapping

- Data Examples
  - Illustrate differences in alternative mappings
  - Use to interactively extend mappings
  - Provide set operations on data examples
- Example operation: Data Chase

Data Chase



- User provides data value for target attribute
  - Funding.proj = 'Data Mining'
- Provide illustration for each different logical mapping

Outline

- Schema Mapping
  - Interpret high-level mapping
    - Relies on semantics of schemas
  - Generate logical (low-level) mapping
- Understand Mapping
  - Explain mappings, resolve ambiguities *using data*
- Data Translation
  - Query generation
    - Retrieve from source (join/unnest)
    - Insert into target (partition/nest)
  - Accommodate differences in data content
    - Skolemization: used to create missing data

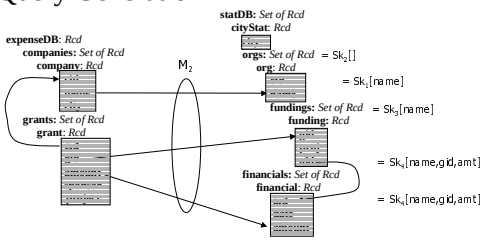
From Dependencies to Queries

- Inter-schema dependencies are a very convenient intermediate representation (in Clio) of mapping:
  - can be reasoned about (query optimization)
  - can even be shown (graphically) to a user
  - but not complete :
    - not all target attributes are determined by source attributes
    - flat representation of correspondence (over logical mapping)
    - **we need to materialize a nested target**

Query Generation Issues

- We translate logical mappings into queries
- Two parts, conceptually:
  - Unnest (join) the source association
  - Nest (partition) according to the shape of the target association
- Skolemization algorithm: the heart of query generation
  - Achieves a good nesting (grouping)
  - Generates new values (ids)
  - Not to many ids, but not too few either !

Query Generation

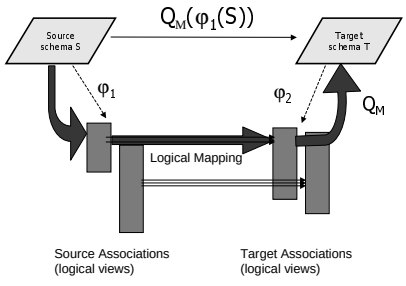


- Correspondences map only into some of the *atomic* attributes
- We use *Skolem functions* to control the creation of the other elements:
  - sets (this controls how we group elements in the target), and
  - atomic values (this enforces the integrity of the target)

Correctness Guarantees

- Information preservation:
  - For every logical mapping $M: \varphi_1 \subseteq \varphi_2$ we generate a query Q_M such that:
    - For any source instance S: $\varphi_2 (Q_M(\varphi_1(S))) = \varphi_1(S)$
  - Thus, we map no less and *no more* than the data that was intended to be mapped
  - In fact, φ_2 and Q_M are inverse to each other (view inversion)
    - lossless, *bidirectional*, data transmission
- Target integrity constraints satisfied!!

Mapping Algorithm



$\varphi_2 (Q_M(\varphi_1(S))) = \varphi_1(S)$

A Formal Semantics of the Mapping

- Logical mapping is a set of s-t dependencies Σ_{st}
- We can reformulate the schema mapping problem as:
  Given:
  source schema S with NRIs Σ_s,
  target schema T with NRIs Σ_t,
  Source-to-target dependencies Σ_{st},
  Source instance I_s
  Output:
  Target instance I_t such that: 1) $<I_s, I_t> \models \Sigma_{st}$, and 2) $I_t \models \Sigma_t$
- A consequence of the mapping algorithm:
  Theorem. Σ_{st} implies Σ_t

Hence, $Q_M(\varphi_1(I_s)) \models \Sigma_t$

When Can This Tool Help?

- In general, whenever:
  - schemas are complex (nesting + constraints) and very different,
  - it matters whether we lose data or not,
  - it is not obvious to a human user that there are many forms of data that can coexist within a schema
- Specific applications:
  - data migration from one schema to another (schema redesign)
  - data transmission (between independently created schemas)
  - data integration and warehousing

Project Status: So Far

- Can map between any combination of XML schemas/DTDs/Relational schemas
- The generated queries are XQuery + Skolem fcts.
- Tested Clio with some complex LifeScience schemas and integration benchmark
- Demos SIGMOD01 (Relational), ICDE02 (XML)
- Papers: VLDB00, SIGMOD01

No commercial tool correctly generates XML transformations that are lossless !

Conclusion

- A schema mapping framework with some good foundations
  - Covers relational and core of XML Schema
  - *"maps source associations to target associations"*
  - We build *data transformations* (XQuery with Skolem functions) rather than just queries
- Separated into 2 phases:
  - Logical mapping (arrows -> dependencies)
  - Data translation (dependencies -> queries)
  - It works (see demo)

Limitations and Future Work

- Only a core of XML Schema is covered
  - No union types, no recursive types
  - No ordering
- More future work:
  - Data integration issue:
    - We guarantee good properties when evaluating a *single* query
    - When taking the *union* of several queries generated, the result can have redundancies (not in PNF anymore)
    - Need *minimal (deep) union*, to combine nested sets with oids
  - How to invert the mapping, in general ?

Answering Queries Using Views

- Answering queries using views
  - Semantics: maximally contained rewriting ([DG97])
    - Query Q on schema T, set views V on T
    - Find Q' on V: Q' ⊑ Q and Q' is maximal
- Schema mapping
  - Semantics: maximally contained mapping
    - Schema T, set views V (associations) on T
    - A query Q is simply *select* * on any relation of T
    - Find Q_M on V: $Q_M \sqsubseteq Q$ and Q_M is maximal
  - Difference: we must ensure target instance satisfies target constraints