Using ontologies for P2P data sharing and ETL design

Dimitrios Skoutas

(joint work with Alkis Simitsis, Verena Kantere, and Timos Sellis)

Dept. of Electrical and Computer Engineering
National Technical University of Athens, Hellas
dskoutas@dblab.ece.ntua.gr
http://www.dblab.ece.ntua.gr

Bertinoro Workshop on Information Integration (INFINT 2007)
Bertinoro, Italy
Outline

- Data sharing in PDMS
- Ontology-based ETL design
Peer Data Management Systems

- **P2P networks**
  - massive sharing of structured or unstructured data

- **Structured vs. unstructured overlays**
  - **Structured**
    - data is indexed and stored at peers with pre-specified characteristics, e.g. peer id
    - peers are connected to specific peers
  - **Unstructured**
    - no pre-specified connection of peers and storage/indexing of data
Problem description

- An unstructured PDMS
  - local relational database
  - pairwise mappings of the forms GAV/LAV/GLAV
  - SPJ queries
    - expressed on the local schema
    - reformulated according to the mappings

- An OWL ontology describing the application domain
Problem description (cont’d)

Semantic peer \( P = (R, O, A) \)
- \( R \): local database schema
- \( O \): the domain ontology
- \( A \): semantic annotation

Two issues
- Semantic similarity between peer schemas
- Semantic similarity between reformulated queries
Sample ontology and peer schemas

P₁ : bands ( name, members, year )

P₂ : bands ( name, singer, year )

Mₚ₁,ₚ₂ : bands ( name, members, year ) :- bands ( name, singer, year )
Semantic annotations

$P_1 : \text{bands having at least one album}$

$P_2 : \text{bands that play Jazz music, were formed before the year 2000, and have released at least 3 albums}$

$\text{Band}_{P_1} \equiv \text{Band} \pi \geq 1 \text{released}$

$\text{Band}_{P_2} \equiv \text{Band} \pi \forall \text{type.Jazz} \pi \geq 3 \text{released}$

$\pi \forall \text{year.}(\leq 2000)$
Comparison of peer schemas

- Express the degree of relevance between the interests of peers (subset, superset, overlap, etc.)
- Semantic similarity is assessed by comparing the classes annotating the peer schemas
- The similarity function needs to be asymmetric
Recall and Precision

Use notions **recall** and **precision** from Information Retrieval

$$\text{recall} = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{relevant}|}$$

$$\text{precision} = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{retrieved}|}$$

$$F_\alpha = \frac{(1+\alpha) \cdot \text{precision} \cdot \text{recall}}{\alpha \cdot \text{precision} + \text{recall}}$$

TP: True Positive
FP: False Positive
FN: False Negative
Adapting recall and precision

We adapt the notions of recall and precision as follows:

\[
\text{recall}(C_1, C_2) = \frac{|\{x \mid x \in (C_1 \cap C_2)\}|}{|\{x \mid x \in C_1\}|}
\]

\[
\text{precision}(C_1, C_2) = \frac{|\{x \mid x \in (C_1 \cap C_2)\}|}{|\{x \mid x \in C_2\}|}
\]

<table>
<thead>
<tr>
<th>Relationship</th>
<th>equivalent</th>
<th>subset</th>
<th>superset</th>
<th>overlap</th>
<th>disjoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>1</td>
<td>1</td>
<td>$&lt;1$</td>
<td>$&lt;1$</td>
<td>0</td>
</tr>
<tr>
<td>Precision</td>
<td>1</td>
<td>$&lt;1$</td>
<td>1</td>
<td>$&lt;1$</td>
<td>0</td>
</tr>
</tbody>
</table>
Semantic similarity

Similarity is assessed by the semantic information provided by the ontology

- class hierarchy
- properties
  - property hierarchy
  - value and cardinality restrictions
Calculating recall and precision

Based on the class hierarchy

- ratio of common ancestors
  - similarity increases with the hierarchy depth

\[
\text{recall}_C(C_1, C_2) = \frac{|A(C_1) \cap A(C_2)|}{|A(C_2)|}
\]

\[
\text{precision}_C(C_1, C_2) = \frac{|A(C_1) \cap A(C_2)|}{|A(C_1)|}
\]
Calculating recall and precision (cont’d)

Based on the properties of classes

- property hierarchy

\[
\text{recall}(p_1, p_2) = \frac{|A(p_1) \cap A(p_2)|}{|A(p_2)|}
\]

\[
\text{precision}(p_1, p_2) = \frac{|A(p_1) \cap A(p_2)|}{|A(p_1)|}
\]
Calculating recall and precision (cont’d)

Based on the properties of classes (cont’d)

- property restrictions
  - value restrictions on object properties: $\forall$ type.Jazz
  - value restrictions on datatype properties: $\forall$ year.$(\leq 2000)$
  - cardinality restrictions: $\geq 3$ released

<table>
<thead>
<tr>
<th>case</th>
<th>recall($R_1, R_2$)</th>
<th>precision($R_1, R_2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1 \equiv R_2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$R_1 \subseteq R_2$</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>$R_1 \supseteq R_2$</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>$R_1 \cap R_2 \neq \emptyset$</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$R_1 \cap R_2 = \emptyset$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Calculating recall and precision (cont’d)

Based on the properties of classes (cont’d)

- combining property hierarchy and restrictions

\[
\text{recall}(p_1, p_2) = \frac{|A(p_1) \cap A(p_2)|}{|A(p_2)|} \cdot \prod_{R(p_2)} \text{recall}(R_i'(p_1), R_i(p_2))
\]

\[
\text{precision}(p_1, p_2) = \frac{|A(p_1) \cap A(p_2)|}{|A(p_1)|} \cdot \prod_{R(p_1)} \text{precision}(R_i(p_1), R_i'(p_2))
\]
Calculating recall and precision (cont’d)

Based on the properties of classes (cont’d)

\[
\text{recall}(C_1, C_2) = \frac{\sum_{P(C_2)} \text{recall}(p'(C_1), p(C_2))}{|P(C_2)|}
\]

\[
\text{precision}(C_1, C_2) = \frac{\sum_{P(C_1)} \text{precision}(p(C_1), p'(C_2))}{|P(C_1)|}
\]
Calculating recall and precision (cont’d)

Extending to sets of classes

\[
\begin{align*}
\text{recall}(C_1, C_2) &= \frac{\sum_{C_i \in C_1} \text{recall}(C_i, C'_i)}{|C_1|} \\
\text{precision}(C_1, C_2) &= \frac{\sum_{C_i \in C_2} \text{precision}(C'_i, C_i)}{|C_2|}
\end{align*}
\]
A query $Q_o$ is forwarded from peer $P_i$ to peer $P_j$, and is reformulated as $Q_r$

- some attributes may not be rewritten
- some attributes may be rewritten approximately
- some conditions may be lost
- some conditions may be inserted
Extending the similarity measure

Rewritten attributes

- t was rewritten to t´
  - recall( t , t´ ) = recall( p_t , p_t´ )
  - precision( t , t´ ) = precision( p_t , p_t´ )

- t was not rewritten
  - recall = 0
  - precision is not affected
Extending the similarity measure (cont’d)

Rewritten conditions

1. A query $Q_o$ is issued at peer $P_i$
2. $C_{Q_o} \leftarrow$ the set of classes annotating the relations in $Q_o$
3. $C_{Q_{o,e}} \leftarrow$ the classes in $C_{Q_o}$ enhanced with additional value restrictions, according to the conditions specified in $Q_o$
4. $Q_o$ is forwarded to peer $P_j$ and is rewritten as $Q_r$
5. $C_{Q_{r,e}} \leftarrow$ apply steps 2 and 3 for $Q_r$
Example

**Q_0:** SELECT name, members, year FROM bands
WHERE year ≥ 1980 AND year < 1990

**C_{Q_0,e}:** Band\_P_1 \equiv \text{Band } \pi \geq 1 \text{ released } \pi \forall \text{ year.}([1980,1990))

**Q_r:** SELECT name, singer, year FROM bands
WHERE year ≥ 1980 AND year < 1990

**C_{Q_r,e}:** Band\_P_2 \equiv \text{Band } \pi \forall \text{ type.Jazz } \pi \geq 3 \text{ released } \pi \forall \text{ year.}([1980,1990))
Similarity measure for rewritten queries

- Combine results for attributes and conditions

\[
\text{recall}(C_{Q_o,e}, C_{Q_r,e}, Q_o, Q_r) = \frac{\sum_{t \in S(Q_o)} \text{recall}(p_t, p_t')}{|S(Q_o)|} \cdot \text{recall}(C_{Q_o,e}, C_{Q_r,e})
\]

\[
\text{precision}(C_{Q_o,e}, C_{Q_r,e}, Q_o, Q_r) = \frac{\sum_{t \in S(Q_r)} \text{precision}(p_t, p_t')}{|S(Q_r)|} \cdot \text{precision}(C_{Q_o,e}, C_{Q_r,e})
\]
Outline

- Data sharing in PDMS
- Ontology-based ETL design
Extract-Transform-Load (ETL)
Problem description

The problem of heterogeneity in data sources

- structural heterogeneity
  - data stored under different schemata

- semantic heterogeneity
  - different naming conventions
    - e.g., homonyms, synonyms
  - different representation formats
    - e.g., units of measurement, currencies, encodings
  - different ranges of values

Two main goals

- specify inter-schema mappings
- identify appropriate transformations
Overview

- **Construct a suitable application ontology**
- **Annotate the datastores**
  - establish mappings between the datastore schemas and the ontology
- **Apply reasoning techniques to**
  - select relevant sources
  - to identify required transformations
## Reference example

<table>
<thead>
<tr>
<th>Source schema</th>
<th>Target schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>emp</em></td>
<td><em>employee</em></td>
</tr>
<tr>
<td><em>fname</em></td>
<td><em>name</em></td>
</tr>
<tr>
<td><em>lname</em></td>
<td><em>project</em></td>
</tr>
<tr>
<td><em>prj</em></td>
<td><em>salary</em></td>
</tr>
<tr>
<td><em>salary</em></td>
<td></td>
</tr>
<tr>
<td><em>address</em></td>
<td><em>city</em></td>
</tr>
<tr>
<td></td>
<td><em>street</em></td>
</tr>
</tbody>
</table>

- at least one
- at least two
- EUR
- USD, above basic
The application ontology

A suitable application ontology is constructed to model

- the concepts of the domain
- the relationships between those concepts
- the attributes characterizing each concept
- the different representation formats or (ranges of) values for each attribute

A graph representation specified for the ontology

- graph nodes → classes
- graph edges → properties
- visual notation
  - different symbols used for each type of class or property
## Ontology graph notation

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Type</th>
<th>Represents</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept-node</td>
<td>a class $c \in C_C$</td>
<td></td>
<td>$C$</td>
</tr>
<tr>
<td>aggregation-node</td>
<td>a class $c \in C_G$</td>
<td></td>
<td>$C$</td>
</tr>
<tr>
<td>type-node</td>
<td>a class $c \in C_{TP}$</td>
<td></td>
<td>$C$</td>
</tr>
<tr>
<td>format-node</td>
<td>a class $c \in C_{TF}$</td>
<td></td>
<td>$C$</td>
</tr>
<tr>
<td>range-node</td>
<td>a class $c \in C_{TR}$</td>
<td></td>
<td>$C$</td>
</tr>
<tr>
<td>aggregated-node</td>
<td>a class $c \in C_{TG}$</td>
<td></td>
<td>$C$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edges</th>
<th>Type</th>
<th>Represents</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>property-edge</td>
<td>a property $p \in P_p$</td>
<td></td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>convertsTo-edge</td>
<td>property convertsTo</td>
<td></td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>aggregates-edge</td>
<td>property aggregates</td>
<td></td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>groups-edge</td>
<td>property groups</td>
<td></td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>subclass-edge</td>
<td>class hierarchy</td>
<td></td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>disjoint-edge</td>
<td>disjointness of classes</td>
<td></td>
<td>$\leftrightarrow$</td>
</tr>
</tbody>
</table>
Reference example (cont’d)

The application ontology graph
Datastore Annotation

- Semantic annotation \(\rightarrow\) correspondences between the datastore schema and the ontology

- Relations are mapped to concept-nodes

- Attributes are mapped to nodes of the following types:
  - type-node
  - format-node
  - range-node
  - aggregated-node

- Defined classes are created to express the semantics of the schema elements
Reference example (cont’d)

- Datastore mappings

<table>
<thead>
<tr>
<th>Source schema</th>
<th>Target schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp</td>
<td>employee</td>
</tr>
<tr>
<td>{1,1} fname</td>
<td>{1,1} name</td>
</tr>
<tr>
<td>{1,1} lname</td>
<td>{2,-} project</td>
</tr>
<tr>
<td>{1,-} prj</td>
<td>{1,1} salary</td>
</tr>
<tr>
<td>{1,1} salary</td>
<td>{1,1} city</td>
</tr>
<tr>
<td>{1,1} address</td>
<td>{1,1} street</td>
</tr>
</tbody>
</table>
Datastore definitions

- \( S.\text{Emp} \equiv \text{Employee} \sqcap \forall \text{hasName.Name} \sqcap =1\text{hasName} \sqcap \forall \text{worksAt.Project} \sqcap \geq 1\text{worksAt} \sqcap \forall \text{receives.EUR} \sqcap \geq 1\text{receives} \sqcap \forall \text{lives.Address} \sqcap =1\text{lives} \)

- \( T.\text{Employee} \equiv \text{Employee} \sqcap \forall \text{hasName.Name} \sqcap =1\text{hasName} \sqcap \forall \text{worksAt.Project} \sqcap \geq 2\text{worksAt} \sqcap \forall \text{receives.AboveBasic} \sqcap =1\text{receives} \sqcap \forall \text{lives.Address} \sqcap =1\text{lives} \)
ETL Transformations

- **Generic types of ETL transformations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETRIEVE(n)</td>
<td>Retrieves recordsets from the underlying provider node n</td>
</tr>
<tr>
<td>EXTRACT(c)</td>
<td>Extracts, from incoming recordsets, the part denoted by c</td>
</tr>
<tr>
<td>MERGE</td>
<td>Merges recordsets from two or more provider nodes</td>
</tr>
<tr>
<td>FILTER(c)</td>
<td>Filters incoming recordsets, allowing only records with values of the template type specified by c</td>
</tr>
<tr>
<td>CONVERT(c₁, c₂)</td>
<td>Converts incoming recordsets from the template type denoted by c₁ to the template type denoted by c₂</td>
</tr>
<tr>
<td>AGGREGATE(fₐ, g₁, ..., gₙ)</td>
<td>Aggregates incoming recordsets over the attributes g₁, ..., gₙ applying the aggregate function denoted by fₐ.</td>
</tr>
<tr>
<td>MINCARD(p, min)</td>
<td>Filters out incoming recordsets having cardinality less than min on property p</td>
</tr>
<tr>
<td>MAXCARD(p, max)</td>
<td>Filters out incoming recordsets having cardinality more than max on property p</td>
</tr>
<tr>
<td>UNION</td>
<td>Unites recordsets from two or more sources</td>
</tr>
<tr>
<td>JOIN</td>
<td>Joins incoming recordsets</td>
</tr>
<tr>
<td>STORE</td>
<td>Stores incoming recordsets to a target datastore</td>
</tr>
</tbody>
</table>
Generating ETL transformations

Two main steps

- select relevant sources to populate a target element
- identify required data transformations

Based on the use of a reasoner to infer subsumption relationships between the defined classes representing the datastores
Generating ETL transformations

Selecting relevant sources

- a source node $n_s$, mapped to class $c_s$
- a target node $n_T$, mapped to class $c_T$
- $n_s$ is provider for $n_T$, if
  - $c_s$ and $c_T$ have a common superclass
  - ensures that the integrated data records refer to the same kind of entity in the domain
  - $c_s$ and $c_T$ are not disjoint
  - prevents integration between datastores with conflicting constraints
Generating ETL transformations

Identifying data transformations (I)

- a RETRIEVE operation for each provider node
- a MERGE operation to combine data from several provider nodes
- an EXTRACT operation to extract a portion of data from a provider node
Generating ETL transformations

Identifying data transformations (II)

- if $C_S \equiv C_T$ or $C_S \sqsubseteq C_T$, no transformations are required
- if $C_T \sqsubseteq C_S$, AGGREGATE, FILTER and/or MINCARD/MAXCARD operations are required
  - choice of specific operation(s) is based on parsing the class definitions and analyzing the value and cardinality restrictions found
- else, as previous plus CONVERT operations
Generating ETL transformations

Identifying data transformations (III)

- a JOIN operation to combine recordsets from nodes, whose corresponding classes are related by a property
- a UNION operation to combine recordsets from nodes, whose corresponding classes have a common superclass
- a STORE operation to denote loading of data to the target datastore
Reference example (cont’d)

- Datastore mappings

<table>
<thead>
<tr>
<th>Source schema</th>
<th>Target schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp</td>
<td>employee</td>
</tr>
<tr>
<td>{1,1} fname</td>
<td>{1,1} name</td>
</tr>
<tr>
<td>{1,1} lname</td>
<td>{2,-} project</td>
</tr>
<tr>
<td>{1,-} prj</td>
<td>{1,1} salary</td>
</tr>
<tr>
<td>{1,1} salary</td>
<td>{1,1} city</td>
</tr>
<tr>
<td>{1,1} address</td>
<td>{1,1} street</td>
</tr>
</tbody>
</table>
Reference example (cont’d)

<table>
<thead>
<tr>
<th>Source elements</th>
<th>Target elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp</td>
<td>employee</td>
</tr>
<tr>
<td>fname</td>
<td>name</td>
</tr>
<tr>
<td>lname</td>
<td>project</td>
</tr>
<tr>
<td>prj</td>
<td>salary</td>
</tr>
<tr>
<td>salary</td>
<td>city</td>
</tr>
<tr>
<td>address</td>
<td>street</td>
</tr>
</tbody>
</table>

Diagram:
- **MERGE**
- **MINCARD(2)**
- **CONVERT(EUR,USD)**
- **FILTER(AboveBasic)**
- **EXTRACT**
Current and Future Work

**PDMS**
- route queries in the network
- apply the approach to social network applications

**ETL**
- NLG techniques for textual representation of datastore descriptions and ETL operations
- impact of schema evolution

**Semantic Web services**
- service ranking
- QoS aspects
Thank You