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

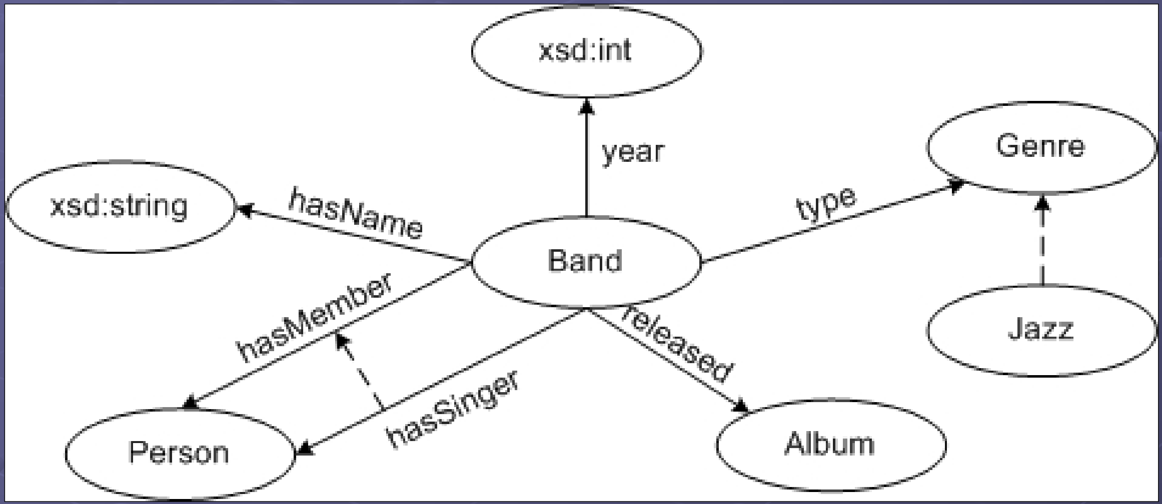
Problem description

- 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 $\mathrm{P} = (\mathrm{R}, \mathrm{O}, \mathrm{A})$
  - 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



- P_1 : bands (name, members, year)
- P_2 : bands (name, singer, year)
- M_{P_1, P_2} : bands (name, members, year) :- bands (name, singer, year)

Semantic annotations

- P_1 : bands having at least one album
- P_2 : bands that play Jazz music, were formed before the year 2000, and have released at least 3 albums

- $Band\_P_1 \equiv Band \sqcap \geq_1 released$
- $Band\_P_2 \equiv Band \sqcap \forall type.Jazz \sqcap \geq_3 released$

  $\sqcap \forall year.(\leq 2000)$

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

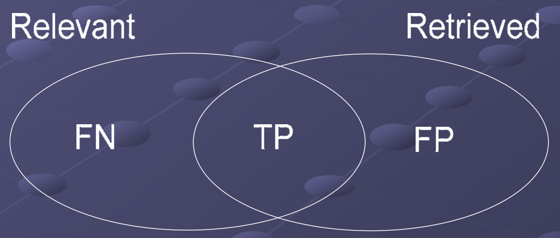
Recall and Precision

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

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

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

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



TP: True Positive

FP: False Positive

FN: False Negative

Adapting recall and precision

- We adapt the notions of recall and precision as follows:

$$recall(C_1,\ C_2) = \frac{\left|\{x \mid x \in (C_1 \sqcap C_2)\}\right|}{\left|\{x \mid x \in C_1\}\right|}$$

$$precision(C_1,\ C_2) = \frac{\left|\{x \mid x \in (C_1 \sqcap C_2)\}\right|}{\left|\{x \mid x \in C_2\}\right|}$$

Relationship	*equivalent*	*subset*	*superset*	*overlap*	*disjoint*
Recall	1	1	<1	<1	0
Precision	1	<1	1	<1	0

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

$$recall_C(C_1,\ C_2)\ =\ \frac{|\,A(C_1)\ \cap\ A(C_2)\,|}{|\,A(C_2)\,|}$$

$$precision_C(C_1,\ C_2)\ =\ \frac{|\,A(C_1)\ \cap\ A(C_2)\,|}{|\,A(C_1)\,|}$$

Calculating recall and precision (cont'd)

- Based on the properties of classes
  - property hierarchy

$$recall(p_1, p_2) = \frac{|A(p_1) \cap A(p_2)|}{|A(p_2)|}$$

$$precision(p_1, p_2) = \frac{|A(p_1) \cap A(p_2)|}{|A(p_1)|}$$

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

$case$	$recall(R_1, R_2)$	$precision(R_1, R_2)$
$R_1 \equiv R_2$	1	1
$R_1 \subseteq R_2$	1	0.5
$R_1 \supseteq R_2$	0.5	1
$R_1 \cap R_2 \neq \varnothing$	0.5	0.5
$R_1 \cap R_2 = \varnothing$	0	0

Calculating recall and precision (cont'd)

- Based on the properties of classes (cont'd)
  - combining property hierarchy and restrictions

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

$$precision(p_1, p_2) = \frac{|A(p_1) \cap A(p_2)|}{|A(p_1)|} \cdot \prod_{R(p_1)} precision(R_i(p_1), R_i'(p_2))$$

Calculating recall and precision (cont'd)

- Based on the properties of classes (cont'd)

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

$$precision(C_1,\ C_2) = \frac{\sum_{P(C_1)} precision(p(C_1), p'(C_2))}{|P(C_1)|}$$

Calculating recall and precision (cont'd)

- Extending to sets of classes

$$recall(C_1, C_2) = \frac{\sum_{C_i \in C_1} recall(C_i, C_i')}{|C_1|}$$

$$precision(C_1, C_2) = \frac{\sum_{C_i \in C_2} precision(C_i', C_i)}{|C_2|}$$

Query reformulation

- 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} ← the set of classes annotating the relations in Q_o
  3. $C_{Q_{o,e}}$ ← 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. $C_{Q_{r,e}}$ ← apply steps 2 and 3 for Q_r

Example

- Q_o: SELECT name, members, year FROM bands WHERE year ≥ 1980 AND year < 1990
- $C_{Q_{o,e}}$: $Band\_P_1 \equiv Band \sqcap \geq_1 released \sqcap \forall\, year.([1980, 1990))$
- Q_r: SELECT name, singer, year FROM bands WHERE year ≥ 1980 AND year < 1990
- $C_{Q_{r,e}}$: $Band\_P_2 \equiv Band \sqcap \forall\, type.Jazz \sqcap \geq_3 released \sqcap \forall\, year.([1980, 1990))$

Similarity measure for rewritten queries

- Combine results for attributes and conditions

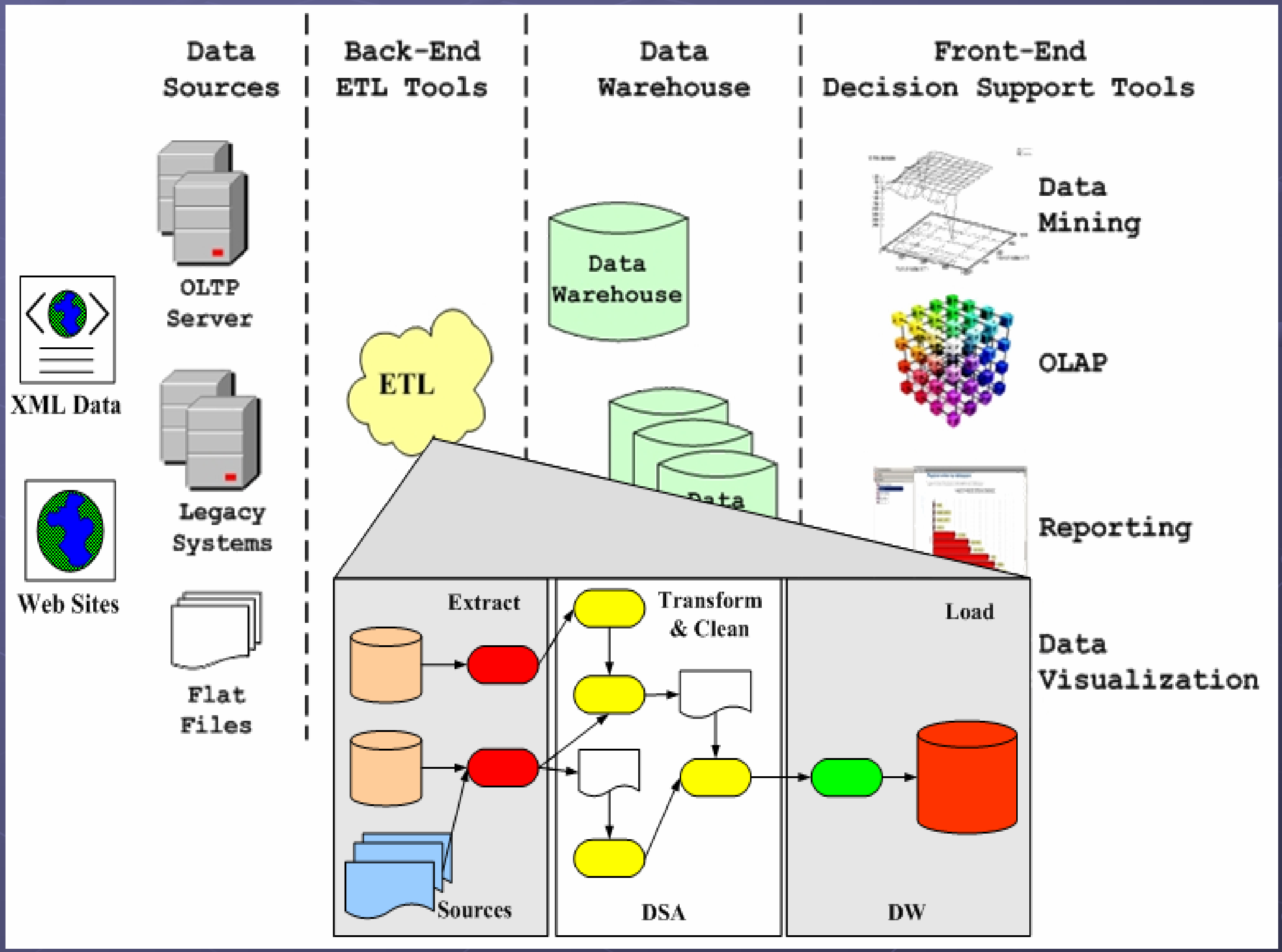
$$recall(C_{Q_{o,e}}, C_{Q_{r,e}}, Q_o, Q_r) = \frac{\sum_{t \in S(Q_o)} recall(p_t, p_{t'})}{|S(Q_o)|} \cdot recall(C_{Q_{o,e}}, C_{Q_{r,e}})$$

$$precision(C_{Q_{o,e}}, C_{Q_{r,e}}, Q_o, Q_r) = \frac{\sum_{t \in S(Q_r)} precision(p_{t'}, p_t)}{|S(Q_r)|} \cdot precision(C_{Q_{o,e}}, C_{Q_{r,e}})$$

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

Source schema	Target schema
emp	**employee**
fname	**name**
lname	**project**
prj	**salary**
salary	**city**
address	**street**

at least one

EUR

at least two

USD, above basic

The application ontology

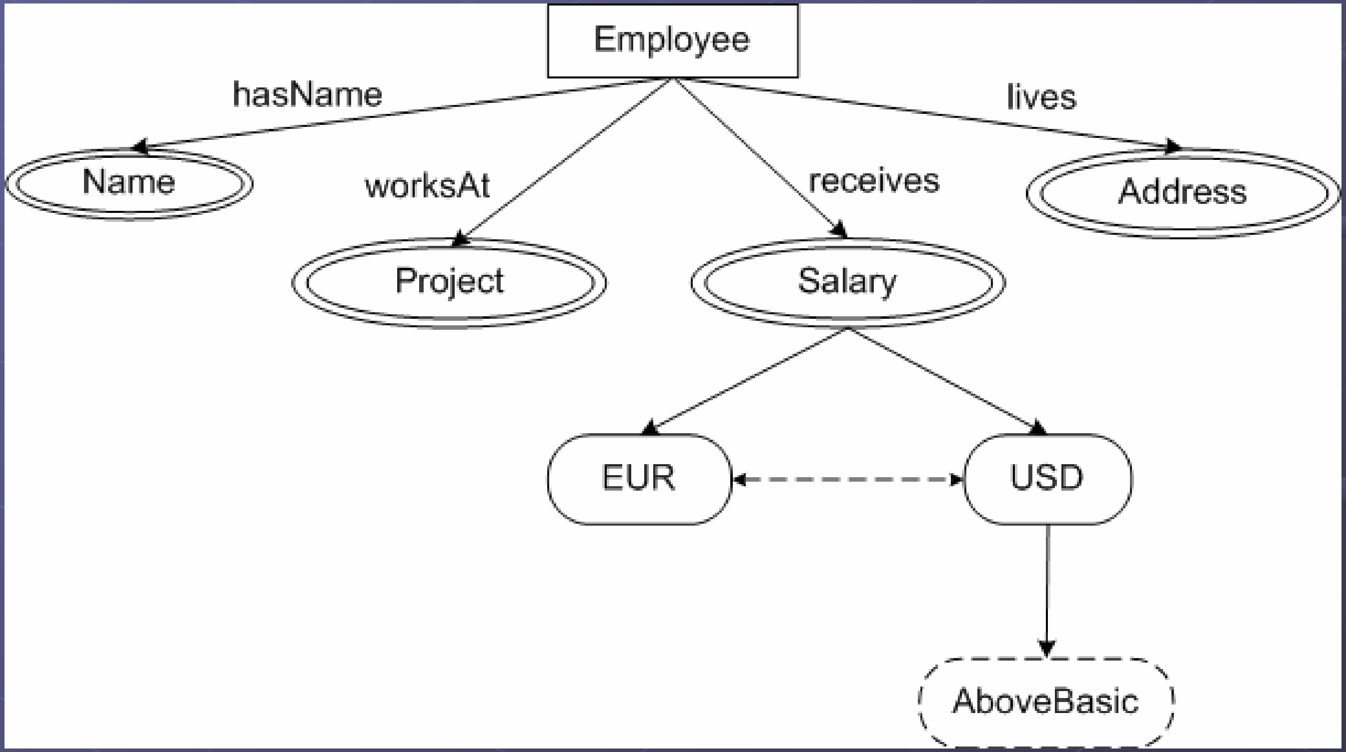
- A suitable application ontology is constructed to model
  - the concepts of the domain
  - the relationships between those concepts
  - the attributes characterizing each concept
  - the different representation formats or (ranges of) values for each attribute
- A graph representation specified for the ontology
  - graph nodes → classes
  - graph edges → properties
  - visual notation
    - different symbols used for each type of class or property

Ontology graph notation

	Type	Represents	Symbol
Nodes	concept-node	a class $c \in C_C$	C
	aggregation-node	a class $c \in C_G$	C
	type-node	a class $c \in C_{TP}$	C
	format-node	a class $c \in C_{TF}$	C
	range-node	a class $c \in C_{TR}$	C
	aggregated-node	a class $c \in C_{TG}$	C
Edges	property-edge	a property $p \in P_P$	– p →
	convertsTo-edge	property `convertsTo`	- - →
	aggregates-edge	property `aggregates`	●→
	groups-edge	property groups	○→
	subclass-edge	class hierarchy	→
	disjoint-edge	disjointness of classes	←X→

Reference example (cont'd)

- The application ontology graph

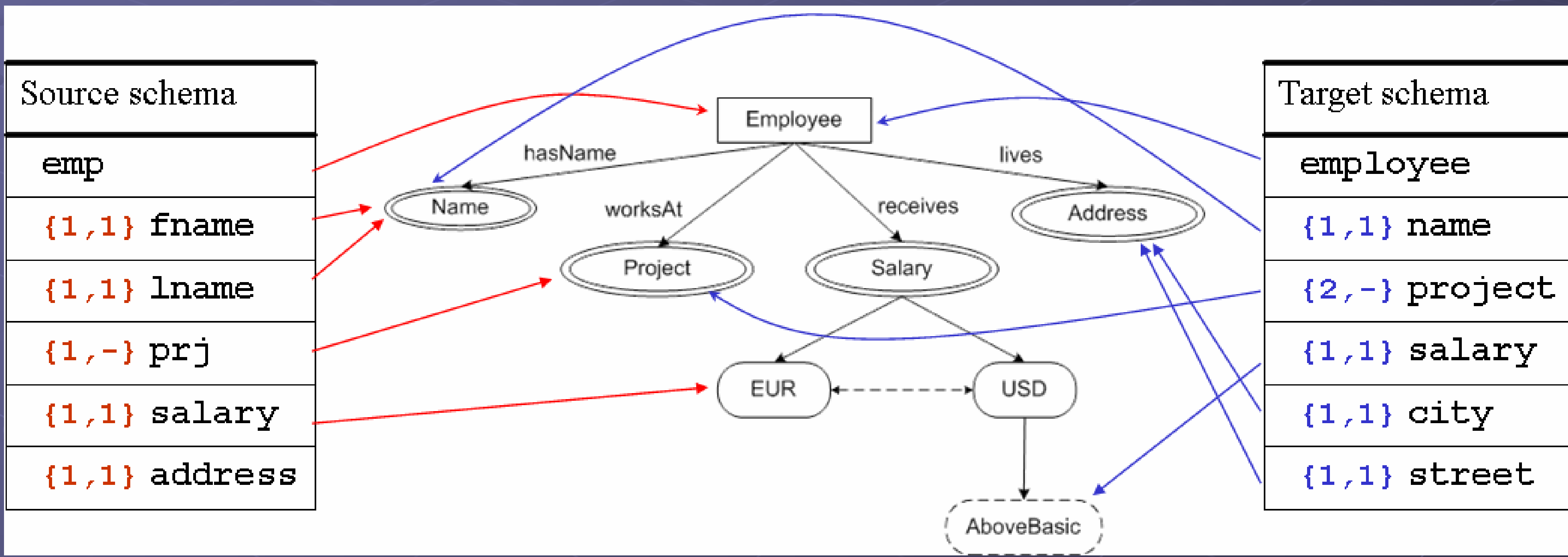


Datastore Annotation

- Semantic annotation → 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



Reference example (cont'd)

- Datastore definitions
  - $\mathtt{S.Emp} \equiv \mathtt{Employee} \sqcap \forall \mathtt{hasName.Name} \sqcap =_1\mathtt{hasName} \sqcap \forall \mathtt{worksAt.Project} \sqcap \geq_1\mathtt{worksAt} \sqcap \forall \mathtt{receives.EUR} \sqcap =_1\mathtt{receives} \sqcap \forall \mathtt{lives.Address} \sqcap =_1\mathtt{lives}$
  - $\mathtt{T.Employee} \equiv \mathtt{Employee} \sqcap \forall \mathtt{hasName.Name} \sqcap =_1\mathtt{hasName} \sqcap \forall \mathtt{worksAt.Project} \sqcap \geq_2\mathtt{worksAt} \sqcap \forall \mathtt{receives.AboveBasic} \sqcap =_1\mathtt{receives} \sqcap \forall \mathtt{lives.Address} \sqcap =_1\mathtt{lives}$

ETL Transformations

- Generic types of ETL transformations

Operation	Description
RETRIEVE(n)	Retrieves recordsets from the underlying provider node n
EXTRACT(c)	Extracts, from incoming recordsets, the part denoted by c
MERGE	Merges recordsets from two or more provider nodes
FILTER(c)	Filters incoming recordsets, allowing only records with values of the template type specified by c
CONVERT(c_1, c_2)	Converts incoming recordsets from the template type denoted by c_1 to the template type denoted by c_2
AGGREGATE(f_g, g_1, …, g_n)	Aggregates incoming recordsets over the attributes g_1, …, g_n applying the aggregate function denoted by f_g.
MINCARD(p,min)	Filters out incoming recordsets having cardinality less than min on property p
MAXCARD(p,max)	Filters out incoming recordsets having cardinality more than max on property p
UNION	Unites recordsets from two or more sources
JOIN	Joins incoming recordsets
STORE	Stores incoming recordsets to a target datastore

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

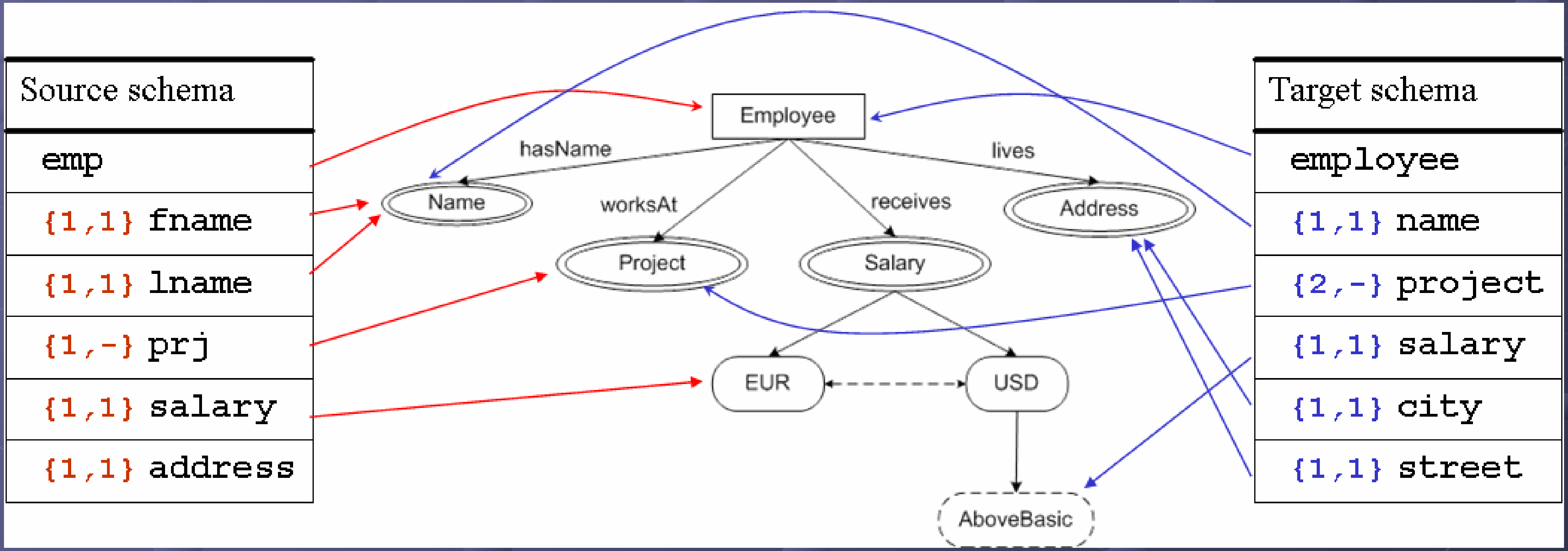
- Identifying data transformations (II)
  - if $C_S \equiv C_T$ or $C_S \sqsubset C_T$, no transformations are required
  - if $C_T \sqsubset C_S$, AGGREGATE, FILTER and/or MINCARD/MAXCARD operations are required
    - choice of specific operation(s) is based on parsing the class definitions and analyzing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

Generating ETL transformations

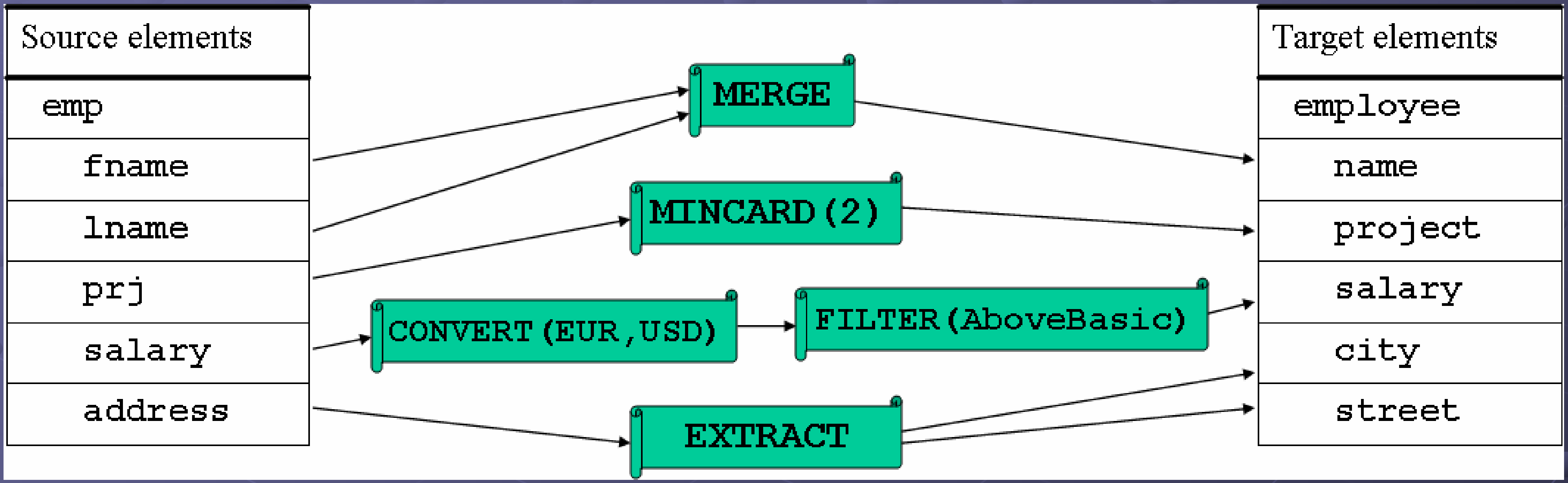
- Identifying data transformations (III)
  - a JOIN operation to combine recordsets from nodes, whose corresponding classes are related by a property
  - a UNION operation to combine recordsets from nodes, whose corresponding classes have a common superclass
  - a STORE operation to denote loading of data to the target datastore

Reference example (cont'd)

- Datastore mappings



Reference example (cont'd)



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