Upgrading Databases to Ontologies

Gisella Bennardo, Giovanni Grasso, Nicola Leone,
Francesco Ricca

University of Calabria, Italy

ALPSWS 2008

Outline

1. Motivation and Contribution
2. OntoDLV
3. Virtual Classes and Relations
4. Data Integration Features
5. Conclusion and Ongoing work

Ontologies and Enterprises

- **Ontology**:
  - Formal representation of a conceptualization [Gruber]
  - Roughly, an abstract formal model of a complex domain
  - Recognized to be a fundamental tool for KRR
- **The strong need of knowledge-based technologies is perceived by industries today**
  - Ontologies start to be exploited
- **Enterprise ontologies**
  - Terms and definitions relevant to business enterprises
  - Clean conceptual view of the enterprise knowledge
  - Improve sharing and manipulation
  - Simplify information retrieval and knowledge discovery

Ontologies and Enterprises

- **Ontology**:
  - Formal representation of a conceptualization [Gruber]
  - Roughly, an abstract formal model of a complex domain
  - Recognized to be a fundamental tool for KRR
- **The strong need of knowledge-based technologies is perceived by industries today**
  - Ontologies start to be exploited
- **Enterprise ontologies**
  - Terms and definitions relevant to business enterprises
  - Clean conceptual view of the enterprise knowledge
  - Improve sharing and manipulation
  - Simplify information retrieval and knowledge discovery

Motivation

- **Enterprise ontologies are not widely used, why?**
- **Two major obstacles:**
  1. Specification of real-world ontologies is an hard task
  2. Often relevant information stored in relational DB
- Indeed, developing by scratch would be time consuming and expensive
  - Knowledge engineers + domain experts
- Ontology must incorporate large amount of data from Enterprise Information Systems
  - mainly regarding instances
  - avoid import: exploit fresh data + legacy system support
  - data from several autonomous systems

    → well known inconsistency problems

    [argw-etal-95,lenz-02,bert-etal-05]

Motivation

- **Enterprise ontologies are not widely used, why?**
- **Two major obstacles:**
  1. Specification of real-world ontologies is an hard task
  2. Often relevant information stored in relational DB
- Indeed, developing by scratch would be time consuming and expensive
  - Knowledge engineers + domain experts
- Ontology must incorporate large amount of data from Enterprise Information Systems
  - mainly regarding instances
  - avoid import: exploit fresh data + legacy system support
  - data from several autonomous systems
    → well known inconsistency problems
    [argw-etal-95,lenz-02,bert-etal-05]

“Lifting” databases to ontologies (1)

- **Combine an ontology representation language, with Large (already existent) databases**
    - *High Expressive power + deal with large amount of data*
- **Deal with inconsistency:**
    - *Data Integration techniques*
- Analyze the schema and recognize entities and relationships
    - Create an ontology specification
    - Obtain a clean view of the enterprise knowledge
- Exploit database data for specifying concept instances
    - Data should be kept at the sources
    - Legacy systems might still work on it
    - Take only *consistent information*

“Lifting” databases to ontologies (2)

- **OntoDLP**
  - Ontology representation language
  - **rule based:** Disjunctive Logic Programming - ASP

\+ **Virtual classes and virtual relations**
  - Link data about instances to the ontology
  - Seamless combination of ontologies and DB [lenz-02] **(GAV approach)**
  - Data are kept to the original sources!

\+ **Consistent Query Answering (CQA)** [lenz-02,bert-etal-05,chom-marcin-05]
  - By rewriting queries in DLP
  - Minimal Change Integrity Maintenance [chom-marcin-05]

OntoDLP [ricca-etal-08]

- **OntoDLP** = DLP +
  - Ontology specification constructs
    - Classes, (Multiple) Inheritance, Relations, ...
    - Data-Types (integer, string, date ...)
  - Consistency control features
    - Strong typing, user defined axioms
  - Rules
    - Support DLP with many linguistic enhancements
      - > Lists and Sets, Aggregate and Plug-in functions, Complex Terms, Named notation
    - Modular Programming: Reasoning modules

Example: Reasoning on Ontology

Example

```
class employee(name:string, salary:int).
class project(numeEmp:int, bud:int, numSk:int, maxSal:int).

module (team_building) {
 inTeam(E,P) v outTeam(E,P) :– E:employee(), P:project().
 :– P:project(numEmp:N), not #countE: inTeam(emp:E)=N.
 :– P:project(numSk:S), not #count{Sk: E:employee(sk:Sk), inTeam(E,P)}≥S.
 :– P:project(budt:B), not #sum{Sa,E: E:employee(sal:Sa), inTeam(E ,P)}≤B.
 :– P:project(maxSal:M),
                          not #max{Sa: E:employee(sal:Sa), inTeam(E ,P)}≤M.
}
X:person(age:18, father:employee(skill:"Java Programmer")), inTeam(X,_)?
```

OntoDLV Main Features

- **Advanced Platform for Ontology Management**
  - Specification, Browsing, Querying, Reasoning
  - Based on OntoDLP
    - Ontology + Disjunctive Logic Programming - ASP
    - High computational power
      → Solve complex problems in a fully declarative way
  - Built on DLV the state-of-the-art DLP System [leone-etal-06]
  - Application Programming Interface (API)
  - OWL Interoperability

- **Able to deal with data-intensive applications**
  - Persistency on DBMS
  - **exploits** $\mathbf{DLV}^{DB}$ (DLV working on mass memory)

Virtual Class and Virtual Relation

- **Virtual Class and Virtual Relations**
  - Usual schema specification
  - Instances are specified by means of mapping rules
    - exploits Sourced Atoms (logical notation)
    - Exploit SQL Atoms (SQL notation)

- *Sources are specified directly in OntoDLP*
  - built-in class dbSource
  - several databases and ...any other kind of sources

Example

```
class dbSource(uri:string, user:string, psw:string).

db1:dbSource(uri:"http : //mydb.mysite.com:3306", user:"me",psw:"myPsw").
```

Virtual Class and Virtual Relation

- **Virtual Class and Virtual Relations**
  - Usual schema specification
  - Instances are specified by means of mapping rules
    - exploits Sourced Atoms (logical notation)
    - Exploit SQL Atoms (SQL notation)

- *Sources are specified directly in OntoDLP*
  - built-in class dbSource
  - several databases and ...any other kind of sources

Example

class dbSource(uri:string, user:string, psw:string).

db1:dbSource(uri:"http : //mydb.mysite.com:3306", user:"me",psw:"myPsw").

Virtual Class Specification

Example

virtual class branch(name : string, city : string, assets : integer)
{
  f(BN) : branch(BN, BC, A) :– branch@db1(branch-name : BN,
                                                  branch-city : BC, assets :A).
}

- Sourced Atoms
    - Attribute types must match the table schema
    - Attributes can be filled in by constants or variables
- Functional Object Identifiers (impedance mismatch)
    - Values vs instances
      → exploit function symbols
    - Each virtual class should use a fresh function symbol
      → distinct oids for distinct classes

Virtual Class with multiple sources

Example

```
virtual class branch(name : string, city : string, assets : integer )
{
  f(BN) : branch(BN, BC, A) :-  branch@db1(branch-name : BN,
                                            branch-city : BC, assets :A).

  f(BN) : branch(BN, BC, A) :- localBranch@db2(bName : BN,
                                            bCity : BC, aS : A, group:_).
}
```

- Multiple sources
    - Just write several "mapping" rules
    - Select the information you need

SQL Notation

Example

```
virtual class branch(name : string, city : string, assets : integer )
{

    f(BN):branch(BN, BC, A) :– [db1, "SELECT branch-name AS BN,
                                         branch-city AS BC, assets AS A
                                    FROM branch "].


}
```

Virtual Entities in OntoDLV

- **Off-line Mode**
  - Extract data from DBMS
  - Store instances in the Persistency Manager
  - Useful for migrating the database
- **On-line Mode**
  - Keep information in the original database
  - Queries are performed directly at the sources
  - Unfolding (query predicates are substituted with the corresponding query at the sources)
- **Evaluation in mass memory**
  - exploit DLV^{DB}
  - *restricted to stratified and non disjunctive programs*

Data Integration Features

- **Virtual Classes and Virtual Relations**
  - instances are virtually populated
  - rules act as a mapping
  - **in presence of multiple source databases**

    → typical Data Integration scenario

    → *Global As View (GAV)* [lenz-02,bert-etal-05]
- Inconsistency Problems
  - Integrity constraint may be violated
    1. Repair manually

       → *Consistency Checking*
    2. Single out as much consistent information as possible

       → *Consistent Query Answering (CQA)*

Data Integration Features

- **Virtual Classes and Virtual Relations**
  - instances are virtually populated
  - rules act as a mapping
  - **in presence of multiple source databases**

    → typical Data Integration scenario

    → *Global As View (GAV)* [lenz-02,bert-etal-05]
- Inconsistency Problems
  - Integrity constraint may be violated
    1. Repair manually

       → *Consistency Checking*
    2. Single out as much consistent information as possible

       → *Consistent Query Answering (CQA)*

Consistent Query Answering

- **Minimal Change Integrity Maintenance** [chom-marc-05]
  - *Complete sources assumption*
    - Common in Data Warehousing
    - Closed World Assumption
  - Integrity restoration by *tuple deletion*
    - Constraints: Arbitrary denial, inclusion dependencies
    - Decidable setting: Π_2^P in the general case [chom-marc-05]
      → implemented by rewriting in DLP

Definition

Given a schema Σ and a set A of integrity constraints, let $\mathcal{O}$ and $\mathcal{O}^r$ be two ontology instances, $\mathcal{O}^r$ is a *repair* [chom-marc-05] of $\mathcal{O}$ w.r.t. A, if

- $\mathcal{O}^r$ satisfies all the constrains in A; and
- the instances in $\mathcal{O}^r$ are a maximal subset of the instances in $\mathcal{O}$.

Given a query Q, the consistent answers to Q are those tuples that are true in *every repair*.

Consistent Query Answering

- **Minimal Change Integrity Maintenance** [chom-marc-05]
  - *Complete sources assumption*
    - Common in Data Warehousing
    - Closed World Assumption
  - Integrity restoration by *tuple deletion*
    - Constraints: Arbitrary denial, inclusion dependencies
    - Decidable setting: Π_2^P in the general case [chom-marc-05]
      → implemented by rewriting in DLP

Definition

Given a schema Σ and a set A of integrity constraints, let $\mathcal{O}$ and $\mathcal{O}^r$ be two ontology instances, $\mathcal{O}^r$ is a *repair* [chom-marc-05] of $\mathcal{O}$ w.r.t. A, if

- $\mathcal{O}^r$ satisfies all the constrains in A; and
- the instances in $\mathcal{O}^r$ are a maximal subset of the instances in $\mathcal{O}$.

Given a query Q, the consistent answers to Q are those tuples that are true in *every repair*.

CQA by Rewriting

Given $\mathcal{O}$, Q, A build program Π_{cqa} and a query Q_{cqa} s.t. $\Pi_{cqa} \models_c Q_{cqa}$

(Q is consistently true in $\mathcal{O}$ w.r.t. A iff Q_{cqa} is true in every answer set of Π_{cqa})

Run Q_{cqa} on Π_{cqa} in mass memory with DLV^{DB}

Example

Given two relations *m(code)*, and *e(code,name)* and *code(X) :- e(X,_).*

$$\begin{array}{ll} :- e(X,Y), e(X,Z), Y <> Z. & \textit{(denial: code is key)} \\ :- m(X), \text{not } code(X) & \textit{(inclusion } m[code] \subseteq e[code]\textit{)} \end{array}$$

become:

$$\overline{e}(X,Y) \vee \overline{e}(X,Z) :- e(X,Y), e(X,Z), Y <> Z.$$

$$e^r(X,Y) :- e(X,Y), \text{not } \overline{e}(X,Y).$$

$$code^*(X) :- e^r(X,\_).$$

$$m^r(M) :- m(M), \text{not } code^*(M).$$

CQA by Rewriting

Given $\mathcal{O}$, Q, A build program Π_{cqa} and a query Q_{cqa} s.t. $\Pi_{cqa} \models_c Q_{cqa}$

(Q is consistently true in $\mathcal{O}$ w.r.t. A iff Q_{cqa} is true in every answer set of Π_{cqa})

Run Q_{cqa} on Π_{cqa} in mass memory with DLV^{DB}

Example

Given two relations *m(code)*, and *e(code,name)* and *code(X) :- e(X,_).*

$$:\!-\ e(X,Y), e(X,Z), Y <> Z. \qquad \textit{(denial: code is key)}$$
$$:\!-\ m(X), \mathrm{not}\ code(X) \qquad \textit{(inclusion } m[code] \subseteq e[code]\textit{)}$$

become:

$$\overline{e}(X,Y) \vee \overline{e}(X,Z) :\!-\ e(X,Y), e(X,Z), Y <> Z.$$
$$e^r(X,Y) :\!-\ e(X,Y), \mathrm{not}\ \overline{e}(X,Y).$$

$$code^*(X) :\!-\ e^r(X,\_).$$
$$m^r(M) :\!-\ m(M), \mathrm{not}\ code^*(M).$$

Conclusion

- **"Lifting" databases to OntoDLV Ontologies:**
  - Define an ontology, and specify instances by logic rules
    - Ontological view of the enterprise knowledge
    - Powerful rule-based reasoning mechanisms
  - Virtual classes and virtual relations

    → Data is kept at the sources

    → Queries are performed at the source
  - Consistent Query Answering:

    → Deal with inconsistencies
- **Ongoing work:**
  - Different input sources: XML, RDF, ...
  - CQA on user constraints