Information Integration and the Semantic Web

José Luis Ambite
USC/Information Sciences Institute
http://www.isi.edu/info-agents/

“Fundamental Axiom of Modeling”
For each domain, either
■ there is OneTrueModel, or
   (please enlighten me)
■ there is more than one model, then either
   → destroy opponents until the VictoriousModel emerges, or
   → merge models, or
   → map among models

Map versus Merge
■ Mapping preferable when:
   ◆ Size of the ground data/instances/tuples is much larger than the size of the schema
   ◆ Source models are autonomous
■ Merge preferable when:
   ◆ Schema and data of similar sizes
   ◆ Sources willing to adopt merged model

Why Map?
■ Because $G$ needs to reason with $S$
   ◆ A proof in model $G$ can only be completed using reasoning available in model $S$
■ Most often
   ◆ $G$ needs data (ground facts, instances) present in $S$

Information Integration: Mappings
Assume two models:
■ $G$ in language $L_G$ over alphabet $A_G$
■ $S$ in language $L_S$ over alphabet $A_S$ ($A_S \cap A_G = \emptyset$)
A mapping is a logical formula $\Psi(G,S)$
   where $G \subseteq A_G$ and $S \subseteq A_S$
All the action lies on what are the formal languages used to describe $G$, $S$, and $\Psi$:
   ◆ Conjunctive queries
   ◆ Datalog
   ◆ First-order logic
   ◆ XML-Schema and XML Query…
   ◆ Description logics (DAML, OWL)

Information Integration: Global and Source Models
To simplify the problem of integrating multiple models which are not under our control ($S_1 \ldots S_n$):
■ define global/domain/mediated model ($G$)
■ mappings are restricted to $\Psi(G, S_i)$ for each source model $S_i$
Information Integration: Mapping styles: GAV & LAV

Useful form of the mappings:

- **Global-as-view (GAV):**
  - \( g \subseteq \Psi(S) \) i.e. \( g \leftarrow \Psi(S) \) (sound)
  - \( g = \Psi(S) \) i.e. \( g \leftrightarrow \Psi(S) \) (exact)

- **Local-as-View (LAV):**
  - \( s \subseteq \Psi(G) \) i.e. \( s \rightarrow \Psi(G) \) (sound)
  - \( s = \Psi(G) \) i.e. \( s \leftrightarrow \Psi(G) \) (exact)

**GLAV**

Global-as-View (GAV)

Each global relation \((\text{MovieActor}, \text{MovieReview} \in G)\) is defined as a formula (view) over source relations \((DB1, DB2, DB3 \in S)\)

\[
\text{MovieActor}(\text{title}, \text{actor}) \leftarrow \text{DB1}(\text{id}, \text{title}, \text{actor}, \text{year})
\]

\[
\text{MovieActor}(\text{title}, \text{actor}) \leftarrow \text{DB2}(\text{title}, \text{director}, \text{actor}, \text{year})
\]

\[
\text{MovieReview}(\text{title}, \text{review}) \leftarrow \text{DB1}(\text{id}, \text{title}, \text{actor}, \text{year}) \land \text{DB3}(\text{id}, \text{review})
\]

[Language for \( \Psi \): conjunctive queries]

Query Reformulation in GAV

Query reformulation = rule unfolding+simplification

Query: Find reviews for 'DeNiro' movies

\[
q(\text{title}, \text{review}) \leftarrow \text{MovieActor}(\text{title}, 'DeNiro'), \text{MovieReview}(\text{title}, \text{review})
\]

1. \( q'(\text{title}, \text{review}) \leftarrow \text{DB1}(\text{title}, 'DeNiro', \text{year}), \text{DB1}(\text{title}, \text{actor}, \text{year'}), \text{DB3}(\text{title}, \text{review}) \)

   \[
   q'(\text{title}, \text{review}) \leftarrow \text{DB1}(\text{title}, 'DeNiro', \text{year}), \text{DB3}(\text{title}, \text{review})
   \]

   Redundant

Local-as-View (LAV)

Each source relation \((v_1, v_2 \in S)\) is defined as a formula (view) over global relations

\[(\text{Movie}, \text{American}, \text{MovieReview} \in G)\]

\[
V1(\text{title}, \text{year}, \text{director}) \leftarrow \text{Movie}(\text{title}, \text{year}, \text{director}, \text{genre}) \land \text{American}(\text{director}) \land \text{year} \geq 1960 \land \text{genre} = 'Comedy'
\]

\[
V2(\text{title}, \text{review}) \leftarrow \text{Movie}(\text{title}, \text{year}, \text{director}, \text{genre}) \land \text{year} \geq 1990 \land \text{MovieReview}(\text{title}, \text{review})
\]

[Language for \( \Psi \): conjunctive queries with comparison predicates]

Query Reformulation in LAV

Query: Reviews for comedies produced after 1950

\[
q(\text{title}, \text{review}) \leftarrow \text{Movie}(\text{title}, \text{year}, \text{director}, 'Comedy') \land \text{year} \geq 1950 \land \text{MovieReview}(\text{title}, \text{review})
\]

Reformulated query:

\[
q'(\text{title}, \text{review}) \leftarrow V1(\text{title}, \text{year}, \text{director}) \land V2(\text{title}, \text{review})\]

\[
q' \subseteq q
\]

S1: \( V1(\text{title}, \text{year}, \text{director}) \leftarrow \text{Movie}(\text{title}, \text{year}, \text{director}, \text{genre}) \land \text{American}(\text{director}) \land \text{year} \geq 1960 \land \text{genre} = 'Comedy'
\]

S2: \( V2(\text{title}, \text{review}) \leftarrow \text{Movie}(\text{title}, \text{year}, \text{director}, \text{genre}) \land \text{year} \geq 1990 \land \text{MovieReview}(\text{title}, \text{review}) \)
Query Reformulation in LAV

The Bucket Algorithm

Given: user query q, source descriptions \{V_i\}

1. Find relevant sources (fill buckets)
   For each relation g in query q
     • Find \( V_j \) that contains relation g
     • Check that constraints in \( V_j \) are compatible with q

2. Combine \( \{V_j\} \) from each bucket into a conjunctive query q' and check for containment (q' \( \subseteq \) q)

Example

Query: Reviews for comedies produced after 1950

\[ q(\text{title}, \text{review}) \leftarrow \text{Movie}(\text{title}, \text{year}, \text{director}, \text{`Comedy'}) \land \text{year} \geq 1950 \land \text{MovieReview}(\text{title}, \text{review}) \]

Reformulated query:

\[ q'(\text{title}, \text{review}) \leftarrow V_1(\text{title}, \text{year}, \text{director}) \land V_2(\text{title}, \text{review}) \]

\[ S_1: V_1(\text{title}, \text{year}, \text{director}) \rightarrow \text{Movie}(\text{title}, \text{year}, \text{director}, \text{genre}) \land \text{American}(\text{director}) \land \text{year} \geq 1960 \land \text{genre} = \text{`Comedy'} \]

\[ S_2: V_2(\text{title}, \text{review}) \rightarrow \text{Movie}(\text{title}, \text{year}, \text{director}, \text{genre}) \land \text{year} \geq 1990 \land \text{MovieReview}(\text{title}, \text{review}) \]

Query Reformulation in LAV

Inverse-Rules Algorithm

Idea: Construct an equivalent logic program which evaluation yields the answer to the query

¬ The antecedent of the query and views is in terms of global predicates
¬ Would like to have source predicates in antecedent so that program can be evaluated

⇒ Invert the rules
  (simply by using standard logical manipulations)

Example

\[ V_5(\text{dept}, \text{course}) \rightarrow \text{Enrolled}(\text{student}, \text{dept}) \land \text{Registered}(\text{student}, \text{course}) \]

\[ q(D) \leftarrow \text{Enrolled}(\text{S,D}) \land \text{Registered}(\text{S,”DB”}) \]

\[ v5(D,C) \rightarrow \text{Enrolled}(\text{S,D}) \land \text{Registered}(\text{S,”DB”}) \]

\[ q(D) \leftarrow \text{Enrolled}(\text{S,D}) \land \text{Registered}(\text{S,”DB”}) \]

\[ \text{Enrolled}(f(D,C),D) \leftarrow v5(D,C) \]

\[ \text{Registered}(f(D,C),C) \leftarrow v5(D,C) \]

\[ \text{Ext}(v5) = \{ (“CS”, “DB”), (“EE”, “DB”), (“CS”, “AI”)) \]

\[ \text{Ext}(\text{Enrolled}) = \{ f(“CS”, “DB”), “CS”), (f(“EE”, “DB”), “EE”), (f(“CS”, “AI”), “CS”\}) \]

\[ \text{Ext}(\text{Registered}) = \{ (f(“CS”, “DB”)), “DB”), (f(“EE”, “DB”)), “DB”), (f(“CS”, “AI”), “AI”)) \]

\[ \text{Ext}(q) = \{ (“CS”), (“EE”)) \]

The Inverse-Rules Algorithm: Example

\[ q(D) \leftarrow \text{Enrolled}(\text{S,D}) \land \text{Registered}(\text{S,”DB”}) \]

\[ v5(D,C) \rightarrow \text{Enrolled}(\text{S,D}) \land \text{Registered}(\text{S,C}) \]

\[ q(D) \leftarrow \text{Enrolled}(\text{S,D}) \land \text{Registered}(\text{S,”DB”}) \]

\[ \text{Enrolled}(f(D,C),D) \leftarrow v5(D,C) \]

\[ \text{Registered}(f(D,C),C) \leftarrow v5(D,C) \]

\[ \text{Ext}(v5) = \{ (“CS”, “DB”), (“EE”, “DB”), (“CS”, “AI”)\}) \]

\[ \text{Ext}(\text{Enrolled}) = \{ (f(“CS”, “DB”)), “CS”), (f(“EE”, “DB”)), “EE”), (f(“CS”, “AI”), “CS”\}) \]

\[ \text{Ext}(\text{Registered}) = \{ (f(“CS”, “DB”)), “DB”), (f(“EE”, “DB”)), “DB”), (f(“CS”, “AI”), “AI”)) \]

\[ \text{Ext}(q) = \{ (“CS”), (“EE”)) \]

LAV in Description Logics

Terminology: happyParent := person \land (\forall \text{child.smart})

Views (sources):

\[ v1(x) \rightarrow \text{person}(x) \land (\forall \text{child}(x)) \]

\[ v2(x) \rightarrow \text{person}(x) \land \text{child}(x,y) \land \text{smart}(y) \]

\[ v3(x) \rightarrow \text{happyParent}(x) \land (\exists 2 \text{child}(x)) \]

\[ v4(x) \rightarrow (\exists 3 \text{child}(x)) \]

Query: q1(x) \leftarrow \text{happyParent}(x)

Maximally contained rewriting:

\[ q_1'(x) \leftarrow \text{v3}(x) \]

\[ q_1'(x) \leftarrow \text{v1}(x) \land \text{v2}(x) \]

Query: q2(x) \leftarrow \text{happyParent}(x) \land (\exists 3 \text{child}(x))

Equivalent rewriting: q2'(x) \leftarrow \text{v3}(x) \land \text{v4}(x)
GAV Integration with XML Query

- GAV mapping in XML Query:
  FOR $c$ IN db(1)/customers/tuple
  RETURN <customer>
  <name> $c/name </name>
  <due> $c/balance </due>
  <orders>
    FOR $o$ IN db(2)/orders/tuple
    WHERE $c/name = $o/name
    RETURN $o
  </orders>
  </customer>
AS int-customers

- Query:
  FOR $c$ IN source(int-customers)/customer
  RETURN <totaldue> sum($c/due) </totaldue>

GAV vs. LAV

- Not modular: adding new sources changes the mediated schema
- Can be awkward to write mediated schema without loss of information
- Easier query reformulation: reduces to view unfolding
- Best when:
  Few, stable, data sources
  Well-known to the mediator (e.g., corporate integration)
  Garlic, TSIMMIS, HERMES

- Modular: adding new sources is easy
- Very flexible: power of the entire query language available to describe sources
- Reformulation is hard: answering queries using views (from NP to undecidable)
- Best when:
  Many, relatively unknown data sources
  Possibility of addition/deletion of sources
  Information Manifold, InfoMaster, Emerac

Modeling Sources with Negative Capabilities: Binding Patterns

Sources:
- AAAIdb (X) → AAAIPapers(X)
- CitationDB (X,Y) → Cites(X,Y)
- AwardDB (X) → AwardPaper(X)

Query: find all the award winning papers:
q(X) ← AwardPaper(X)

Recursive Rewritings

q(X) ← AwardPaper(X)

- Problem: Unbounded union of conjunctive queries

q1(X) ← AAAIdb(X), AwardDB(X)
q1(X) ← AAAIdb(X1), CitationDB(X1,X), AwardDB(X)
... q1(X) ← AAAIdb(X1), CitationDB(X1,X2), ..., CitationDB(Xn,X), AwardDB(X)

- Solution: Recursive Rewriting
p(X) ← AAAIdb(X)
p(X) ← papers(Y), CitationDB(Y,X)
q'(X) ← papers(X), AwardDB(X)

Inverse-Rules Algorithm

Inverted Rules:
- AAAIPapers(X) ← AAAIdb(X)
- Cites(X,Y) ← dom(X) ^ CitationDB(X,Y)
- AwardPaper(X) ← dom(X) ^ AwardDB(X)

Domain Rules:
- dom(Y) ← dom(X) ^ CitationDB(X,Y)
- dom(X) ← AAAIdb(X)

Query:
q(X) ← AwardPaper(X)
Inverse-Rules Algorithm
Inverse + Domain Rules (2)

Simplyfing the program:

- \( q(X) \leftarrow \text{dom}(X) \land \text{AwardDB}(X) \)
- \( \text{dom}(Y) \leftarrow \text{dom}(X) \land \text{CitationDB}(X,Y) \)
- \( \text{dom}(X) \leftarrow \text{AAAIdb}(X) \)

Application to the Semantic Web

- Schema/Ontology integration
  - Very expressive mappings
    - CQ, Datalog, First-order logic
    - Description Logics \( \rightarrow \) DAML+OIL (OWL)
    - XML Query
  - Types of mappings: LAV, GAV, GLAV
- Composing semantic web services
  - Binding pattern restrictions \( \rightarrow \) Recursion