



Carleton
UNIVERSITY

**Virtual Data Integration:
Query Answering, Consistency and
Specification**

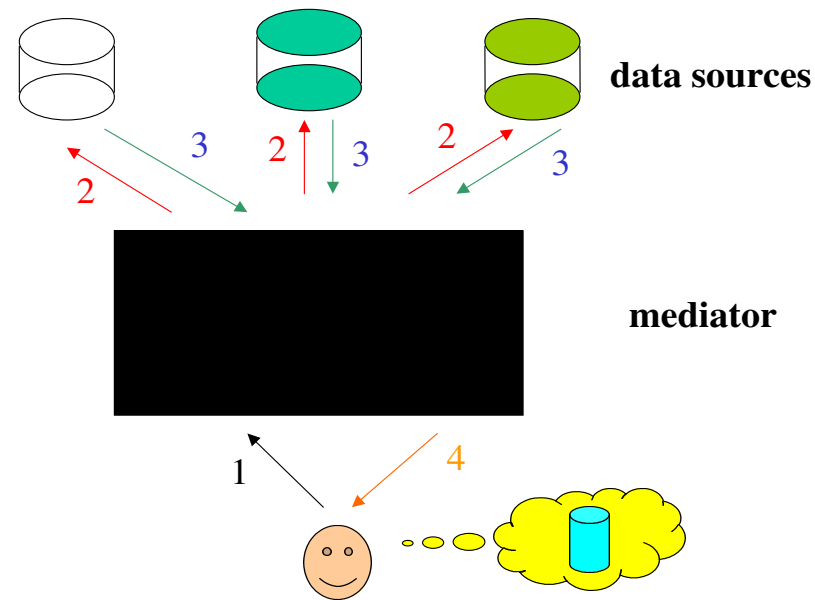
Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

Joint work with:

Loreto Bravo (U. Concepcion, Chile) and Gayathri Jayaraman (CU)

Virtual Data Integration



We consider a **mediated approach** to data integration: data stay at the sources, and a **virtual integration system** is created

Mediator: A software system that offers a **common query interface** to a set of data sources (assume all schemas are relational)

- Data sources are heterogeneous and mutually independent
- Data kept at the local sources

Data extracted at the mediator's request, at query time

Interaction with the mediator is through queries and answers

- Individual data sources are updated independently

Updates on data sources via the mediator not allowed

- System should allow sources to get in and out

Class of participating sources should be flexible and open

- Data sources have their own schema

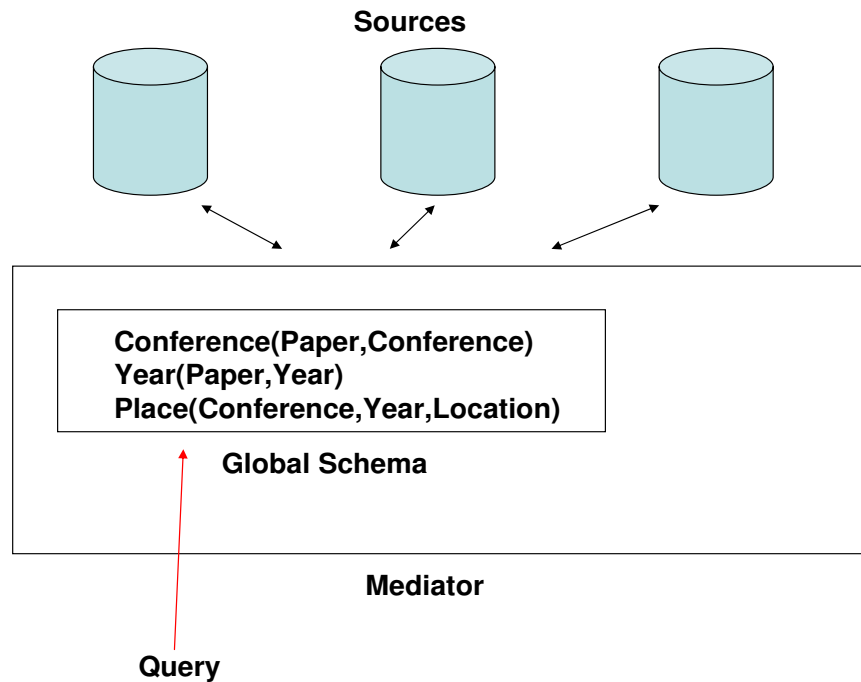
Virtual database has its own global, unifying presentation schema

- Mediator has to know what kind of data is offered by the sources and how they relate to the global schema

A problem of describing data and specifying mappings between data schemas

A form metadata management

Example: Global schema for a DB “containing” information about scientific publications:



*Conference(Paper, Conference), Year(Paper, Year),
Place(Conference, Year, Location)*

User poses queries in terms of the relations in the global schema

Query about where conference PODS'89 was held:

$$Ans(x) \leftarrow Place(pods, 1989, x)$$

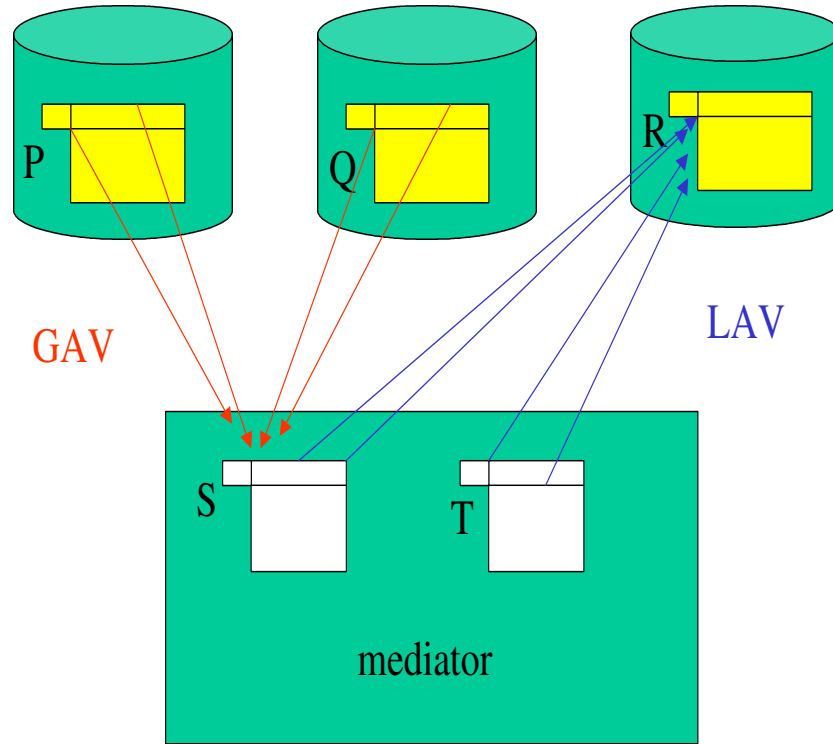
Relationships between the global schema and local sources are specified at the mediator level

This metadata determines how query plans are computed

The sources are described by means of database views

- **Global-as-View (GAV)**: Relations in the global schema are described as views over the tables in the local schemas
- **Local-as-View (LAV)**: Relations in the local, source schemas are described as views over the global schema

Each source relation is “put in correspondence” with a query (view) over the global relations



Example: Global system \mathcal{G} with global predicates P, R , and sources V_1, V_2 defined according to LAV by:

$$\begin{aligned} V_1(x, y) &\leftarrow P(x, z), R(z, y); & v_1 &= \{(a, b)\} \\ V_2(x, y) &\leftarrow P(x, y); & v_2 &= \{(a, c)\} \end{aligned}$$

From the perspective of V_2 , there could be other sources contributing with data of the same kind to P , actually so does V_1

In this sense, information in V_2 is **incomplete** wrt what \mathcal{G} “contains” (or might contain)

As usual, we assumed sources are incomplete (open)

Provisionally assume: There are **no global integrity constraints**, i.e. constraints on (the combination of) P, R

When we pose queries to a VDIS, we expect to receive answers

What answers? **What are the correct answers?**

It depends on the semantics of the system

So, **what are the semantically correct answers?**

There is no global instance and no answer to a global query in the classical sense

We have to indicate what are the **intended global instances** of \mathcal{G}

The set of admissible, legal global instances will give a meaning to the system

What global instances D if we decided to materialize global instances using the data at v_1, v_2 ? (usually we won't)

At this point the openness of the sources is taken into account

$$Legal(\mathcal{G}) := \{ \text{global } D \mid v_i \subseteq V_i(D), \quad i = 1, 2 \}$$

$V_i(D)$: Contents of view V_i evaluated on global instance D

Example: (cont.)

$$V_1(x, y) \leftarrow P(x, z), R(z, y); \quad v_1 = \{(a, b)\}$$

$$V_2(x, y) \leftarrow P(x, y); \quad v_2 = \{(a, c)\}$$

The legal instances of \mathcal{G} are all the **supersets of instances of the form:** $\{P(a, c), P(a, z), R(z, b) \mid z \in Dom\}$

1. $\{P(a, c), R(c, b)\} \in Legal(\mathcal{G})$

A **minimal legal instance**: It is legal and any proper subset is not legal

2. $\{P(a, c), P(a, e), R(e, b)\} \in Legal(\mathcal{G})$

Also a **minimal legal instance**

3. $\{P(a, c), R(c, b), P(e, e), R(e, a), R(d, d), R(a, c)\} \in Legal(\mathcal{G})$

Legal, but **not minimal**

$$4. \{P(a, c), R(e, b)\} \notin \text{Legal}(\mathcal{G})$$

We have minimal legal instances that are incomparable under set inclusion: 1. $\not\subseteq$ 2. and 2. $\not\subseteq$ 1.

MinLegal(\mathcal{G}) denotes the class of minimal legal instances

$$\text{MinLegal}(\mathcal{G}) \subsetneq \text{Legal}(\mathcal{G})$$

Global query: $Q_1: Ans(x, y) \leftarrow P(x, y)$

What are the **semantically correct answers**?

The **certain answers** to a global query are those that can be obtained from every legal global instance

$$Certain(Q) := \bigcap \{Q(D) \mid D \in Legal(\mathcal{G})\}$$

Global $Q_1: Ans(x, y) \leftarrow P(x, y)$ $Certain(Q_1) = \{(a, c)\}$

Global $Q_2: Ans(x) \leftarrow R(x, y)$ $Certain(Q_2) = \{\}$

Global $Q_3: Ans(y) \leftarrow R(x, y)$ $Certain(Q_3) = \{(b)\}$

We did not use any query plan to get them, only the semantics

Algorithms for computing certain answers?

Algorithms to produce query plans that eventually access the sources?

Some Remarks:

- **Monotone queries:** $D_1 \subseteq D_2 \Rightarrow Q(D_1) \subseteq Q(D_2)$
- Conjunctive queries with built-ins, and disjunctions thereof are monotone
Negation spoils monotonicity
- Notion of certain answer (as defined above) is not adequate for non-monotone queries
- Monotone queries Q can be correctly answered by restriction to the minimal legal instances

$$\text{Certain}(Q) = \bigcap \{Q(D) \mid D \in \text{MinLegal}(\mathcal{G})\}$$

- We will provide an algorithm for computing the certain answers to monotone queries

It is based on a specification of the minimal legal instances of \mathcal{G} (we'll see another use of specification of minimal legal instances)

Specifying Minimal Legal Instances

Example: Global system \mathcal{G} $Dom = \{a, b, c, \dots\}$

$$V_1(x, y) \leftarrow P(x, z), R(z, y) \quad v_1 = \{(a, b)\}$$

$$V_2(x, y) \leftarrow P(x, y) \quad v_2 = \{(a, c)\}$$

$$MinLegal(\mathcal{G}) = \{ \{P(a, c), P(a, z), R(z, b)\} \mid z \in Dom \}$$

Specification of minimal instances: logic program $\Pi(\mathcal{G})$

$$P(x, z) \leftarrow V_1(x, y), F_1(x, y, z) \quad (*)$$

$$P(x, y) \leftarrow V_2(x, y)$$

$$R(z, y) \leftarrow V_1(x, y), F_1(x, y, z) \quad (**)$$

$$F_1(x, y, z) \leftarrow V_1(x, y), dom(z), choice((x, y), z) \quad (***)$$

$$dom(a)., \quad dom(b)., \quad dom(c)., \quad \dots, \quad V_1(a, b)., \quad V_2(a, c).$$

Specifies global predicates in terms of source relations!

Inspired by **inverse rules algorithm** for computing certain answers
 $\text{choice}((x, y), z)$: **non-deterministically chooses** a unique value
for z for each combination of values for x, y

Programs with **choice operator** can be transformed into (usual)
programs with stable models semantics

1-1 correspondence between stable models of $\Pi(\mathcal{G})$ and minimal instances

Remarks:

- In the general case, it holds:

$$\textit{MinLegal}(\mathcal{G}) \subseteq \textit{StableMod}(\Pi(\mathcal{G})) \subseteq \textit{Legal}(\mathcal{G})$$

- In consequence, the program can be used to compute the certain answers to monotone queries

More general than any other algorithm for LAV and open sources

- The program can be refined to compute all and only the minimal legal instances

Refinement not relevant to compute certain answers to monotone queries

- The program can be adapted in order to deal with combinations of open, closed and clopen sources

Cf. Bertossi, L. and Bravo, L. “Consistent Query Answers in Virtual Data Integration Systems”. Springer LNCS 3300, 2004, pp. 42-83.

Now, if we have a global query, say $Q_2: Ans(x) \leftarrow R(x, y)$

- Combined program $\Pi' := \Pi(\mathcal{G}) \cup \{Ans(x) \leftarrow R(x, y)\}$
- Evaluate Π' under the **skeptical stable model semantics**
It makes true what is true of all stable models
- That is, the certain answers are those in the intersection of the extensions of the *Ans* predicate on all stable models

$$Certain(Q_2) = \bigcap \{Ans(M) \mid M \text{ is a stable model of } \Pi'\}$$

The same program $\Pi(\mathcal{G})$ can be used with all the queries

Systems like DLV can be used for program evaluation

For a query only the portion of program $\Pi(\mathcal{G})$ that is relevant can be built and used

Data Integration and Consistency

Still many scientific and technical issues in virtual data integration (among others):

- Uncertain data
- Quality data, preferences, provenance, etc.
- Inconsistent data

Consistency: Two sources may be individually consistent, but taken together, possibly not

E.g. Same ID number may be assigned to different people in different sources

Existing mediated integration systems (MISs) offer almost no support for consistency handling

(Even commercial DBMSs for stand alone databases offer limited general purpose support)

Mediated systems have no global IC maintenance mechanism

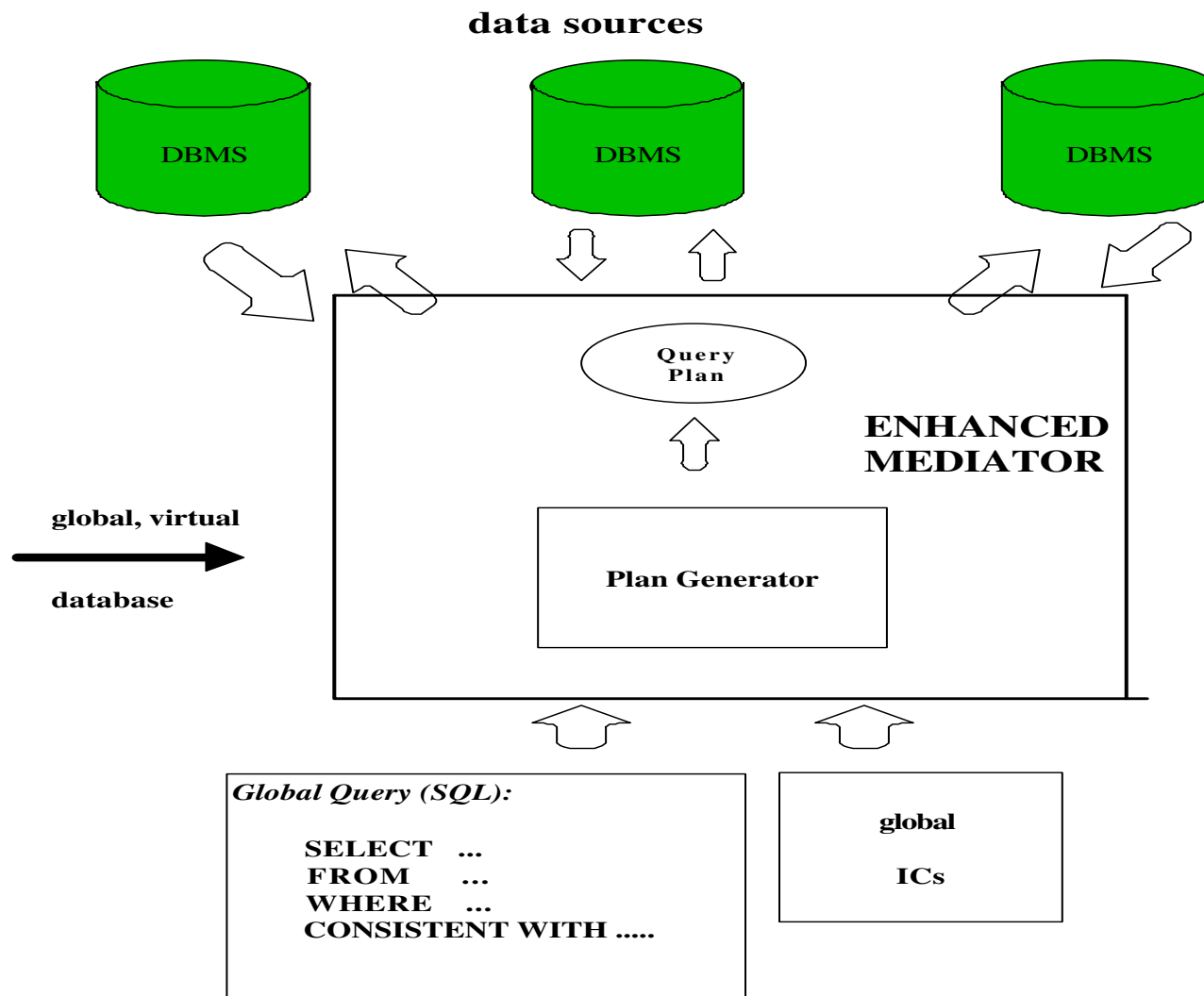
No guarantee that global ICs hold

In the virtual approach to data integration, one usually **assumes** that certain ICs hold at the global level

A possible approach: **Consistent Query Answering!**

- Do not try to enforce the consistency of the data “contained” in the integration system
- Better deal with the problem at query time
- When a query is are posed to the system, **retrieve only those answers from the global database that are “consistent with” the global ICs**
- **Obtain semantically correct answers on-the-fly!**

Cf. Bertossi, L. “Consistent Query Answering in Databases” . ACM Sigmod Record, June 2006, 35(2):68-76.



Example: System \mathcal{G}_1 with sources

$$V_1(x, y) \leftarrow R(x, y), \quad v_1 = \{(a, b), (c, d)\}$$

$$V_2(x, y) \leftarrow R(x, y), \quad v_2 = \{(c, a), (e, d)\}$$

$D = \{R(a, b), R(c, d), R(c, a), R(e, d)\}$ and its supersets are the legal instances

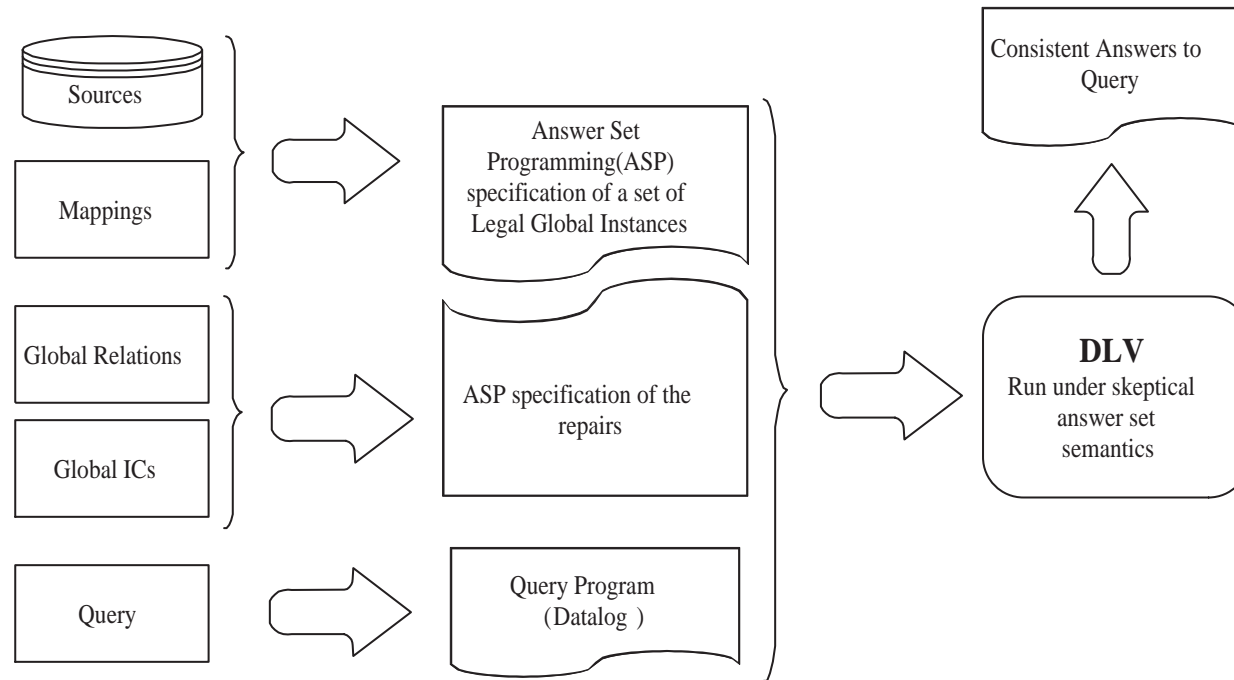
Global query Q : $R(x, y)$?

$$\text{Certain}(Q) = \{(a, b), (c, d), (c, a), (e, d)\}$$

What if we had a global functional dependency $R: X \rightarrow Y$?

Global FD not satisfied by D , nor by its supersets

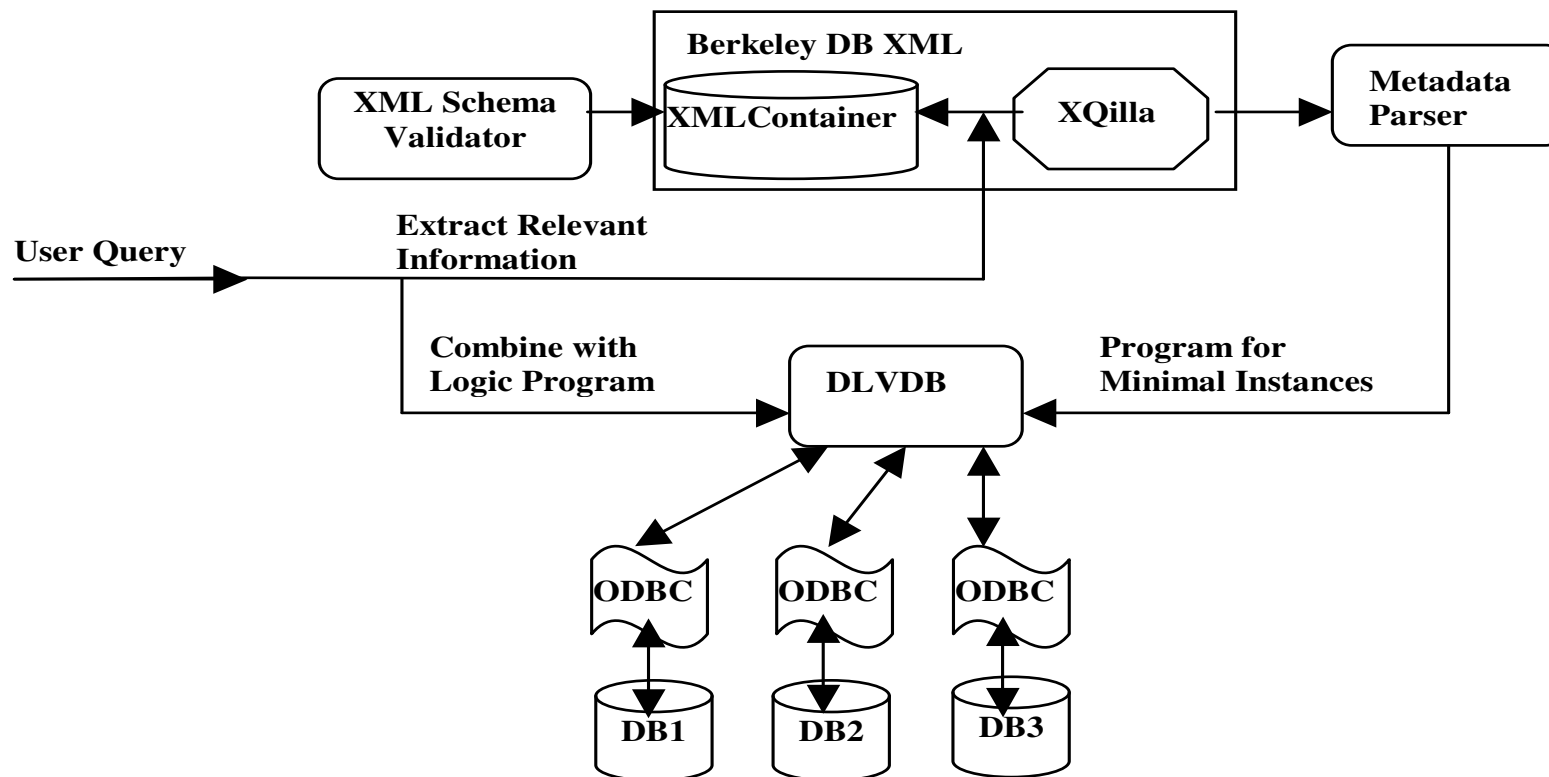
Only $(a, b), (e, d)$ should be consistent answers



Combine in one single program the programs that specify:

- The minimal legal instances
- The **repairs of minimal legal instances** wrt the global ICs
- The global query to be consistently answered (as before)

Specifying Metadata for a LAV MIS



Architecture of VISS

The mediator requires information about:

- Participating sources, their schemas, and
- How they relate to the global schema

This metadata has to be represented at the mediator level, considering the following issues:

- Sources have possibly very different schemas
They will differ in terms of predicate names, names and number of attributes, data types, etc.
As a whole, this is **unstructured (meta)data**
- Mediators might want to easily and seamlessly **share meta-data**

- We may want a **vendor-independent representation**
So that the metadata can be easily processed by any DBMS that the mediator may want to use (if any)
Commercial DBMSs differ in the way they capture, store, and provide access to metadata
- **Mappings are also metadata, that has to be represented**
Keeping track of their syntactic structure and subformulas is crucial (e.g. to build inverse rules)
Particularly important under LAV

On this basis, we use a combination of

- **XML** (in native form)
- **RuleML**, actually also a standardized XML representation
- **XQuery**, to query the former

Example: Two sources, *AnimalKingdom* and *AnimalHabitat*, with the following relations, resp.

<i>V1</i>	<i>Name</i>	<i>Class</i>	<i>Food</i>
	dolphin	mammal	fish
	camel	mammal	plant
	shark	fish	fish
	frog	amphibian	insect
	nightingale	bird	insect

<i>V2</i>	<i>Name</i>	<i>Habitat</i>
	dolphin	ocean
	camel	desert
	frog	wetlands

Global schema \mathcal{G} : *Animal*(*Name*, *Class*, *Food*), *Vertebrate*(*Name*), *Habitat*(*Name*, *Habitat*)

V1 and *V2* are defined as a views over \mathcal{G} :

$V1(\textit{Name}, \textit{Class}, \textit{Food}) \leftarrow \textit{Animal}(\textit{Name}, \textit{Class}, \textit{Food}), \textit{Vertebrate}(\textit{Name})$

$V2(\textit{Name}, \textit{Habitat}) \leftarrow \textit{Animal}(\textit{Name}, \textit{Class}, \textit{Food}), \textit{Habitat}(\textit{Name}, \textit{Habitat})$

In an XML document we **represent database schemas**:

- Source names
- DBMS at each source
- Database names at each source
- Access information for each DB
- Relation names and their schemas at each source
- Global predicates and their schemas

```

<VirInt>
<Schema>
  <Local>
    <Source name="animalkingdom">
      <Type>sqlexpress </Type>
      <Hostname>animalkingdom </Hostname>
      <Databasename>animalkingdom </Databasename>
      <Userid>test </Userid>
      <Password>test </Password>
      <Atom>
        <Rel>V1</Rel> <Var>Name</Var> <Var>Class </Var> <Var>Food</Var>
      </Atom>
    </Source>
    <Source name="animalhabitat">
      <Type>mysql </Type>
      <Hostname>animalhabitat </Hostname>
      <Databasename>animalhabitat </Databasename>
      <Userid>test1 </Userid>
      <Password>test1 </Password>
      <Atom>
        <Rel>V2</Rel> <Var>Name</Var> <Var>Habitat </Var>
      </Atom>
    </Source>
  </Local>
  <Global>
    <Atom>
      <Rel>Animal</Rel> <Var>Name</Var> <Var>Class </Var> <Var>Food</Var>
    </Atom>
    <Atom>
      <Rel>Habitat </Rel> <Var>Name</Var> <Var>Habitat </Var>
    </Atom>
    <Atom>
      <Rel>Vertebrate </Rel> <Var>Name</Var>
    </Atom>
  </Global>
</Schema>
</VirInt>

```

Next we have to **represent the mappings**

- We use RuleML
- It was developed for rule representation and exchange in the context of the semantic web
- It is based on XML
- The right XML schemas have to be invoked
- We partially used it in the specification of schemas above
- We are able to specify the syntactic components of the mapping, for further processing
- They are represented as implications of heads by bodies, etc.

```

<RuleML xmlns:rule="http://www.ruleml.org/0.91/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ruleml.org/0.91/xsd
file:///C:/thesis/VDI/datalog.xsd">
<Assert>
  <Implies>
    <head>
      <Atom>
        <Rel>V1</Rel> <Var>Name</Var> <Var>Class</Var> <Var>Food</Var>
      </Atom>
    </head>
    <body>
      <And>
        <Atom>
          <Rel>Animal</Rel> <Var>Name</Var> <Var>Class</Var> <Var>Food</Var>
        </Atom>
        <Atom>
          <Rel>Vertebrate</Rel> <Var>Name</Var>
        </Atom>
      </And>
    </body>
  </Implies>
  <Implies>
    <head>
      <Atom>
        <Rel>V2</Rel> <Var>Name</Var> <Var>Habitat</Var>
      </Atom>
    </head>
    <body>
      <And>
        <Atom>
          <Rel>Animal</Rel> <Var>Name</Var> <Var>Class</Var> <Var>Food</Var>
        </Atom>
        <Atom>
          <Rel>Habitat</Rel> <Var>Name</Var> <Var>Habitat</Var>
        </Atom>
      </And>
    </body>
  </Implies>
</Assert>
</RuleML>

```

In order to compute the certain answers to a **global monotone query** Q , e.g.

$$Ans(Name, Habitat) \leftarrow Animal(Name, Class, Food), \\ Habitat(Name, Habitat)$$

- The right fragment of the specification $\Pi(\mathcal{G})$ of minimal legal instances has to be built
The one relative to global predicates *Animal, Habitat*
- Query the XML and RuleML representations of metadata to obtain the pieces needed
- In order to identify the sources and the relations therein that are relevant to Q
- Those are the source predicates that are defined in terms of global relations that are mentioned by Q
- This is good enough: No global ICs (cf. later)
- XQuery is used

```

for $n in collection('mappingAlias')/VirInt return
<rules>
  {for $x in $n/RuleML/Assert/Implies
   where distinct-values($x/body/And/Atom/Rel) =
     ("Animal", "Habitat")
   return
   <rule>
     {for $d in $x return
      <head r1='{ $d/head/Atom/Rel}'>
        { for $v in $x/head
         where $v/Atom/Rel=$d/head/Atom/Rel
         return
           concat($d/head/Atom/Rel, '(' ,
             string-join($v/Atom/Var, ', '), ')')}
        </head>
      }
     }
   {for $b in distinct-values($x/body/And/Atom/Rel)
   return
     <body r1='{ $b}'>
       { for $m in $n/Schema/Global/Atom
        where $m/Rel=$b
        return concat($b, '(' ,
          string-join($m/Rel/following-sibling::Var, ', '), ')')}
       }
     </body>
   }
   <body r1=''>
   { for $q in distinct-values($x/body/And/Atom/Ind)
   let $l as xs:integer := index-of($x/body/And/Atom/*,$q)-1
   let $r := $x/body/And/Atom/Ind/preceding-sibling::Rel/text()
   let $s := $n/Schema/Global/Atom/Rel[text()=$r]/../Var[$l]
   return
   if ($q != '') then
     (concat('(',$s,'=', $q, ')', ',', ','))
   else ()
   }
   </body>
   </rule>
   }
</rules>

```


The query first identifies the source predicates that are defined in terms of global relations *Animal*, *Habitat* that appear in the query

For each of them, it identifies the parts of the bodies of their definitions, including built-ins (the last part)

The following view definitions are used later to built $\Pi(\mathcal{G})$

```
<rules xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='mappingrule.xsd'>
  <rule>
    <head r1="V1">V1(Name, Class, Food)</head>
    <body r1="Animal">Animal(Name, Class, Food)</body>
    <body r1="Vertebrate">Vertebrate(Name)</body>
  </rule>
  <rule>
    <head r1="V2">V2(Name, Habitat)</head>
    <body r1="Animal">Animal(Name, Class, Food)</body>
    <body r1="Habitat">Habitat(Name, Habitat)</body>
  </rule>
</rules>
```

This result is used to:

- Build $\Pi(\mathcal{G})$ as required by DLV
- Identify via XQuery sources that are relevant to the query
- For the later appropriate **data import commands** are generated and added to $\Pi(\mathcal{G})$
- DLV uses them to retrieve the facts directly from the data sources via ODBC

```

#import(animalkingdom , " test" , " test" , "SELECT * FROM V1" , V1,
        type : Q_CONST, Q_CONST, Q_CONST).
#import(animalhabitat , " root" , " root" , "SELECT * FROM V2" , V2,
        type : Q_CONST, Q_CONST).

Animal(Name, Class , Food) :- V1(Name, Class , Food).
Animal(Name, Class , Food) :- V2(Name,Habitat),
        f1(Name,Habitat , Class) , f2(Name, Habitat , Food).
f1(Name, Habitat , Class) :- V2(Name, Habitat) , dom(Class) ,
        chosen1(Name, Habitat , Class).
chosen1(Name,Habitat , Class) :- V2(Name,Habitat) , dom(Class) ,
        not diffchoice1(Name,Habitat , Class).
diffchoice1(Name, Habitat , Class) :- chosen1(Name, Habitat , U) ,
        dom(Class) , U != Class.
f2(Name, Habitat , Food) :- V2(Name, Habitat) , dom(Food) ,
        chosen2(Name, Habitat , Food).
chosen2(Name, Habitat , Food) :- V2(Name,Habitat) , dom(Food) ,
        not diffchoice2(Name,Habitat , Food).
diffchoice2(Name, Habitat , Food) :- chosen2(Name, Habitat , U) ,
        dom(Food) , U != Food.

Habitat(Name, Habitat) :- V2(Name, Habitat).

```

Run with DLV:

```

dl.exe -silent -cautious test2.dlv
dolphins, ocean
camels, desert
frogs, wetlands

```

With Global ICs

If there are global ICs:

- Global ICs can also be represented with RuleML
- Dependencies between global predicates may appear
- This may add new relevant sources for a global query
Those that become indirectly relevant to the query
- Inconsistencies may arise
- In this case a relevant fragment of the refined version of $\Pi(\mathcal{G})$ has to be used (as a first layer)
- Repair programs have to be built on top
- Everything is run with DLV as before

This presentation: www.scs.carleton.ca/~bertossi