Description Logics for Conceptual Design, Information Access, and Ontology Integration (ISWC-2002)

Enrico Franconi

franconi@cs.man.ac.uk
http://www.cs.man.ac.uk/~franconi

Department of Computer Science, University of Manchester
Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology
- Ontology Integration
Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology
- Ontology Integration
What is an Ontology

- An ontology is a formal conceptualisation of the world: a conceptual schema.
What is an Ontology

- An ontology is a formal conceptualisation of the world: a **conceptual schema**.
- An ontology specifies a set of **constraints**, which declare what should necessarily hold in any possible world.
What is an Ontology

- An ontology is a formal conceptualisation of the world: a conceptual schema.
- An ontology specifies a set of constraints, which declare what should necessarily hold in any possible world.
- Any possible world should conform to the constraints expressed by the ontology.
What is an Ontology

• An ontology is a formal conceptualisation of the world: a conceptual schema.

• An ontology specifies a set of constraints, which declare what should necessarily hold in any possible world.

• Any possible world should conform to the constraints expressed by the ontology.

• Given an ontology, a legal world description is a possible world satisfying the constraints.
The role of a Conceptual Schema
The role of a Conceptual Schema

Diagram:

- **Constraints**
  - Conceptual Schema
    - Logical Schema
    - Data Store
The role of a Conceptual Schema

- **Constraints**
  - Conceptual Schema

- Logical Schema

- Data Store

- Query

- Result
The role of a Conceptual Schema

![Diagram showing the role of a Conceptual Schema]

- **Deduction**
  - **Constraints**
    - Conceptual Schema
  - Logical Schema
- **Data Store**
- **Query**
- **Result**
The role of a Conceptual Schema
Ontology languages and Conceptual Data Models

- An ontology language usually introduces concepts (aka classes, entities), properties of concepts (aka slots, attributes, roles), relationships between concepts (aka associations), and additional constraints.
Ontology languages and Conceptual Data Models

- An ontology language usually introduces concepts (aka classes, entities), properties of concepts (aka slots, attributes, roles), relationships between concepts (aka associations), and additional constraints.

- Ontology languages may be simple (e.g., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua and DAML+OIL).
Ontology languages and Conceptual Data Models

• An ontology language usually introduces concepts (aka classes, entities), properties of concepts (aka slots, attributes, roles), relationships between concepts (aka associations), and additional constraints.

• Ontology languages may be simple (e.g., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua and DAML+OIL).

• Ontology languages are typically expressed by means of diagrams.
Ontology languages and Conceptual Data Models

- An ontology language usually introduces **concepts** (aka classes, entities), **properties** of concepts (aka slots, attributes, roles), **relationships** between concepts (aka associations), and additional **constraints**.

- Ontology languages may be simple (e.g., having only concepts and taxonomies), frame-based (having only concepts and properties), or logic-based (e.g. Ontolingua and DAML+OIL).

- Ontology languages are typically expressed by means of diagrams.

- **Entity-Relationship** schemas and **UML** class diagrams can be considered as ontologies.
UML Class Diagram

Employee
- PaySlipNumber: Integer
- Salary: Integer

Manager

AreaManager

TopManager

Project
- ProjectCode: String

Manager

Works-for

{disjoint, complete}

1..1

Manages

Project

1..1

Employee

1..*

TopManager

(7/56)
Entity-Relationship Schema

- **Employee**
  - PaySlipNumber (Integer)
  - Salary (Integer)

- **Manager**

- **Project**
  - ProjectCode (String)

- **AreaManager**, **TopManager**

- **Works-for** (1:n)

- **Manages** (1:1)
Semantics

In a specific world:

- A class is a set of instances;
- a n-ary relationship is a set of n-tuples of instances;
- an attribute is a set of pairs of an instance and a domain element.
Semantics

In a specific world:

- A class is a \textit{set of instances};
- a n-ary relationship is a \textit{set of n-tuples of instances};
- an attribute is a \textit{set of pairs of an instance and a domain element}. 

```
<table>
<thead>
<tr>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
<td>P5</td>
</tr>
</tbody>
</table>
```

```
“P12a”
“P02b”
“P2a/1”
“P9”
```

Employee  Project  String
Works-for
In a specific world:

- A class is a set of instances;
- a n-ary relationship is a set of n-tuples of instances;
- an attribute is a set of pairs of an instance and a domain element.
A world is described by sets of instances
The relational representation of a world

<table>
<thead>
<tr>
<th>Employee</th>
<th>Project</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>emploeeId</td>
<td>projectId</td>
<td>anystring</td>
</tr>
<tr>
<td>E₁</td>
<td>P₁</td>
<td>“P12a”</td>
</tr>
<tr>
<td>E₂</td>
<td>P₁</td>
<td>“P02b”</td>
</tr>
<tr>
<td>E₃</td>
<td>P₂</td>
<td>“P2a/1”</td>
</tr>
<tr>
<td>E₄</td>
<td>P₃</td>
<td>“P9”</td>
</tr>
<tr>
<td>E₅</td>
<td>P₃</td>
<td>…</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Works-for</th>
<th>ProjectCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>emploeeId</td>
<td>projectld</td>
</tr>
<tr>
<td>E₁</td>
<td>P₁</td>
</tr>
<tr>
<td>E₂</td>
<td>P₁</td>
</tr>
<tr>
<td>E₂</td>
<td>P₂</td>
</tr>
<tr>
<td>E₃</td>
<td>P₁</td>
</tr>
<tr>
<td>E₄</td>
<td>P₂</td>
</tr>
<tr>
<td>E₅</td>
<td>P₃</td>
</tr>
</tbody>
</table>
The graph representation of a world – e.g. RDF triples

E₁:Employee \(\xrightarrow{\text{Works-for}}\) P₃:Project
E₂:Employee \(\xrightarrow{\text{Works-for}}\) P₁:Project
E₃:Employee \(\xrightarrow{\text{Works-for}}\) P₁:Project
E₄:Employee \(\xrightarrow{\text{Works-for}}\) P₁:Project
E₅:Employee

P₁:Project \(\xrightarrow{\text{ProjectCode}}\) “P12a”:String
P₂:Project \(\xrightarrow{\text{ProjectCode}}\) “P02b”:String
P₃:Project \(\xrightarrow{\text{ProjectCode}}\) “P2a/1”:String
P₄:Project \(\xrightarrow{\text{ProjectCode}}\) “P9”:String
Constraints introduced by Relationships

Employee

Works-for

Project
Constraints introduced by Relationships

Employee \quad \text{Works-for} \quad Project

\text{Works-for} \subseteq \text{Employee} \times \text{Project}
Constraints introduced by Relationships

Employee \( A_1 \) \[\text{Works-for}\] \( A_2 \) Project

Works-for \( \subseteq \) Employee \( \times \) Project
Constraints introduced by Attributes

| Project
| ProjectCode : String |
Constraints introduced by Attributes

Project

| ProjectCode : String |

Project \subseteq \{ p \mid \#(\text{ProjectCode} \cap (\{p\} \times \text{String})) \geq 1\}
Constraints introduced by Cardinality Constraints

Employee \((\text{min}, \text{max})\) Works-for Project
Constraints introduced by Cardinality Constraints

TopManager \subseteq \{ m \mid \text{max} \geq \#(\text{Manages} \cap (\{m\} \times \Omega)) \geq \text{min}\}

(where \(\Omega\) is the set of all instances)
Constraints introduced by Cardinality Constraints

TopManager $\subseteq \{ m \mid \max \geq \#(\text{Manages} \cap (\{m\} \times \Omega)) \geq \min \}$

(where $\Omega$ is the set of all instances)
Constraints introduced by ISA

Employee

Manager
Constraints introduced by ISA

Manager ⊆ Employee
Disjoint and Total constraints

Manager

{disjoint,complete}

AreaManager  TopManager
**Disjoint and Total constraints**

- **ISA:** AreaManager $\subseteq$ Manager
- **ISA:** TopManager $\subseteq$ Manager
- **disjoint:** AreaManager $\cap$ TopManager = $\emptyset$
- **total:** Manager $\subseteq$ AreaManager $\cup$ TopManager
Constraints introduced by the initial diagram

- Works-for $\subseteq$ Employee $\times$ Project
- Manages $\subseteq$ TopManager $\times$ Project

Employee $\subseteq \{ e | \#(\text{Pay Slip Number} \cap (\{e\} \times \text{Integer})) \geq 1 \}$

Employee $\subseteq \{ e | \#(\text{Salary} \cap (\{e\} \times \text{Integer})) \geq 1 \}$

Project $\subseteq \{ p | \#(\text{Project Code} \cap (\{p\} \times \text{String})) \geq 1 \}$

TopManager $\subseteq \{ m | 1 \geq \#(\text{Manages} \cap (\{m\} \times \Omega)) \geq 1 \}$

Project $\subseteq \{ p | 1 \geq \#(\text{Manages} \cap (\Omega \times \{p\})) \geq 1 \}$

Project $\subseteq \{ p | \#(\text{Works-for} \cap (\Omega \times \{p\})) \geq 1 \}$

Manager $\subseteq$ Employee

AreaManager $\subseteq$ Manager

TopManager $\subseteq$ Manager

AreaManager $\cap$ TopManager $= \emptyset$

Manager $\subseteq$ AreaManager $\cup$ TopManager
Reasoning

Given an ontology – seen as a collection of constraints – it is possible that additional constraints can be inferred.

- A class is **inconsistent** if it denotes the empty set in any legal world description.
- A class is a **subclass** of another class if the former denotes a subset of the set denoted by the latter in any legal world description.
- Two classes are **equivalent** if they denote the same set in any legal world description.
- A **stricter** contraint is inferred – e.g., a **cardinality** contraint – if it holds in any legal world description.
- ...
Simple reasoning example

Person

{{disjoint}}

{disjoint, covering}

Italian

Lazy

LatinLover

English

Gentleman

Hooligan
implies

\[ \text{LatinLover} = \emptyset \]

\[ \text{Italian} \subseteq \text{Lazy} \]

\[ \text{Italian} \equiv \text{Lazy} \]
Reasoning by cases

- Italian
- Lazy
- Mafioso
- LatinLover
- ItalianProf

{disjoint, complete}

{disjoint}
Reasoning by cases

\begin{align*}
\text{Italian} & \quad \{\text{disjoint, complete}\} \\
\text{Lazy} & \quad \text{Mafioso} & \quad \text{LatinLover} & \quad \text{ItalianProf}
\end{align*}

\text{implies}

\text{ItalianProf} \subseteq \text{LatinLover}
ISA and Inheritance

Employee

Salary: Integer

Manager

implies

Manager from

Salary: Integer
implies

$$\text{Manager} \subseteq \{m \mid \#(\text{Salary} \cap \{m\} \times \text{Integer}) \geq 1\}$$
Bijection between Classes

Natural Number \rightarrow Even Number

rel

1..1

implies the classes 'Natural Number' and 'Even Number' contain the same number of instances.
Bijection between Classes

implies

“the classes ’Natural Number’ and ’Even Number’ contain the same number of instances”.
Bijection between Classes

implies

“the classes ‘Natural Number’ and ‘Even Number’ contain the same number of instances”.

If the domain is finite:   Natural Number \equiv Even Number
Infinite worlds

The classes Root and Node contain an infinite number of instances.
Infinite worlds

implies

“the classes Root and Node contain an infinite number of instances”.
Ontologies in First Order Logic

- We have introduced ontology languages that specify a set of constraints that should be satisfied by the world of interest.

- The *interpretation* of an ontology is therefore defined as the collection of all the *legal world descriptions* – i.e., all the (finite) relational structures which conform to the constraints imposed by the ontology.

- An alternative way to define the interpretation: an ontology is mapped into a set of *First Order Logic* (FOL) formulas.

- The legal world descriptions (i.e., the interpretation) of an ontology are all the *models* of the FOL theory associated to it.
∀x, y. Works-for(x, y) → Employee(x) ∧ Project(y)
∀x, y. Manages(x, y) → Top-Manager(x) ∧ Project(y)
∀y. Project(y) → ∃x. Works-for(x, y)
∀y. Project(y) → ∃=1 x. Manages(x, y)
∀x. Top-Manager(x) → ∃=1 y. Manages(x, y)
∀x. Manager(x) → Employee(x)
∀x. Manager(x) → Area-Manager(x) ∨ Top-Manager(x)
∀x. Area-Manager(x) → Manager(x) ∧ ¬Top-Manager(x)
∀x. Top-Manager(x) → Manager(x)
Additional constraints

- Managers do not work for a project (she/he just manages it):
  \[ \forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x, y) \]
Additional constraints

- Managers do not work for a project (she/he just manages it):
  \[ \forall x. \text{Manager}(x) \rightarrow \forall y. \neg \text{WORKS-FOR}(x, y) \]

- If the minimum cardinality for the participation of employees to the *works-for* relationship is increased, then . . .
Key constraints

A key is a set of attributes of a class whose value uniquely identify elements of the class itself.

\[ \forall x. \text{Project}(x) \rightarrow \exists^=1 y. \text{ProjectCode}(x,y) \land \text{String}(y) \]

\[ \forall y. \exists x. \text{ProjectCode}(x,y) \rightarrow \exists^=1 x. \text{ProjectCode}(x,y) \land \text{Project}(x) \]
Summary

• Logic and Conceptual Modelling

• Description Logics for Conceptual Modelling

• Queries with an Ontology

• Ontology Integration
The **DLR** Description Logic – a fragment of FOL

- **relationships**: interpreted as **sets of tuples** of a given arity

\[ R \rightarrow \top_n \mid RN \mid \neg R \mid R_1 \sqcap R_2 \mid R_1 \sqcup R_2 \mid i/n : C \]

- **classes**: interpreted as **sets of objects**

\[ C \rightarrow \top \mid CN \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists^{k}[i] R \]

- **conceptual schema**: \[ R \sqsubseteq R' \mid C \sqsubseteq C' \mid R \not\sqsubseteq R' \mid C \not\sqsubseteq C' \]

Works-for \( \sqsubseteq \) subj/2 : Employee \( \sqcap \) obj/2 : Project

TopManager \( \sqsubseteq \) Manager \( \sqcap \) \( \exists^{-1} \) [man] Manages
Encoding conceptual data models in $DLR$

- Object-oriented data models (e.g., UML and ODMG)
- Semantic data models (e.g., EER and ORM)
- Frame-based ontology languages (e.g., DAML+OIL)
Encoding conceptual data models in $\text{DLR}$

- Object-oriented data models (e.g., UML and ODMG)
- Semantic data models (e.g., EER and ORM)
- Frame-based ontology languages (e.g., DAML+OIL)
- Theorems prove that an ontology and its encoding as $\text{DLR}$ knowledge bases constrain every world description in the same way – i.e., the models of the $\text{DLR}$ theory correspond to the legal world descriptions of the ontology, and vice-versa.
Works-for $\equiv$ \text{emp}/2 : \text{Employee} \sqcap \text{act}/2 : \text{Project}

Manages $\equiv$ \text{man}/2 : \text{TopManager} \sqcap \text{prj}/2 : \text{Project}

Employee $\equiv$ \exists^=1[\text{worker}] (\text{PaySlipNumber} \sqcap \text{num}/2 : \text{Integer}) \sqcap$
\quad \exists^=1[\text{payee}] (\text{Salary} \sqcap \text{amount}/2 : \text{Integer})
\quad \top
\quad \exists^\leq1[\text{num}] (\text{PaySlipNumber} \sqcap \text{worker}/2 : \text{Employee})

Manager $\equiv$ \text{Employee} \sqcap (\text{AreaManager} \sqcap \neg \text{TopManager})

AreaManager $\equiv$ \text{Manager} \sqcap \neg \text{TopManager}

TopManager $\equiv$ \text{Manager} \sqcap \exists^=1[\text{man}] \text{Manages}$

Project $\equiv$ \exists^\geq1[\text{act}] \text{Works-for} \sqcap \exists^=1[\text{prj}] \text{Manages}$

...
Managers are employees who do not work for a project (she/he just manages it):

\[ \text{Employee} \cap \neg (\exists \geq 1 \text{[emp]} \text{Works-for}) \subseteq \text{Manager} \]

\[ \text{Manager} \subseteq \neg (\exists \geq 1 \text{[emp]} \text{Works-for}) \]
Managers are employees who do not work for a project (she/he just manages it):

\[
\text{Employee} \sqcap \neg (\exists \geq 1 [\text{emp}] \text{Works-for}) \sqsubseteq \text{Manager}
\]

\[
\text{Manager} \sqsubseteq \neg (\exists \geq 1 [\text{emp}] \text{Works-for})
\]

For every project, there is at least one employee who is not a manager:

\[
\text{Project} \sqsubseteq \exists \geq 1 [\text{act}] (\text{Works-for} \sqcap \text{emp} : \neg \text{Manager})
\]
Extensions of $\mathcal{DLR}$

- $\mathcal{DLR}_{\text{reg}}$: regular expressions and recursive views (beyond FOL)
- $\mathcal{DLR}_{\text{US}}$: temporal constructs to model temporal databases (temporal logic)
- $\mathcal{DLR}_{\text{key}}$: general key constraints
Reasoning with Ontologies

- Exploit the $\mathcal{DLR}$ reasoning procedures for solving reasoning problems in the ontology enriched with constraints.

- Logical implication and consistency for $\mathcal{DLR}$ knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.
Reasoning with Ontologies

- Exploit the $\mathcal{DLR}$ reasoning procedures for solving reasoning problems in the ontology enriched with constraints.

- Logical implication and consistency for $\mathcal{DLR}$ knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.

- Ontology consistency checking with constraints and logical implication of constraints in ontologies are all decidable EXPTIME-complete problems.
Reasoning with Ontologies

- Exploit the $\mathcal{DLR}$ reasoning procedures for solving reasoning problems in the ontology enriched with constraints.

- Logical implication and consistency for $\mathcal{DLR}$ knowledge bases is decidable and EXPTIME-complete, and practical, proved correct and complete algorithms exist in implemented systems.

- $\leadsto$ Ontology consistency checking with constraints and logical implication of constraints in ontologies are all decidable EXPTIME-complete problems.

- i•com is an implemented conceptual modelling tool using in the background a $\mathcal{DLR}$ ontology server supporting the ontology design.
Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology
- Ontology Integration
The role of a Conceptual Schema – revisited
The role of a Conceptual Schema – revisited

Diagram: Constraints
- Conceptual Schema
  - Logical Schema
  - Data Store
The role of a Conceptual Schema – revisited
The role of a Conceptual Schema – revisited
The role of a Conceptual Schema – revisited

Deduction

Constraints

Conceptual Schema

Logical Schema

Data Store

Query

Result
The role of a Conceptual Schema – revisited
The role of a Conceptual Schema – revisited

Deduction

Constraints

Conceptual Schema

Logical Schema

Data Store

Query → Result

Query → Result
The role of a Conceptual Schema – revisited
The role of a Conceptual Schema – revisited
The role of a Conceptual Schema – revisited
Adapting standard DB query technology

- Assumption 1: complete information about each term appearing in the ontology
- Assumption 2: consistent information with respect to the constraints introduced by the ontology
- Problem: answer a query over the ontology vocabulary
Adapting standard DB query technology

- Assumption 1: *complete information about each term* appearing in the ontology
- Assumption 2: *consistent* information with respect to the constraints introduced by the ontology
- Problem: answer a query over the ontology vocabulary
- Solution: use a standard DB technology (e.g., SQL, datalog, etc)
Adapting standard DB query technology

- Assumption 1: complete information about each term appearing in the ontology

- Assumption 2: consistent information with respect to the constraints introduced by the ontology

- Problem: answer a query over the ontology vocabulary

- Solution: use a standard DB technology (e.g., SQL, datalog, etc)

- Assumption 1 is against the principle that an ontology presents a richer vocabulary than the data stores.
Weakening the assumptions

- Assumption 1\textsubscript{weak}: complete information about \textit{some term} appearing in the ontology

- Standard DB technologies do not apply

- The query answering problem in this context is inherently complex
Example

OfficeMate

Employee

Manager

\{\text{disjoint, complete}\}

AreaManager

TopManager

AreaManager_p

TopManager_p

Supervises

Supervises

OfficeMate
Example

Employee = \{ John, Andrea, Mary, Paul \}
Manager = \{ John, Andrea \}
AreaManager_p = \{ Paul \}
TopManager_p = \{ Mary \}
Supervises = \{ (John, Andrea), (John, Mary) \}
OfficeMate = \{ (Mary, Andrea), (Andrea, Paul) \}
Example

Employee = \{ \text{John, Andrea, Mary, Paul} \}
Manager = \{ \text{John, Andrea} \}
AreaManager_p = \{ \text{Paul} \}
TopManager_p = \{ \text{Mary} \}
Supervises = \{ (\text{John, Andrea}), (\text{John, Mary}) \}
OfficeMate = \{ (\text{Mary, Andrea}), (\text{Andrea, Paul}) \}

John: Manager

Supervises

Andrea: Manager

OfficeMate

Mary: TopManager_p

OfficeMate

Paul: AreaManager_p
Example (cont.)

OfficeMate

Employee

Manager

\{ disjoint, complete \}

AreaManager

TopManager

AreaManager\_p

TopManager\_p
Example (cont.)

Supervises

**John**: Manager

**Andrea**: Manager
**OfficeMate**

**Mary**: TopManager

**Paul**: AreaManager

**Employee**

Supervises

**Manager**

Supervises

**AreaManager**
**TopManager**

{disjoint, complete}
Example (cont.)

John: Manager

Andrea: Manager

Mary: TopManager

Paul: AreaManager

Q :- Supervises(John, X), TopManager(X), Officemate(X, Y), AreaManager(Y)
Example (cont.)

\[ Q := \text{Supervises}(\text{John}, X), \text{TopManager}(X), \text{Officemate}(X, Y), \text{AreaManager}(Y) \]

YES!
Weakening the assumptions, II

In general, the link of the ontology with the information source (called mapping) can be given in terms of a set of views:

- **GAV** (global-as-view): for each ontology term one view over the information source is given;
- **LAV** (local-as-view): for each information source term one view over the ontology terms is given;
An Information Source Example

CompanyEmployee/2; CompanyProject/3

<table>
<thead>
<tr>
<th>CompanyEmployee</th>
<th>CompanyProject</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>project</td>
</tr>
<tr>
<td>john</td>
<td>esprit-dwq</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>project</th>
<th>manager</th>
<th>department</th>
</tr>
</thead>
<tbody>
<tr>
<td>esprit-dwq</td>
<td>enrico</td>
<td>cs-uman</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
An Information Source Example

CompanyEmployee/2; CompanyProject/3

<table>
<thead>
<tr>
<th>CompanyEmployee</th>
<th>CompanyProject</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>project</td>
</tr>
<tr>
<td>john</td>
<td>esprit-dwq</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>project</th>
<th>manager</th>
<th>department</th>
</tr>
</thead>
<tbody>
<tr>
<td>esprit-dwq</td>
<td>enrico</td>
<td>cs-uman</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Q = “Tell me the projects in which John works, and their managers and departments.”
Q = “Tell me the projects in which John works, and their managers and departments.”

SELECT project, manager, department
FROM CompanyEmployee, CompanyProject
WHERE CompanyEmployee.name = “john” AND
    CompanyEmployee.project = CompanyProject.project
An Information Source Example

CompanyEmployee/2; CompanyProject/3

<table>
<thead>
<tr>
<th>CompanyEmployee</th>
<th>CompanyProject</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>project</td>
</tr>
<tr>
<td>john</td>
<td>esprit-dwq</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>project</th>
<th>manager</th>
<th>department</th>
</tr>
</thead>
<tbody>
<tr>
<td>esprit-dwq</td>
<td>enrico</td>
<td>cs-uman</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Q = “Tell me the projects in which John works, and their managers and departments.”

SELECT project, manager, department
FROM CompanyEmployee, CompanyProject
WHERE CompanyEmployee.name = "john" AND
    CompanyEmployee.project = CompanyProject.project

Q ≡ π_{proj., manager, dept.} \sigma_{name=\text{john}} (CompanyEmployee \bowtie_{proj.} CompanyProject)
An Information Source Example

CompanyEmployee/2; CompanyProject/3

<table>
<thead>
<tr>
<th>CompanyEmployee</th>
<th>CompanyProject</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>project</td>
</tr>
<tr>
<td>john</td>
<td>esprit-dwq</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Q = “Tell me the projects in which John works, and their managers and departments.”

```
SELECT project, manager, department
FROM CompanyEmployee, CompanyProject
WHERE CompanyEmployee.name = “john” AND
    CompanyEmployee.project = CompanyProject.project
```

Q ≡ π_{proj., manager, dept.} σ_{name=john} (CompanyEmployee ⊗_{project} CompanyProject)

Q(x, y, z) ← CompanyEmployee(john, x) ∧ CompanyProject(x, y, z)
LAV: local-as-view

- **Employee**
  - PaySlipNumber: Integer
  - Salary: Integer

- **Manager**
  - {disjoint, complete}

- **Project**
  - ProjectCode: String

- **AreaManager**
  - 1..1

- **TopManager**
  - 1..1

- **Works-for**
  - 1..*

- **Manages**
  - 1..1

- **CompanyEmployee**
  - Employee(x, y, z)
    - Project(x, y)
    - Works-for(x, y)

- **CompanyProject**
  - Project(x, y, z)
    - Manager(y, x)
    - Department(z, x)
    - Manages(y, x)
    - Resp-for(z, x)
LAV: local-as-view

CompanyEmployee(x, y) ← Employee(x) ∧ Project(y) ∧ Works-for(x, y).

CompanyProject(x, y, z) ← Project(x) ∧ Manager(y) ∧ Department(z) ∧
                      Manages(y, x) ∧ Resp-for(z, x).
GAV: global-as-view

Employee
PaySlipNumber: Integer
Salary: Integer

Manager

AreaManager

TopManager

Project
ProjectCode: String

Works-for
1..*

Manages
1..1

1..1

{disjoint, complete}
GAV: global-as-view

Project(x) ⇐ CompanyEmployee(y, x) ∪ CompanyProject(x, y, z)
Works-for(x, y) ⇐ CompanyEmployee(x, y)
TopManager(x) ⇐ CompanyProject(y, x, z)
Manages(x, y) ⇐ CompanyProject(y, x, z)
Querying via the Ontology (local-as-view)

\[ Q(x, y, z) \leftarrow Project(x) \land Works-for(john, x) \land TopManager(y) \land Manages(y, x) \land \neg InterestGroup(z) \land Resp-for(z, x). \]

```
CompanyEmployee(x, y) \leftarrow 
   Employee(x) \land Project(y) \land Works-for(x, y).

CompanyProject(x, y, z) \leftarrow 
   Project(x) \land Manager(y) \land Department(z) \land 
   Manages(y, x) \land Resp-for(z, x).
```
Querying via the Ontology (local-as-view)

\[
Q(x, y, z) \leftarrow Project(x) \land Works-for(john, x) \land TopManager(y) \land Manages(y, x) \land \\
\neg \text{InterestGroup}(z) \land \text{Resp-for}(z, x).
\]

\[
\begin{align*}
\text{CompanyEmployee}(x, y) & \leftarrow \\
\text{Employee}(x) \land Project(y) \land Works-for(x, y).
\end{align*}
\]

\[
\begin{align*}
\text{CompanyProject}(x, y, z) & \leftarrow \\
Project(x) \land Manager(y) \land Department(z) \land \\
\text{Manages}(y, x) \land \text{Resp-for}(z, x).
\end{align*}
\]

\[
\sim Q(x, y, z) \leftarrow \text{CompanyEmployee}(john, x) \land \text{CompanyProject}(x, y, z)
\]
Querying via the Ontology (global-as-view)

\[ Q(x, y, z) \triangleq \text{Project}(x) \land \text{Works-for}(\text{john}, x) \land \text{TopManager}(y) \land \text{Manages}(y, x) \land \\
\neg \text{InterestGroup}(z) \land \text{Resp-for}(z, x). \]

\begin{align*}
\text{Project}(x) & \triangleq \text{CompanyEmployee}(y, x) \cup \\
& \quad \text{CompanyProject}(x, y, z) \\
\text{Works-for}(x, y) & \triangleq \text{CompanyEmployee}(x, y) \\
\text{TopManager}(x) & \triangleq \text{CompanyProject}(y, x, z) \\
\text{Manages}(x, y) & \triangleq \text{CompanyProject}(y, x, z) \\
\end{align*}
Querying via the Ontology (global-as-view)

\[ Q(x, y, z) \Leftarrow \text{Project}(x) \land \text{Works-for}(\text{john}, x) \land \text{TopManager}(y) \land \text{Manages}(y, x) \land \\
\neg \text{InterestGroup}(z) \land \text{Resp-for}(z, x). \]

\[
\begin{align*}
\text{Project}(x) & \Leftarrow \text{CompanyEmployee}(y, x) \cup \\
& \text{CompanyProject}(x, y, z)
\end{align*}
\]

\[
\begin{align*}
\text{Works-for}(x, y) & \Leftarrow \text{CompanyEmployee}(x, y) \\
\text{TopManager}(x) & \Leftarrow \text{CompanyProject}(y, x, z) \\
\text{Manages}(x, y) & \Leftarrow \text{CompanyProject}(y, x, z) \\
\ldots
\end{align*}
\]

\[ \neg \sim Q(x, y, z) \Leftarrow \text{CompanyEmployee}(\text{john}, x) \land \text{CompanyProject}(x, y, z) \]
Reasoning over queries

\[ Q(x, y) \iff \text{Employee}(x) \land \text{Works-for}(x, y) \land \text{Manages}(x, y) \]

Manager \subseteq \neg(\exists^{\geq 1}[emp]\text{Works-for})
Reasoning over queries

\[ Q(x, y) \leftarrow \text{Employee}(x) \land \text{Works-for}(x, y) \land \text{Manages}(x, y) \]

Manager \subseteq \neg(\exists^{\geq 1}[\text{emp}]\text{Works-for})

\[ \text{\textless\textgreater \ INCONSISTENT QUERY!} \]
Summary

- Logic and Conceptual Modelling
- Description Logics for Conceptual Modelling
- Queries with an Ontology
- Ontology Integration
Mediator Architecture for Ontology Integration

Inter-schema Constraints

Conceptual Schema

Logical Schema

Database

Query

Result
DWQ Ontology Integration Architecture

Integration System

Meta Model

Conceptual Data Warehouse Model

Enterprise Model

Source Model 1

Source Model n

Source Model 1

Source Model n

Data Schema

Data Warehouse Schema

Source Schema 1

Source Schema n

Source Schema 1

Source Schema n

Mediators

Data Warehouse Store

Wrappers

Sources

Source Data Store 1

Source Data Store n

Interface

---

conceptual link

logical link

conceptual/logical mapping

physical/logical mapping

data flow
Local-as-view vs. Global-as-view

Local-as-view (Information Manifold, DWQ, Picsel):

- High modularity and reusability (when a source changes, only its view definition is changed).
- Relationships between sources can be inferred.
- Computationally more difficult (query reformulation).

Global-as-view (Carnot, SIMS, TSIMMIS, Tambis, Observer, . . .):

- Whenever the source changes or a new one is added, the view needs to be reconsidered.
- Needs to understand the relationships between the sources.
- Query processing sometimes easy (unfolding), when the ontology is very simple. Otherwise it requires sophisticated query evaluation procedures.
Possible scenarios

• Empty ontology / very simple Ontology
  • Global-as-view

• Local-as-view

• Full Ontology / Integrity Constraints
  • Global-as-view

• Local-as-view
Possible scenarios

• Empty ontology / very simple Ontology
  • Global-as-view
    • The problem reduces to standard DB technology.
    • Can not express Ontology Integration needs.
    • Not modular.
  • Local-as-view

• Full Ontology / Integrity Constraints
  • Global-as-view

  • Local-as-view
Possible scenarios

• Empty ontology / very simple Ontology
  • Global-as-view
    • The problem reduces to standard DB technology.
    • Can not express Ontology Integration needs.
    • Not modular.
  • Local-as-view
    • “Standard” view-based query processing.
    • Can express only few Ontology Integration needs.
    • Modular.

• Full Ontology / Integrity Constraints
  • Global-as-view

  • Local-as-view
Possible scenarios

- Empty ontology / very simple Ontology
  - Global-as-view
    - The problem reduces to standard DB technology.
    - Can not express Ontology Integration needs.
    - Not modular.
  - Local-as-view
    - “Standard” view-based query processing.
    - Can express only few Ontology Integration needs.
    - Modular.

- Full Ontology / Integrity Constraints
  - Global-as-view
    - Requires sophisticated query evaluation procedures (involving deduction).
    - Can express Ontology Integration needs.
    - Not modular.
  - Local-as-view
Possible scenarios

- Empty ontology / very simple Ontology
  - Global-as-view
    - The problem reduces to standard DB technology.
    - Can not express Ontology Integration needs.
    - Not modular.
  - Local-as-view
    - "Standard" view-based query processing.
    - Can express only few Ontology Integration needs.
    - Modular.

- Full Ontology / Integrity Constraints
  - Global-as-view
    - Requires sophisticated query evaluation procedures (involving deduction).
    - Can express Ontology Integration needs.
    - Not modular.
  - Local-as-view
    - View-based query processing under constraints.
    - Can express Ontology Integration needs.
    - Modular.
Current (sad) Practice

- Most implemented Ontology Integration systems:
Current (sad) Practice

• Most implemented Ontology Integration systems:
  • either assume no Ontology or a very simple Ontology with a global-as-view approach,
Current (sad) Practice

- Most implemented Ontology Integration systems:
  - either assume no Ontology or a very simple Ontology with a global-as-view approach,
  - or include an Ontology or Integrity Constraints in their framework, but adopt a naive query evaluation procedure, based on query unfolding: no correctness of the query answering can be proved.
Conclusions

• All the things presented in this tutorial require heavy logical and technical machineries.

• Nonetheless, we believe that
  • it is feasible in practice,
  • it will lead to more usable information systems,
  • it is a lot of fun from the point of view of research.
iocom: Intelligent Conceptual Modelling tool

- iocom allows for the specification of multiple EER (or UML) diagrams and inter- and intra-schema constraints;

- Complete logical reasoning is employed by the tool using a hidden underlying DLR inference engine;

- iocom verifies the specification, infers implicit facts and stricter constraints, and manifests any inconsistencies during the conceptual modelling phase.

- [www.cs.man.ac.uk/~franconi/icom/](http://www.cs.man.ac.uk/~franconi/icom/)