Ontology-Based Data Integration

Diego Calvanese[1], Giuseppe De Giacomo[2]

[1] Free University of Bozen-Bolzano,

[2] Sapienza Università di Roma

Tutorial at Semantic Days
May 18, 2009
Stavanger, Norway

Outline

Outline

Outline

Information integration: relevance

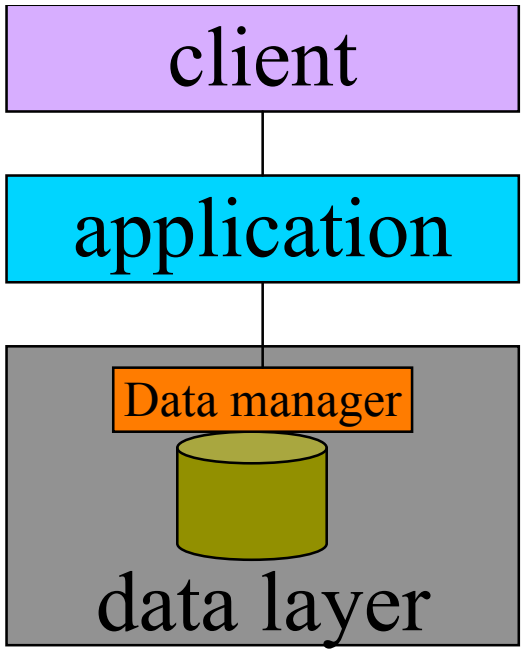
From [Bernstein & Haas, CACM Sept. 2008]:

- Large enterprises spend a great deal of time and money on information integration (e.g., 40% of information-technology shops' budget).
- Market for data integration software estimated to grow from $2.5 billion in 2007 to $3.8 billion in 2012 (+8.7% per year)
  [IDC. Worldwide Data Integration and Access Software 2008-2012 Forecast. Doc No. 211636 (Apr. 2008)]
- One of the major challenges for the future of IT.

At least two general contexts

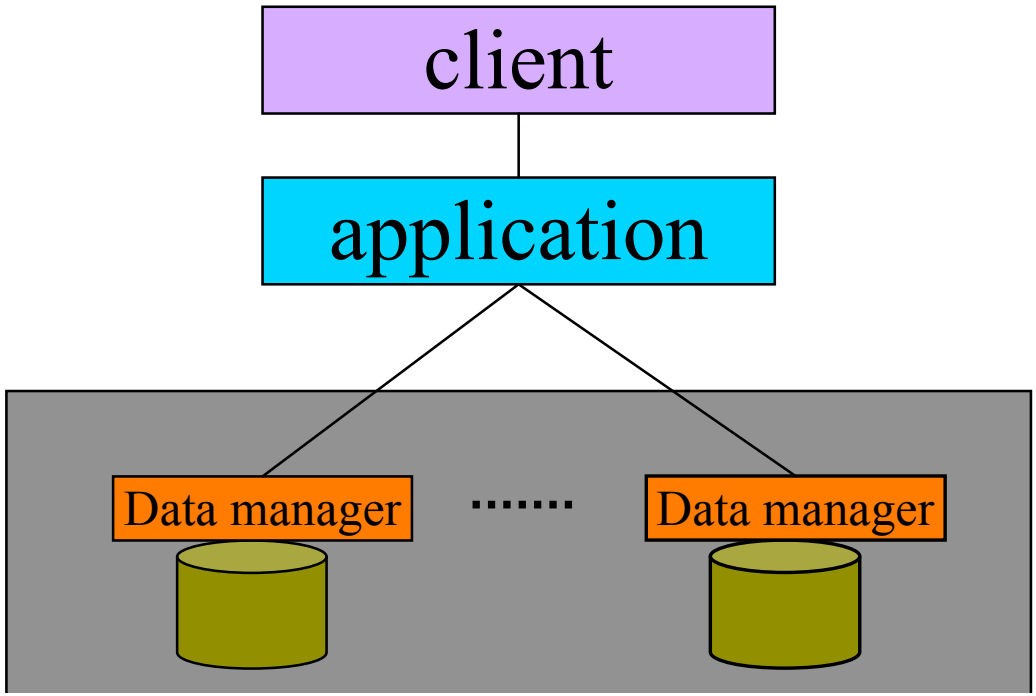
- Intra-organization information integration (e.g., EIS)
- Inter-organization information integration (e.g., integration on the Web)

Integration in data management: : evolution



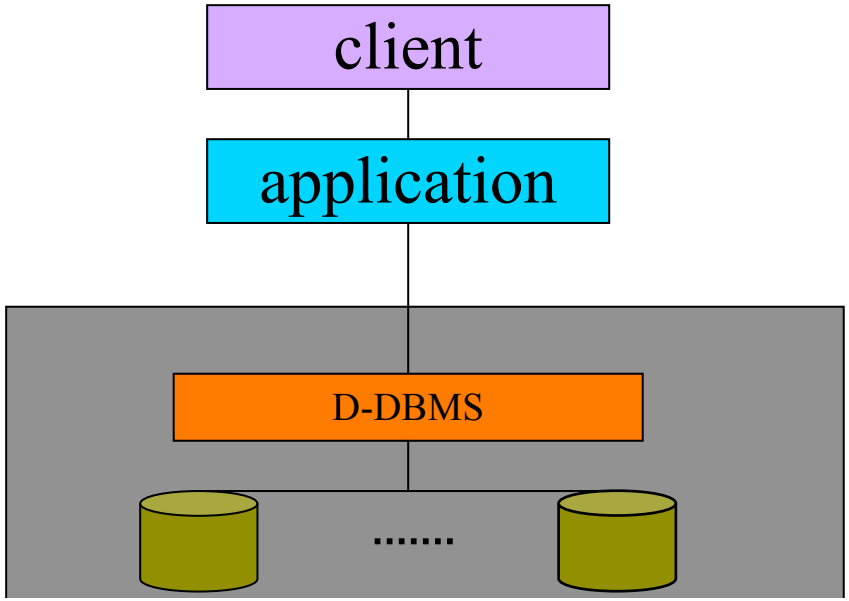
- Centralized system with three-tier architecture
- Implicit integration: integration supported by the Data Base Management System (DBMS), i.e., the data manager)

Integration in data management: evolution



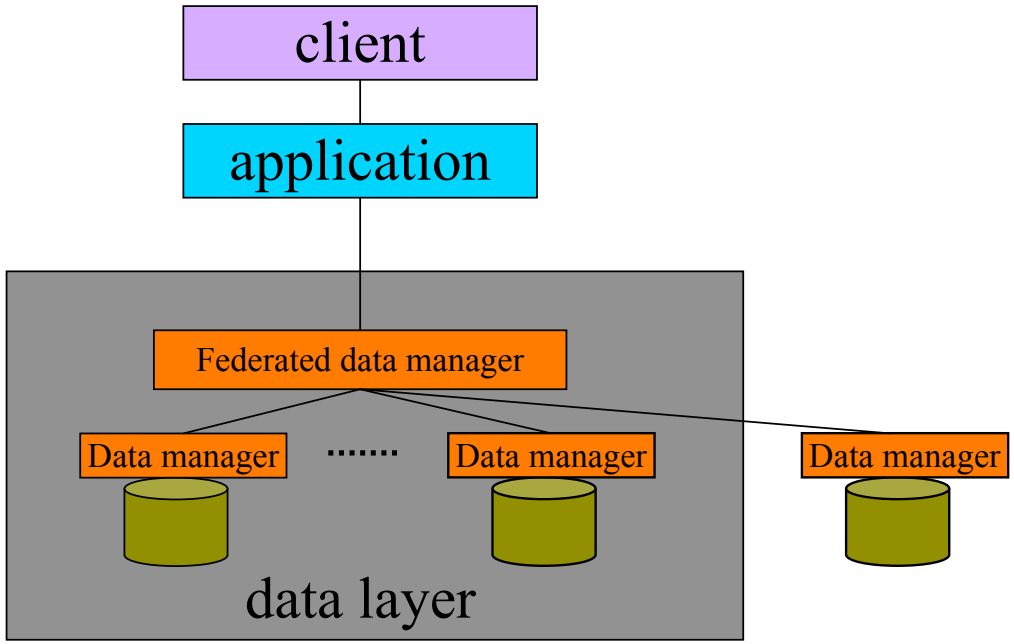
- Centralized system with three-tier architecture and multiple stores
- Application-hidden integration: integration "embedded" within application

Integration in data management: evolution



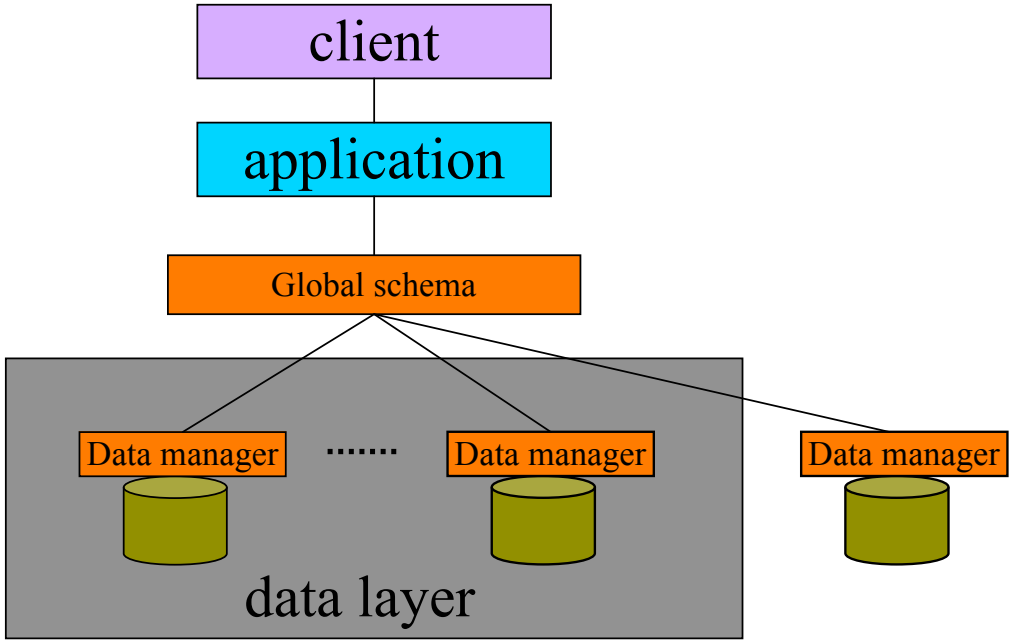
- Centralized system with three-tier architecture and multiple stores
- Distributed data management: different data sources of the same type, under the control of the organization, managed by a Distributed DBMS

Integration in data management: evolution



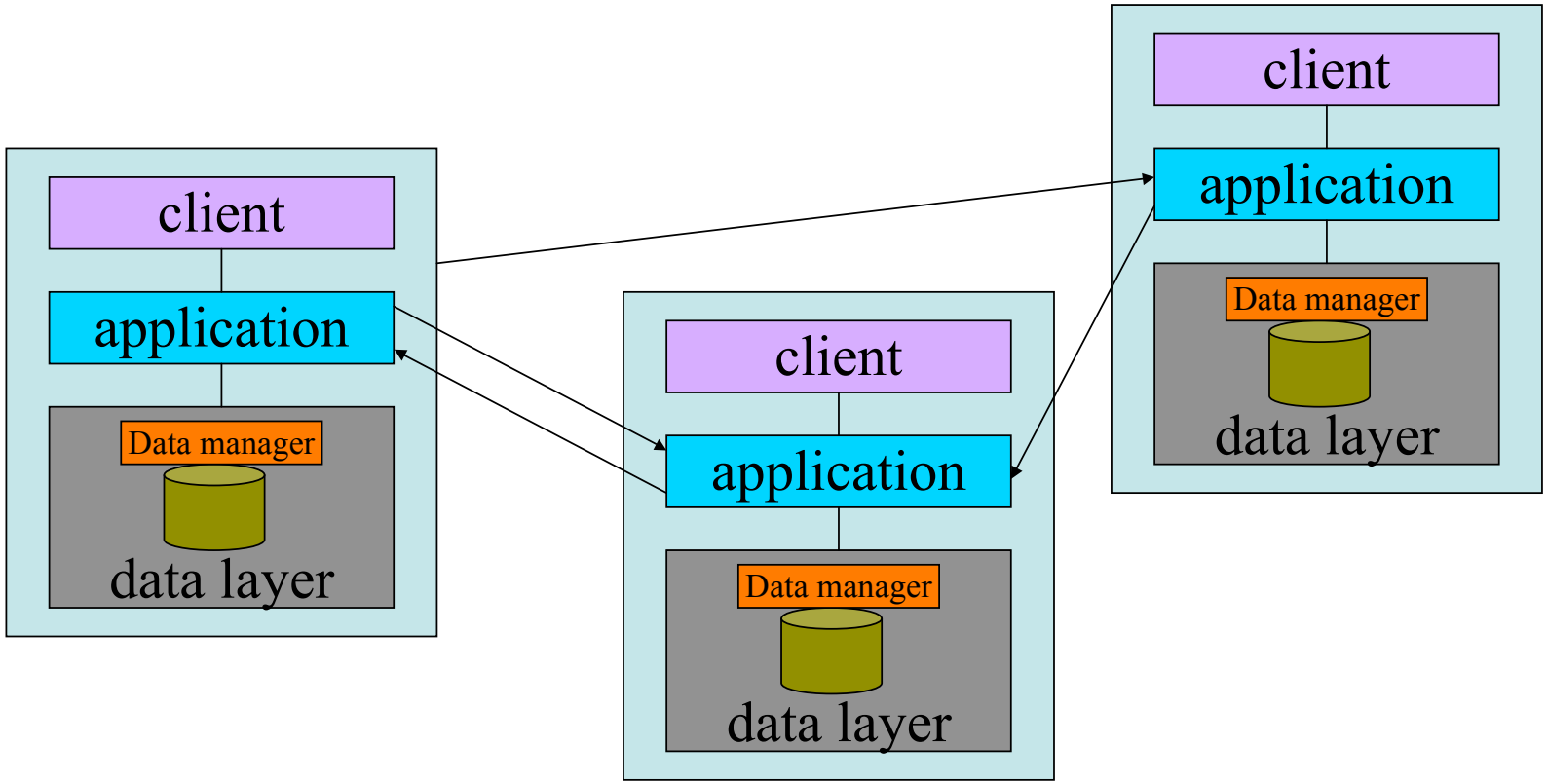
- Centralized system with three-tier architecture and distributed stores
- Data federation: different data sources, not necessarily of the same type, or under the control of the organization, federated within one data layer

Integration in data management: evolution



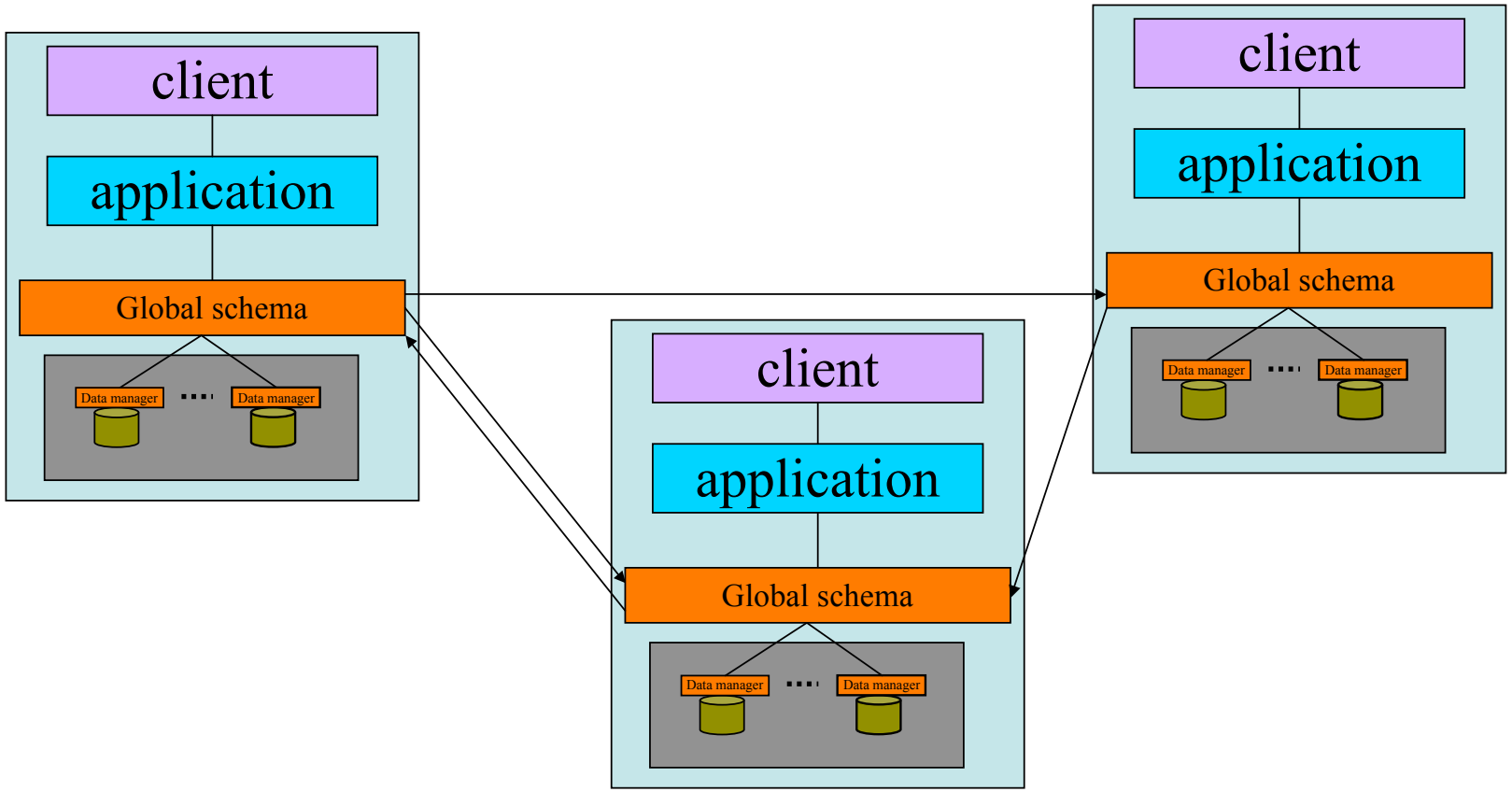
- Centralized system with four-tier architecture and distributed stores
- Data integration: the global schema is "independent" from the different data sources, which are heterogeneous, and not necessarily under the control of a single organization

Integration in data management: evolution



- Decentralized system
- Application-based distribution: distributed integration realized within application

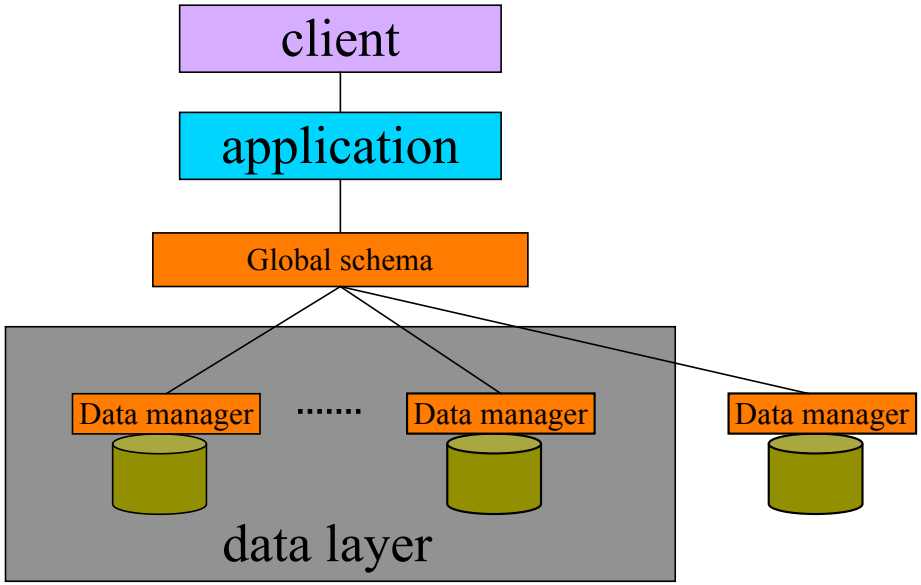
Integration in data management: evolution



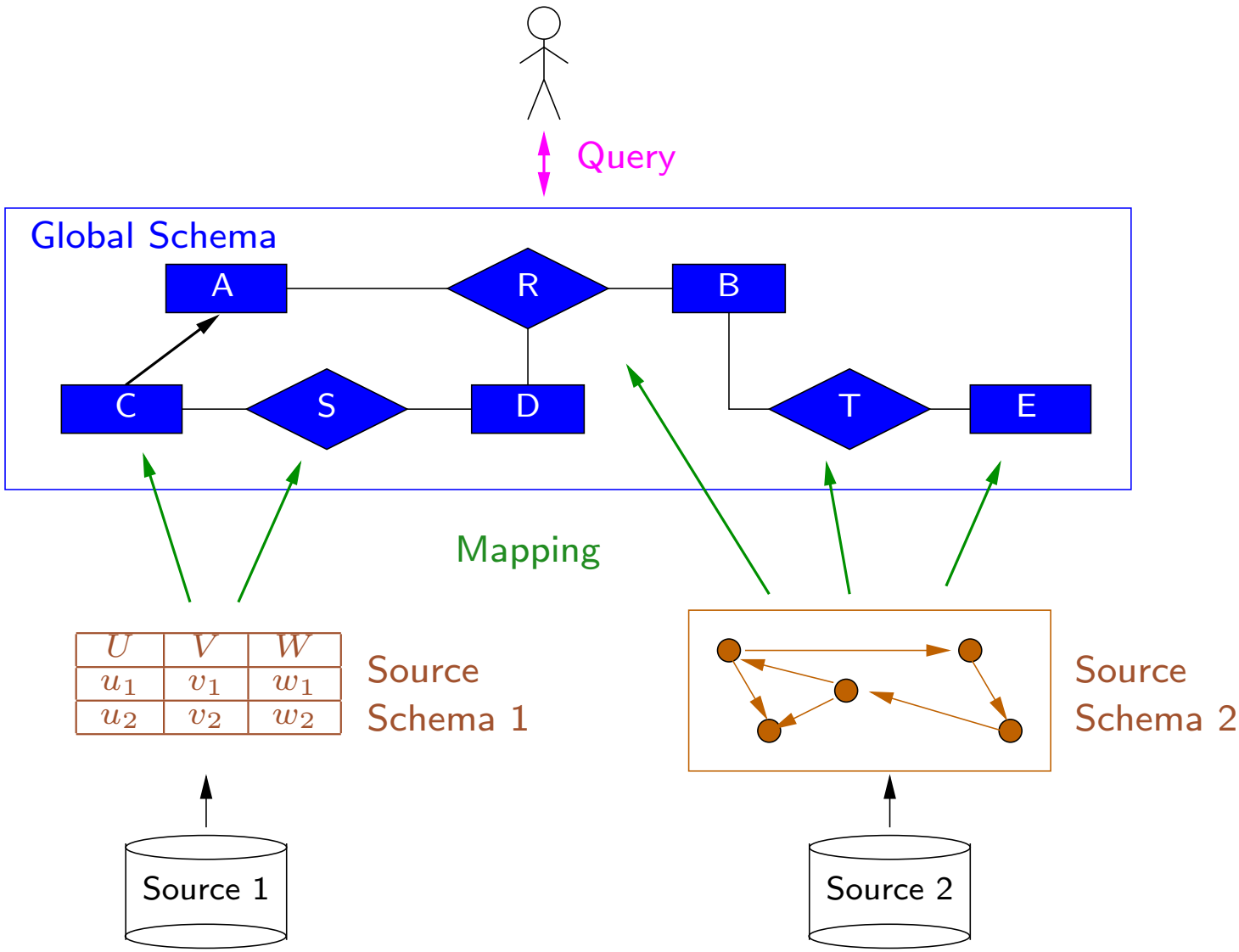
- Decentralized system
- Peer-to-peer data integration: distributed data integration realized with no central global schemas

What is information integration?

Information Integration is the problem of providing unified and transparent access, through a global (or target) schema to a collection of data stored in **multiple**, **autonomous**, and **heterogeneous** data sources



Conceptual architecture of information integration



Information integration: available industrial efforts

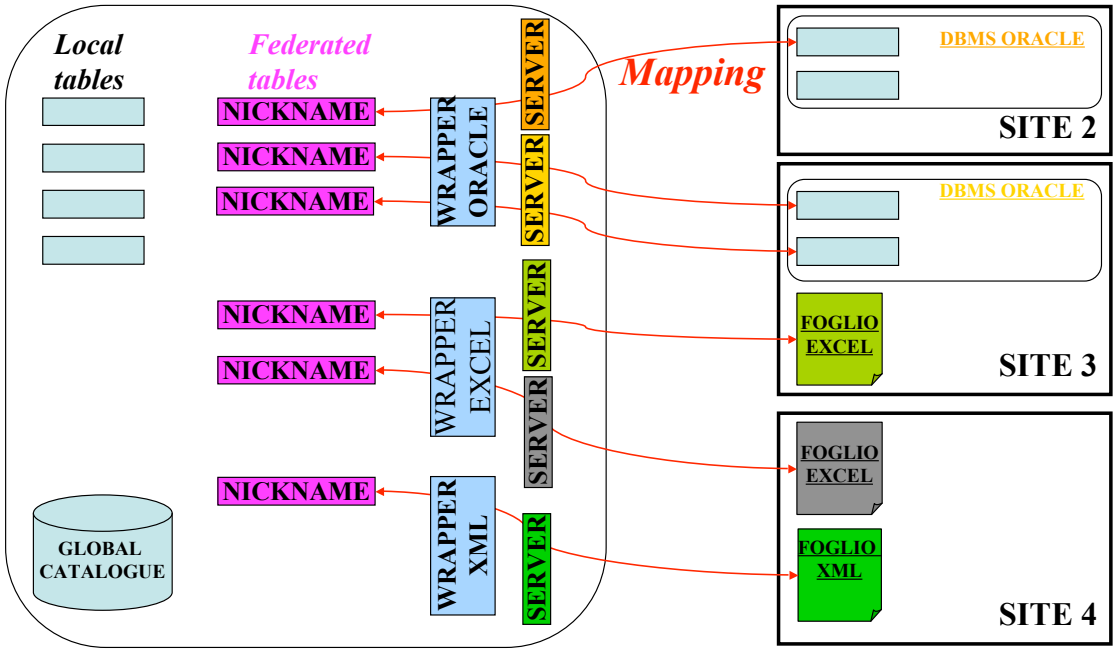
- Distributed database systems
- Tools for source wrapping
- Tools for ETL (Extraction, Transformation and Loading)
- Data warehousing
- Tools based on database federation, e.g., IBM Information Integrator
- Distributed query optimization

Current information integration tools: characteristics

- **Physical transparency**, i.e., masking from the user the physical characteristics of the sources
- **Heterogeinity**, i.e., federating highly diverse types of sources
- **Extensibility**
- **Autonomy** of data sources
- **Performance**, through distributed query optimization

However, current tools do not (directly) support the so-called **logical (or semantic) transparency**

IBM Information integrator



Nicknames are the "images" of the sources in the global schema. The "logical" structure of the sources (i.e., how data are stored in the sources) influence the global schema: logical transparency is hampered.

Logical transparency

Basic ingredients for achieving logical transparency:

- The global schema provides a **semantic view** of the information that is independent on how such information is stored at the sources
- The global schema is described with a **semantically rich formalism**
- The mappings which are also lifted to the **semantic level** are the crucial tools for realizing the independence of the global schema from the sources
- Obviously, it is crucial to choose the **right formalisms** for specifying
  - the global view
  - the mappings

All the above aspects are not appropriately dealt with by current tools.
This means that information integration cannot be simply addressed on a tool basis

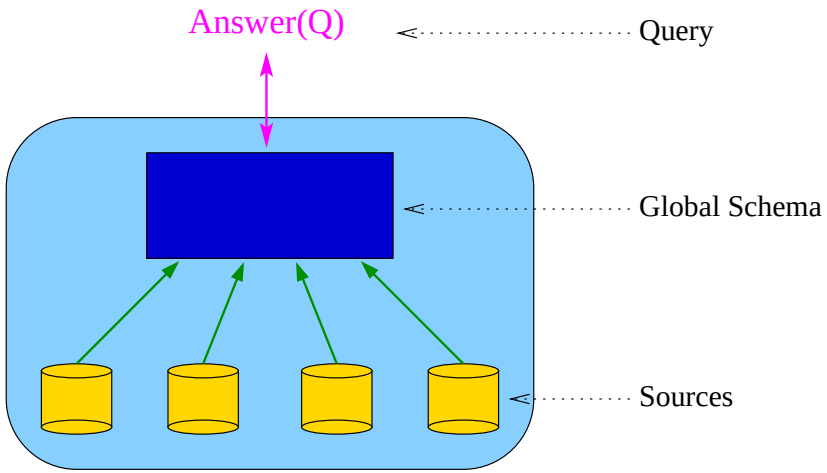
Outline

Approaches to information integration

- **(Mediator-based) data integration** *. . . is the focus of this tutorial*
  - Virtual: data remain at the sources
  - Main service provided is query answering: query the virtual global view, get answer from concrete sources
- Data exchange [Fagin & al. TCS'05, Kolaitis PODS'05]
  - materialization of the global view
  - allows for query answering without accessing the sources
- P2P data integration and exchange [Halevy & al. ICDE'03, Calvanese & al. PODS'04, Calvanese & al. DBPL'05, De Giacomo & al. PODS'07]
  - several peers
  - each peer with local and external sources
  - queries over one peer

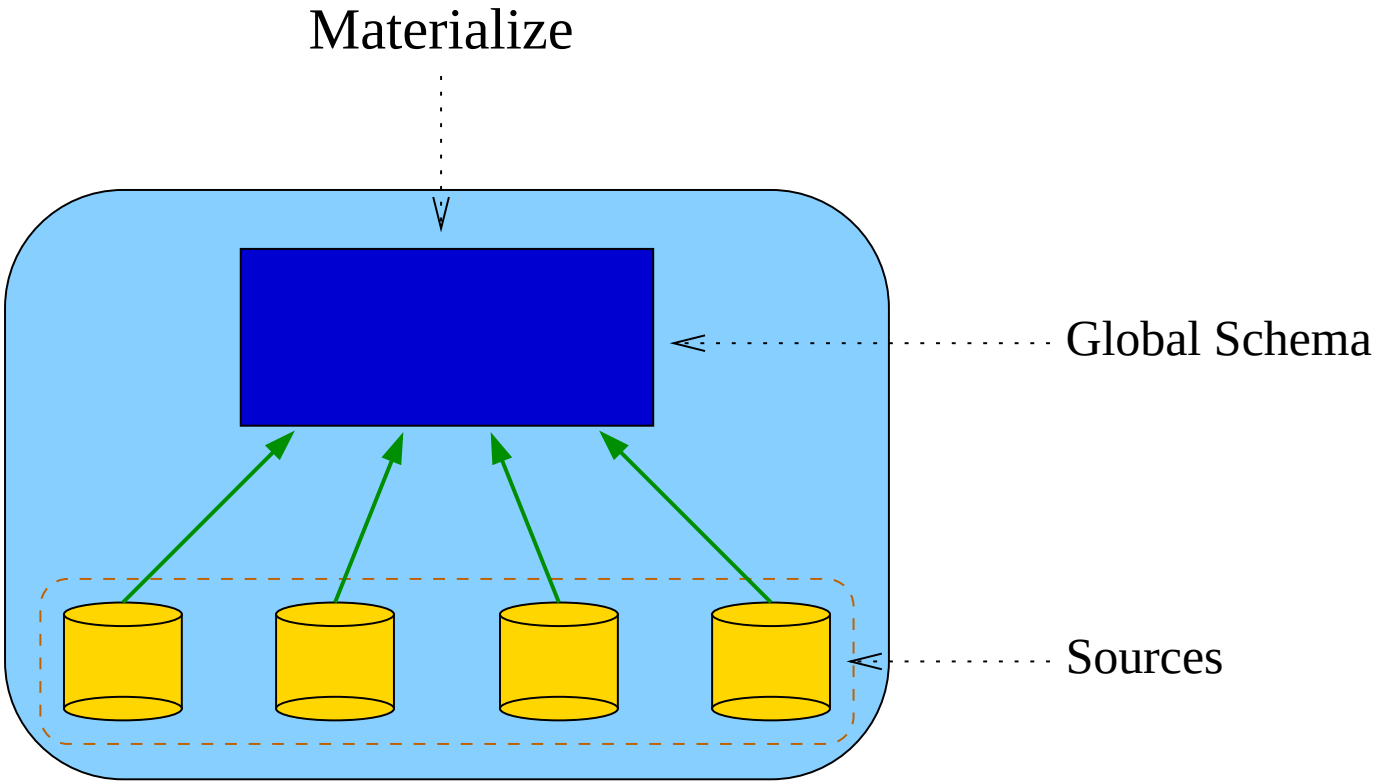
Mediator based data integration

- Queries are expressed over a **global schema** (a.k.a. mediated schema, enterprise model, . . .)
- Data are stored in a set of sources
- **Wrappers** access the sources (provide a view in a uniform data model of the data stored in the sources)
- **Mediators** combine answers coming from wrappers and/or other mediators

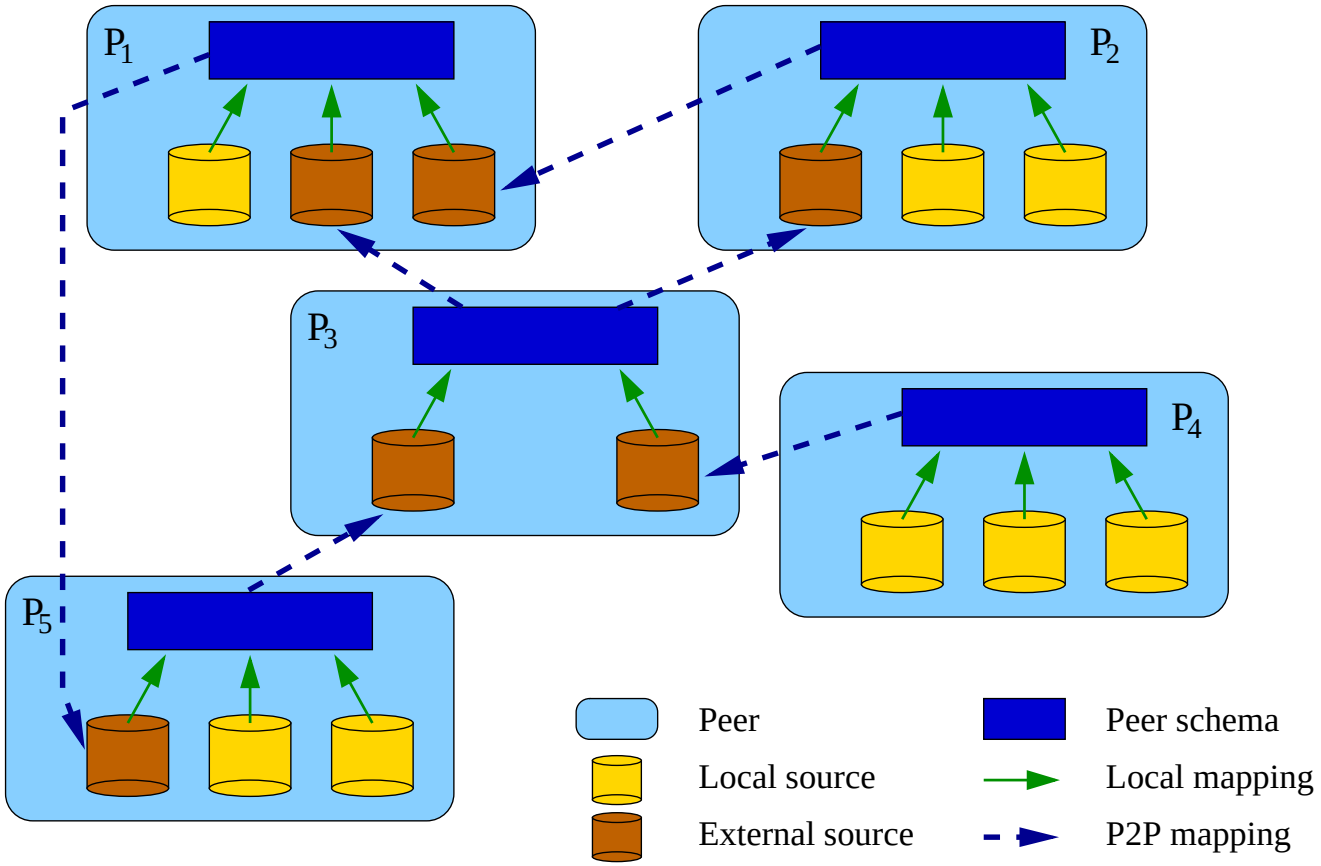


Data exchange

- Materialization of the global schema



Peer-to-peer data integration



Operations: – Answer(Q,P_i) – Materialize(P_i)

Outline

Main problems in information integration

1. How to construct the global schema
2. (Automatic) source wrapping
3. How to discover mappings between sources and global schema
4. Limitations in mechanisms for accessing sources
5. Data extraction, cleaning, and reconciliation
6. How to optimize query answering
7. **How to model the global schema, the sources, and the mappings between the two**
8. **How to answer queries expressed on the global schema**
9. How to process updates expressed on the global schema and/or the sources ("read/write" vs. "read-only" data integration)

The modeling problem

Basic questions:

- How to model the global schema
  - capture the information of interest from the **semantic point of view**
  - allow for **intensional information** (e.g., constraints)
  - deal with **incomplete information**
- How to model the sources
  - data model as provided by the wrapping
  - access limitations
  - data values (common vs. different domains)
- How to model the mapping between global schemas and sources
  - from a **semantic point of view**
  - from an interoperation point of view
- How to verify the quality of the modeling process

A word of caution: Data modeling (in data integration) is an art. Theoretical frameworks can help humans, not replace them

The querying problem

- A query expressed in terms of the global schema must be **reformulated** in terms of (a set of) queries over the sources and/or materialized views
- The computed sub-queries are shipped to the sources, and the results are collected and **assembled** into the final answer
- The computed query plan should guarantee
  - completeness of the obtained answers wrt the semantics
  - efficiency of the whole query answering process
  - efficiency in accessing sources
- This process heavily depends on the approach adopted for modeling the data integration system

This is one of the main problems that we want to address in this tutorial

Outline

Global view: ontologies

The global view ought to provide a description of the data of interest in **semantic terms** ...

- Represent the global view as a **conceptual schema** as those used at design time to design a database, but ...
- ... allow for using it at **runtime** as the actual schema over which queries are asked!

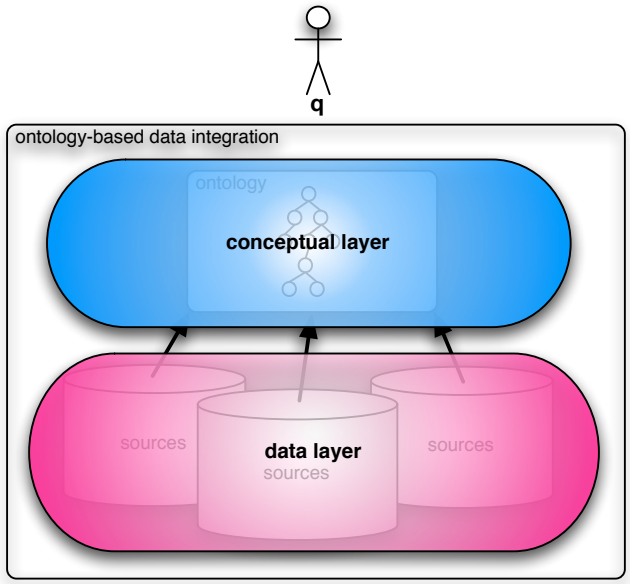
How can we do this?

- Formalize the conceptual schema in logic (e.g., FOL or a Description Logic – see later)
- Use reasoning for query answering.

We call **ontology** the artifact that is the runtime counterpart of the conceptual schema.

Ontology-based data integration: conceptual layer & data layer

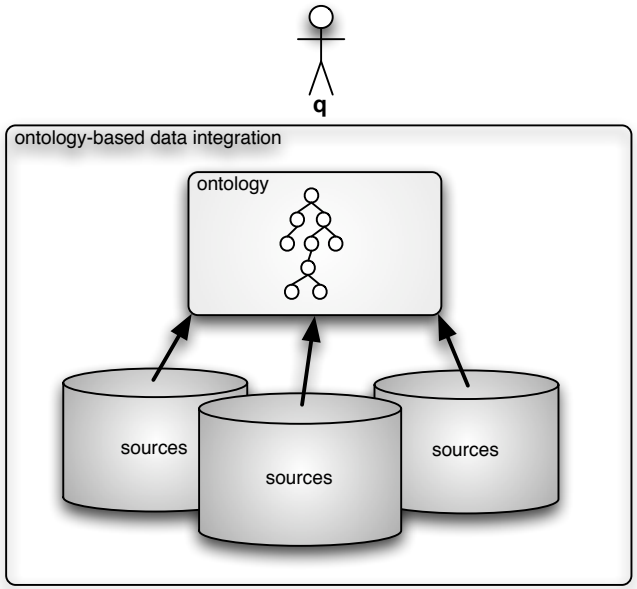
Ontology-based data integration is based on the idea of decoupling information semantics from data storage.



Clients access only the **conceptual layer** ... while the **data layer**, hidden to clients, manages the data.

$\rightsquigarrow$ *Technological concerns (and changes) on the managed data become fully transparent to the clients.*

Ontology-based data integration: architecture

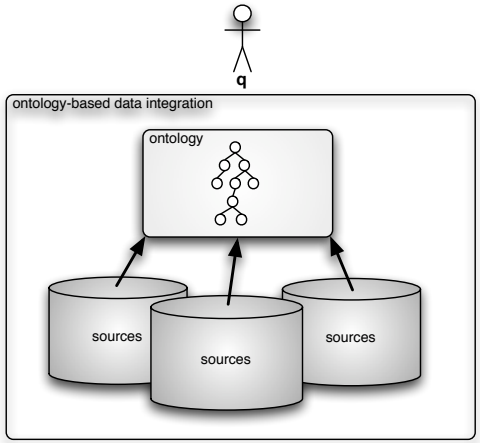


Based on three main components:

- **Ontology**, used as the conceptual layer to give clients a unified conceptual "global view" of the data.
- **Data sources**, these are external, independent, heterogeneous, multiple information systems.
- **Mappings**, which semantically link data at the sources with the ontology.

Ontology-based data integration: the conceptual layer

The ontology is used as the conceptual layer, to give clients a unified conceptual global view of the data.

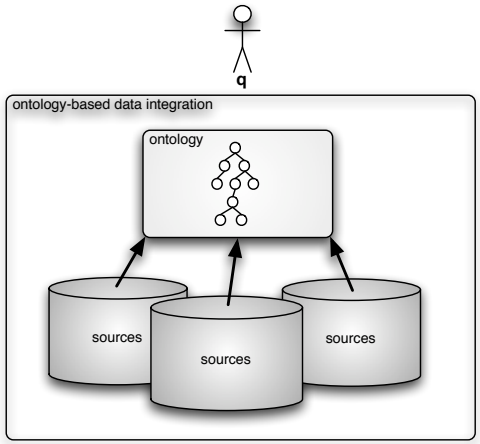


Note: in standard information systems, UML Class Diagram or ER is used at **design time**, ...

... here we use ontologies at **runtime**!

Ontology-based data integration: the sources

Data sources are external, independent, heterogeneous, multiple information systems.

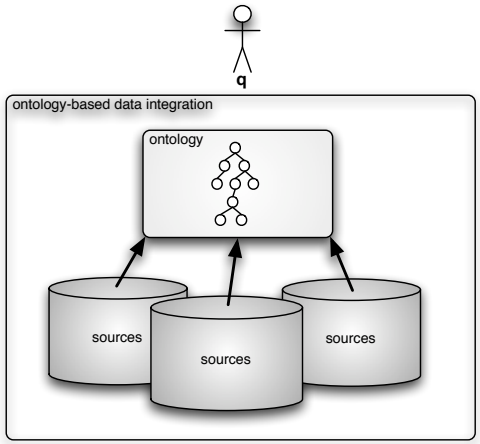


By now we have industrial solutions for:

- Distributed database systems
- Distributed query optimization
- Tools for source wrapping
- Systems for database federation, e.g., IBM Information Integrator
  but notice no open-source federated databases yet!

Ontology-based data integration: the sources

Data sources are external, independent, heterogeneous, multiple information systems.



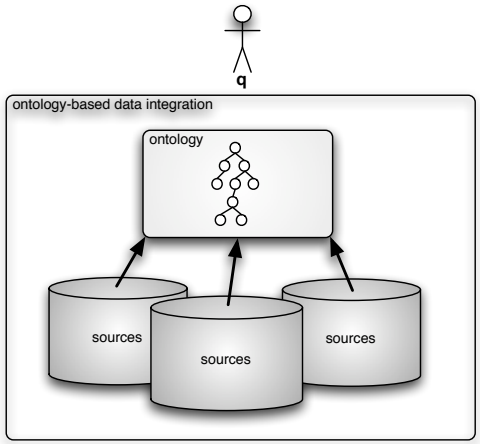
Based on these industrial solutions we can:

1. Wrap the sources and see all of them as relational databases.
2. Use federated database tools to see the multiple sources as a single one.

$\rightsquigarrow$ We can see the sources as a single (remote) relational database.

Ontology-based data integration: mappings

Mappings semantically link data at the sources with the ontology.



Scientific literature on data integration in databases has shown that ...

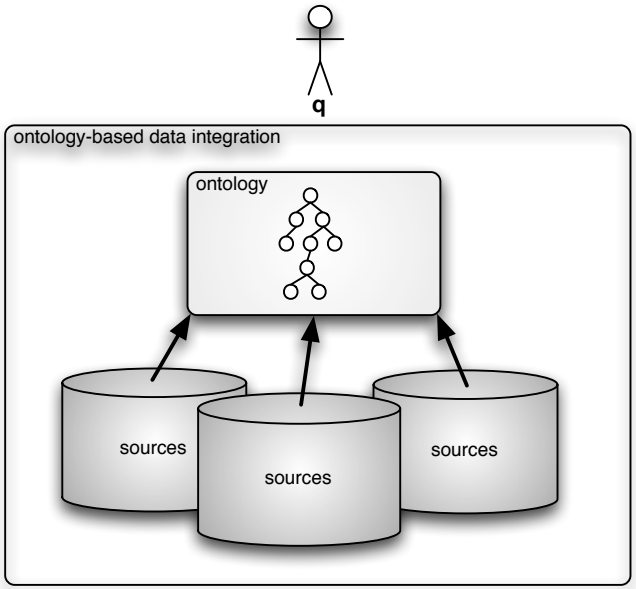
... generally we cannot simply **map** single relations to single elements of the global view (the ontology) ...

... we need to rely on **queries**!
Several general forms of mappings based on queries have been considered:

- GAV: map a query over the source to an element in the global view
  – *most used form of mappings*

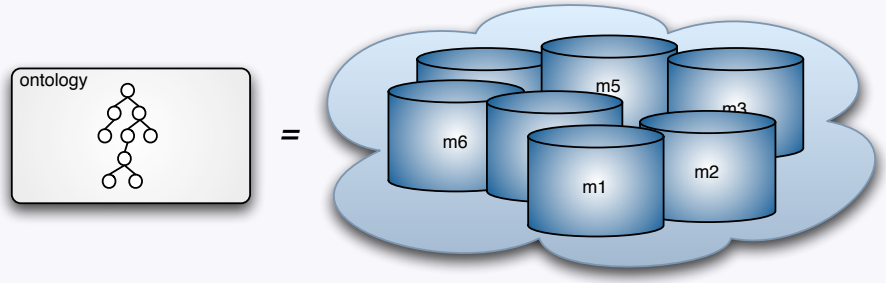
Ontology-based data integration: incomplete information

It is assumed, even in standard data integration, that the information that the global view has on the data is incomplete!



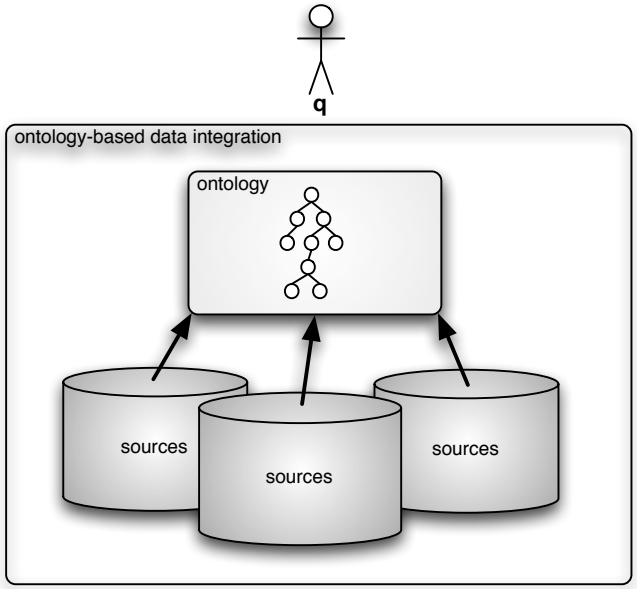
Important

Ontologies are logical theories $\rightsquigarrow$ they are perfectly suited to deal with **incomplete information**!



- Query answering amounts to compute **certain answers**, given the global view, the mapping and the data at the sources ...
- ... but query answering may be costly in ontologies (even without mapping and sources).

Ontology-based data integration: the *DL-Lite*$_{\mathcal{A}}$ solution



- We require the data sources to be **wrapped** and presented as relational sources. $\rightsquigarrow$ *"standard technology"*
- We make use of a **data federation tool**, such as IBM Information Integrator, to present the yet to be (semantically) integrated sources as a single relational database. $\rightsquigarrow$ *"standard technology"*
- We make use of the **DL-Lite**$_{\mathcal{A}}$ technology presented in the following for the conceptual view on the data, to **exploit effectiveness of query answering**. *"new technology"*

Outline

Outline

Modeling the global view

The global view ought to provide a description of the data of interest in **semantic terms** ...

Programme:

- Represent the global view as a **conceptual schema** as those used at design time to design a database.
- Formalize the conceptual schema as **logical theory**, namely the **ontology**.
- Use the resulting logical theory for **reasoning** and **query answering**.

Let's start with an exercise ...

Requirements: We are interested in building a software application to manage filmed scenes for realizing a movie, by following the so-called "Hollywood Approach".

Every **scene** is identified by a code (a string) and it is described by a text in natural language.

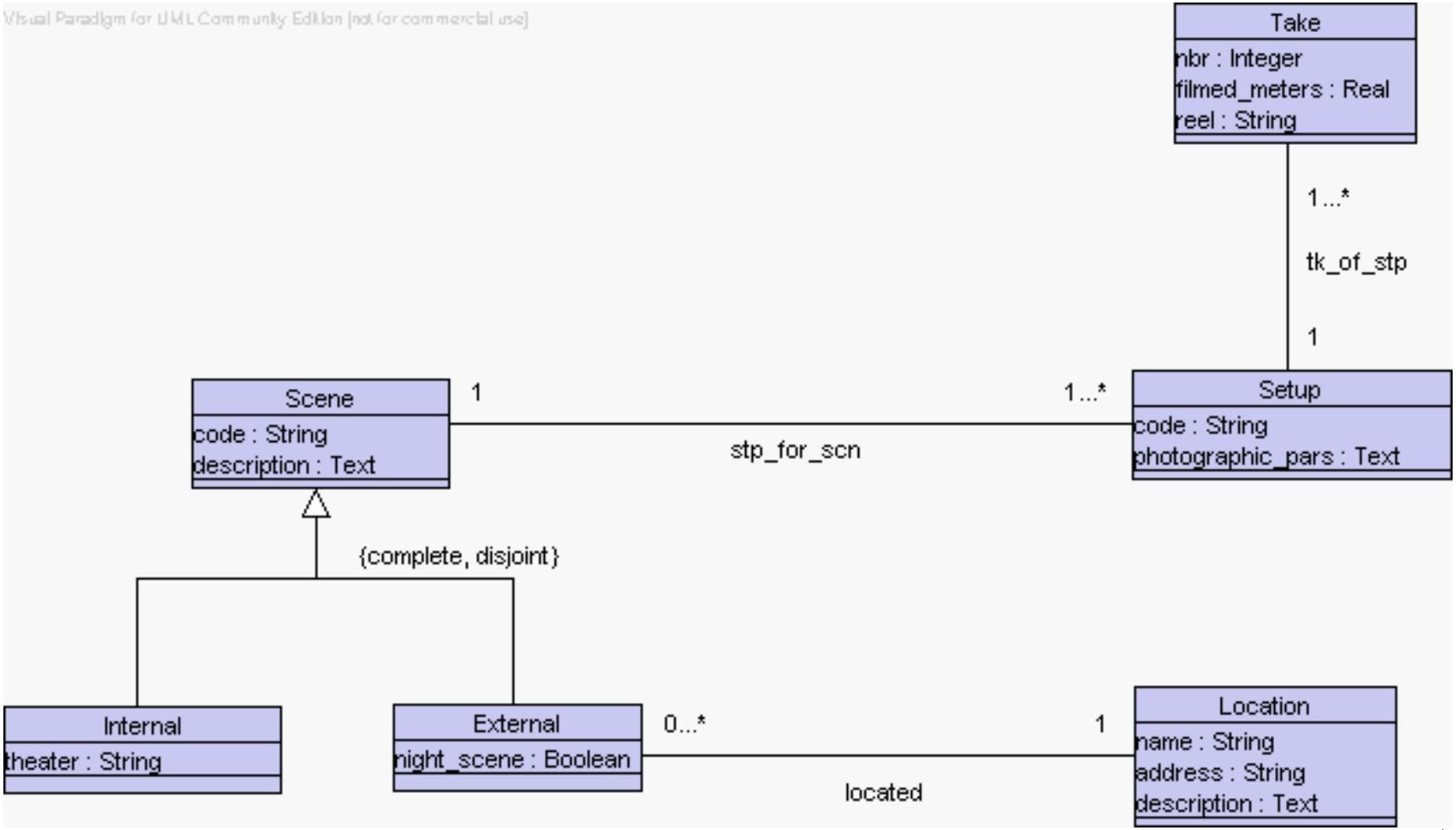
Every scene is filmed from different positions (at least one), each of this is called a **setup**. Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related to a single scene.

For every setup, several **takes** may be filmed (at least one). Every take is characterized by a (positive) natural number, a real number representing the number of meters of film that have been used for shooting the take, and the code (a string) of the reel where the film is stored. Note that a take is associated to a single setup.

Scenes are divided into **internals** that are filmed in a theater, and **externals** that are filmed in a **location** and can either be "day scene" or "night scene". Locations are characterized by a code (a string) and the address of the location, and a text describing them in natural language.

Write a precise specification of this domain using any formalism you like.

Solution 1: ... use conceptual modeling diagrams (UML)!!!



Solution 1: ... use conceptual modeling diagrams (discussion)

Good points:

- Easy to generate (it's the standard in software design)
- Easy to understand for humans
- Well disciplined, well-established methodologies available

Bad points:

- No precise semantics (people that use it wave hands about it)
- Verification (or better validation) done informally by humans
- Machine incomprehensible (because of lack of formal semantics)
- Automated reasoning and query answering out of question
- Limited expressiveness*

**Not really a bad point, in fact.*

Solution 2: ... use logic!!!

Alphabet:

$Scene(x), Setup(x), Take(x), Internal(x), External(x), Location(x), stp\_for\_scn(x, y), ck\_of\_stp(x, y), located($

Axioms:

$$\forall x, y.\, code_{Scene}(x, y) \rightarrow Scene(x) \wedge String(y)$$
$$\forall x, y.\, description(x, y) \rightarrow Scene(x) \wedge Text(y)$$

$$\forall x, y.\, code_{Setup}(x, y) \rightarrow Setup(x) \wedge String(y)$$
$$\forall x, y.\, photographic\_pars(x, y) \rightarrow Setup(x) \wedge Text(y)$$

$$\forall x, y.\, nbr(x, y) \rightarrow Take(x) \wedge Integer(y)$$
$$\forall x, y.\, filmed\_meters(x, y) \rightarrow Take(x) \wedge Real(y)$$
$$\forall x, y.\, reel(x, y) \rightarrow Take(x) \wedge String(y)$$

$$\forall x, y.\, theater(x, y) \rightarrow Internal(x) \wedge String(y)$$

$$\forall x, y.\, night\_scene(x, y) \rightarrow External(x) \wedge Boolean(y)$$

$$\forall x, y.\, name(x, y) \rightarrow Location(x) \wedge String(y)$$
$$\forall x, y.\, address(x, y) \rightarrow Location(x) \wedge String(y)$$
$$\forall x, y.\, description(x, y) \rightarrow Location(x) \wedge Text(y)$$

$$\forall x.\, Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$$
$$\ldots$$

$$\forall x, y.\, stp\_for\_scn(x, y) \rightarrow Setup(x) \wedge Scene(y)$$
$$\forall x, y.\, tk\_of\_stp(x, y) \rightarrow Take(x) \wedge Setup(y)$$
$$\forall x, y.\, located(x, y) \rightarrow External(x) \wedge Location(y)$$

$$\forall x.\, Setup(x) \rightarrow 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$$
$$\forall y.\, Scene(y) \rightarrow 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$$
$$\forall x.\, Take(x) \rightarrow 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$$
$$\forall x.\, Setup(y) \rightarrow 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$$
$$\forall x.\, External(x) \rightarrow 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$$

$$\forall x.\, Internal(x) \rightarrow Scene(x)$$
$$\forall x.\, External(x) \rightarrow Scene(x)$$
$$\forall x.\, Internal(x) \rightarrow \neg External(x)$$
$$\forall x.\, Scene(x) \rightarrow Internal(x) \vee External(x)$$

Solution 2: ... use logic (discussion)

Good points:

- Precise semantics
- Formal verification
- Allows for query answering
- Machine comprehensible
- Virtually unlimited expressiveness*

Bad points:

- Difficult to generate
- Difficult to understand for humans
- Too unstructured (making reasoning difficult), no well-established methodologies available
- Automated reasoning may be impossible

**Not really a good point, in fact.*

Solution 3: ... use both!!!

Note these two approaches look as being orthogonal, but they can in fact be used together cooperatively!!!
Basic idea:

- Assign formal semantics to constructs of the conceptual design diagrams
- Use conceptual design diagrams as usual, taking advantage of methodologies developed for them in Software Engineering
- Read diagrams as logical theories when needed, i.e., for formal understanding, verification, automated reasoning, etc.

Added values:

- inherit from conceptual modeling diagrams: ease-to-use for humans
- inherit from logic: formal semantics and reasoning tasks, which are needed for formal verification and machine manipulation

Solution 3:... use both!!! (cont.)

Important: by using conceptual modeling diagrams one gets logical theories of a specific form.

- One gets limited (or better, well-disciplined) expressiveness
- One can exploit the particular form of the corresponding logical theory to simplify reasoning, hopefully getting:
  - decidability
  - reasoning procedures that match intrinsic computational complexity

In this part of the tutorial ...

We will illustrate what we get from interpreting conceptual modeling diagrams in logic.
We will use ...

- as conceptual modeling diagrams:
  - UML Class Diagrams*
- as logic:
  - First-Order Logic to formally capture semantics and reasoning.

**Note, we could equivalently use Entity-Relationship Diagrams instead of UML.*

Outline

Unified Modeling Language (UML)

UML stands for Unified Modeling Language. It was developed in 1994 by unifying and integrating the most prominent object-oriented modeling approaches of that age:

- Booch
- Rumbaugh: Object Modeling Technique (OMT)
- Jacobson: Object-Oriented Software Engineering (OOSE)

History:

- 1995, version 0.8, Booch, Rumbaugh; 1996, version 0.9, Booch, Rumbaugh, Jacobson; version 1.0 BRJ + Digital, IBM, HP, ...
- Best known version: 1.2–1.5 (1999–2004)
- Current version: 2.0 (2005)
- 1999/today: de facto standard object-oriented modeling language

References:

- Grady Booch, James Rumbaugh, Ivar Jacobson, "The unified modeling language user guide", Addison Wesley, 1999 (second edition, 2005)
- `http://www.omg.org` → UML
- `http://www.uml.org`

UML Class Diagrams

In this tutorial we deal with one of the most prominent components of UML: UML Class Diagrams.

A UML Class Diagram is used to represent explicitly the information on a domain of interest (typically the application domain of a software).

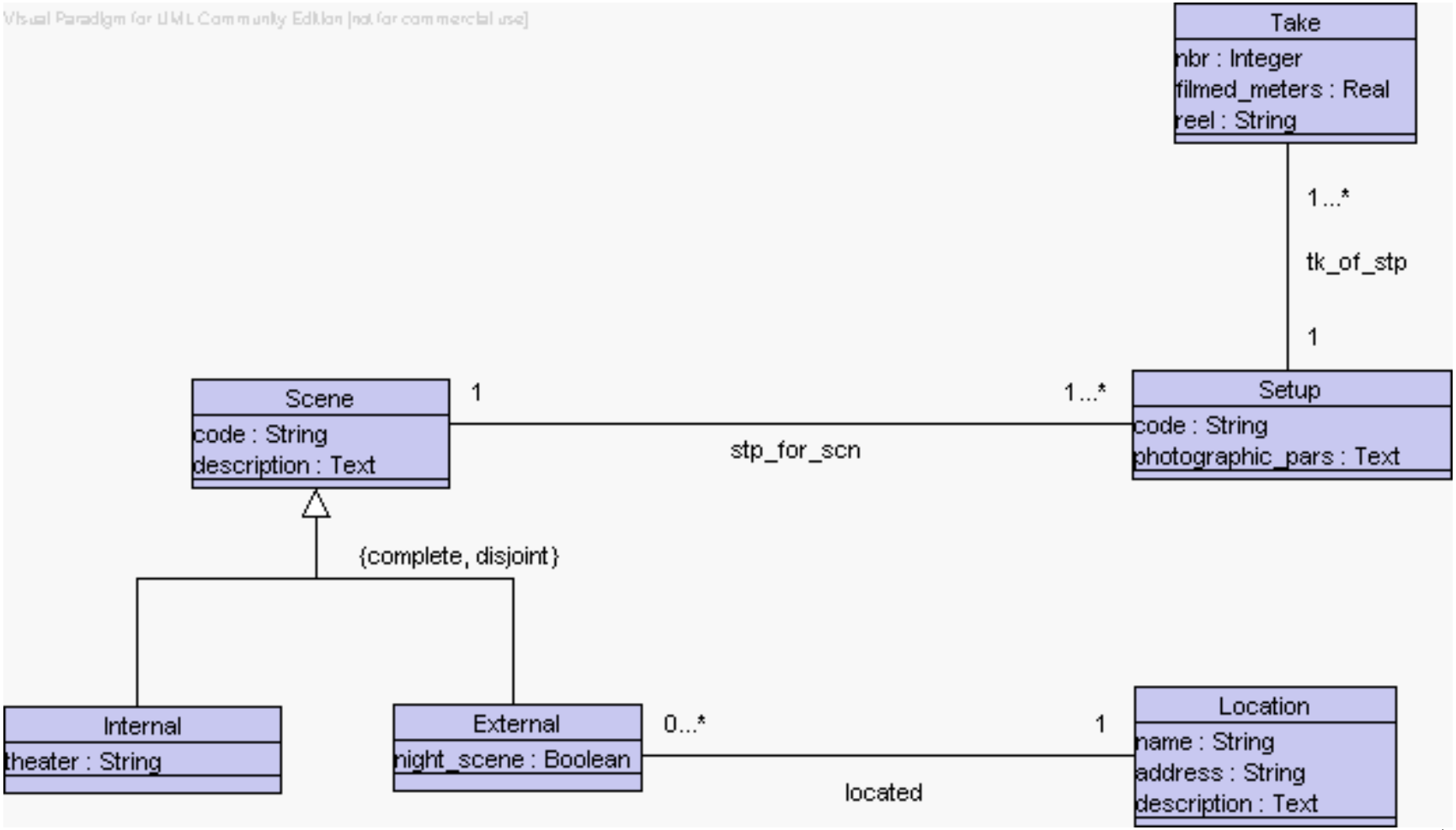
Note: This is exactly the goal of all conceptual modeling formalism, such as the Entity-Relationship Schemas (standard in Database design) or Ontologies (now in vogue due to the Semantic Web – see later) .

UML Class Diagrams (cont.1)

The UML class diagram models the domain of interest in terms of:

- objects grouped into classes
- relationships (associations) between classes
- simple properties of classes: "attributes" (we do not deal with "operations")
- sub-classing i.e., ISA/Generalization relationships

Example of an UML Class Diagram



UML Class Diagrams (cont.2)

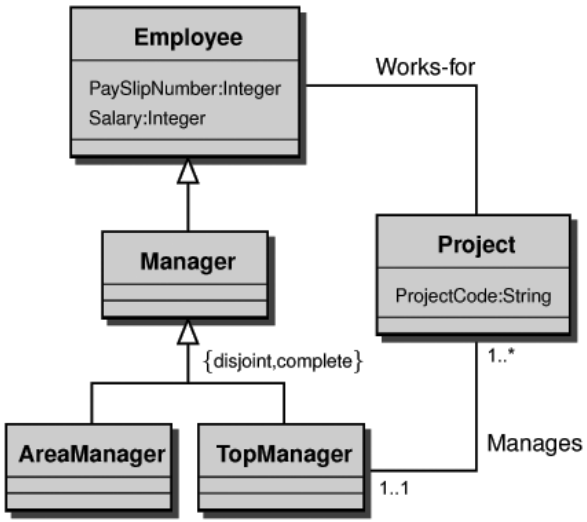
In fact UML class diagrams are used in various phase of a software design:

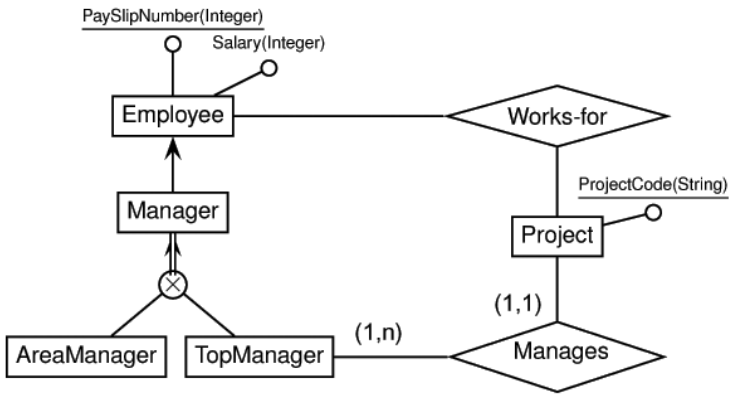
1. during the so-called analysis, where an abstract precise view of the domain of interest needs to be developed – the so-call "conceptual perspective"
2. during software development to maintain an abstract view of the software to be developed – the so-called "implementation perspective"

In this course we focus on 1!

UML Class Diagrams and ER Schemas

UML class diagrams are heavily influenced by Entity-Relationship Schemas.
Example of UML vs. ER:





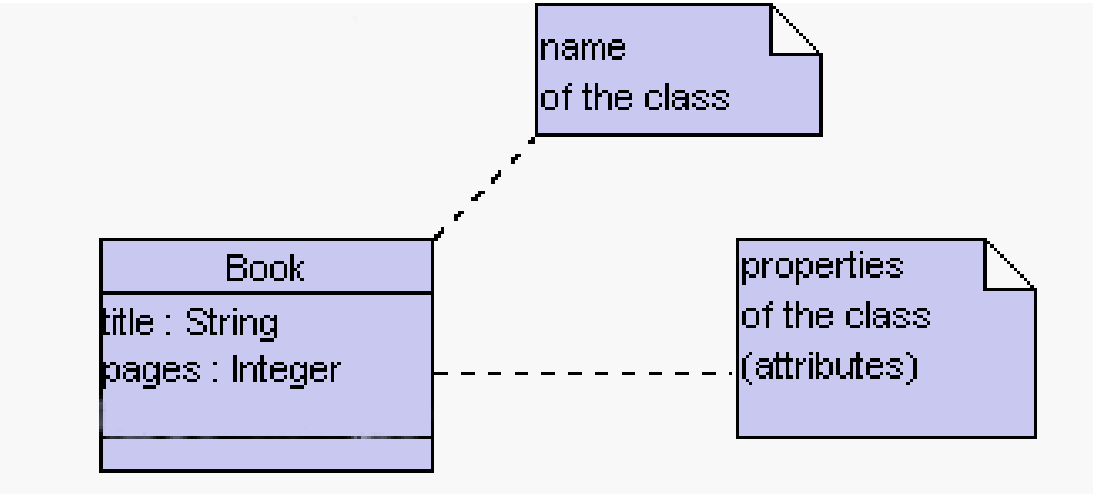
Classes in UML

A class in UML models a set of objects (its "instances") that share certain common properties: attributes, operations, etc.

Each class is characterized by:

- a name (which must be unique in the whole class diagram)
- a set of (local) properties, namely attributes and operations (see later).

Example:



Classes in UML: instances

The objects that belong to a class are called instances of the class. They form a so-called instantiation (or extension) of the class.

Example:

Here are some possible instantiations for our class $Book$.

$$\{book_1, book_2, book_3, \ldots\}$$

$$\{book_\alpha, book_\beta, book_\gamma, \ldots\}$$

Which is the actual instantiation? *We will know it only at run-time!!! – we are now at design time!*

Classes in UML: semantics

A class represent set of objects ... but which set? We don't actually know.

So, how can we assign such a semantics to a class?

Use a FOL unary predicate!!!

Example:
For our class $Book$, we introduce a predicate $Book(x)$.

Associations

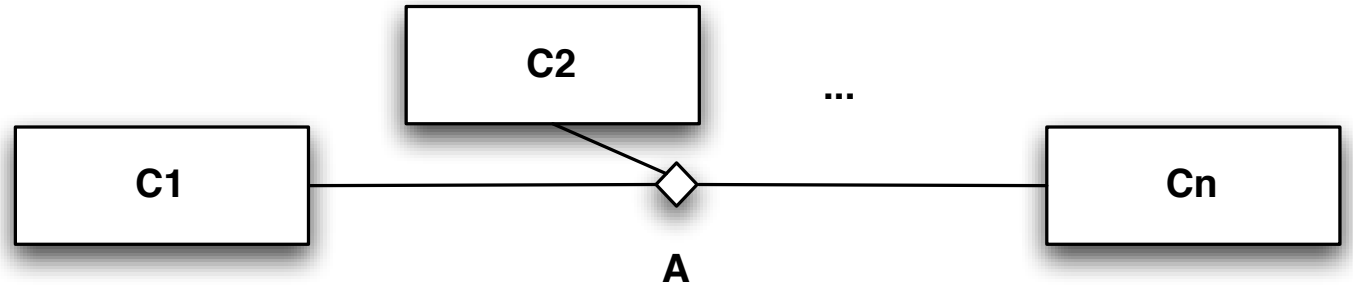
Relationships between classes are modeled in UML Class Diagrams as Associations.

An association in UML is a relation between the instances of two or more classes.

Association model properties of classes that are non-local, in the sense that they involve other classes.
An association between two classes is a property of both classes.

Example:

Associations: formalization



We can represent an n-ary association A among classes $C_1, \ldots, C_n$ as n-ary predicate A in FOL.
We assert that the components of the predicate must belong to correct classes:

$$\forall x_1, \ldots, x_n.\ A(x_1, \ldots, x_n) \rightarrow C_1(x_1) \wedge \ldots \wedge C_n(x_n)$$

Example:

$$\forall x_1, x_2.\ written\_by(x_1, x_2) \rightarrow Book(x_1) \wedge Author(x_2)$$

Associations: multiplicity

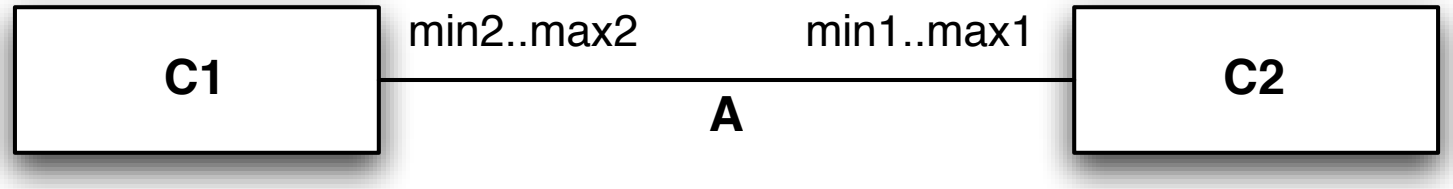
On binary associations we can place multiplicity constrains, i.e., a minimal and maximal number of tuples in which every object partecipates as first (second) component:

Example:

Note: UML multiplicities for associations are look-across and are not easy to use in an intuitive way for n-ary associations, so typically they are not used at all.

In contrast, in ER Schemas, multiplicities are not look-across and are easy to use, and widely used.

Associations: formalization (cont.)



Multiplicities of binary associations are easily expressible in FOL:

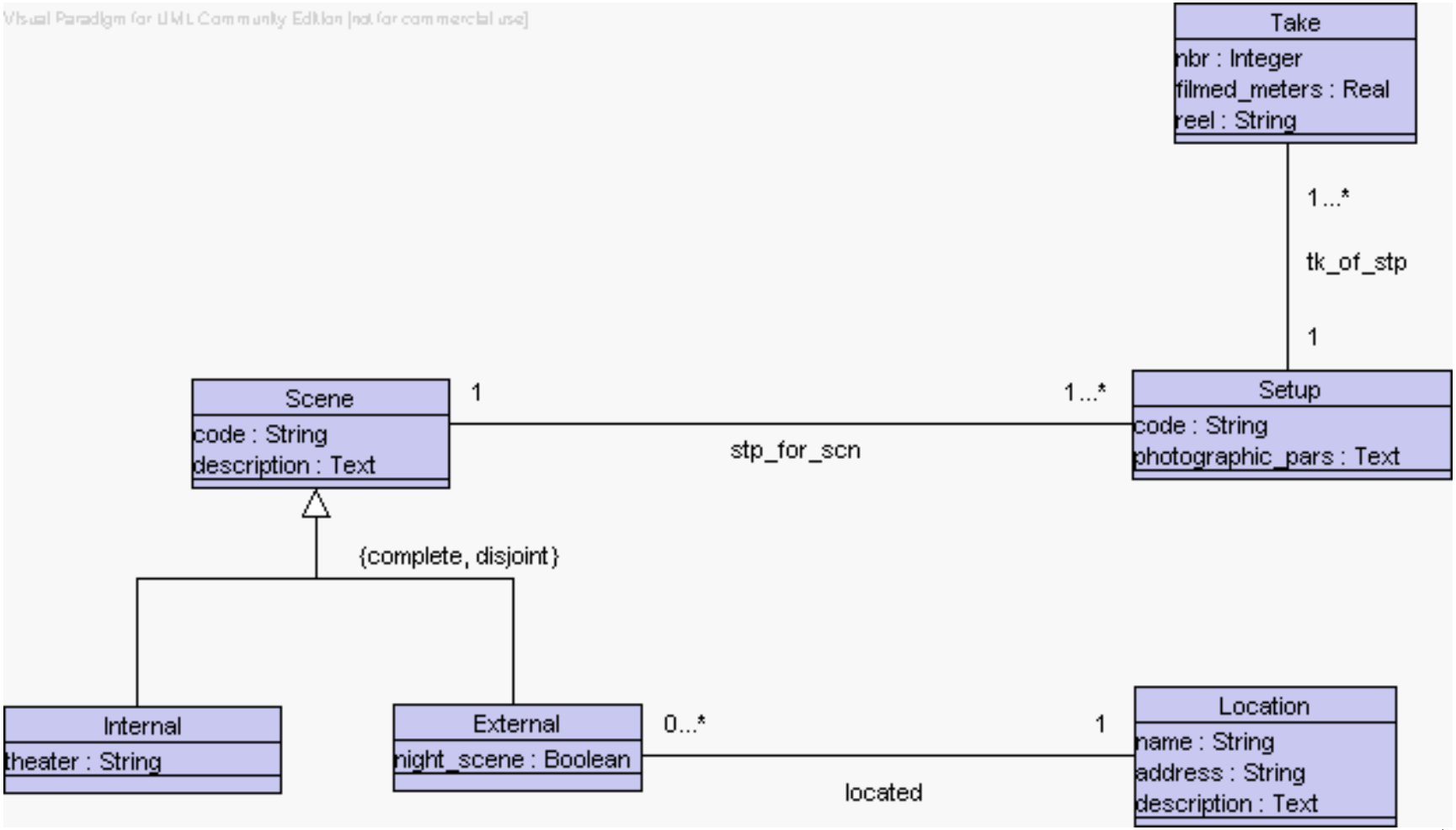
$$\forall x_1.\ C_1(x_1) \rightarrow (min_1 \leq \sharp\{x_2 \mid A(x_1, x_2)\} \leq max_1)$$
$$\forall x_2.\ C_2(x_2) \rightarrow (min_2 \leq \sharp\{x_1 \mid A(x_1, x_2)\} \leq max_2)$$

Example:

$$\forall x.\ Book(x) \rightarrow (1 \leq \sharp\{y \mid written\_by(x, y)\})$$

Note: this is a shorthand for a FOL formula expressing cardinality of the possible values for y.

In our example ...



In our example ...

Alphabet:

$Scene(x), Setup(x), Take(x), Internal(x), External(x), Location(x), stp\_for\_scn(x, y), ck\_of\_stp(x, y), located($

Axioms:

$\forall x, y.\, code_{Scene}(x, y) \rightarrow Scene(x) \wedge String(y)$
$\forall x, y.\, description(x, y) \rightarrow Scene(x) \wedge Text(y)$

$\forall x, y.\, code_{Setup}(x, y) \rightarrow Setup(x) \wedge String(y)$
$\forall x, y.\, photographic\_pars(x, y) \rightarrow Setup(x) \wedge Text(y)$

$\forall x, y.\, nbr(x, y) \rightarrow Take(x) \wedge Integer(y)$
$\forall x, y.\, filmed\_meters(x, y) \rightarrow Take(x) \wedge Real(y)$
$\forall x, y.\, reel(x, y) \rightarrow Take(x) \wedge String(y)$

$\forall x, y.\, theater(x, y) \rightarrow Internal(x) \wedge String(y)$

$\forall x, y.\, night\_scene(x, y) \rightarrow External(x) \wedge Boolean(y)$

$\forall x, y.\, name(x, y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.\, address(x, y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.\, description(x, y) \rightarrow Location(x) \wedge Text(y)$

$\forall x.\, Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$
$\ldots$

$\forall x, y.\, stp\_for\_scn(x, y) \rightarrow Setup(x) \wedge Scene(y)$
$\forall x, y.\, tk\_of\_stp(x, y) \rightarrow Take(x) \wedge Setup(y)$
$\forall x, y.\, located(x, y) \rightarrow External(x) \wedge Location(y)$

$\forall x.\, Setup(x) \rightarrow 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$
$\forall y.\, Scene(y) \rightarrow 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$
$\forall x.\, Take(x) \rightarrow 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$
$\forall x.\, Setup(y) \rightarrow 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$
$\forall x.\, External(x) \rightarrow 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$

$\forall x.\, Internal(x) \rightarrow Scene(x)$
$\forall x.\, External(x) \rightarrow Scene(x)$
$\forall x.\, Internal(x) \rightarrow \neg External(x)$
$\forall x.\, Scene(x) \rightarrow Internal(x) \vee External(x)$

Associations: most interesting multiplicities

The most interesting multiplicities are:

- $0..*$: unconstrained
- $1..*$: mandatory participation
- $0..1$: functional participation (the association is a partial function)
- $1..1$: mandatory and functional participation (the association is a total faction)

In FOL:

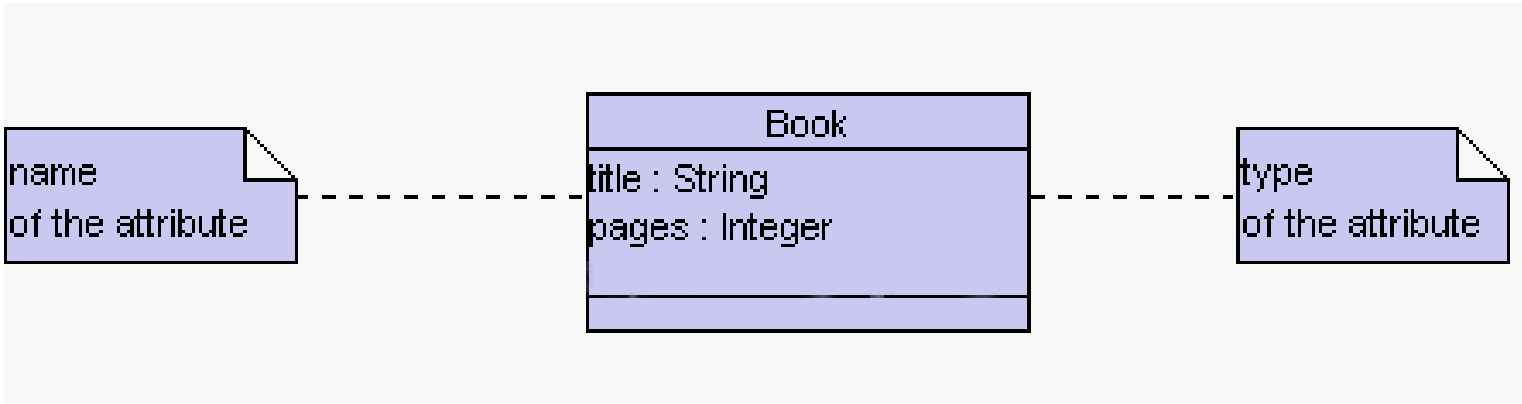
- $0..*$: no constrain
- $1..*$: $\forall x.\ C_1(x) \rightarrow \exists y.A(x,y)$
- $0..1$: $\forall x.\ C_1(x) \rightarrow \forall y,y'.A(x,y) \wedge A(x,y') \rightarrow y = y'$ (or simply $\forall x,y,y'.A(x,y) \wedge A(x,y') \rightarrow y = y'$)
- $1..1$: $(\forall x.\ C_1(x) \rightarrow \exists y.A(x,y)) \wedge (\forall x,y,y'.A(x,y) \wedge A(x,y') \rightarrow y = y')$

Attributes

An attribute models a local property of a class.
It is characterized by:

- a name (which is unique only in the class it belongs to)
- and a type (a collection of possible values)
- and possibly a multiplicity

Example:



Attributes (cont.1)

Attributes without explicit multiplicity are:

- mandatory (must have at least a value)
- single-valued (must have at most a value)

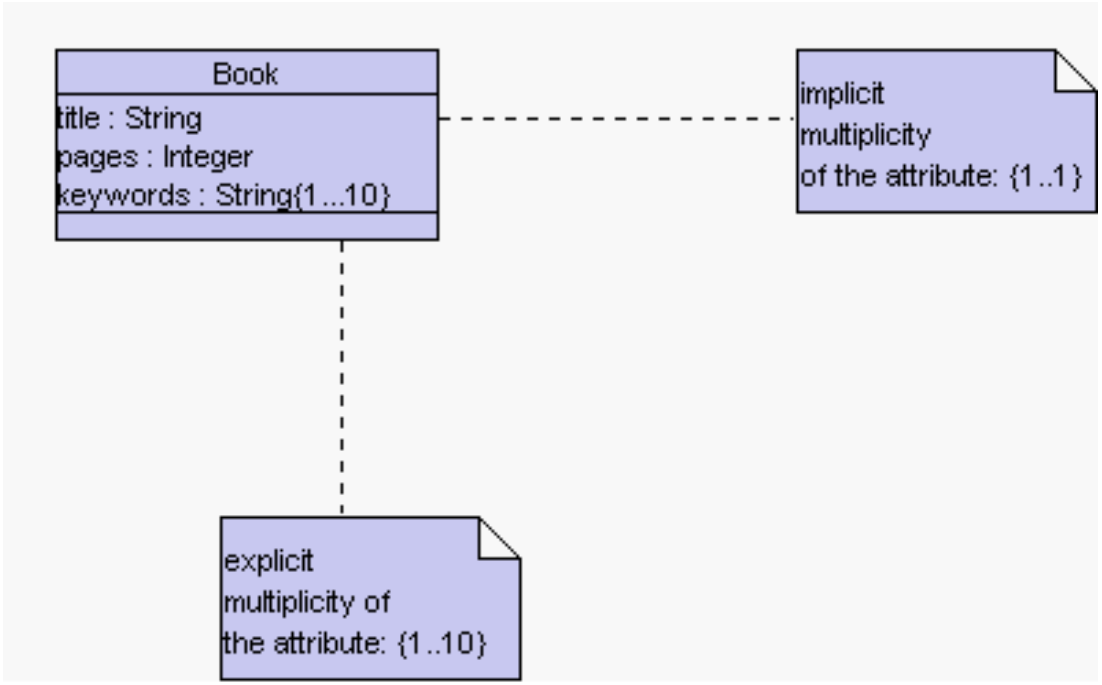
That is, they are functions from the instances of the class to the values of the type they have.

Example:
$book_1$ has as value for the attribute name the $String$: "*The little digital video book*".

Attributes (cont.2)

More generally attributes may have an explicit multiplicity (as associations).

Example:



When multiplicity is implicit then it is assumed to be $1 \ldots 1$.

Attributes: formalization

Since attributes may have a multiplicity different from $1 \ldots 1$ they are better formalized as binary predicates, with suitable assertions representing types and multiplicity:
Given an attribute a of a class C with type T and multiplicity $i \ldots j$ we capture it in FOL as a binary predicate $a_C(x, y)$ with the following assertions:

- Assertion for the attribute type

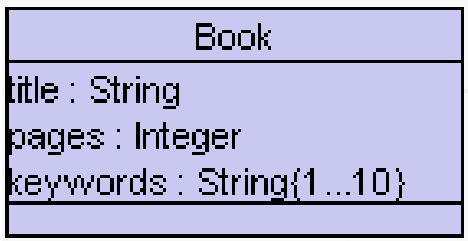
$$\forall x, y.\ a(x, y) \rightarrow C(x) \wedge T(y)$$

- Assertion for the multiplicity

$$\forall x.\ C(x) \rightarrow (i \leq \sharp\{y \mid a(x, y)\} \leq j)$$

Attributes: formalization (cont.)

Example:

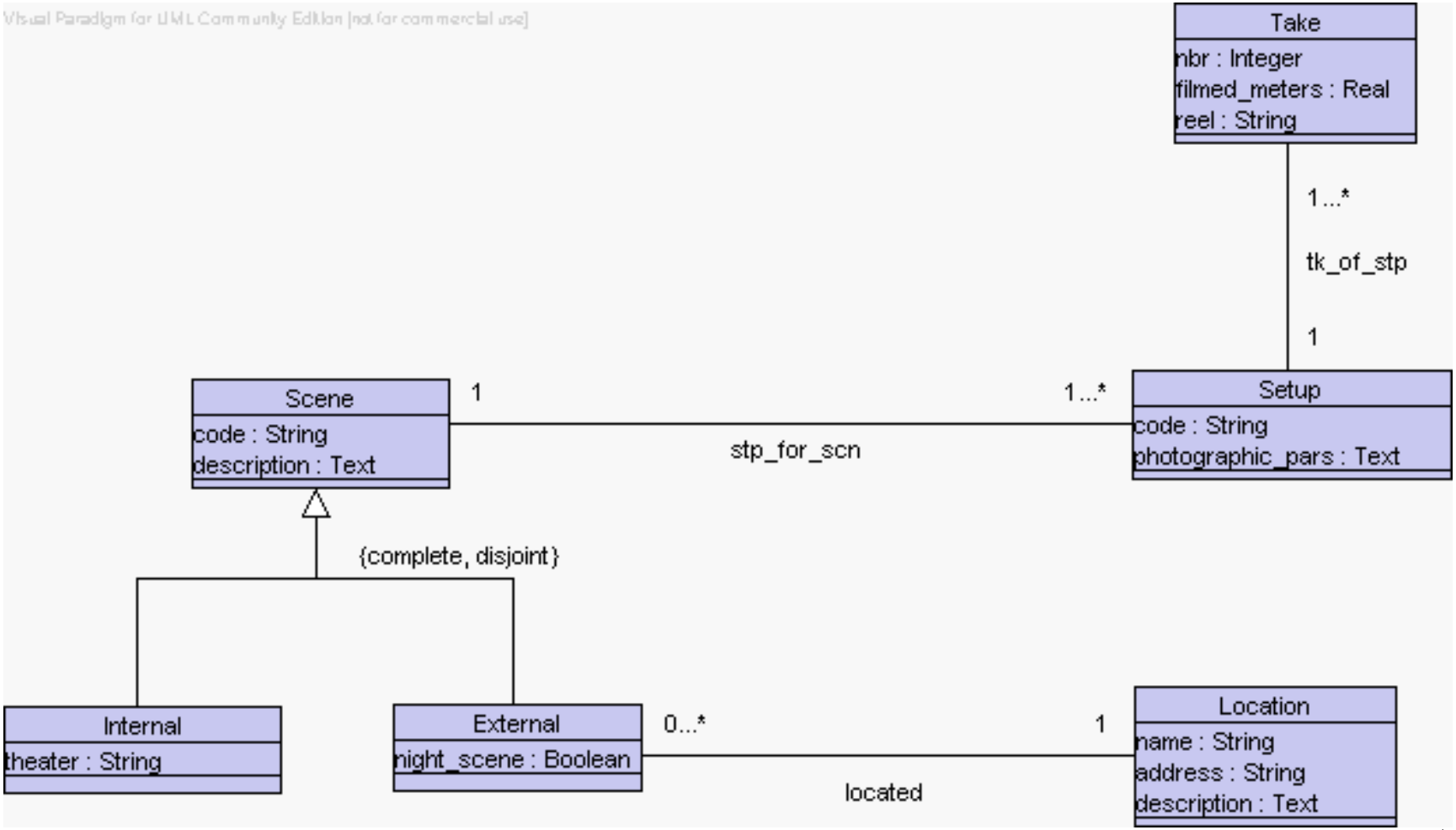


$$\forall x, y.\ title(x, y) \rightarrow Book(x) \wedge String(y)$$
$$\forall x.\ Book(x) \rightarrow (1 \leq \sharp\{y \mid name(x, y)\} \leq 1)$$

$$\forall x, y.\ pages(x, y) \rightarrow Book(x) \wedge Integer(y)$$
$$\forall x.\ Book(x) \rightarrow (1 \leq \sharp\{y \mid pages(x, y)\} \leq 1)$$

$$\forall x, y.\ keywords(x, y) \rightarrow Book(x) \wedge String(y)$$
$$\forall x.\ Book(x) \rightarrow (1 \leq \sharp\{y \mid keywords(x, y)\} \leq 10)$$

In our example ...



In our example ...

Alphabet:
$Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stp\_for\_scn(x, y)$, $ck\_of\_stp(x, y)$, $located($
Axioms:

$\forall x, y.\, code_{Scene}(x, y) \rightarrow Scene(x) \wedge String(y)$
$\forall x, y.\, description(x, y) \rightarrow Scene(x) \wedge Text(y)$

$\forall x, y.\, code_{Setup}(x, y) \rightarrow Setup(x) \wedge String(y)$
$\forall x, y.\, photographic\_pars(x, y) \rightarrow Setup(x) \wedge Text(y)$

$\forall x, y.\, nbr(x, y) \rightarrow Take(x) \wedge Integer(y)$
$\forall x, y.\, filmed\_meters(x, y) \rightarrow Take(x) \wedge Real(y)$
$\forall x, y.\, reel(x, y) \rightarrow Take(x) \wedge String(y)$

$\forall x, y.\, theater(x, y) \rightarrow Internal(x) \wedge String(y)$

$\forall x, y.\, night\_scene(x, y) \rightarrow External(x) \wedge Boolean(y)$

$\forall x, y.\, name(x, y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.\, address(x, y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.\, description(x, y) \rightarrow Location(x) \wedge Text(y)$

$\forall x.\, Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$
$\ldots$

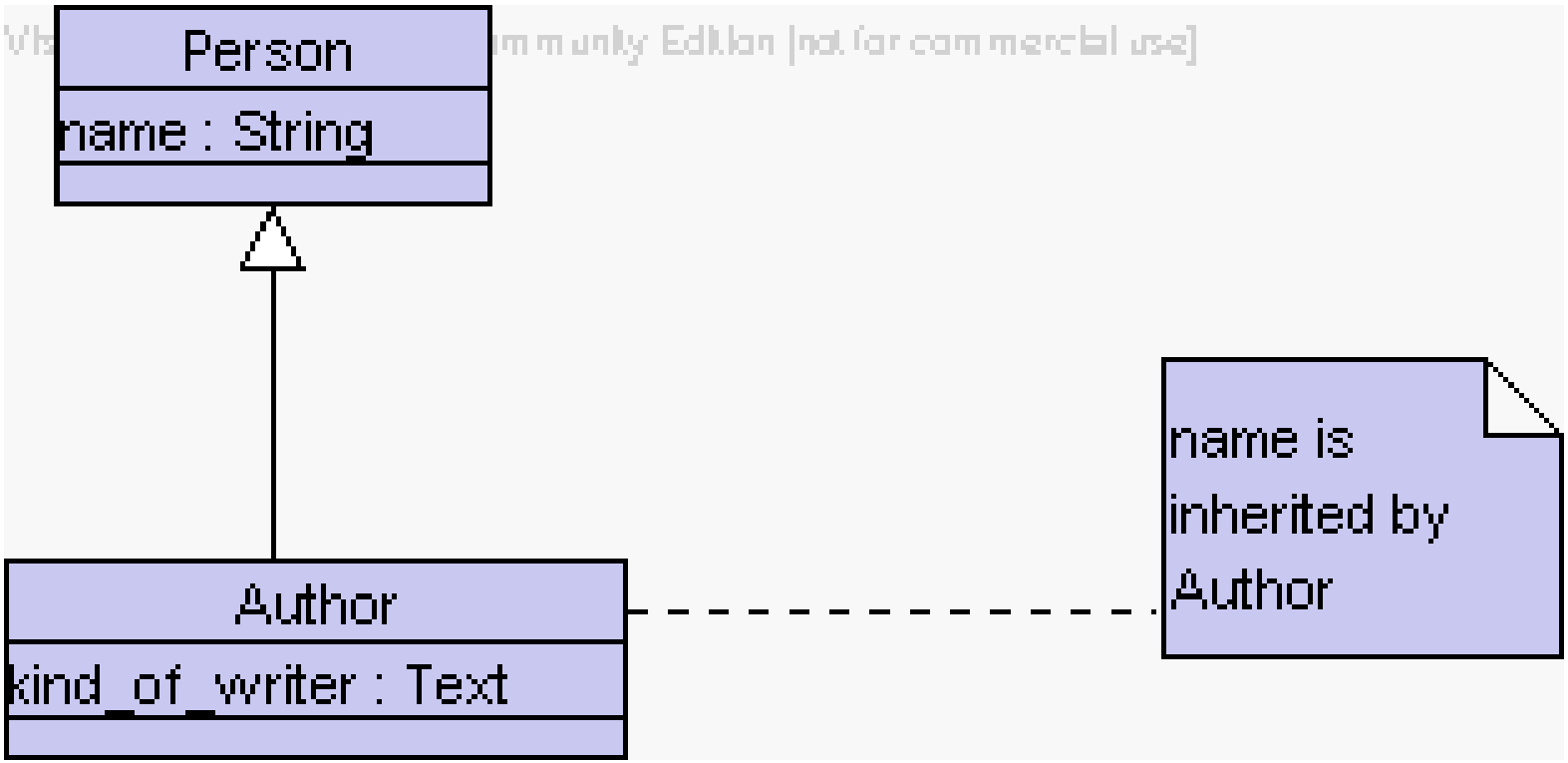
$\forall x, y.\, stp\_for\_scn(x, y) \rightarrow Setup(x) \wedge Scene(y)$
$\forall x, y.\, tk\_of\_stp(x, y) \rightarrow Take(x) \wedge Setup(y)$
$\forall x, y.\, located(x, y) \rightarrow External(x) \wedge Location(y)$

$\forall x.\, Setup(x) \rightarrow 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$
$\forall y.\, Scene(y) \rightarrow 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$
$\forall x.\, Take(x) \rightarrow 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$
$\forall x.\, Setup(y) \rightarrow 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$
$\forall x.\, External(x) \rightarrow 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$

$\forall x.\, Internal(x) \rightarrow Scene(x)$
$\forall x.\, External(x) \rightarrow Scene(x)$
$\forall x.\, Internal(x) \rightarrow \neg External(x)$
$\forall x.\, Scene(x) \rightarrow Internal(x) \vee External(x)$

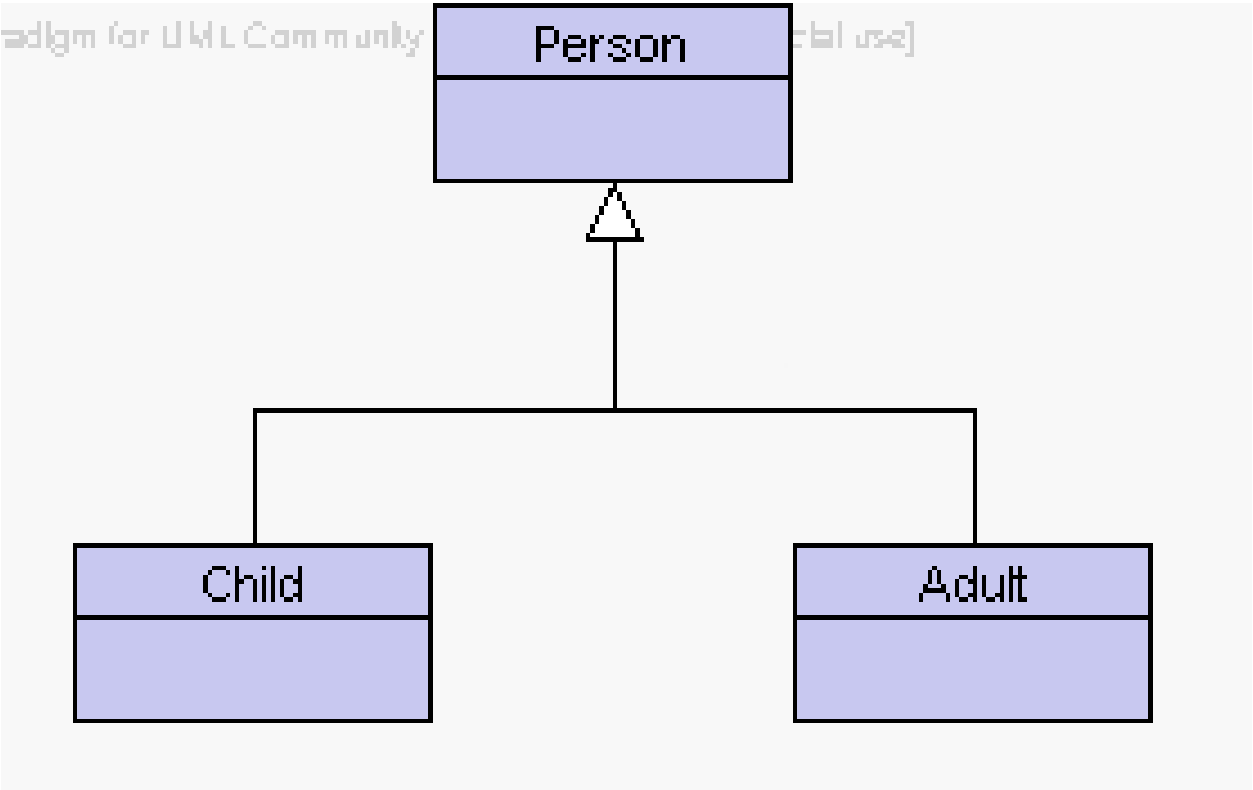
ISA/Generalizations

The ISA relationship is of particular importance in conceptual modeling: a class C ISA a class C' if every instance of C is also an instance of C'.
In UML the ISA relationship is modeled through the notion of generalization.
Example:



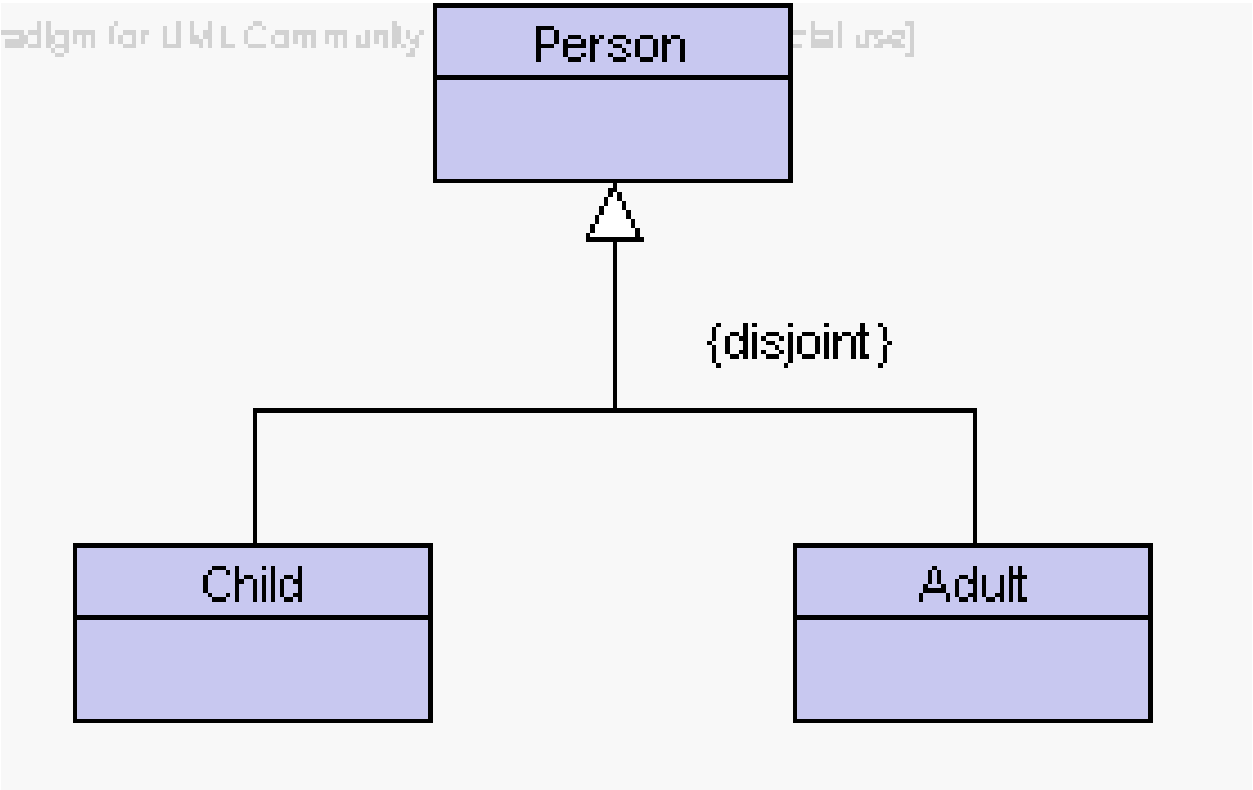
Generalizations (cont.1)

A generalization involves a superclass (base class) and one or more subclasses: every instance of each subclass is also an instance of the superclass.
Example:



Generalizations (cont.2)

The ability of having more subclasses in the same generalization, allows for placing suitable constraints on the classes involved in the generalization:
Example:

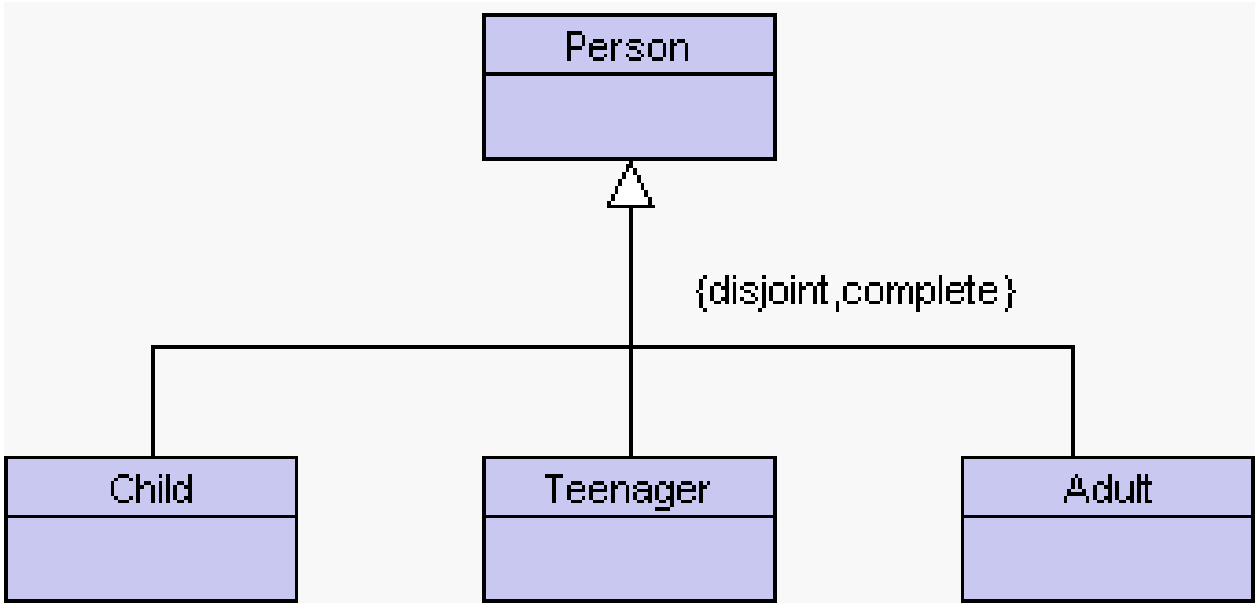


Generalizations (cont.3)

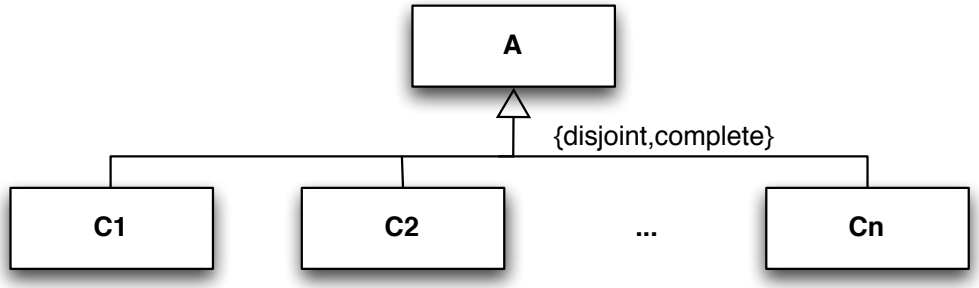
The most notable and used constraints are disjointness and completeness:

- disjointness asserts that different subclasses cannot have common instances (i.e., an object cannot be at the same time instance of two disjoint subclasses).
- completeness (aka "covering") asserts that every instances of the superclass is also an instance of at least one of the subclasses.

Example:



Generalizations: formalization



ISA:

$$\forall x.\ C_i(x) \rightarrow C(x), \qquad \text{for } i = 1, \ldots, n$$

Disjointness:

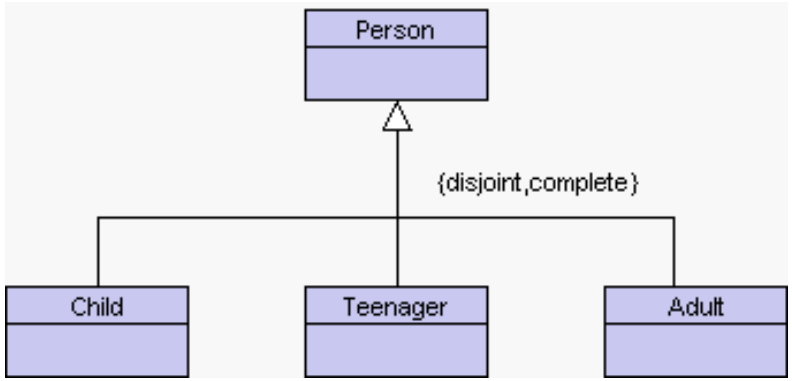
$$\forall x.\ C_i(x) \rightarrow \neg C_j(x), \qquad \text{for } i \neq j$$

Completeness:

$$\forall x.\ C(x) \rightarrow \bigvee_{i=1}^{n} C_i(x)$$

Generalizations: formalization (cont.)

Example:

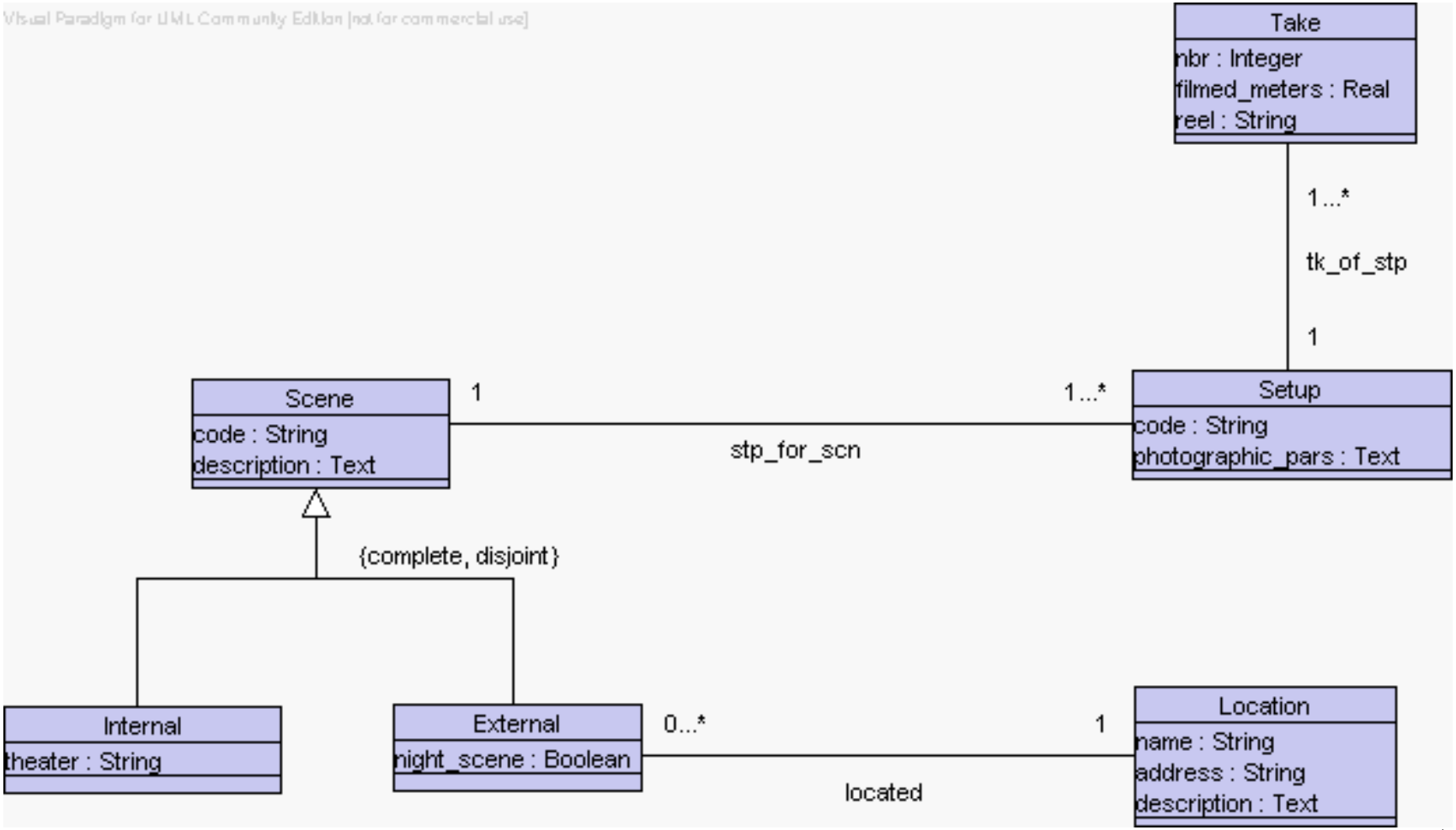


$\forall x.\, Child(x) \rightarrow Person(x)$
$\forall x.\, Teenager(x) \rightarrow Person(x)$
$\forall x.\, Adult(x) \rightarrow Person(x)$

$\forall x.\, Child(x) \rightarrow \neg Teenager(x)$
$\forall x.\, Child(x) \rightarrow \neg Adult(x)$
$\forall x.\, Teenager(x) \rightarrow \neg Adult(x)$

$\forall x.\, Person(x) \rightarrow (Child(x) \vee Teenager(x) \vee Adult(x))$

In our example ...



In our example ...

Alphabet:

$Scene(x), Setup(x), Take(x), Internal(x), External(x), Location(x), stp\_for\_scn(x, y), ck\_of\_stp(x, y), located($

Axioms:

$$\forall x, y.\, code_{Scene}(x, y) \rightarrow Scene(x) \wedge String(y)$$
$$\forall x, y.\, description(x, y) \rightarrow Scene(x) \wedge Text(y)$$

$$\forall x, y.\, code_{Setup}(x, y) \rightarrow Setup(x) \wedge String(y)$$
$$\forall x, y.\, photographic\_pars(x, y) \rightarrow Setup(x) \wedge Text(y)$$

$$\forall x, y.\, nbr(x, y) \rightarrow Take(x) \wedge Integer(y)$$
$$\forall x, y.\, filmed\_meters(x, y) \rightarrow Take(x) \wedge Real(y)$$
$$\forall x, y.\, reel(x, y) \rightarrow Take(x) \wedge String(y)$$

$$\forall x, y.\, theater(x, y) \rightarrow Internal(x) \wedge String(y)$$

$$\forall x, y.\, night\_scene(x, y) \rightarrow External(x) \wedge Boolean(y)$$

$$\forall x, y.\, name(x, y) \rightarrow Location(x) \wedge String(y)$$
$$\forall x, y.\, address(x, y) \rightarrow Location(x) \wedge String(y)$$
$$\forall x, y.\, description(x, y) \rightarrow Location(x) \wedge Text(y)$$

$$\forall x.\, Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x, y)\} \leq 1)$$
$$\ldots$$

$$\forall x, y.\, stp\_for\_scn(x, y) \rightarrow Setup(x) \wedge Scene(y)$$
$$\forall x, y.\, tk\_of\_stp(x, y) \rightarrow Take(x) \wedge Setup(y)$$
$$\forall x, y.\, located(x, y) \rightarrow External(x) \wedge Location(y)$$

$$\forall x.\, Setup(x) \rightarrow 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$$
$$\forall y.\, Scene(y) \rightarrow 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$$
$$\forall x.\, Take(x) \rightarrow 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$$
$$\forall x.\, Setup(y) \rightarrow 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$$
$$\forall x.\, External(x) \rightarrow 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$$

$$\forall x.\, Internal(x) \rightarrow Scene(x)$$
$$\forall x.\, External(x) \rightarrow Scene(x)$$
$$\forall x.\, Internal(x) \rightarrow \neg External(x)$$
$$\forall x.\, Scene(x) \rightarrow Internal(x) \vee External(x)$$

Association classes

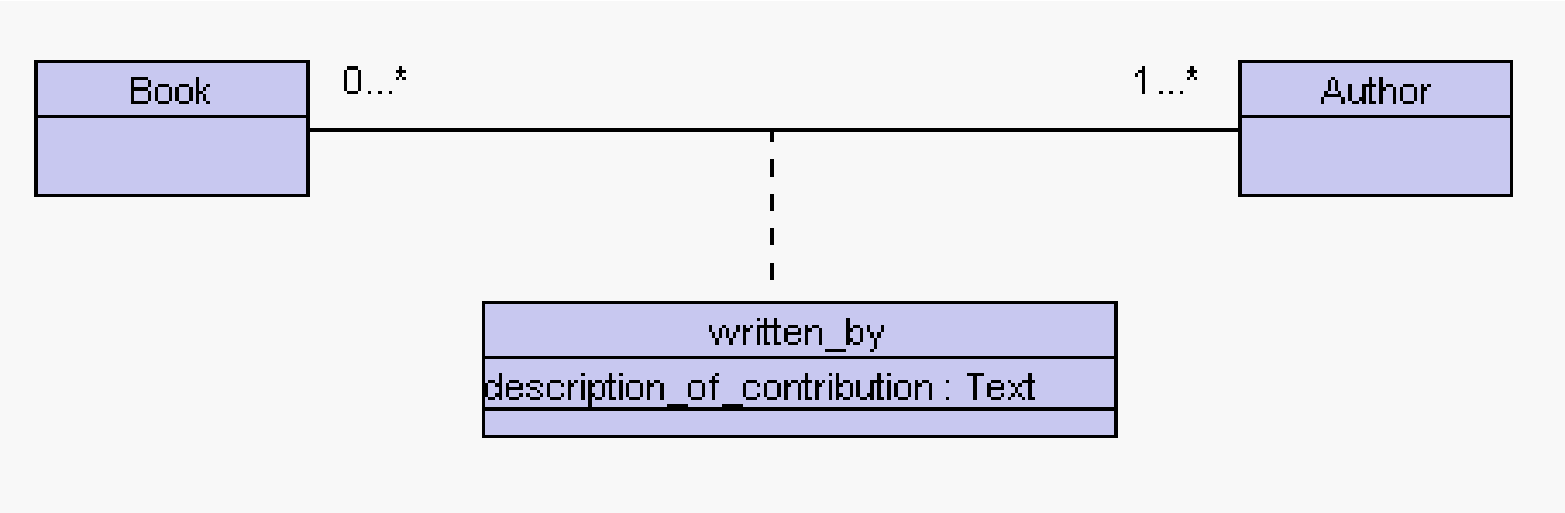
Sometimes we may want to assert properties of associations. In UML to do so we resort to Association Classes.

That is, we associate to an association a class whose instances are in bijection with the tuples of the association.

Then we use the association class exactly as a UML class (modeling local and non-local properties).

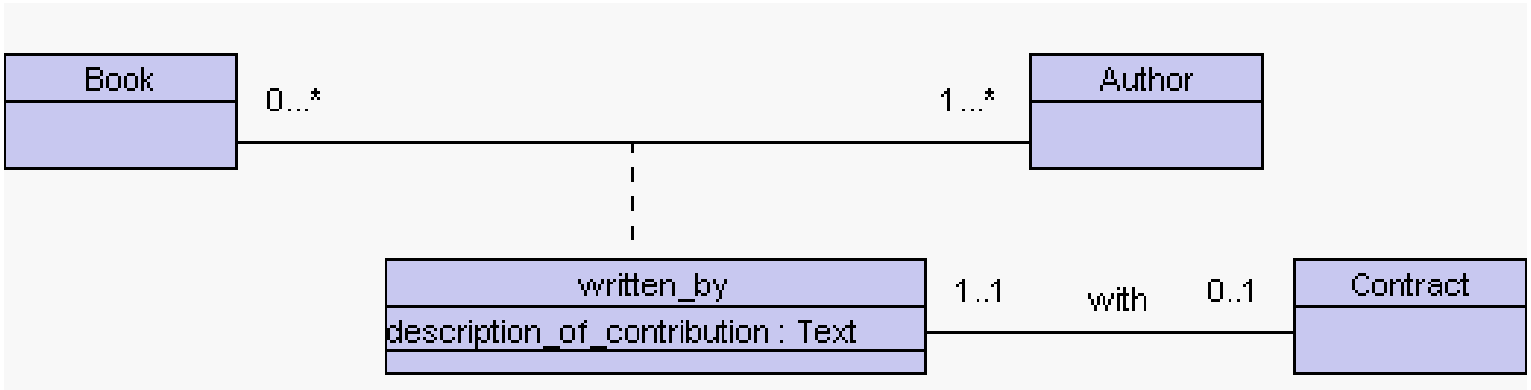
Association classes (cont.1)

Example:

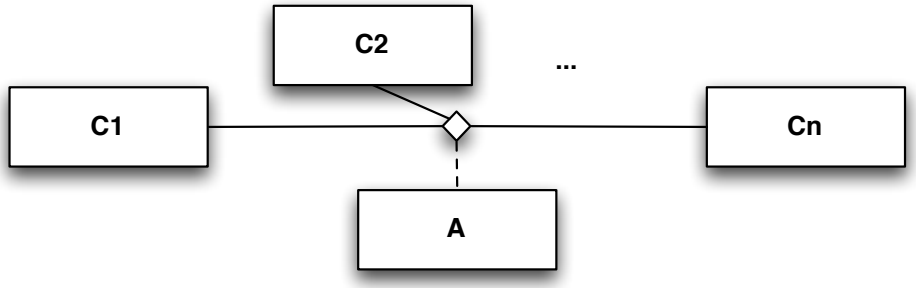


Association classes (cont.2)

Example:



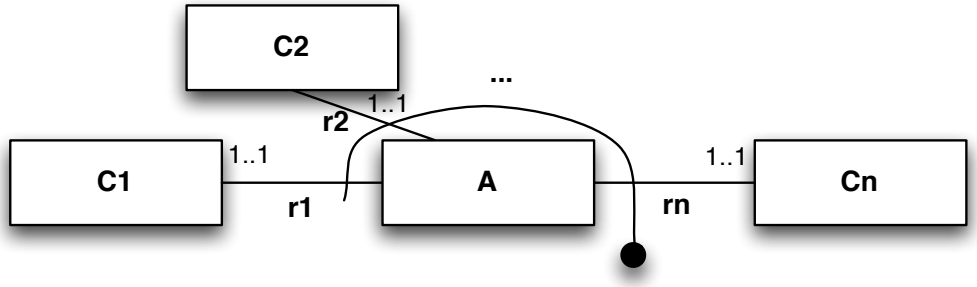
Association classes: formalization



The process of putting in correspondence objects of a class (the association class) with tuples in an association is formally described as reification.
That is:

- we introduce a unary predicate A for the association class A
- we introduce n new binary predicates $r_1, \ldots, r_n$, one for each of the components of the association
- we introduce suitable assertions so that objects in the extension of unary-predicate A are in bijection with tuples in n-ary association A.

Association classes: formalization (cont.1)



FOL Assertions needed for stating bijection between association class and association:

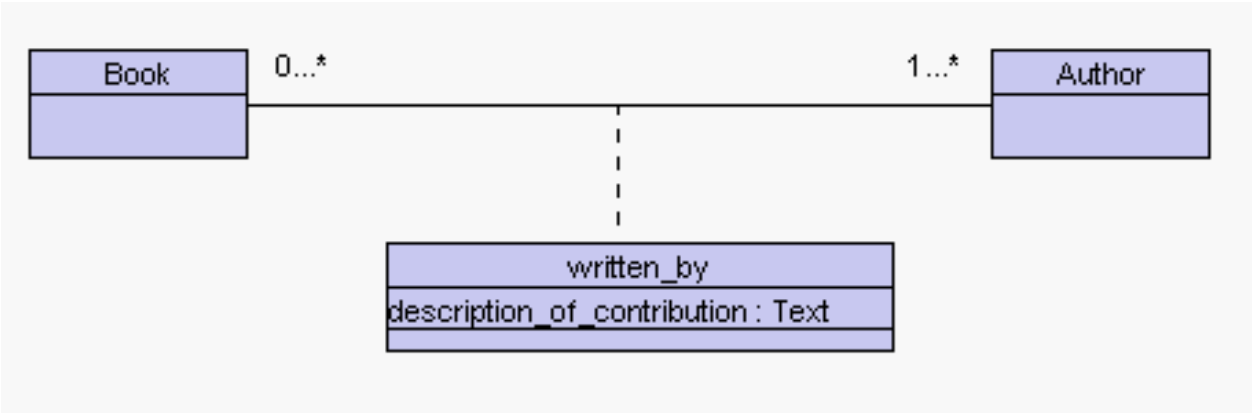
$$\forall x, y.\ r_i(x, y) \rightarrow A(x) \wedge C_i(y), \qquad \text{for } i = 1, \ldots, n$$

$$\forall x.\ A(x) \rightarrow \exists y.\ r_i(x, y), \qquad \text{for } i = 1, \ldots, n$$

$$\forall x, y, y'.\ r_i(x, y) \wedge r_i(x, y') \rightarrow y = y', \qquad \text{for } i = 1, \ldots, n$$

$$\forall y_1, \ldots, y_n, x, x'.\ \bigwedge_{i=1}^{n} (r_i(x, y_i) \wedge r_i(x', y_i)) \rightarrow x = x'$$

Association classes: formalization (example)



$$\forall x, y.\ rw_1(x, y) \rightarrow written\_by(x) \wedge Book(y)$$
$$\forall x, y.\ rw_2(x, y) \rightarrow written\_by(x) \wedge Author(y)$$

$$\forall x.\ written\_by(x) \rightarrow \exists y.\ rw_1(x, y)$$
$$\forall x.\ written\_by(x) \rightarrow \exists y.\ rw_2(x, y)$$
$$\forall x, y, y'.\ rw_1(x, y) \wedge rw_1(x, y') \rightarrow y = y'$$
$$\forall x, y, y'.\ rw_2(x, y) \wedge rw_2(x, y') \rightarrow y = y'$$

$$\forall x, x', y_1, y_2.\ rw_1(x, y_1) \wedge rw_1(x', y_1) \wedge rw_2(x, y_2) \wedge rw_2(x', y_2) \rightarrow x = x'$$

Outline

Forms of reasoning: class consistency

A class is consistent, if the class diagram admits an instantiation in which the class has a non-empty set of instances.

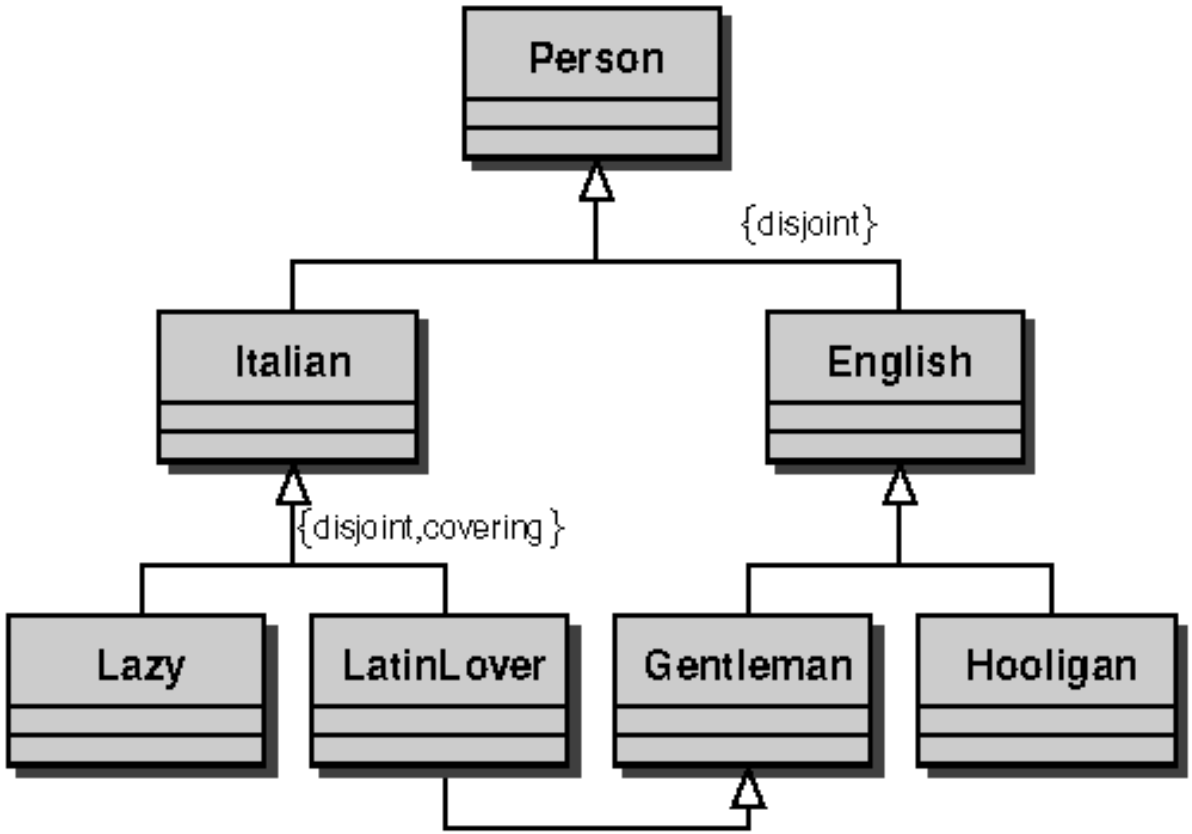
Let Γ be the set of FOL assertions (i.e., the **ontology**) corresponding to the UML Class Diagram, and $C(x)$ the predicate corresponding to a class C of the diagram.
Then C is consistent iff

$$\Gamma \not\models \forall x.\, C(x) \rightarrow false$$

i.e., there exists a model of Γ where the extension of $C(x)$ is not the empty set.

Note: FOL reasoning task: satisfiability

Example (by E. Franconi)



$$\Gamma \models \forall x.\ LatinLover(x) \rightarrow false$$

Forms of reasoning: whole diagram consistency

A class diagram is consistent, if it admits an instantiation, i.e., if its classes can be populated without violating any of the conditions imposed by the diagram.

Let Γ be the set of FOL assertions (i.e., ontology) corresponding to the UML Class Diagram.
Then, the diagram is consistent iff

$$\Gamma \text{ is satisfiable}$$

i.e., Γ admits at least a model (remember that FOL models cannot be empty).

Note: FOL reasoning task: satisfiability.

Forms of reasoning: class subsumption

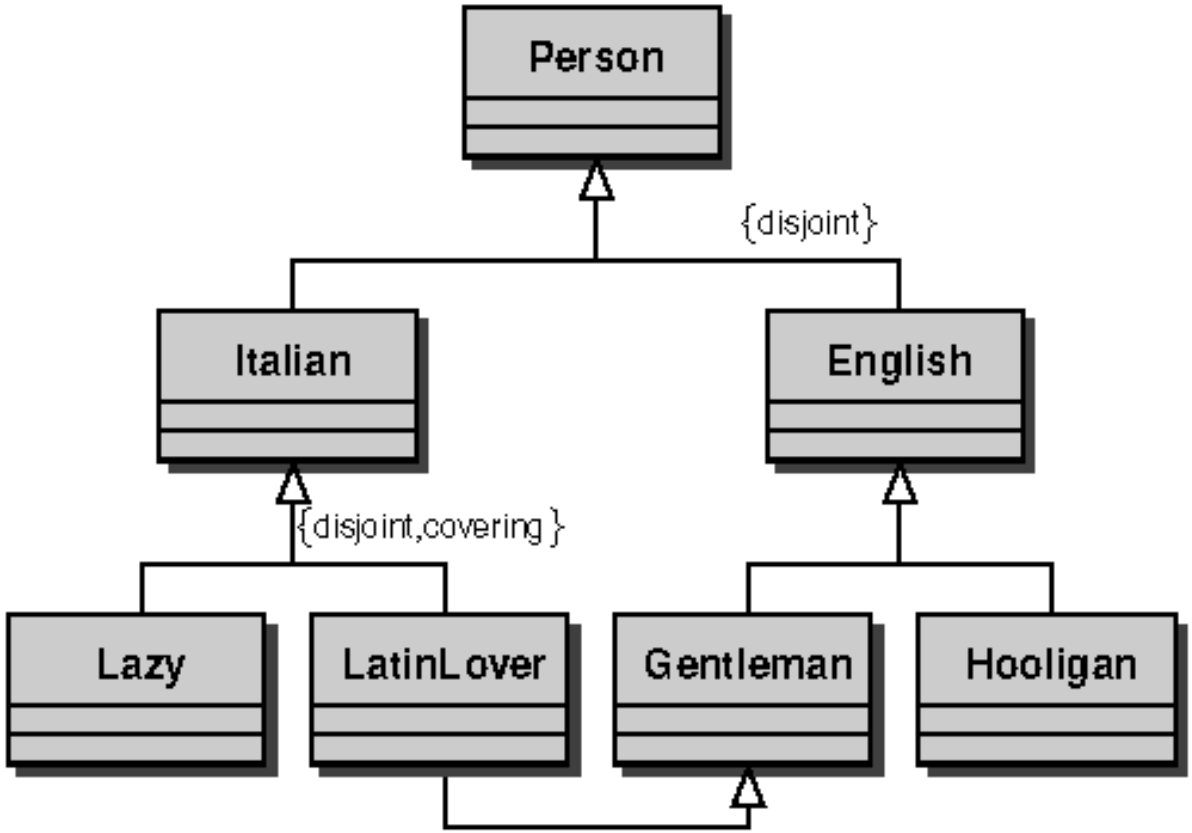
A class C_1 is subsumed by a class C_2 (or C_2 subsumes C_1), if the class diagram implies that C_2 is a generalization of C_1.

Let Γ be the set of FOL assertions (i.e., the ontology) corresponding to the UML Class Diagram, and $C_1(x), C_2(x)$ the predicates corresponding to the class C_1, C_2 of the diagram.
Then C_1 is subsumed by C_2 iff

$$\Gamma \models \forall x.\ C_1(x) \rightarrow C_2(x)$$

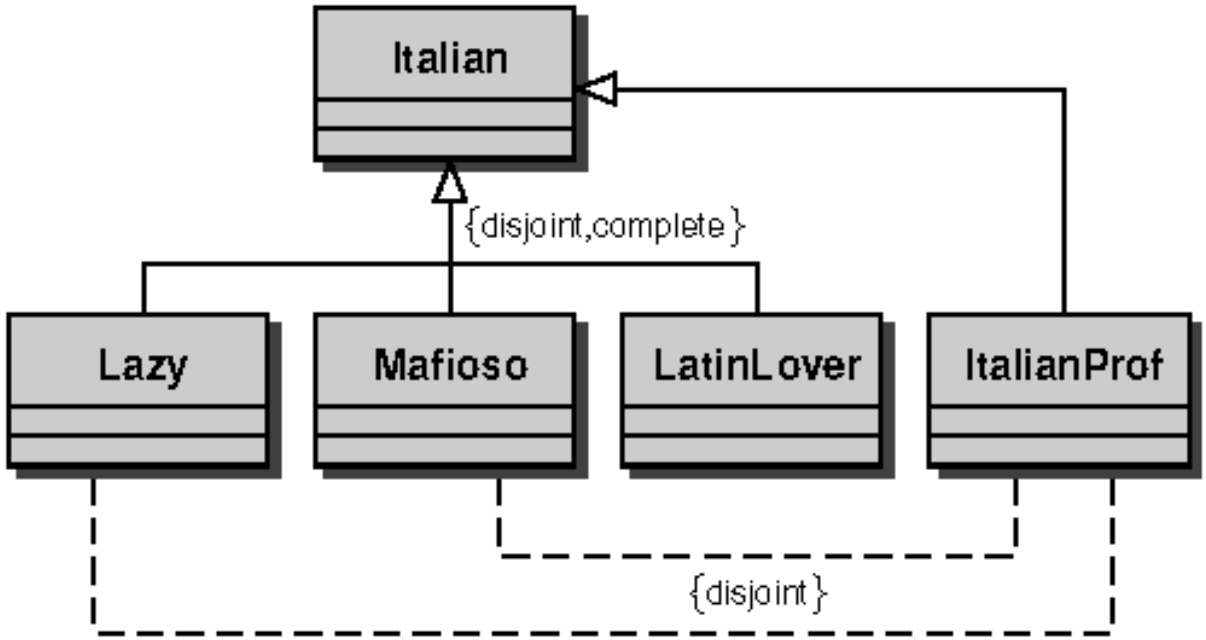
Note: FOL reasoning task: logical implication

Example



$$\Gamma \models \forall x.\ LatinLover(x) \rightarrow false$$
$$\Gamma \models \forall x.\ Italian(x) \rightarrow Lazy(x)$$

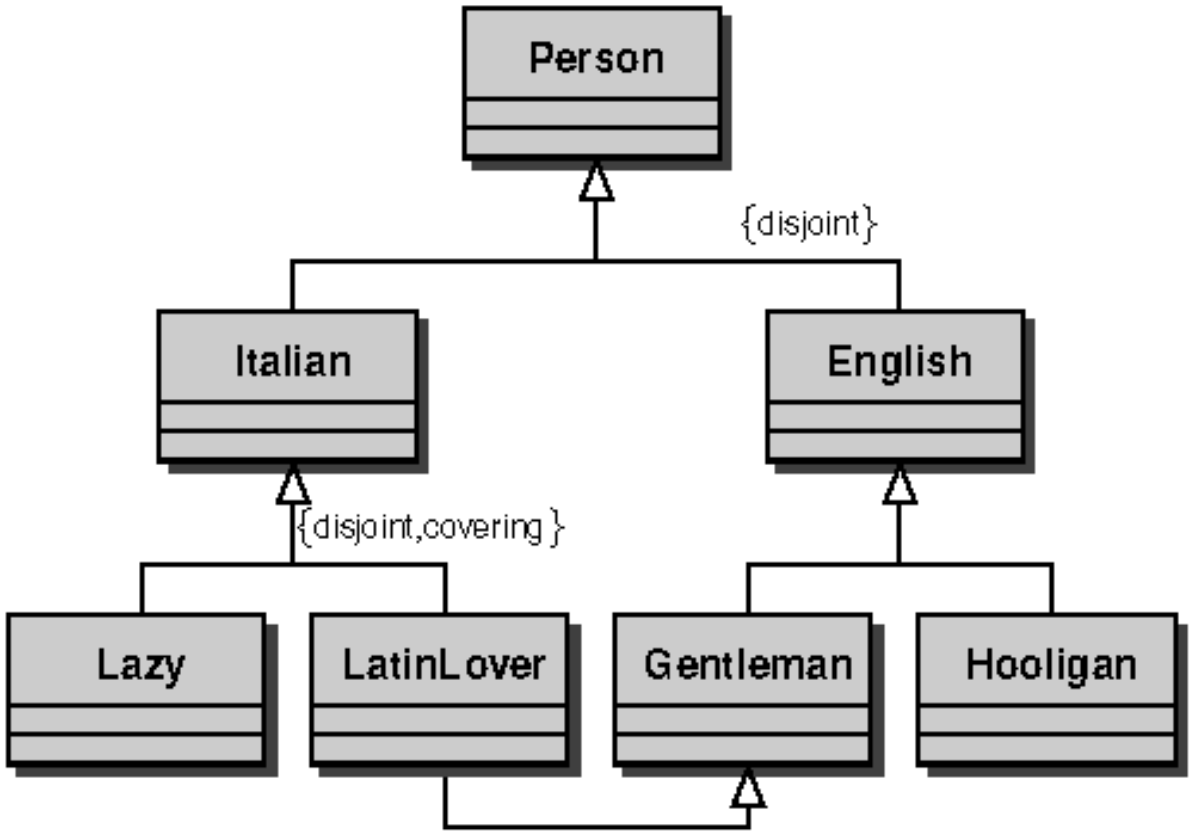
Another Example (by E. Franconi)



(reasoning by cases)

$$\Gamma \models \forall x.\ ItalianProf(x) \rightarrow LatinLover(x)$$

Example



$$\Gamma \models \forall x.\ LatinLover(x) \rightarrow false$$
$$\Gamma \models \forall x.\ Italian(x) \rightarrow Lazy(x)$$
$$\Gamma \models \forall x.\ Lazy(x) \equiv Italian(x)$$

Forms of reasoning: implicit consequence

The properties of various classes and associations may interact to yield stricter multiplicities or typing than those explicitly specified in the diagram.

More generally...
A property $\mathcal{P}$ is an (implicit) consequence of a class diagram if $\mathcal{P}$ holds whenever all conditions imposed by the diagram are satisfied.

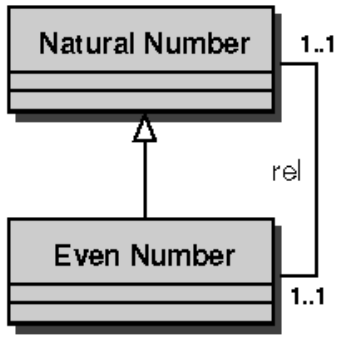
Let Γ be the set of FOL assertion (i.e., the ontology) corresponding to the UML Class Diagram, and $\mathcal{P}$ (the formalization in FOL of) the property of interest
Then $\mathcal{P}$ is an implicit consequence iff

$$\Gamma \models \mathcal{P}$$

i.e., the property $\mathcal{P}$ holds in every model of Γ.

Note: FOL reasoning task: logical implication

Unrestricted vs. finite model reasoning



The classes $NaturalNumber$ and $EvenNumber$ are in bijection.
this implies: "the classes $NaturalNumber$ and $EvenNumber$ contains the same number of objects"

Unrestricted vs. finite model reasoning (cont.)

- If the domain is finite then:

$$\forall x.\ NaturalNumber(x) \rightarrow EvenNumber(x)$$

- if the domain is infinite we do not get the subsumption!

Finite model reasoning: look only at models with finite domains (very interesting for standard Databases).

In UML Class Diagrams finite model reasoning is different form unrestricted reasoning.

Questions

In above examples reasoning could be easily carried out on intuitive grounds, but ...

- *Can we develop sound, complete, and* **terminating** *reasoning procedures for reasoning on UML Class Diagrams?*
  - not using FOL ...
  - ... but we can in fact use Description Logics instead of FOL (see later).
  - $\leadsto$ reasoning on UML Class Diagrams can be done in EXPTIME in general (and actually carried out by current DLs-based systems such as FACT++, PELLET or RACER-PRO).
- *How hard is it to reason on UML Class Diagrams in general?*
  - It's EXPTIME-hard in general! [Berardi, Calvanese, De Giacomo - AIJ 2005].
  - This is somewhat surprising, since UML Class Diagrams are so widely used and yet reasoning on them (and hence fully understand the implication the may give rise to) is not easy at all in general.
  - *Note: that there are very interesting fragments that are PTIME (see later)!*
- *What about query answering?*

  See next.

Note that these results hold for Entity-Relationship Diagrams as well!!!

Outline

Query answering

In ontology-based information integration we are interested in a reasoning service that is not typical in ontologies (or a FOL theory, or UML class diagram, or knowledge base) but it is very common in databases: **query answering**.

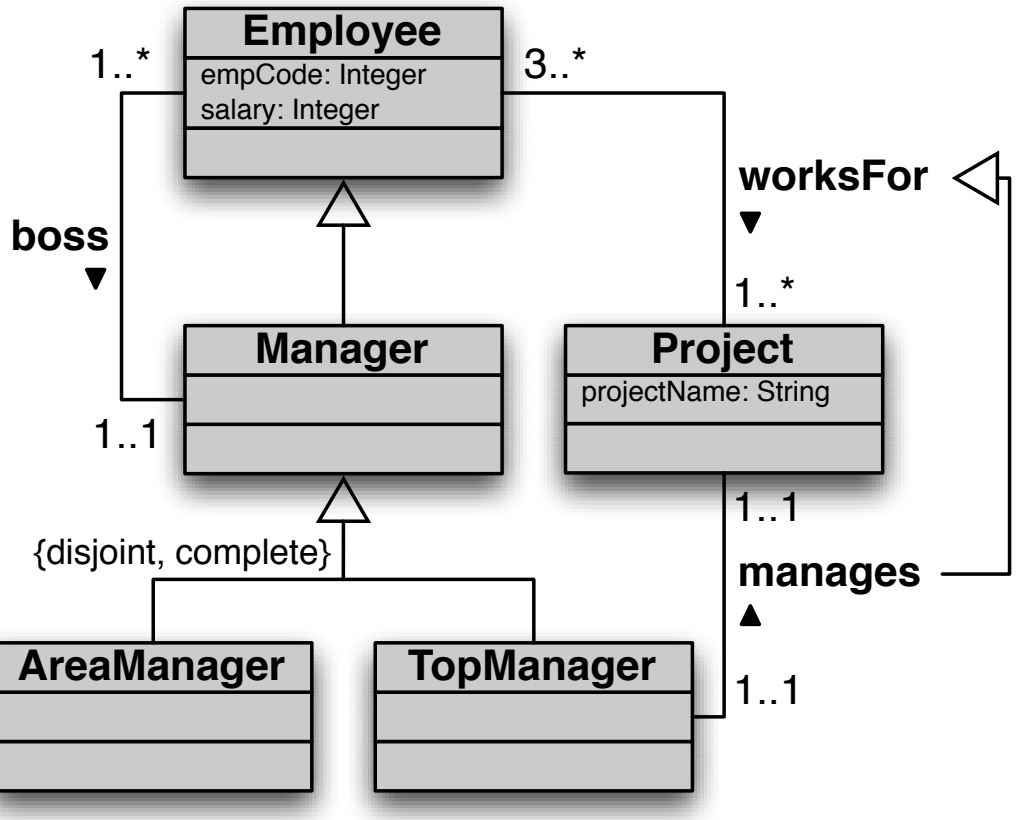
Def.: Query

Is an expression at the intensional level denoting set of tuples of individuals satisfying a given condition.

Def.: Query Answering

Is the reasoning services that actually compute the answer to a query.

Example of query



$$q(ce, cm, sa) \leftarrow \exists e, p, m.$$
$$\text{worksFor}(e, p) \wedge \text{manages}(m, p) \wedge \text{boss}(m, e) \wedge \text{empCode}(e, ce) \wedge$$
$$\text{empCode}(m, cm) \wedge \text{salary}(e, sa) \wedge \text{salary}(m, sa)$$

Query answering under different assumptions

There are two fundamental different assumptions when addressing query answering:

- **complete information** on the data, as in traditional databases.
- **incomplete information** on the data, as in ontologies (aka knowledge bases), but also information integration in databases.

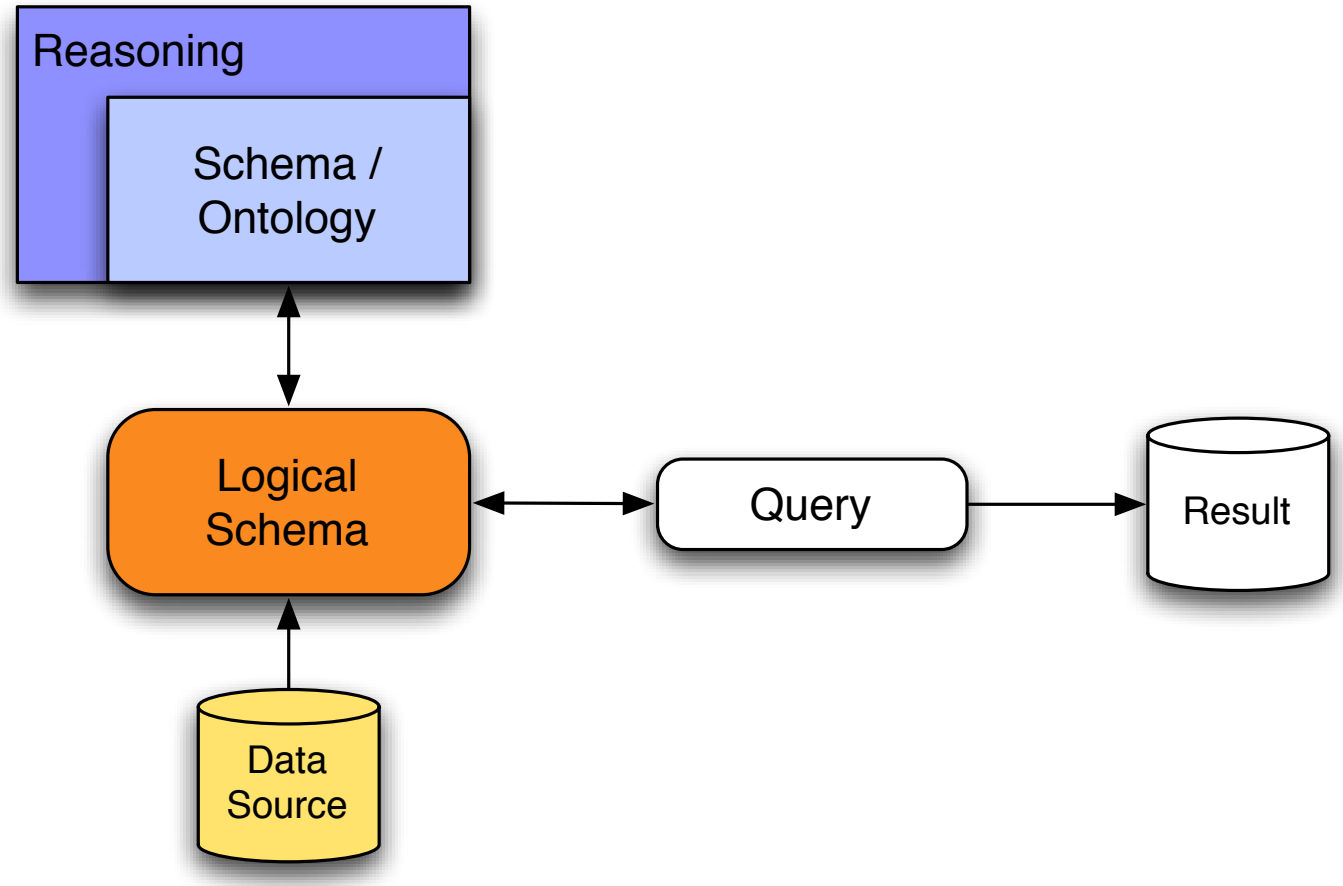
Outline

Query answering in traditional databases

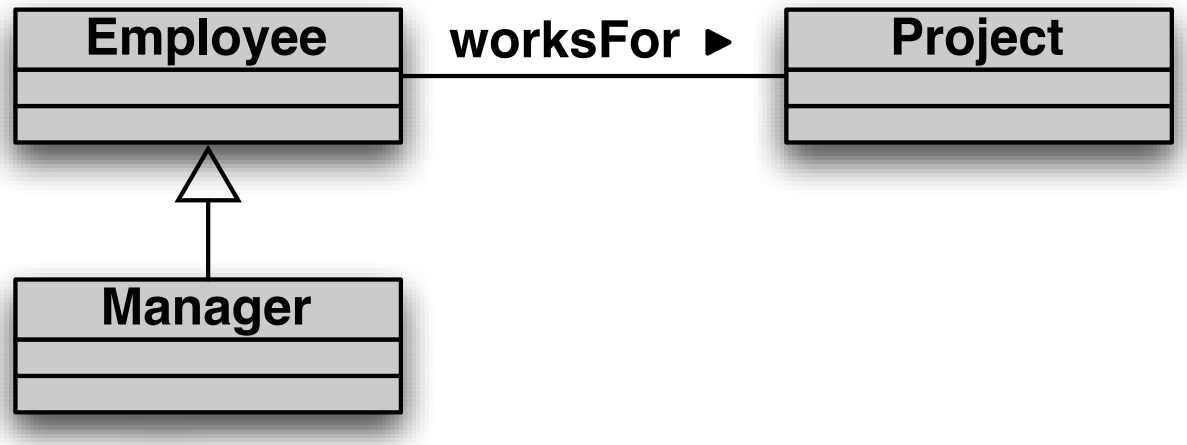
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

$\rightsquigarrow$ Query answering amounts to **query evaluation**, which is computationally easy.

Query answering in traditional databases (cont'd)



Query answering in traditional databases – Example



For each class/association/property we have a (complete) table in the DB.

DB: Employee = { john, mary, nick }
Manager = { john, nick }
Project = { prA, prB }
worksFor = { (john,prA), (mary,prB) }

Query: $q(x) \leftarrow \exists p.\, \mathsf{Manager}(x), \mathsf{Project}(p), \mathsf{worksFor}(x, p)$

Answer: { john }

Outline

Query answering in ontologies

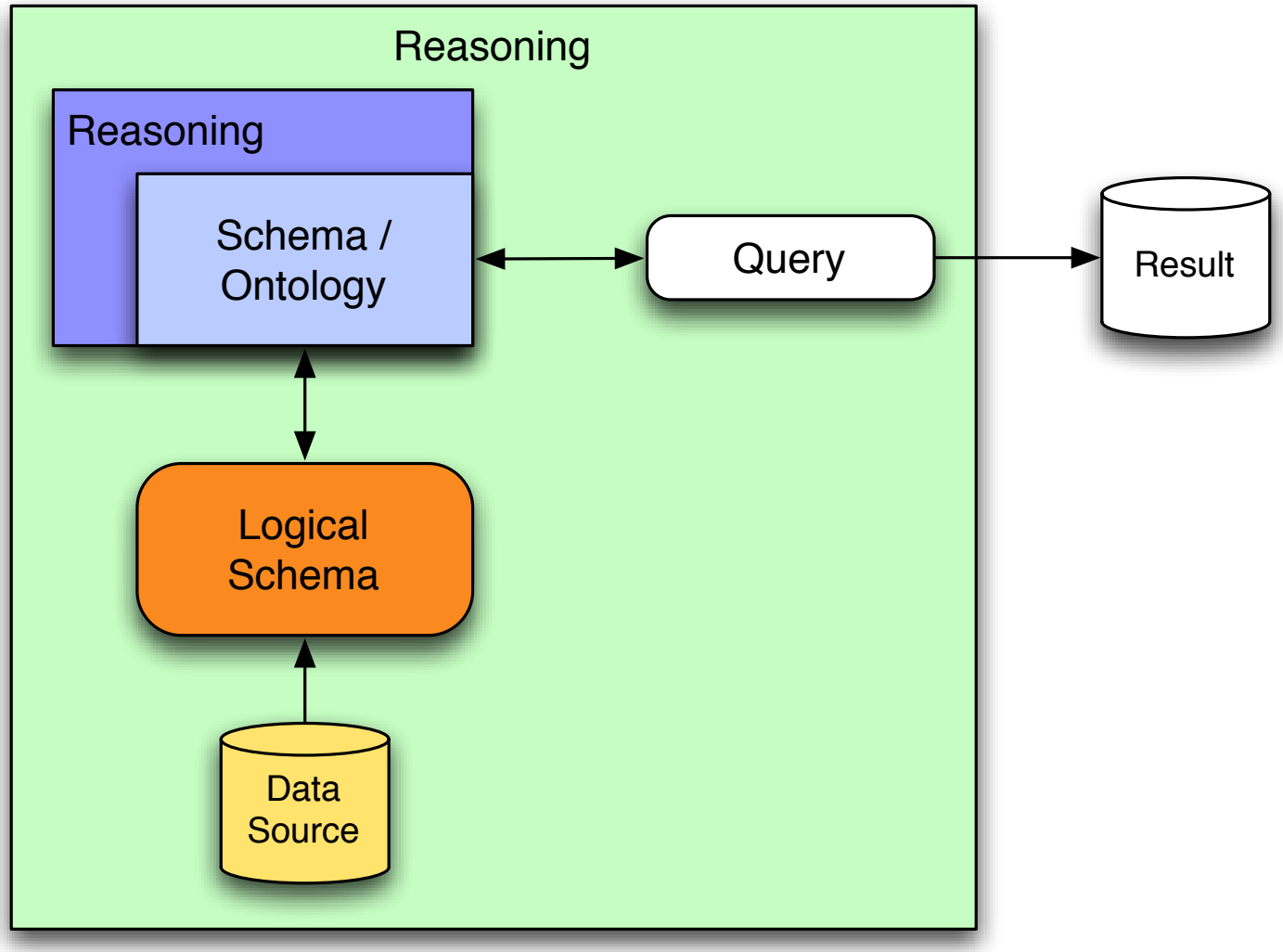
- An ontology (aka knowledge base) imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

$\rightsquigarrow$ Query answering amounts to **logical inference**, which is computationally more costly.

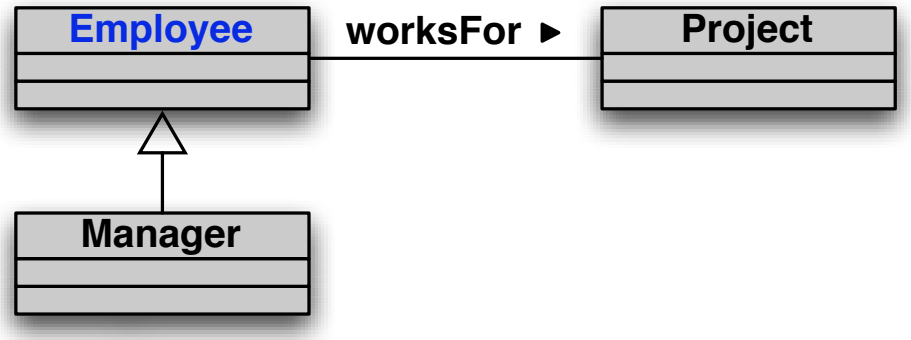
Note:

- The size of the data is not considered critical (comparable to the size of the intensional information).
- Queries are typically simple, i.e., atomic (a class name), and query answering amounts to instance checking.

Query answering in ontologies (cont'd)



Query answering in ontologies – Example



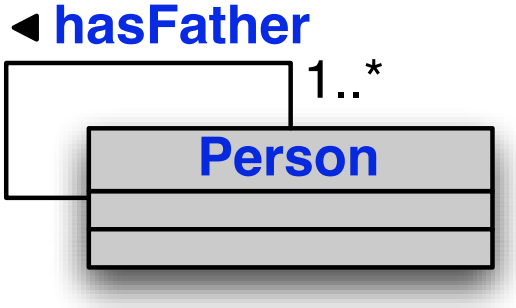
The tables in the database may be **incompletely specified**, or even missing for some classes/associations/properties.

DB: Manager $\supseteq$ { `john`, `nick` }
Project $\supseteq$ { `prA`, `prB` }
worksFor $\supseteq$ { (`john`,`prA`), (`mary`,`prB`) }

Query: $q(x) \leftarrow \text{Employee}(x)$

Answer: { `john`, `nick`, `mary` }

Query answering in ontologies – Example 2



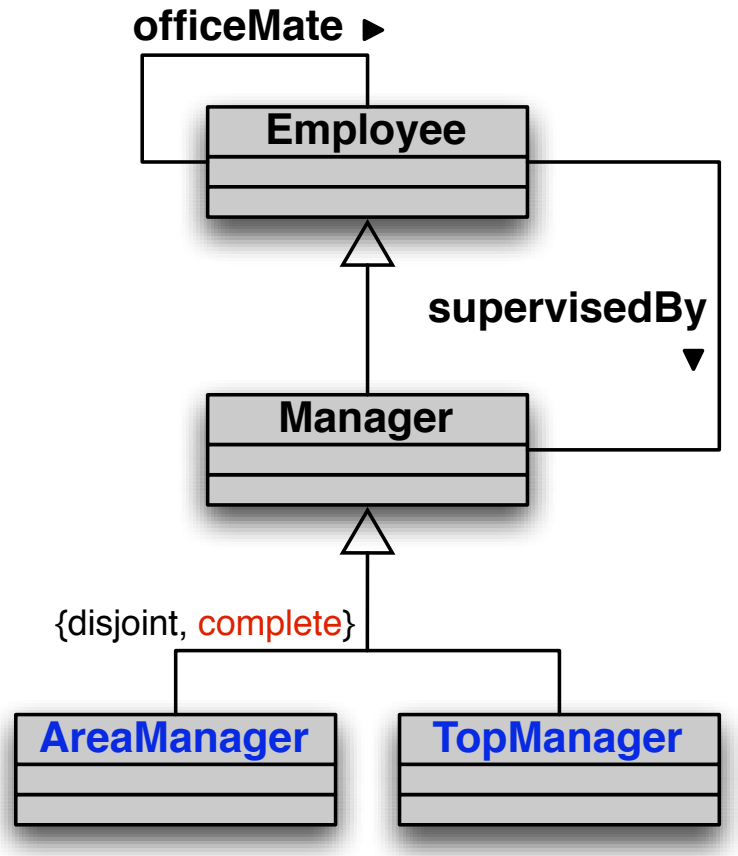
Each person has a father, who is a person.

DB: Person $\supseteq$ { john, nick, toni }
hasFather $\supseteq$ { (john,nick), (nick,toni) }

Queries: $q_1(x, y) \leftarrow \mathsf{hasFather}(x, y)$
$q_2(x) \leftarrow \exists y.\, \mathsf{hasFather}(x, y)$
$q_3(x) \leftarrow \exists y_1, y_2, y_3.\, \mathsf{hasFather}(x, y_1), \mathsf{hasFather}(y_1, y_2), \mathsf{hasFather}(y_2, y_3)$
$q_4(x, y_3) \leftarrow \exists y_1, y_2.\, \mathsf{hasFather}(x, y_1), \mathsf{hasFather}(y_1, y_2), \mathsf{hasFather}(y_2, y_3)$

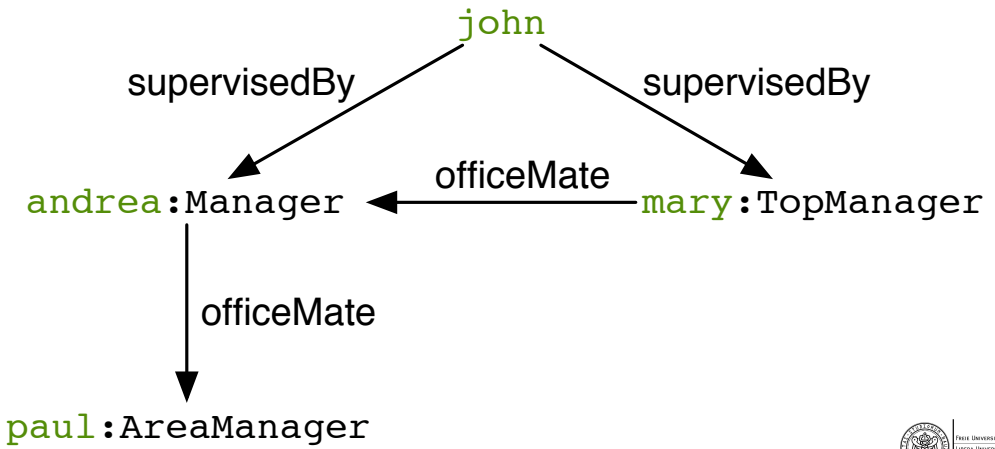
Answers: to q_1: { (john,nick), (nick,toni) }
to q_2: { john, nick, toni }
to q_3: { john, nick, toni }
to q_4: { }

QA in ontologies – Andrea's Example(*)



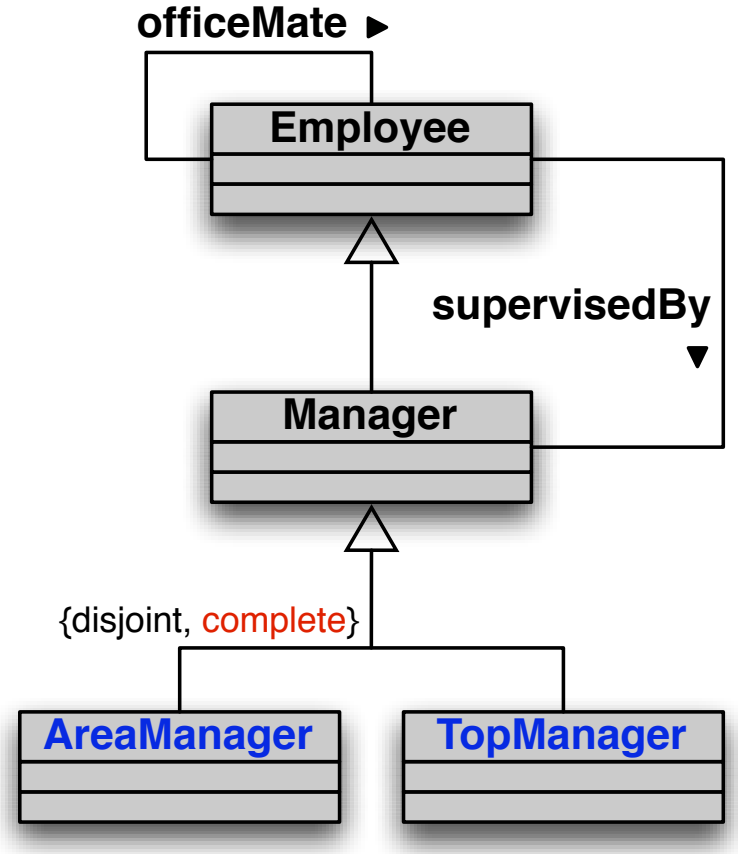
Manager is **partitioned into** AreaManager and TopManager.

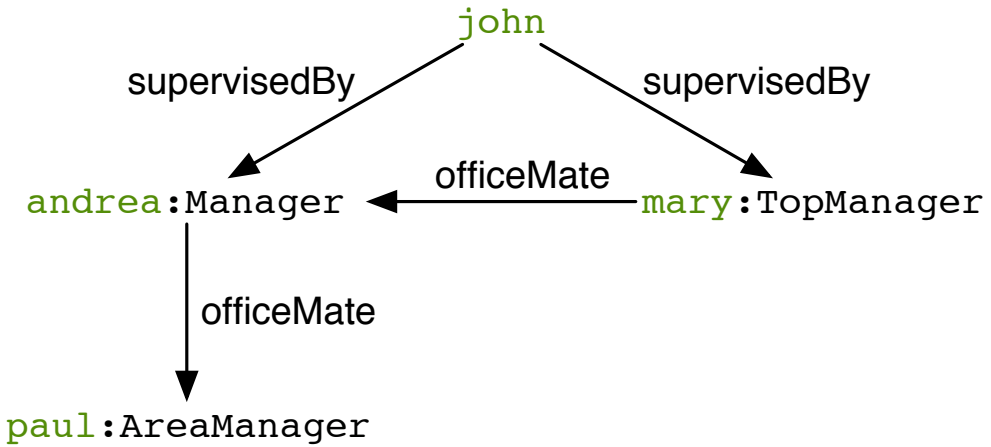
$$\begin{aligned}
\text{Employee} &\supseteq \{\ \texttt{andrea}, \texttt{nick}, \texttt{mary}, \texttt{john}\ \} \\
\text{Manager} &\supseteq \{\ \texttt{andrea}, \texttt{nick}, \texttt{mary}\ \} \\
\text{AreaManager} &\supseteq \{\ \texttt{nick}\ \} \\
\text{TopManager} &\supseteq \{\ \texttt{mary}\ \} \\
\text{supervisedBy} &\supseteq \{\ (\texttt{john},\texttt{andrea}), (\texttt{john},\texttt{mary})\ \} \\
\text{officeMate} &\supseteq \{\ (\texttt{mary},\texttt{andrea}), (\texttt{andrea},\texttt{nick})\ \}
\end{aligned}$$



(*) Due to [Sch93].

QA in ontologies – Andrea's Example (cont'd)





$$q(x) \leftarrow \exists y, z.\, \mathsf{supervisedBy}(x, y),\ \mathsf{TopManager}(y),\ \mathsf{officeMate}(y, z),\ \mathsf{AreaManager}(z)$$

Answer: { john }

To determine this answer, we need to resort to **reasoning by cases**.

Outline

QA in ontology-based information integration

In ontology-based information integration, we face the difficulties of both assumptions:

- The actual **data** is stored in external information sources (i.e., databases), and thus its size is typically **very large**.
- The global view which captures available information at the semantic/conceptual level has to deal with **incompleteness** of information, and we have to perform logical inference, rather than query evaluation.
- We want to take into account at **runtime** the **constraints** expressed in the global view.
- We want to answer **complex database-like queries**.
- We have to deal with multiple information sources, and thus face also the problems that are typical of data integration.

Next we will report on the state-of-the-art technology in this area.

Outline

Outline

What are Description Logics?

Description Logics (DLs) [BCM+03] are **logics** specifically designed to represent and reason on structured knowledge.

The domain of interest is composed of objects and is structured into:

- concepts, which correspond to classes, and denote sets of objects
- roles, which correspond to (binary) relationships, and denote binary relations on objects

The knowledge is asserted through so-called assertions, i.e., logical axioms.

Brief history of Description Logics

1977 *KL-ONE* Workshop: from Semantic Networks and Frames to Description Logics (aka Terminological Languages, Concept Languages)

1984 Trade-off expressiveness – complexity of inference [BL84]

1986 Description logics for conceptual modeling [Lenzerini, Borgida]

1989 *Classic* system [BBMR89] – polynomial inference, but no assertions

1990' Expressive DLs – tableaux [Baa90], correspondence with modal logic and PDLs [Sch91, DG95], automata [Cal96]

1995 Conceptual models fully captured in DLs [CDGL95]

1998 Optimized tableaux make expressive DLs practical [Hor98]
Query answering in DLs [CDGL98]

2000's Standardization efforts – OIL, DAML+OIL, OWL, OWL2

2005 Polynomial DLs with assertions – $\mathcal{EL}$ [BBL05], *DL-Lite* [CDGL$^+$05]

Current applications of Description Logics

DLs have evolved from being used "just" in KR.

Novel applications of DLs:

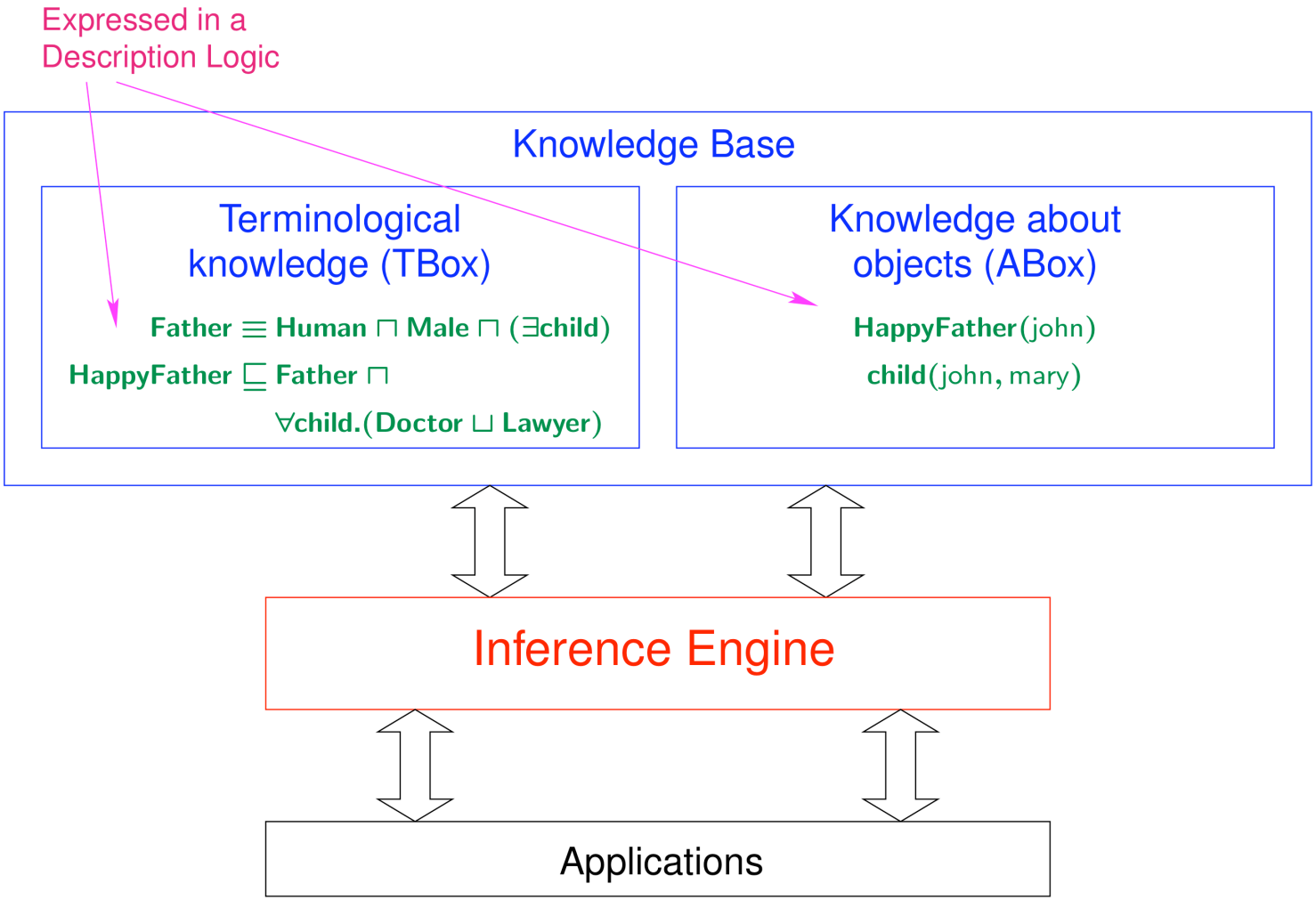
- Databases:
  - schema design, schema evolution
  - query optimization
  - integration of heterogeneous data sources, data warehousing
- Conceptual modeling
- Foundation for the Semantic Web (variants of OWL correspond to specific DLs)
- ...

Ingredients of a Description Logic

A **DL** is characterized by:

1. A **description language**: how to form concepts and roles
   $\mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
2. A mechanism to **specify knowledge** about concepts and roles (i.e., a TBox)
   $\mathcal{T} = \{ \mathsf{Father} \equiv \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild},$
   $\mathsf{HappyFather} \sqsubseteq \mathsf{Father} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer}) \}$
3. A mechanism to specify **properties of objects** (i.e., an ABox)
   $\mathcal{A} = \{ \mathsf{HappyFather}(\mathtt{john}), \quad \mathsf{hasChild}(\mathtt{john}, \mathtt{mary}) \}$
4. A set of **inference services**: how to reason on a given KB
   $\mathcal{T} \models \mathsf{HappyFather} \sqsubseteq \exists \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
   $\mathcal{T} \cup \mathcal{A} \models (\mathsf{Doctor} \sqcup \mathsf{Lawyer})(\mathtt{mary})$

Architecture of a Description Logic system



Outline

Description language

A description language provides the means for defining:

- concepts, corresponding to classes: interpreted as sets of objects;
- roles, corresponding to relationships: interpreted as binary relations on objects.

To define concepts and roles:

- We start from a (finite) alphabet of **atomic concepts** and **atomic roles**, i.e., simply names for concept and roles.
- Then, by applying specific **constructors**, we can build **complex concepts** and **roles**, starting from the atomic ones.

A **description language** is characterized by the set of constructs that are available for that.

Semantics of a description language

The **formal semantics** of DLs is given in terms of interpretations.

Def.: An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of:

- a nonempty set $\Delta^{\mathcal{I}}$, the domain of $\mathcal{I}$
- an interpretation function $\cdot^{\mathcal{I}}$, which maps
  - each individual a to an element $a^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - each atomic concept A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - each atomic role P to a subset $P^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

The interpretation function is extended to complex concepts and roles according to their syntactic structure.

Concept constructors

Construct	Syntax	Example	Semantics
atomic concept	A	Doctor	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	P	hasChild	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
atomic negation	$\neg A$	¬Doctor	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conjunction	$C \sqcap D$	Hum ⊓ Male	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
(unqual.) exist. res.	$\exists R$	∃hasChild	$\{ a \mid \exists b. (a, b) \in R^{\mathcal{I}} \}$
value restriction	$\forall R.C$	∀hasChild.Male	$\{a \mid \forall b. (a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
bottom	$\bot$		$\emptyset$

(C, D denote arbitrary concepts and R an arbitrary role)

The above constructs form the basic language $\mathcal{AL}$ of the family of $\mathcal{AL}$ languages.

Further examples of DL constructs

- Disjunction $\mathcal{U}$: $\mathsf{Doctor} \sqcup \mathsf{Lawyer}$
- Qualified existential restriction $\mathcal{E}$: $\exists \mathsf{hasChild}.\mathsf{Doctor}$
- Full negation $\mathcal{C}$: $\neg(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
- Number restrictions $\mathcal{N}$: $(\geq 2\,\mathsf{hasChild})$ $(\leq 1\,\mathsf{sibling})$
- Qualified number restrictions $\mathcal{Q}$: $(\geq 2\,\mathsf{hasChild}.\,\mathsf{Doctor})$
- Inverse role $\mathcal{I}$: $\exists \mathsf{hasChild}^{-}.\mathsf{Doctor}$
- Reflexive-transitive role closure $_{reg}$: $\exists \mathsf{hasChild}^{*}.\mathsf{Doctor}$

Structural properties vs. asserted properties

We have seen how to build complex **concept and roles expressions**, which allow one to denote classes with a complex structure.

However, in order to represent real world domains, one needs the ability to **assert properties** of classes and relationships between them (e.g., as done in UML class diagrams).

The assertion of properties is done in DLs by means of an **ontology**.

Description Logics ontology

Is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a **TBox** and $\mathcal{A}$ is an **ABox**:

Def.: Description Logics **TBox**

Consists of a set of **assertions** on concepts and roles:

- Inclusion assertions on concepts: $C_1 \sqsubseteq C_2$
- Inclusion assertions on roles: $R_1 \sqsubseteq R_2$
- Property assertions on (atomic) roles:
  (**transitive** P) (**symmetric** P) (**domain** P C)
  (**functional** P) (**reflexive** P) (**range** P C) $\cdots$

Def.: Description Logics **ABox**

Consists of a set of **membership assertions** on individuals:

- for concepts: $A(c)$
- for roles: $P(c_1, c_2)$

(we use c_i to denote individuals)

Description Logics ontology – Example

Note: We use $C_1 \equiv C_2$ as an abbreviation for $C_1 \sqsubseteq C_2,\ C_2 \sqsubseteq C_1$.

TBox assertions:

- Inclusion assertions on concepts:

$$\begin{aligned} \text{Father} &\equiv \text{Human} \sqcap \text{Male} \sqcap \exists \text{hasChild} \\ \text{HappyFather} &\sqsubseteq \text{Father} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{HappyPerson}) \\ \text{HappyAnc} &\sqsubseteq \forall \text{descendant}.\text{HappyFather} \\ \text{Teacher} &\sqsubseteq \neg\text{Doctor} \sqcap \neg\text{Lawyer} \end{aligned}$$

- Inclusion assertions on roles:

$$\text{hasChild} \sqsubseteq \text{descendant} \qquad \text{hasFather} \sqsubseteq \text{hasChild}^-$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant), (**functional** hasFather)

ABox membership assertions:

- $\text{Teacher}(\texttt{mary})$, $\text{hasFather}(\texttt{mary}, \texttt{john})$, $\text{HappyAnc}(\texttt{john})$

Semantics of a Description Logics ontology

The semantics is given by specifying when an interpretation $\mathcal{I}$ **satisfies** an assertion:

- $C_1 \sqsubseteq C_2$ is satisfied by $\mathcal{I}$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- $R_1 \sqsubseteq R_2$ is satisfied by $\mathcal{I}$ if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$.
- A property assertion (**prop** P) is satisfied by $\mathcal{I}$ if $P^{\mathcal{I}}$ is a relation that has the property **prop**.
- $A(c)$ is satisfied by $\mathcal{I}$ if $c^{\mathcal{I}} \in A^{\mathcal{I}}$.
- $P(c_1, c_2)$ is satisfied by $\mathcal{I}$ if $(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

This leads to the notion of:

Def.: Model of a DL ontology

An interpretation $\mathcal{I}$ is a **model** of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ if it satisfies all assertions in $\mathcal{T}$ and all assertions in $\mathcal{A}$.

Encoding UML Class Diagrams in DLs

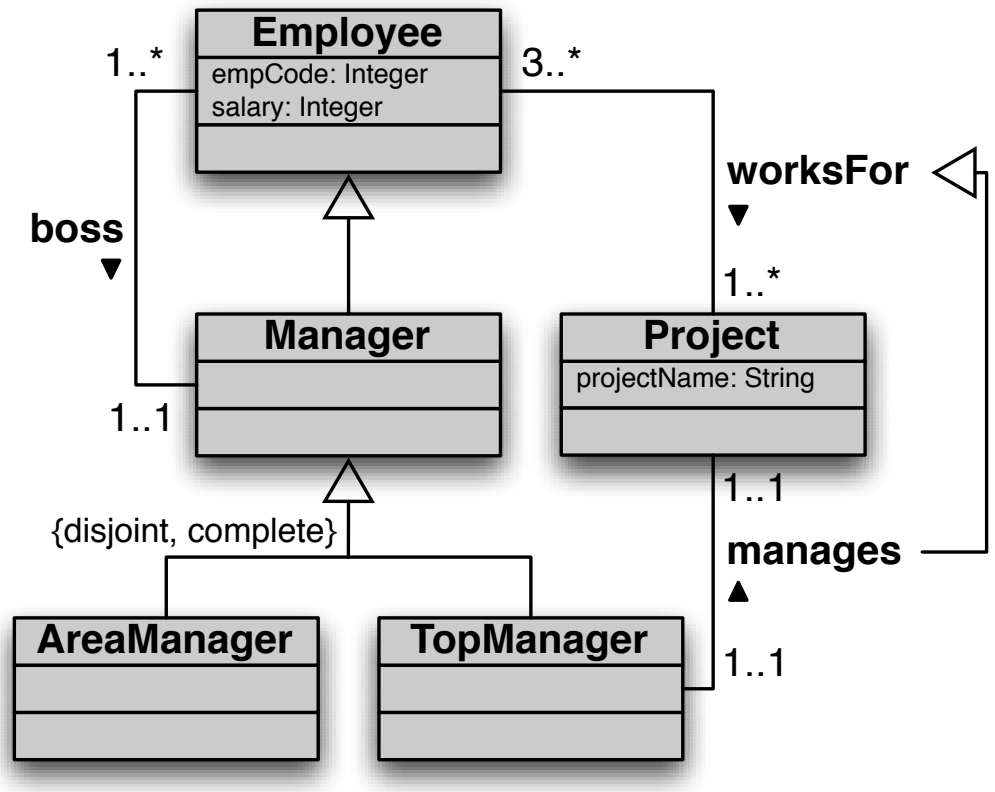
There is a tight correspondence between variants of DLs and UML Class Diagrams [BCDG05].

UML Class Diagrams can be quite naturally encoded in DL TBoxes:

- Each class is represented by an atomic concept.
- Each attribute is represented by a role (or an OWL data property).
- Each binary association is represented by a role (or an OWL object property).
- Each non-binary association is reified, i.e., represented as a concept connected to its components by roles.
- Each part of the diagram is encoded by suitable assertions.

We illustrate the encoding by means of an example.

Encoding UML Class Diagrams in DLs – Example



$$
\begin{aligned}
\text{Manager} &\sqsubseteq \text{Employee}\\
\text{AreaManager} &\sqsubseteq \text{Manager}\\
\text{TopManager} &\sqsubseteq \text{Manager}\\
\text{Manager} &\sqsubseteq \text{AreaManager} \sqcup \text{TopManager}\\
\text{AreaManager} &\sqsubseteq \neg\text{TopManager}\\
\text{Employee} &\sqsubseteq \exists \text{salary}\\
\exists \text{salary}^- &\sqsubseteq \text{Integer}\\
\exists \text{worksFor} &\sqsubseteq \text{Employee}\\
\exists \text{worksFor}^- &\sqsubseteq \text{Project}\\
\text{Employee} &\sqsubseteq \exists \text{worksFor}\\
\text{Project} &\sqsubseteq (\geq 3\, \text{worksFor}^-)\\
&(\textbf{funct}\ \text{manages})\\
&(\textbf{funct}\ \text{manages}^-)\\
\text{manages} &\sqsubseteq \text{worksFor}\\
&\ldots
\end{aligned}
$$

Note: Domain and range of associations are expressed by means of concept inclusions.

Outline

DLs vs. OWL

DLs provide the foundations for standard ontology languages.

Different versions of the W3C standard **Web Ontology Language (OWL)** have been defined as syntactic variants of certain DLs:

- **OWL1 Lite** is a variant of the DL $\mathcal{SHIF}(D)$, where:
  - $\mathcal{S}$ stands for $\mathcal{ALC}$ extended with transitive roles,
  - $\mathcal{H}$ stands for role hierarchies (i.e., role inclusion assertions),
  - $\mathcal{I}$ stands for inverse roles,
  - $\mathcal{F}$ stands for functionality of roles,
  - (D) stand for data types, which are necessary in any practical knowledge representation language.
- **OWL1 DL** is a variant of $\mathcal{SHOIN}(D)$, where:
  - $\mathcal{O}$ stands for nominals, which means the possibility of using individuals in the TBox (i.e., the intensional part of the ontology),
  - $\mathcal{N}$ stands for (unqualified) number restrictions.

Description Logics vs. OWL2

- A new version of OWL, **OWL2**, is currently being standardized by the W3C.
- The design aim of OWL2 was to address user requirements for more expressivity of the language, while still preserving decidability of reasoning.
- **OWL2 DL** is a variant of $\mathcal{SROIQ}(D)$, which adds to OWL1 DL several features:
  - qualified number restrictions ($\mathcal{Q}$)
  - regular role hierarchies ($\mathcal{R}$)
  - better treatment of datatypes

DL constructs vs. OWL constructs

OWL contructor	DL constructor	Example
`ObjectIntersectionOf`	$C_1 \sqcap \cdots \sqcap C_n$	Human $\sqcap$ Male
`ObjectUnionOf`	$C_1 \sqcup \cdots \sqcup C_n$	Doctor $\sqcup$ Lawyer
`ObjectComplementOf`	$\neg C$	$\neg$Male
`ObjectOneOf`	$\{a_1\} \sqcup \cdots \sqcup \{a_n\}$	$\{$john$\} \sqcup \{$mary$\}$
`ObjectAllValuesFrom`	$\forall P.C$	$\forall$hasChild.Doctor
`ObjectSomeValuesFrom`	$\exists P.C$	$\exists$hasChild.Lawyer
`ObjectMaxCardinality`	$(\leq n\, P)$	$(\leq 1$ hasChild$)$
`ObjectMinCardinality`	$(\geq n\, P)$	$(\geq 2$ hasChild$)$

...

Note: all constructs come also in the `Data...` instead of `Object...` variant.

DL axioms vs. OWL axioms

OWL axiom	DL syntax	Example
`SubClassOf`	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
`EquivalentClasses`	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
`DisjointClasseses`	$C_1 \sqsubseteq \neg C_2$	Man $\sqsubseteq \neg$Female
`SameIndividual`	$\{a_1\} \equiv \{a_2\}$	$\{$presBush$\} \equiv \{$G.W.Bush$\}$
`DifferentIndividuals`	$\{a_1\} \sqsubseteq \neg\{a_2\}$	$\{$john$\} \sqsubseteq \neg\{$peter$\}$
`SubObjectPropertyOf`	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
`EquivalentObjectProperties`	$P_1 \equiv P_2$	hasCost $\equiv$ hasPrice
`InverseObjectProperties`	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
`TransitiveObjectProperty`	$P^+ \sqsubseteq P$	ancestor$^+ \sqsubseteq$ ancestor
`FunctionalObjectProperty`	$\top \sqsubseteq (\leq 1\, P)$	$\top \sqsubseteq (\leq 1$ hasFather$)$

...

Outline

TBox reasoning

- **Concept Satisfiability**:
  C is satisfiable wrt $\mathcal{T}$, if $C^{\mathcal{I}}$ is not empty for some model $\mathcal{I}$ of $\mathcal{T}$.
- **Subsumption**:
  C_1 is subsumed by C_2 wrt $\mathcal{T}$, if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.
- **Equivalence**:
  C_1 and C_2 are equivalent wrt $\mathcal{T}$, if $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.
- **Disjointness**:
  C_1 and C_2 are disjoint wrt $\mathcal{T}$, if $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$.

Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.

Reasoning over an ontology

- **Ontology Satisfiability:** Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.
- **Concept Instance Checking:** Verify whether an individual c is an instance of a concept C in every model of $\mathcal{O}$.
- **Role Instance Checking:** Verify whether a pair (c_1, c_2) of individuals is an instance of a role R in every model of $\mathcal{O}$.
- **Query Answering:** see later ...

Reasoning in Description Logics – Example

TBox:

- Inclusion assertions on concepts:

$$\begin{aligned} \text{Father} &\equiv \text{Human} \sqcap \text{Male} \sqcap \exists \text{hasChild} \\ \text{HappyFather} &\sqsubseteq \text{Father} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{HappyPerson}) \\ \text{HappyAnc} &\sqsubseteq \forall \text{descendant}.\text{HappyFather} \\ \text{Teacher} &\sqsubseteq \neg\text{Doctor} \sqcap \neg\text{Lawyer} \end{aligned}$$

- Inclusion assertions on roles:

$$\text{hasChild} \sqsubseteq \text{descendant} \qquad \text{hasFather} \sqsubseteq \text{hasChild}^{-}$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant), (**functional** hasFather)

The above TBox logically implies: $\text{HappyAncestor} \sqsubseteq \text{Father}$.

- Membership assertions:
  $\text{Teacher}(\texttt{mary})$, $\text{hasFather}(\texttt{mary}, \texttt{john})$, $\text{HappyAnc}(\texttt{john})$

The above TBox and ABox logically imply: $\text{HappyPerson}(\texttt{mary})$

Complexity of reasoning over DL ontologies

Reasoning over DL ontologies is in general a complex task:

- Reasoning over ontologies in virtually all traditional DLs is ExpTime-**hard** (see, e.g., [Don03]).
- Stays in ExpTime even in the most expressive DLs (except when using nominals, i.e., `ObjectOneOf`).
- There are DL reasoners that perform reasonably well in practice for such DLs (e.g, Racer, Pellet, Fact++, ...) [MH03].

Reasoning over UML class diagrams can be done in ExpTime by using DLs.

Outline

Outline

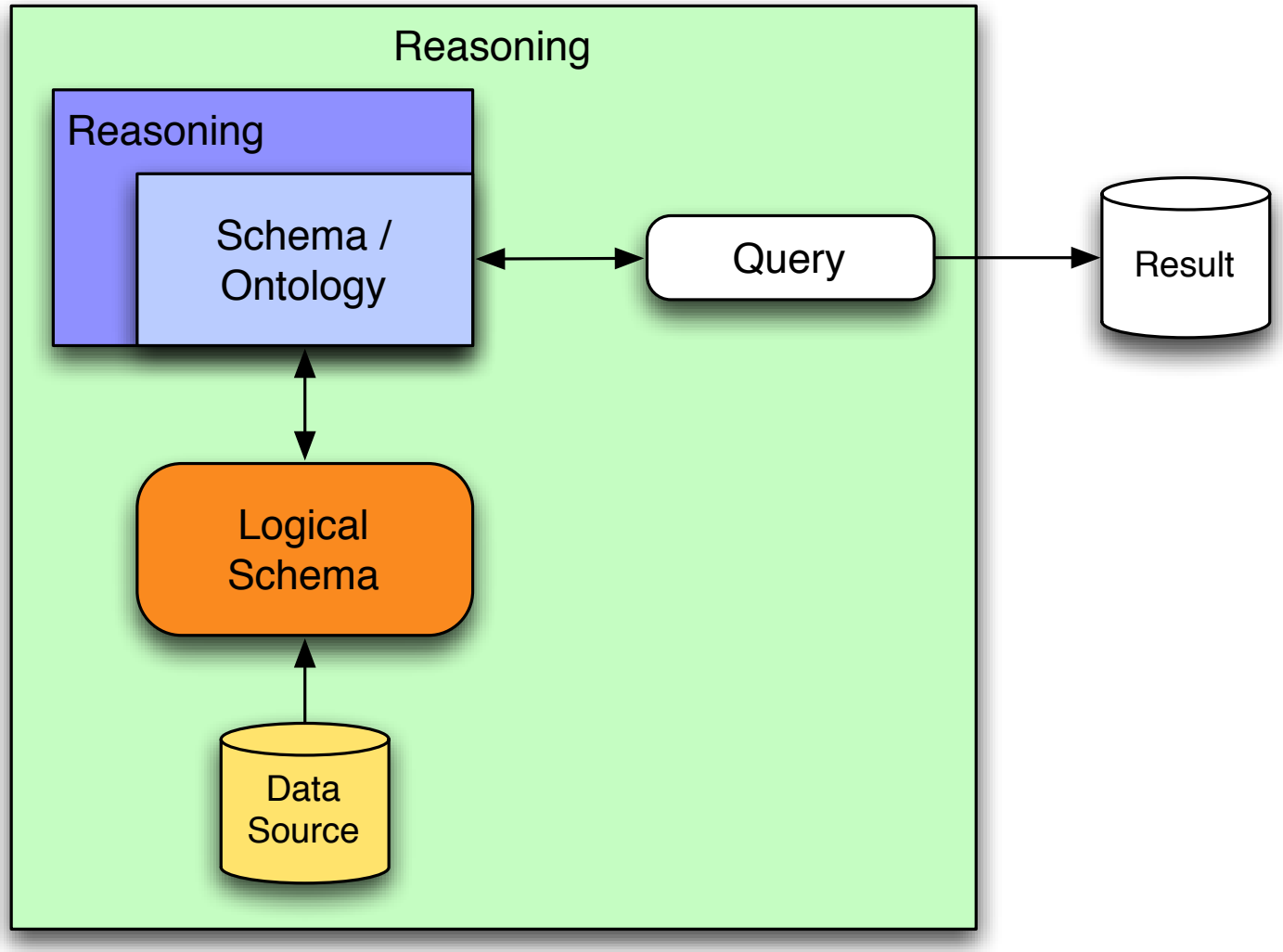
Query answering over ontologies

- An ontology imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

$\rightsquigarrow$ Query answering amounts to **logical inference**, which is computationally more costly than query evaluation in databases.

In other words, we are interested in finding those answers to a query that hold in all models of an ontology, i.e., the **certain answers**.

Query answering over ontologies (cont'd)



Questions that need to be addressed

In the context of Ontology-Based Data Integration:

1. Which is the "right" **query language**?
2. Which is the "right" **ontology language**?
3. How can we bridge the **semantic mismatch** between the ontology and the data sources?
4. How can **tools for ontology-based data integration** take into account these issues?

Which language to use for querying ontologies?

Two borderline cases:

1. Just classes and properties of the ontology $\rightsquigarrow$ instance checking
   - Ontology languages are tailored for capturing intensional relationships.
   - They are quite **poor as query languages**:
     Cannot refer to same object via multiple navigation paths in the ontology, i.e., allow only for a limited form of JOIN, namely chaining.

2. Full SQL (or equivalently, first-order logic)
   - Problem: in the presence of incomplete information, query answering becomes **undecidable** (FOL validity).

Unions of conjunctive queries

A good tradeoff is to use (unions of) **conjunctive queries** (UCQs):

- A (U)CQ is a first-order query using only conjunction, existential quantification (and disjunction).
- Hence, UCQs contain no negation, no universal quantification, and no function symbols besides constants.
- Correspond to SQL/relational algebra **(union) select-project-join (SPJ) queries** – the most frequently asked queries.
- For (U)CQs over an ontology, the predicates in atoms are concepts and roles of the ontology.

Queries over Description Logics ontologies

Def.: A conjunctive query $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$

has the form $q(\vec{x}) \leftarrow \exists \vec{y}.\, conj(\vec{x}, \vec{y})$ where $conj(\vec{x}, \vec{y})$ is a conjunction of atoms

- that has **as predicate symbol an atomic concept or role** of $\mathcal{T}$, and
- may use variables in $\vec{x}$ and $\vec{y}$, and constants that are individuals of $\mathcal{A}$.

The **answer** to $q(\vec{x})$ over $\mathcal{I}$, denoted $q^{\mathcal{I}}$ is the set of **tuples $\vec{c}$ of constants of** $\mathcal{A}$ such that the formula $\exists \vec{y}.\, conj(\vec{c}, \vec{y})$ evaluates to true in $\mathcal{I}$.

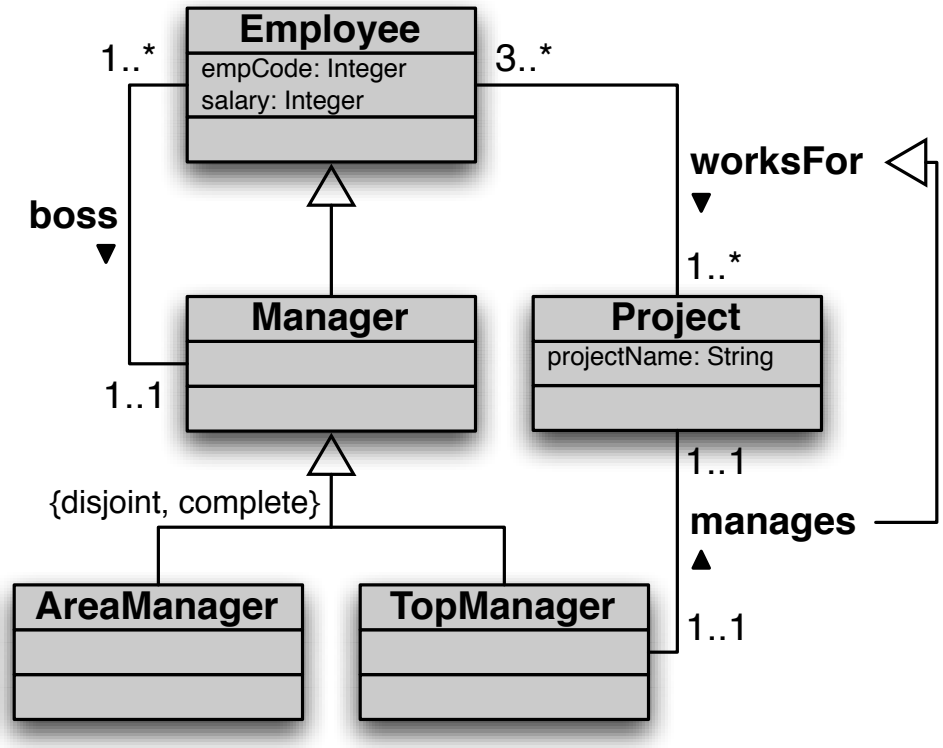
An UCQ is a union of CQs.

We are interested in finding those answers that hold in all models of an ontology.

Def.: The certain answers to $q(\vec{x})$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, denoted $cert(q, \mathcal{O})$

are the **tuples $\vec{c}$ of constants of** $\mathcal{A}$ such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}$.

Queries over Description Logics ontologies – Example



Conjunctive query over the above ontology:

$$q(x, y) \leftarrow \exists p.\, \mathsf{Employee}(x), \mathsf{Employee}(y), \mathsf{Project}(p), \mathsf{boss}(x, y), \mathsf{worksFor}(x, p), \mathsf{worksFor}(y, p)$$

Data complexity

Various parameters affect the complexity of query answering over an ontology.

Depending on which parameters we consider, we get different complexity measures:

- **Data complexity**: only the size of the ABox (i.e., the data) matters. TBox and query are considered fixed.
- **Schema complexity**: only the size of the TBox (i.e., the schema) matters. ABox and query are considered fixed.
- **Combined complexity**: no parameter is considered fixed.

In the integration setting, **the size of the data largely dominates** the size of the conceptual layer (and of the query).

$\rightsquigarrow$ **Data complexity** is the relevant complexity measure.

Complexity of query answering in DLs

Problem of rewriting is related to **complexity of query answering**.

Studied extensively for (unions of) CQs and various ontology languages:

	Combined complexity	Data complexity
Plain databases	NP-complete	in AC^0 (2)
OWL 2 (and less)	2ExpTime-complete	coNP-hard (1)

(1) Already for a TBox with a single disjunction.
(2) This is what we need to scale with the data.

Questions

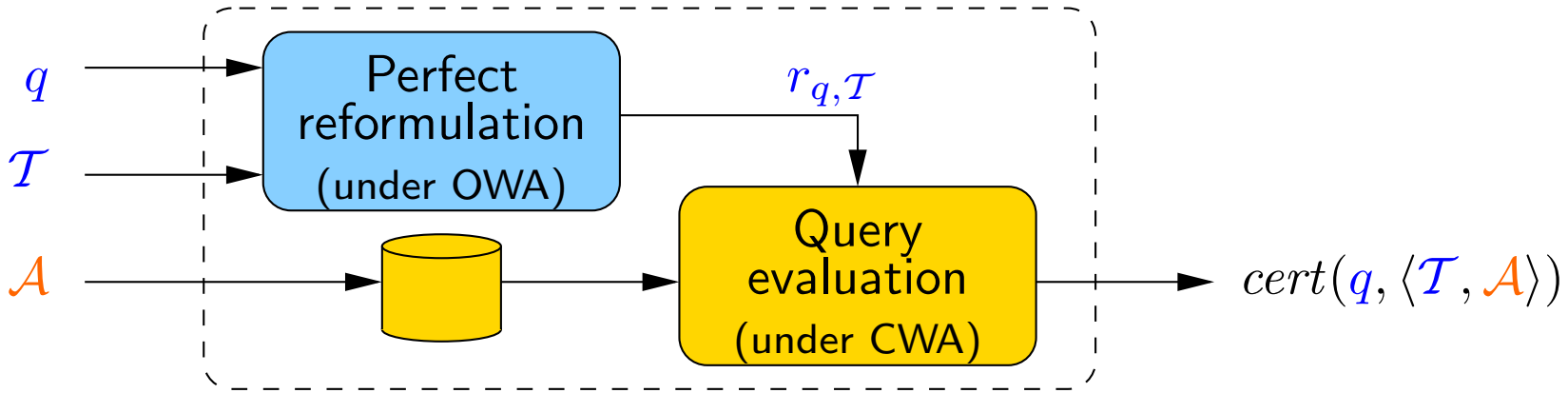
- Can we find interesting (description) logics for which query answering can be done efficiently (i.e., in AC^0)?
- If yes, can we leverage relational database technology for query answering?

Inference in query answering

To be able to deal with data efficiently, we need to separate the contribution of $\mathcal{A}$ from the contribution of q and $\mathcal{T}$.

$\rightsquigarrow$ Query answering by **query rewriting**.

Query rewriting



Query answering can **always** be thought as done in two phases:

1. **Perfect rewriting**: produce from q and the TBox $\mathcal{T}$ a new query $r_{q,\mathcal{T}}$ (called the perfect rewriting of q w.r.t. $\mathcal{T}$).
2. **Query evaluation**: evaluate $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ seen as a complete database (and without considering the TBox $\mathcal{T}$).
   $\rightsquigarrow$ Produces $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$.

Note: The "always" holds if we pose no restriction on the language in which to express the rewriting $r_{q,\mathcal{T}}$.

Language of the rewriting

The **expressiveness of the ontology language affects the rewriting language**, i.e., the language into which we are able to rewrite UCQs:

- When we can rewrite into FOL/SQL.
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an RDBMS (*Note:* FOL is in AC^0).
- When we can rewrite into an NLogSpace-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\rightsquigarrow$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) power of Disjunctive Datalog.

Outline

The *DL-Lite* family

- A family of DLs optimized according to the tradeoff between expressive power and **complexity** of query answering, with emphasis on **data**.
- Carefully designed to have nice computational properties for answering UCQs (i.e., computing certain answers):
  - The same data complexity as relational databases.
  - In fact, query answering can be delegated to a relational DB engine.
  - The DLs of the *DL-Lite* family are essentially the maximally expressive ontology languages enjoying these nice computational properties.
- Captures conceptual modeling formalism.

The *DL-Lite* family provides new foundations for Ontology-Based Data Integration.

Basic features of *DL-Lite*$_{\mathcal{A}}$

DL-Lite$_{\mathcal{A}}$ is an expressive member of the *DL-Lite* family.

- Takes into account the distinction between **objects** and **values**:
  - Objects are elements of an abstract interpretation domain.
  - Values are elements of concrete data types, such as integers, strings, ecc.
  - Values are connected to objects through **attributes** (rather than roles).
- Is equipped with identification constraints.
- Captures most of UML class diagrams and Extended ER diagrams.
- Enjoys **FOL-rewritability**, and hence is AC^0 in data complexity.

Syntax of the *DL-Lite*$_{\mathcal{A}}$ description language

- Concept expressions: atomic concept A

$$\begin{array}{lcl} B & \longrightarrow & A \mid \exists Q \mid \delta(U) \\ C & \longrightarrow & \top_C \mid B \mid \neg B \end{array}$$

- Role expressions: atomic role P

$$\begin{array}{lcl} Q & \longrightarrow & P \mid P^- \\ R & \longrightarrow & Q \mid \neg Q \end{array}$$

- Value-domain expressions: each T_i is one of the RDF datatypes

$$\begin{array}{lcl} E & \longrightarrow & \rho(U) \\ F & \longrightarrow & \top_D \mid T_1 \mid \cdots \mid T_n \end{array}$$

- Attribute expressions: atomic attribute U

$$V \longrightarrow U \mid \neg U$$

Semantics of *DL-Lite*$_\mathcal{A}$ – Objects vs. values

	Objects	Values
Interpretation domain $\Delta^{\mathcal{I}}$	Domain of objects $\Delta_O^{\mathcal{I}}$	Domain of values $\Delta_V^{\mathcal{I}}$
Alphabet Γ of constants	Object constants Γ_O $c^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$	Value constants Γ_V $d^{\mathcal{I}} = \mathit{val}(d)$ given a priori
Unary predicates	Concept C $C^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$	RDF datatype T_i $T_i^{\mathcal{I}} \in \Delta_V^{\mathcal{I}}$ is given a priori
Binary predicates	Role R $R^{\mathcal{I}} \in \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$	Attribute V $V^{\mathcal{I}} \in \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$

Semantics of the *DL-Lite*$_\mathcal{A}$ constructs

Construct	Syntax	Example	Semantics
top concept	$\top_C$		$\top_C^\mathcal{I} = \Delta_O^\mathcal{I}$
atomic concept	A	Doctor	$A^\mathcal{I} \subseteq \Delta_O^\mathcal{I}$
existential restriction	$\exists Q$	$\exists$child$^-$	$\{o \mid \exists o'.\, (o, o') \in Q^\mathcal{I}\}$
concept negation	$\neg B$	$\neg\exists$child	$\Delta^\mathcal{I} \setminus B^\mathcal{I}$
attribute domain	$\delta(U)$	δ(salary)	$\{o \mid \exists v.\, (o, v) \in U^\mathcal{I}\}$
atomic role	P	child	$P^\mathcal{I} \subseteq \Delta_O^\mathcal{I} \times \Delta_O^\mathcal{I}$
inverse role	P^-	child$^-$	$\{(o, o') \mid (o', o) \in P^\mathcal{I}\}$
role negation	$\neg Q$	$\neg$manages	$(\Delta_O^\mathcal{I} \times \Delta_O^\mathcal{I}) \setminus Q^\mathcal{I}$
top domain	$\top_D$		$\top_D^\mathcal{I} = \Delta_V^\mathcal{I}$
datatype	T_i	`xsd:int`	$val(T_i) \subseteq \Delta_V^\mathcal{I}$
attribute range	$\rho(U)$	ρ(salary)	$\{v \mid \exists o.\, (o, v) \in U^\mathcal{I}\}$
atomic attribute	U	salary	$U^\mathcal{I} \subseteq \Delta_O^\mathcal{I} \times \Delta_V^\mathcal{I}$
attribute negation	$\neg U$	$\neg$salary	$(\Delta_O^\mathcal{I} \times \Delta_V^\mathcal{I}) \setminus U^\mathcal{I}$
object constant	c	`john`	$c^\mathcal{I} \in \Delta_O^\mathcal{I}$
value constant	d	`'john'`	$val(d) \in \Delta_V^\mathcal{I}$

DL-Lite$_{\mathcal{A}}$ assertions

TBox assertions can have the following forms:

- Inclusion assertions:

B	$\sqsubseteq$	C	concept inclusion	$E \sqsubseteq F$	value-domain inclusion
Q	$\sqsubseteq$	R	role inclusion	$U \sqsubseteq V$	attribute inclusion

- Functionality assertions:

$(\textbf{funct}\ Q)$ role functionality $(\textbf{funct}\ U)$ attribute functionality

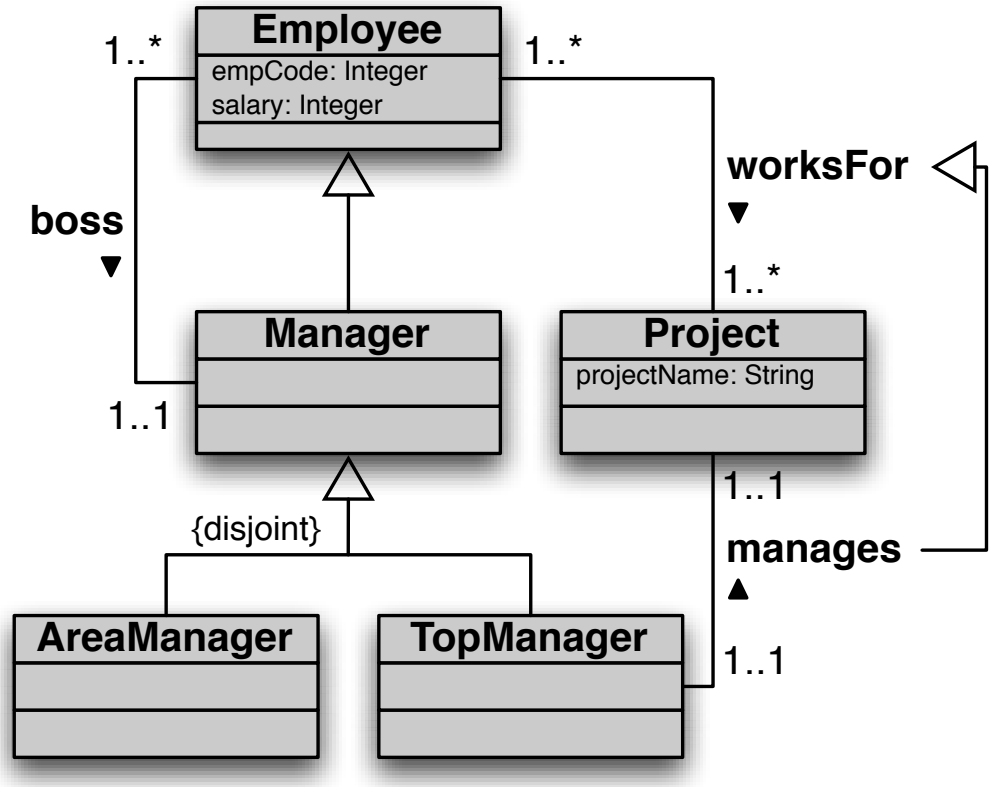
- Identification constraints: $(\textbf{id}\ B\ I_1, \ldots, I_n)$
  where each I_j is a role, an inverse role, or an attribute

ABox assertions: $A(c)$, $P(c, c')$, $U(c, d)$,
where c, c' are object constants and d is a value constant

Semantics of the *DL-Lite*$_{\mathcal{A}}$ assertions

Assertion	Syntax	Example	Semantics
conc. incl.	$B \sqsubseteq C$	Father $\sqsubseteq \exists$child	$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
role incl.	$Q \sqsubseteq R$	father $\sqsubseteq$ anc	$Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
v.dom. incl.	$E \sqsubseteq F$	ρ(age) $\sqsubseteq$ `xsd:int`	$E^{\mathcal{I}} \subseteq F^{\mathcal{I}}$
attr. incl.	$U \sqsubseteq V$	offPhone $\sqsubseteq$ phone	$U^{\mathcal{I}} \subseteq V^{\mathcal{I}}$
role funct.	(**funct** Q)	(**funct** father)	$\forall o, o_1, o_2.(o, o_1) \in Q^{\mathcal{I}} \wedge (o, o_2) \in Q^{\mathcal{I}} \rightarrow o_1 = o_2$
att. funct.	(**funct** U)	(**funct** ssn)	$\forall o, v, v'.(o, v) \in U^{\mathcal{I}} \wedge (o, v') \in U^{\mathcal{I}} \rightarrow v = v'$
id const.	(**id** B $I_1, \ldots, I_n$)	(**id** Person name, dob)	$I_1, \ldots, I_n$ identify instances of B
mem. asser.	$A(c)$	Father(`bob`)	$c^{\mathcal{I}} \in A^{\mathcal{I}}$
mem. asser.	$P(c_1, c_2)$	child(`bob`, `ann`)	$(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$
mem. asser.	$U(c, d)$	phone(`bob`, `'2345'`)	$(c^{\mathcal{I}}, \mathit{val}(d)) \in U^{\mathcal{I}}$

$DL\text{-}Lite_{\mathcal{A}}$ – Example



$$
\begin{aligned}
\text{Manager} &\sqsubseteq \text{Employee}\\
\text{AreaManager} &\sqsubseteq \text{Manager}\\
\text{TopManager} &\sqsubseteq \text{Manager}\\
\text{AreaManager} &\sqsubseteq \neg\text{TopManager}\\
\text{Employee} &\sqsubseteq \delta(\text{empCode})\\
\delta(\text{empCode}) &\sqsubseteq \text{Employee}\\
\rho(\text{empCode}) &\sqsubseteq \texttt{xsd:int}\\
&(\textbf{funct}\ \text{empCode})\\
&(\textbf{id}\ \text{Employee empCode})\\
\exists\text{worksFor} &\sqsubseteq \text{Employee}\\
\exists\text{worksFor}^{-} &\sqsubseteq \text{Project}\\
\text{Employee} &\sqsubseteq \exists\text{worksFor}\\
\text{Project} &\sqsubseteq \exists\text{worksFor}^{-}\\
&(\textbf{funct}\ \text{manages})\\
&(\textbf{funct}\ \text{manages}^{-})\\
\text{manages} &\sqsubseteq \text{worksFor}\\
&\vdots
\end{aligned}
$$

Note: $DL\text{-}Lite_{\mathcal{A}}$ cannot capture completeness of a hierarchy. This would require **disjunction** (i.e., **OR**).

Capturing basic ontology constructs in *DL-Lite*$_{\mathcal{A}}$

ISA between classes	$A_1 \sqsubseteq A_2$
Disjointness between classes	$A_1 \sqsubseteq \neg A_2$
Mandatory participation to relations	$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$
Domain and range of relations	$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$
Functionality of relations	(**funct** P) (**funct** P^-)
ISA between relations	$Q_1 \sqsubseteq Q_2$
Disjointness between relations	$Q_1 \sqsubseteq \neg Q_2$
Domain and range of attributes	$\delta(U) \sqsubseteq A \quad \rho(U) \sqsubseteq T_i$
Mandatory and functional attributes	$A \sqsubseteq \delta(U)$ (**funct** U)
Identification constraints	(**id** $A\ P, \ldots, P'^-, \ldots, U, \ldots$)

Observations on *DL-Lite*$_{\mathcal{A}}$

- Captures all the basic constructs of **UML Class Diagrams** and of the **ER Model** ...
- ... **except covering constraints** in generalizations.
- Extends (the DL fragment of) the ontology language **RDFS**.
- Is completely symmetric w.r.t. **direct and inverse properties**.
- Is at the basis of the **OWL2 QL** profile of OWL2.

The OWL2 QL Profile

OWL2 defines three **profiles**: OWL2 QL, OWL2 EL, OWL2 RL.

- Each profile corresponds to a syntactic fragment (i.e., a sub-language) of OWL2 DL that is targeted towards a specific use.
- The restrictions in each profile guarantee better computational properties than those of OWL2 DL.

The **OWL2 QL** profile is derived from the DLs of the *DL-Lite* family:

- "[It] includes most of the main features of conceptual models such as UML class diagrams and ER diagrams."
- "[It] is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL2 QL, conjunctive query answering can be implemented using conventional relational database systems."

Restriction on TBox assertions in *DL-Lite*$_\mathcal{A}$ ontologies

To ensure FOL-rewritability, we have to impose a **restriction** on the use of functionality and role/attribute inclusions.

Restriction on *DL-Lite*$_\mathcal{A}$ TBoxes

No functional or identifying role or attribute can be specialized by using it in the right-hand side of a role or attribute inclusion assertion.

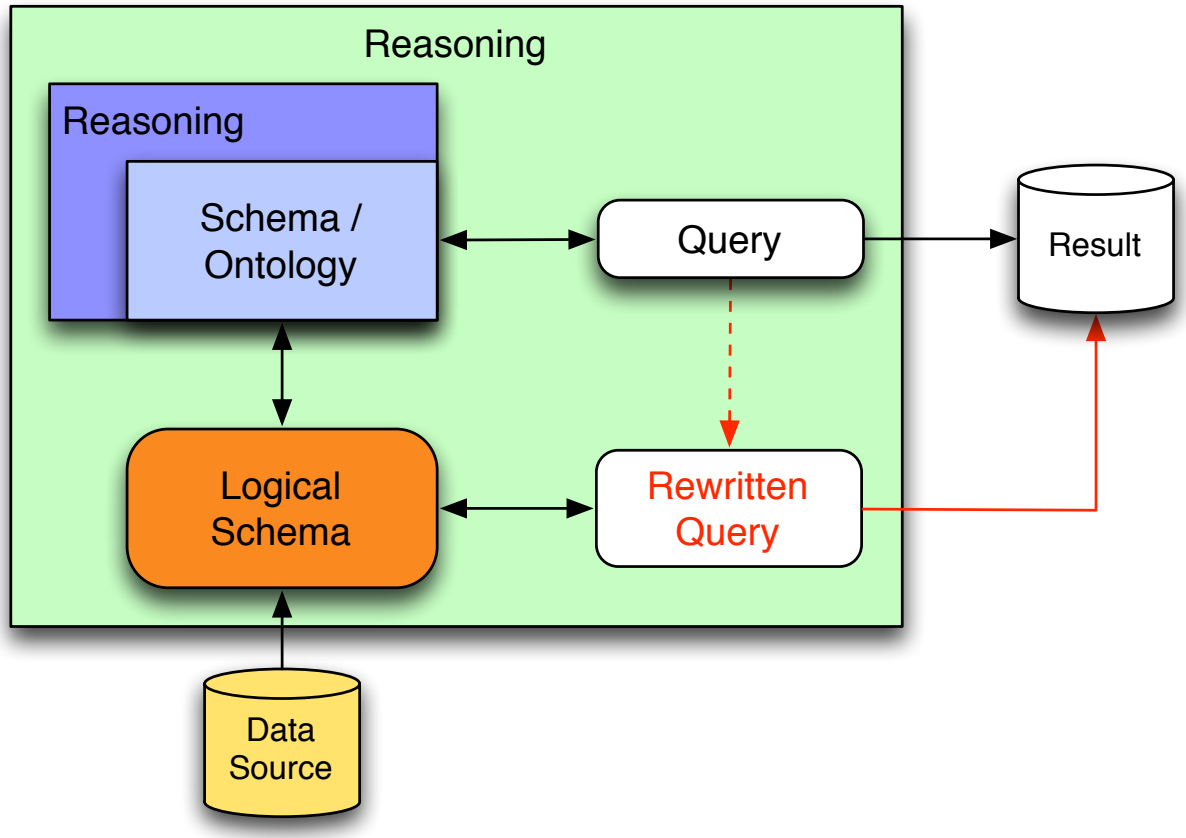
Formally:

- If $Q \sqsubseteq P$, or $Q \sqsubseteq P^-$, or (**id** B $\ldots, P, \ldots$), or (**id** B $\ldots, P^-, \ldots$) is in $\mathcal{T}$, then (**funct** P) and (**funct** P^-) are **not in** $\mathcal{T}$.
- If $U' \sqsubseteq U$ or (**id** B $\ldots, U, \ldots$) is in $\mathcal{T}$, then (**funct** U) is **not in** $\mathcal{T}$.

Query answering in *DL-Lite*$_{\mathcal{A}}$

Based on query reformulation: given an (U)CQ and an ontology:

1. **Compute its perfect rewriting**, which turns out to be a UCQ.
2. **Evaluate the perfect rewriting** on the ABox seen as a DB.



Perfect rewriting algorithm in *DL-Lite*$_{\mathcal{A}}$

To compute the perfect rewriting, starting from the original (U)CQ:

Iteratively get a CQ to be processed and either:

- **expand** positive inclusions & **simplify** redundant atoms, or
- **unify** atoms in the CQ to obtain a more specific CQ to be further expanded.

Each result of the above steps is added to the queries to be processed, until no further CQ can be added.

Note: negative inclusions, functionalities, and identification constraints play a role in ontology satisfiability, but not in query answering (i.e., we have **separability**).

Query answering in *DL-Lite*$_{\mathcal{A}}$ – Example

TBox:

$\mathsf{Manager} \sqsubseteq \exists \mathsf{worksFor}$ $\quad \forall x(\mathsf{Manager}(x) \rightarrow \exists y(\mathsf{worksFor}(x, y)))$

$\exists \mathsf{worksFor}^- \sqsubseteq \mathsf{Project}$ $\quad \forall x(\exists y(\mathsf{worksFor}(y, x)) \rightarrow \mathsf{Project}(x))$

Query: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{Project}(y)$

Perfect Reformulation: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{Project}(y)$

$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{worksFor}(\_, y)$

$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, \_)$

$\mathsf{q}(x) \leftarrow \mathsf{Manager}(x)$

ABox: $\mathsf{worksFor}(\mathtt{john}, \mathtt{prA})$ $\quad \mathsf{Manager}(\mathtt{john})$

$\mathsf{worksFor}(\mathtt{tim}, \mathtt{prB})$ $\quad \mathsf{Manager}(\mathtt{rick})$

Evaluating the last two queries over the ABox (seen as a DB) produces as answer $\{\mathtt{john}, \mathtt{tim}, \mathtt{rick}\}$.

Complexity of reasoning in *DL-Lite*$_{\mathcal{A}}$

Ontology satisfiability and all classical DL reasoning tasks are:

- Efficiently tractable in the size of the TBox (i.e., PTIME).
- Very efficiently tractable in the size of the ABox (i.e., AC^0).

In fact, reasoning can be done by constructing suitable FOL/SQL queries and evaluating them over the ABox (**FOL-rewritability**).

Query answering for CQs and UCQs is:

- PTIME in the size of the TBox.
- AC^0 in the size of the ABox.
- Exponential in the size of the **query** (NP-complete).
  Bad? . . . not really, this is exactly as in relational DBs.

Can we go beyond *DL-Lite*$_{\mathcal{A}}$?

No! By adding essentially any additional constructor we lose these nice computational properties [CDGL$^+$06].

Beyond *DL-Lite*$_\mathcal{A}$: results on data complexity

	Lhs	Rhs	funct.	Role incl.	Data complexity of query answering
0	*DL-Lite*$_\mathcal{A}$		$\surd^*$	$\surd^*$	in AC^0
1	$A \mid \exists P.A$	A	–	–	NLogSpace-hard
2	A	$A \mid \forall P.A$	–	–	NLogSpace-hard
3	A	$A \mid \exists P.A$	$\surd$	–	NLogSpace-hard
4	$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	–	–	PTime-hard
5	$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	–	–	PTime-hard
6	$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	$\surd$	–	PTime-hard
7	$A \mid \exists P.A \mid \exists P^-.A$	$A \mid \exists P$	–	–	PTime-hard
8	$A \mid \exists P \mid \exists P^-$	$A \mid \exists P \mid \exists P^-$	$\surd$	$\surd$	PTime-hard
9	$A \mid \neg A$	A	–	–	coNP-**hard**
10	A	$A \mid A_1 \sqcup A_2$	–	–	coNP-**hard**
11	$A \mid \forall P.A$	A	–	–	coNP-**hard**

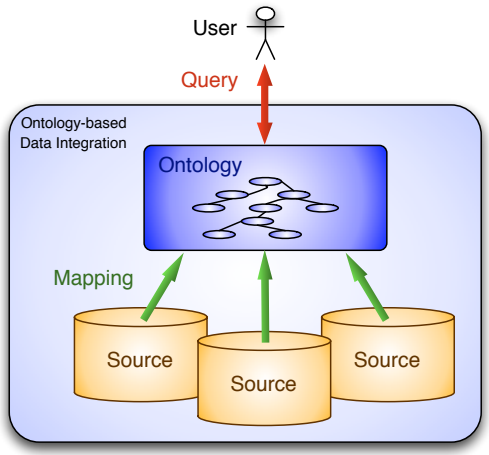
Notes:

- * with the "proviso" of not specializing functional properties.
- NLogSpace and PTime hardness holds already for instance checking.
- For coNP-hardness in line 10, a TBox with a single assertion $A_L \sqsubseteq A_T \sqcup A_F$ suffices! $\rightsquigarrow$ **No** hope of including **covering constraints**.

Outline

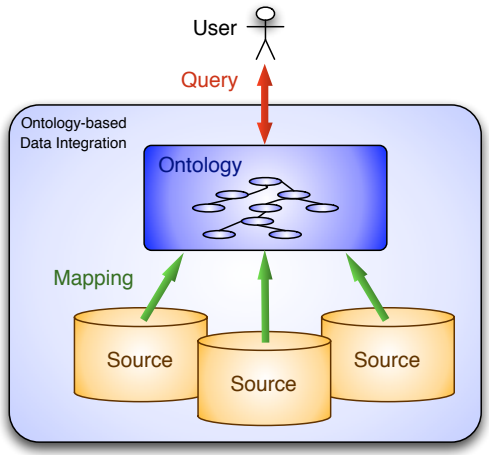
Outline

Ontology-based data integration: The *DL-Lite*$_\mathcal{A}$ solution



- We require the data sources to be **wrapped** and presented as relational sources. $\rightsquigarrow$ *"Standard technology"*
- We make use of a **data federation tool**, such as IBM Information Integrator, to present the yet to be (semantically) integrated sources as a single relational database. $\rightsquigarrow$ *"Standard technology"*
- We make use of the **DL-Lite**$_\mathcal{A}$ technology presented above for the conceptual view on the data, to **exploit effectiveness of query answering**. $\rightsquigarrow$ *"New technology"*

Mappings in ontology based data integration



- The (federated) source DB is **external** and **independent** from the conceptual view (the ontology).
- **Mappings** relate information in the sources to the ontology. $\rightsquigarrow$ sort of virtual ABox
  We use GAV (global-as-view) mappings: the result of an (arbitrary) SQL query on the source DB provides a (partial) extension of a concept/role.
- We exploit the distinction between **objects** and **values** in *DL-Lite*$_{\mathcal{A}}$ to deal with the notorious **impedance mismatch problem**!

Impedance mismatch problem

We have to deal with the **impedance mismatch problem**:

- In **relational databases**, information is represented in forms of tuples of **values**.
- In **ontologies** (or more generally object-oriented systems or conceptual models), information is represented using both **objects** and values ...
  - ... with objects playing the main role, ...
  - ... and values a subsidiary role as fillers of object's attributes.

How do we reconcile these views?

- We need to specify how to construct from the data values in the relational sources the (abstract) objects and values that populate the ABox of the ontology.
- This specification is embedded in the mappings between the data sources and the ontology.

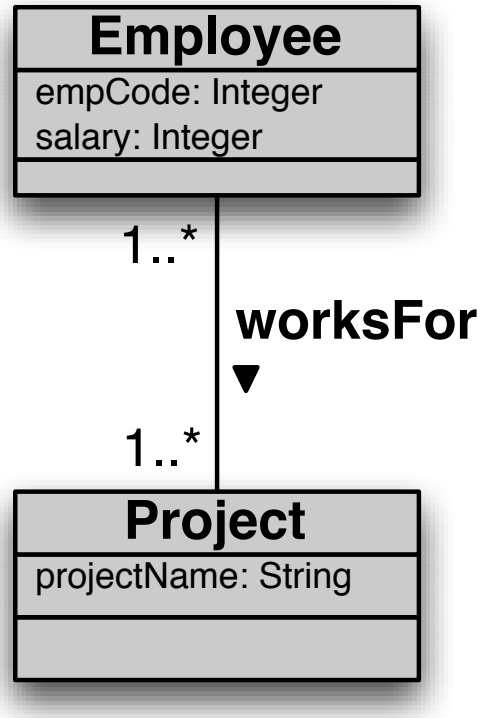
Note: the **ABox** is only **virtual**, and the objects are not materialized.

Solution to the impedance mismatch problem

We need to define a **mapping language** that allows for specifying how to transform data into abstract objects:

- Each mapping assertion maps:
  - a query that retrieves values from a data source to ...
  - a set of atoms specified over the ontology.
- Basic idea: use **Skolem functions** in the atoms over the ontology to "generate" the objects from the data values.
- Semantics of mappings:
  - Objects are denoted by terms (of exactly one level of nesting).
  - Different terms denote different objects (i.e., we make the unique name assumption on terms).

Impedance mismatch – Example



Actual data is stored in a DB:
– An employee is identified by her SSN.
– A project is identified by its name.

$D_1[SSN: \text{String}, PrName: \text{String}]$
Employees and projects they work for

$D_2[Code: \text{String}, Salary: \text{Int}]$
Employee's code with salary

$D_3[Code: \text{String}, SSN: \text{String}]$
Employee's Code with SSN

...

Intuitively:

- An employee should be created from her SSN: **pers**(*SSN*)
- A project should be created from its name: **proj**(*PrName*)

Creating object identifiers

We need to associate to the data in the tables objects in the ontology.

- We introduce an alphabet Λ of **function symbols**, each with an associated arity.
- To denote values, we use value constants from an alphabet Γ_V.
- To denote objects, we use **object terms** (instead of object constants). An object term has the form $\mathbf{f}(d_1, \ldots, d_n)$, with $\mathbf{f} \in \Lambda$, and each d_i a value constant in Γ_V.

Example

- If a person is identified by her *SSN*, we can introduce a function symbol **pers**$/1$. If `VRD56B25` is a *SSN*, then **pers**(`VRD56B25`) denotes a person.
- If a person is identified by her *name* and *dateOfBirth*, we can introduce a function symbol **pers**$/2$. Then **pers**(`Vardi`, `25/2/56`) denotes a person.

Outline

Ontology-Based Data Integration System

The mapping assertions are a crucial part of an Ontology-Based Data Integration System.

Def.: Ontology-Based Data Integration System

is a triple $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, where

- $\mathcal{T}$ is a TBox.
- $\mathcal{S}$ is a (federated) relational database representing the sources.
- $\mathcal{M}$ is a set of mapping assertions between $\mathcal{T}$ and $\mathcal{S}$.

The mapping assertions are used to extract the data from the sources to populate the ontology.

We need to specify their syntax and semantics.

Mapping assertions

A mapping assertion in $\mathcal{M}$ has the form

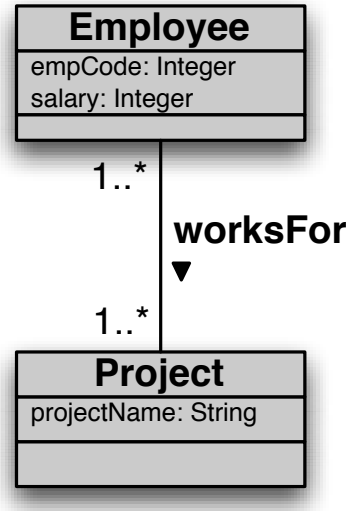
$$\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{t}, \vec{y})$$

where

- Φ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{S}$;
- Ψ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ **without non-distinguished variables**;
- $\vec{x}$, $\vec{y}$ are variables, with $\vec{y} \subseteq \vec{x}$;
- $\vec{t}$ are variable terms of the form $\mathbf{f}(\vec{z})$, with $\mathbf{f} \in \Lambda$ and $\vec{z} \subseteq \vec{x}$.

Note: we could consider also mapping assertions between the datatypes of the database and those of the ontology.

Mapping assertions – Example



$D_1[SSN: \text{String}, PrName: \text{String}]$
Employees and Projects they work for

$D_2[Code: \text{String}, Salary: \text{Int}]$
Employee's code with salary

$D_3[Code: \text{String}, SSN: \text{String}]$
Employee's code with SSN

...

m_1: `SELECT SSN, PrName FROM` D_1 $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_2: `SELECT SSN, Salary FROM` D_2`, `D_3 `WHERE` D_2`.Code =` D_3`.Code` $\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

Semantics of mappings

To define the semantics of an OBDI system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, we first need to define the semantics of mappings.

Intuitively, $\mathcal{I}$ **satisfies** a mapping assertion $\Phi \rightsquigarrow \Psi$ w.r.t. $\mathcal{S}$ if all facts obtained by evaluating Φ over $\mathcal{S}$ and then propagating the answers to Ψ, hold in $\mathcal{I}$.

Def.: Satisfaction of a mapping assertion with respect to a database

An interpretation $\mathcal{I}$ **satisfies** a mapping assertion $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{t}, \vec{y})$ in $\mathcal{M}$ **with respect to a source database** $\mathcal{S}$, if for each tuple of values $\vec{v} \in \mathit{Eval}(\Phi, \mathcal{S})$, and for each ground atom X in $\Psi[\vec{x}/\vec{v}]$, we have that:

- if X is $A(s)$, then $s^{\mathcal{I}} \in A^{\mathcal{I}}$.
- if X is $T(s)$, then $s^{\mathcal{I}} \in T^{\mathcal{I}}$.
- if X is $P(s_1, s_2)$, then $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.
- if X is $U(s_1, s_2)$, then $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in U^{\mathcal{I}}$.

Note: $\mathit{Eval}(\Phi, \mathcal{S})$ denotes the result of evaluating Φ over the database $\mathcal{S}$.
$\Psi[\vec{x}/\vec{v}]$ denotes Ψ where each x_i has been substituted with v_i.

Semantics of an OBDI system

Def.: Model of an OBDI system

An interpretation $\mathcal{I}$ is a **model** of $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ if:

- $\mathcal{I}$ is a model of $\mathcal{T}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ w.r.t. $\mathcal{S}$, i.e., $\mathcal{I}$ satisfies every assertion in $\mathcal{M}$ w.r.t. $\mathcal{S}$.

An OBDI system $\mathcal{O}$ is **satisfiable** if it admits at least one model.

Outline

Answering queries over an OBDI system

In an OBDI system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$

- Queries are posed over the TBox $\mathcal{T}$.
- The data needed to answer queries is stored in the data source $\mathcal{S}$.
- The mapping $\mathcal{M}$ is used to bridge the gap between $\mathcal{T}$ and $\mathcal{S}$.

Two approaches to exploit the mapping:

- bottom-up approach: simpler, but less efficient
- top-down approach: more sophisticated, but also more efficient

Note: Both approaches require to first **split** the TBox queries in the mapping assertions into their constituent atoms.

Splitting of mappings

A mapping assertion $\Phi \rightsquigarrow \Psi$, where the TBox query Ψ is constituted by the atoms $X_1,\ldots,X_k$, can be split into several mapping assertions:

$$\Phi \rightsquigarrow X_1 \qquad \cdots \qquad \Phi \rightsquigarrow X_k$$

This is possible, since Ψ does not contain non-distinguished variables.

Example

m_1: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

is split into

m_1^1: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Employee(**pers**(*SSN*))

m_1^2: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Project(**proj**(*PrName*))

m_1^3: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ projectName(**proj**(*PrName*), *PrName*)

m_1^4: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ worksFor(**pers**(*SSN*), **proj**(*PrName*))

Bottom-up approach to query answering

Consists in a straightforward application of the mappings:

1. Propagate the data from $\mathcal{S}$ through $\mathcal{M}$, materializing an ABox $\mathcal{A}_{\mathcal{M},\mathcal{S}}$ (the constants in such an ABox are values and object terms).
2. Apply to $\mathcal{A}_{\mathcal{M},\mathcal{S}}$ and to the TBox $\mathcal{T}$, the satisfiability and query answering algorithms developed for *DL-Lite*$_{\mathcal{A}}$.

This approach has several drawbacks (hence is only theoretical):

- The technique is no more AC^0 (or even LOGSPACE) in the data, since the ABox $\mathcal{A}_{\mathcal{M},\mathcal{S}}$ to materialize is in general polynomial in the size of the data.
- $\mathcal{A}_{\mathcal{M},\mathcal{S}}$ may be very large, and thus it may be infeasible to actually materialize it.
- Freshness of $\mathcal{A}_{\mathcal{M},\mathcal{S}}$ with respect to the underlying data source(s) may be an issue, and one would need to propagate source updates (cf. Data Warehousing).

Top-down approach to query answering

Given an OBDI system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, the computation of the certain answers to an UCQ q consists of three steps:

1. **Rewriting:** Compute the perfect rewriting $q_{pr} = \mathit{PerfectRew}(q, \mathcal{T}_P)$ of the original query q, using the inclusion assertions of the TBox $\mathcal{T}$.

2. **Unfolding:** Compute from q_{pr} a new query q_{unf} by unfolding q_{pr} using (the split version of) the mappings $\mathcal{M}$.
   - Essentially, each atom in q_{pr} that unifies with an atom in Ψ is substituted with the corresponding query Φ over the database.
   - The unfolded query is such that $\mathit{Eval}(q_{unf}, \mathcal{S}) = \mathit{Eval}(q_{pr}, \mathcal{A}_{\mathcal{M},\mathcal{S}})$.

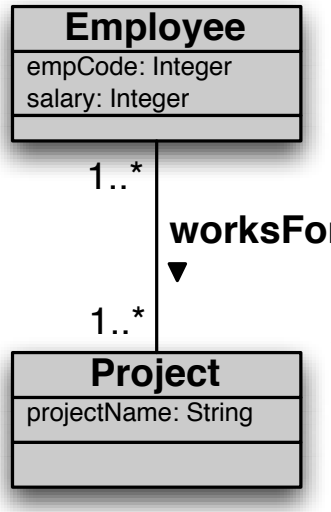
3. **Evaluation:** Delegate the evaluation of q_{unf} to the relational DBMS managing $\mathcal{S}$.

Unfolding

To unfold a query q_{pr} with respect to a set of mapping assertions:

1. For each non-split mapping assertion $\Phi_i(\vec{x}) \rightsquigarrow \Psi_i(\vec{t}, \vec{y})$:
   1. Introduce a **view symbol** Aux_i of arity equal to that of Φ_i.
   2. Add a **view definition** $\text{Aux}_i(\vec{x}) \leftarrow \Phi_i(\vec{x})$.
2. For each split version $\Phi_i(\vec{x}) \rightsquigarrow X_j(\vec{t}, \vec{y})$ of a mapping assertion, introduce a **clause** $X_j(\vec{t}, \vec{y}) \leftarrow \text{Aux}_i(\vec{x})$.
3. Obtain from q_{pr} in all possible ways queries q_{aux} defined over the view symbols Aux_i as follows:
   1. Find a most general unifier ϑ that unifies each atom $X(\vec{z})$ in the body of q_{pr} with the head of a clause $X(\vec{t}, \vec{y}) \leftarrow \text{Aux}_i(\vec{x})$.
   2. Substitute each atom $X(\vec{z})$ with $\vartheta(\text{Aux}_i(\vec{x}))$, i.e., with the body the unified clause to which the unifier ϑ is applied.
4. The unfolded query q_{unf} is the **union** of all queries q_{aux}, together with the view definitions for the predicates Aux_i appearing in q_{aux}.

Unfolding – Example



m_1: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_2: `SELECT SSN, Salary FROM D2, D3 WHERE D2.Code = D3.Code` $\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

We define a view Aux_i for the source query of each mapping m_i.

For each (split) mapping assertion, we introduce a clause:

$$\begin{aligned}
\text{Employee}(\mathbf{pers}(\mathit{SSN})) &\leftarrow \text{Aux}_1(\mathit{SSN}, \mathit{PrName}) \\
\text{projectName}(\mathbf{proj}(\mathit{PrName}), \mathit{PrName}) &\leftarrow \text{Aux}_1(\mathit{SSN}, \mathit{PrName}) \\
\text{Project}(\mathbf{proj}(\mathit{PrName})) &\leftarrow \text{Aux}_1(\mathit{SSN}, \mathit{PrName}) \\
\text{worksFor}(\mathbf{pers}(\mathit{SSN}), \mathbf{proj}(\mathit{PrName})) &\leftarrow \text{Aux}_1(\mathit{SSN}, \mathit{PrName}) \\
\text{Employee}(\mathbf{pers}(\mathit{SSN})) &\leftarrow \text{Aux}_2(\mathit{SSN}, \mathit{Salary}) \\
\text{salary}(\mathbf{pers}(\mathit{SSN}), \mathit{Salary}) &\leftarrow \text{Aux}_2(\mathit{SSN}, \mathit{Salary})
\end{aligned}$$

Unfolding – Example (cont'd)

Query over ontology: employees who work for `tones` and their salary:

$q(e, s) \leftarrow \mathsf{Employee}(e), \mathsf{salary}(e, s), \mathsf{worksFor}(e, p), \mathsf{projectName}(p, \texttt{tones})$

A unifier between the atoms in q and the clause heads is:

$$\vartheta(e) = \mathbf{pers}(\mathit{SSN}) \qquad \vartheta(s) = \mathit{Salary}$$
$$\vartheta(\mathit{PrName}) = \texttt{tones} \qquad \vartheta(p) = \mathbf{proj}(\texttt{tones})$$

After applying ϑ to q, we obtain:

$$q(\mathbf{pers}(\mathit{SSN}), \mathit{Salary}) \leftarrow \mathsf{Employee}(\mathbf{pers}(\mathit{SSN})),\ \mathsf{salary}(\mathbf{pers}(\mathit{SSN}), \mathit{Salary}),\ \mathsf{worksFor}(\mathbf{pers}(\mathit{SSN}), \mathbf{proj}(\texttt{tones})),\ \mathsf{projectName}(\mathbf{proj}(\texttt{tones}), \texttt{tones})$$

Substituting the atoms with the bodies of the unified clauses, we obtain:

$$q(\mathbf{pers}(\mathit{SSN}), \mathit{Salary}) \leftarrow \mathsf{Aux}_1(\mathit{SSN}, \texttt{tones}),\ \mathsf{Aux}_2(\mathit{SSN}, \mathit{Salary}),\ \mathsf{Aux}_1(\mathit{SSN}, \texttt{tones}),\ \mathsf{Aux}_1(\mathit{SSN}, \texttt{tones})$$

Computational complexity of query answering

From the top-down approach to query answering, and the complexity results for *DL-Lite*$_{\mathcal{A}}$, we obtain that **query answering** in a *DL-Lite*$_{\mathcal{A}}$ ontology with mappings $\mathcal{O} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ is:

- Very efficiently tractable in the size of the **database** $\mathcal{S}$ (i.e., AC^0, and in fact FOL-rewritable).
- Efficiently tractable in the size of the **TBox** $\mathcal{T}$ and the **mappings** $\mathcal{M}$ (i.e., PTIME).
- Exponential in the size of the **query** (i.e., NP-complete).

Can we move to LAV or GLAV mappings?
No, if we want to stay in AC^0 [CDGL$^+$08a].

Implementation of top-down approach to query answering

To implement the top-down approach, we need to generate an SQL query.

We can follow different strategies:

1. Substitute each view predicate in the unfolded queries with the corresponding SQL query over the source:
   - \+ joins are performed on the DB attributes;
   - \+ does not generate doubly nested queries;
   - – the number of unfolded queries may be exponential.

2. Construct for each atom in the original query a new view. This view takes the union of all SQL queries corresponding to the view predicates, and constructs also the Skolem terms:
   - \+ avoids exponential blow-up of the resulting query, since the union (of the queries coming from multiple mappings) is done before the joins;
   - – joins are performed on Skolem terms;
   - – generates doubly nested queries.

Which method is better, depends on various parameters.
Experiments have shown that (1) behaves better in most cases.

Towards answering arbitrary SQL queries

- We have seen that answering full SQL (i.e., FOL) queries is undecidable.
- However, we can treat the answers to an UCQ, as "knowledge", and perform further computations on that knowledge.
- This corresponds to applying a knowledge operator to UCQs that are embedded into an arbitrary SQL query (EQL queries)
  - The UCQs are answered according to the certain answer semantics.
  - The SQL query is evaluated on the facts returned by the UCQs.
- The approach can be implemented by rewriting the UCQs and embedding the rewritten UCQs into SQL.
- The user "sees" arbitrary SQL queries, but these SQL queries are evaluated according to a weakened semantics.

Implemented system

- **QuOnto + Integration Module**: *DL-Lite*$_{\mathcal{A}}$ reasoning system developed at SAPIENZA University of Rome:
  - Connects to external relational (federated) DB (several RDBMSs supported).
  - Supports general mappings, various forms of constraints, EQL queries.
  - Reformulation and unfolding process implemented in Java.
  - Evaluation of reformulated and unfolded queries delegated to RDBMS.
- **ODBA Protégé Plugin**: Protégé interface for QuOnto developed at Free University of Bozen-Bolzano:
  - Based on OWL-DL.
  - Extends standard APIs for communication with DL reasoners (DIG).
  - Supports the design of mappings.
  - Supports connection to external sources.

Outline

Experimentations and experiences

Several experimentations have been and are being carried out:

- Selex: world leading radar producer
- National Accessibility Portal of South Africa
- Monte dei Paschi di Siena
- Horizontal Gene Transfer data and ontology

Observations:

- Approach highly effective for bridging impedance mismatch.
- Rewriting technique effective against incompleteness in the data.
- Performance is an issue:
  - Optimizations in the generation of the rewriting are crucial.
  - Management of mappings is still problematic, due to complex joins related to skolem terms.

Conclusions

- Ontology-based Data Integration is a challenging problem with great practical relevance.
- In this setting, the size of the data is the relevant parameter that must guide technological choices.
- Currently, scalability w.r.t. the size of the data can be achieved only by relying on commercial technologies for managing the data, i.e., relational DBMS systems.
- In order to tailor semantic technologies so as to provide a good compromise between expressivity and efficiency, requires a thorough understanding of the semantic and computational properties of the adopted formalisms.
- We have now gained such an understanding, that allowed us to study and develop good solutions to the Ontology-based Data Integration problem.

Other work related to this talk

- Extensions of *DL-Lite*$_{\mathcal{A}}$ with additional constructs (in particular, other forms of constraints) [CDGL$^+$08b]
- Going beyond (unions) of conjunctive queries (weaker semantics wrt FOL) using dynamic CWA offered by epistemic operators [CDGL$^+$07]
- Update on stand-alone ontologies [DGLPR06, DGLPR07]
- Privacy-aware access [CDGLR08]
- Restricting the attention to finite models only [Ros08]
- Meta-reasoning a la RDFS [DGLR08]

Ongoing and future work:

- Provenance and explanation [BCRM08]
- Write-also access: updating the data sources through an ontology
- LAV, and the new *DL-Lite* family: AC^0 query answering after NLogSpace/PTime preprocessing of data.

Acknowledgements

People involved in this work:

- Domenico Lembo
- Maurizio Lenzerini
- Marco Ruzzi
- Antonella Poggi
- Mariano Rodriguez Muro
- Riccardo Rosati
- several master and PhD students

This work has been carried out within the EU Strep FET project TONES (Thinking ONtologiES).
http://www.tonesproject.org/

Outline

References I

[Baa90] F. Baader.

Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles.

Technical Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern (Germany), 1990.

An abridged version appeared in *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pp. 446–451.

[BBL05] F. Baader, S. Brandt, and C. Lutz.

Pushing the $\mathcal{EL}$ envelope.

In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, 2005.

[BBMR89] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick.

CLASSIC: A structural data model for objects.

In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 59–67, 1989.

References II

[BCDG05] D. Berardi, D. Calvanese, and G. De Giacomo.
Reasoning on UML class diagrams.
Artificial Intelligence, 168(1–2):70–118, 2005.

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation and Applications.
Cambridge University Press, 2003.

[BCRM08] A. Borgida, D. Calvanese, and M. Rodriguez-Muro.
Explanation in the *DL-Lite* family of description logics.
In *Proc. of the 7th Int. Conf. on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008)*, volume 5332 of *Lecture Notes in Computer Science*, pages 1440–1457. Springer, 2008.

[BL84] R. J. Brachman and H. J. Levesque.
The tractability of subsumption in frame-based description languages.
In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI'84)*, pages 34–37, 1984.

References III

[Cal96] D. Calvanese.

Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms.

PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1996.

[CDGL95] D. Calvanese, G. De Giacomo, and M. Lenzerini.

Structured objects: Modeling and reasoning.

In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD'95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 229–246. Springer, 1995.

[CDGL98] D. Calvanese, G. De Giacomo, and M. Lenzerini.

On the decidability of query containment under constraints.

In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

References IV

[CDGL+05] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
DL-Lite: Tractable description logics for ontologies.
In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[CDGL+06] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Data complexity of query answering in description logics.
In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.

[CDGL+07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
EQL-Lite: Effective first-order query processing in description logics.
In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 274–279, 2007.

References V

[CDGL+08a] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, and M. Ruzzi.

Data integration through *DL-Lite*$_{\mathcal{A}}$ ontologies.

In K.-D. Schewe and B. Thalheim, editors, *Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)*, volume 4925 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 2008.

[CDGL+08b] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.

Path-based identification constraints in description logics.

In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 231–241, 2008.

[CDGLR08] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati.

View-based query answering over description logic ontologies.

In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 242–251, 2008.

References VI

[DG95] G. De Giacomo.
Decidability of Class-Based Knowledge Representation Formalisms.
PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.

[DGLPR06] G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati.
On the update of description logic ontologies at the instance level.
In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*, pages 1271–1276, 2006.

[DGLPR07] G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati.
On the approximation of instance level update and erasure in description logics.
In *Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007)*, pages 403–408, 2007.

[DGLR08] G. De Giacomo, M. Lenzerini, and R. Rosati.
Towards higher-order *DL-Lite*.
In *Proc. of the 2008 Description Logic Workshop (DL 2008)*, volume 353 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, 2008.

References VII

[Don03] F. M. Donini.
Complexity of reasoning.
In Baader et al. [BCM+03], chapter 3, pages 96–136.

[Hor98] I. Horrocks.
Using an expressive description logic: FaCT or fiction?
In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[MH03] R. Möller and V. Haarslev.
Description logic systems.
In Baader et al. [BCM+03], chapter 8, pages 282–305.

[Ros08] R. Rosati.
Finite model reasoning in *DL-Lite*.
In *Proc. of the 5th European Semantic Web Conf. (ESWC 2008)*, 2008.

References VIII

[Sch91] K. Schild.

A correspondence theory for terminological logics: Preliminary report.

In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.

[Sch93] A. Schaerf.

On the complexity of the instance checking problem in concept languages with existential quantification.

J. of Intelligent Information Systems, 2:265–278, 1993.