Knowledge Representation and Ontologies

Part 1: Modeling Information through Ontologies

Diego Calvanese, Luciano Serafini

Faculty of Computer Science
Master of Science in Computer Science

A.Y. 2010/2011

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Overview of the Course

1. Modeling information through ontologies
   1. Introduction to ontologies
   2. Ontology languages
   3. UML class diagrams as FOL ontologies
2. Using logic for knowledge representation
   1. Main components of a logic
   2. Reasoning methods in logics
   3. Exercises on analyzing logics
3. Description Logics
   1. Introduction to DLs
   2. Reasoning in simple DLs
   3. More expressive DLs
   4. Fuzzy DLs
   5. Ontology modularization, integration, and contextualization
4. Ontology based data access
   1. Description Logics for data access
   2. Query answering over databases and ontologies
   3. Linking ontologies to relational data
   4. Reasoning in the *DL-Lite* family
5. Conclusions and references

Part 1

Modeling Information through Ontologies

Outline of Part 1

Outline of Part 1

Outline of Part 1

New challenges in information management

One of the key challenges in complex systems today is the management of information:

- The **amount** of information has increased enormously.
- The **complexity** of information has increased:
  structured $\rightsquigarrow$ semi-structured $\rightsquigarrow$ unstructured
- The underlying data may be of **low quality**, e.g., incomplete, inconsistent, not *crisp*.
- Information is increasingly **distributed** and **heterogeneous**, but nevertheless needs to be accessed in a uniform way.
- Information is consumed not only by humans, but also by machines.

Traditional data management systems are not sufficient anymore to fulfill today's information management requirements.

Addressing information management challenges

Several efforts come from the database area:

- New kinds of databases are studied, to manage semi-structured (XML), and probabilistic data.
- Information integration is one of the major challenges for the future or IT. E.g., the market for information integration software has been 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 (2008)].

On the other hand, management of complex kinds of information has traditionally been the concern of **Knowledge Representation** in AI:

- Research in AI and KR can bring new insights, solutions, techniques, and technologies.
- **However, what has been done in KR needs to be adapted / extended / tuned to address the new challenges coming from today's requirements for information management.**

Description logics

Description Logics [Baader *et al.*, 2003] are an important area of KR, studied for the last 25 years, that provide the foundations for the structured representation of information:

- By grounding the used formalisms in logic, the information is provided with a **formal semantics** (i.e., a meaning).
- The logic-based formalization allows one to provide **automated support** for tasks related to data management, by means of **logic-based inference**.
- **Computational aspects** are of concern, so that **tools** can provide **effective support** for automated reasoning.

In this course we are looking into using description logics for data management.

Ontologies

Description logics provide the formal foundations for ontology languages.

Def.: Ontology

is a representation scheme that describes a **formal conceptualization** of a domain of interest.

The specification of an ontology usually comprises two distinct levels:

- **Intensional level**: specifies a set of conceptual elements and of constraints/axioms describing the conceptual structures of the domain.
- **Extensional level**: specifies a set of instances of the conceptual elements described at the intensional level.

Note: an ontology may contain also a **meta-level**, which specifies a set of modeling categories of which the conceptual elements are instances.

Outline of Part 1

1. Introduction to ontologies
   - Ontologies for information management
   - Ontologies in information systems
   - Issues in ontology-based information management
2. Ontology languages
3. UML class diagrams as FOL ontologies
4. References

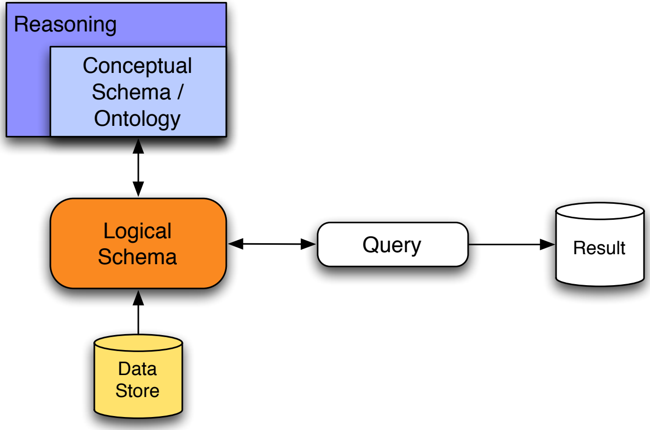
Conceptual schemas in information systems

Intensional information has traditionally played an important role in information systems.

Design phase of the information system:

1. From the requirements, a **conceptual schema** of the domain of interest is produced.
2. The conceptual schema is used to produce the logical data schema.
3. The data are stored according to the logical schema, and queried through it.

Conceptual schemas used at design-time



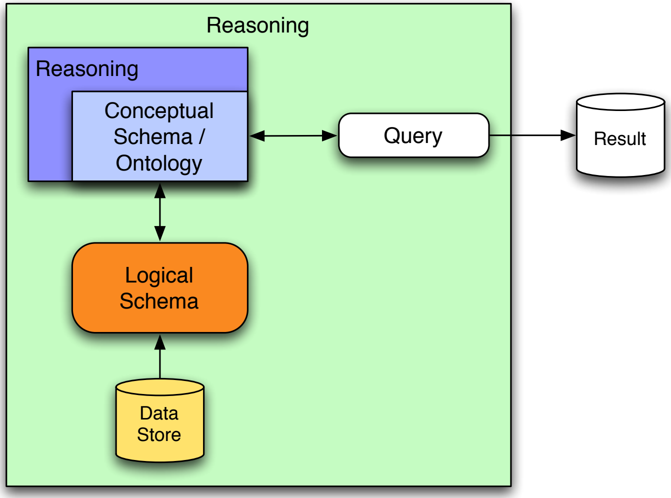
Ontologies in information systems

The role of ontologies in information systems goes beyond that of conceptual schemas.

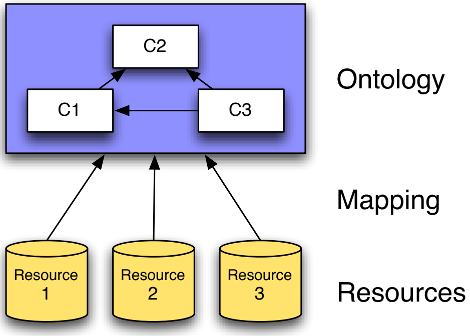
Ontologies affect the whole life-cycle of the information system:

- Ontologies, with the associated reasoning capabilities and inference tools, can provide support at design time.
- The use of ontologies can significantly simplify **maintenance** of the information system's data assets.
- The ontology is used also to support the interaction with the information system, i.e., at run-time.
  $\leadsto$ **Reasoning** to take into account the constraints coming from the ontology has to be **done at run-time**.

Ontologies used at run-time



Ontologies at the core of information systems



The usage of all system resources (data and services) is done through the domain conceptualization.

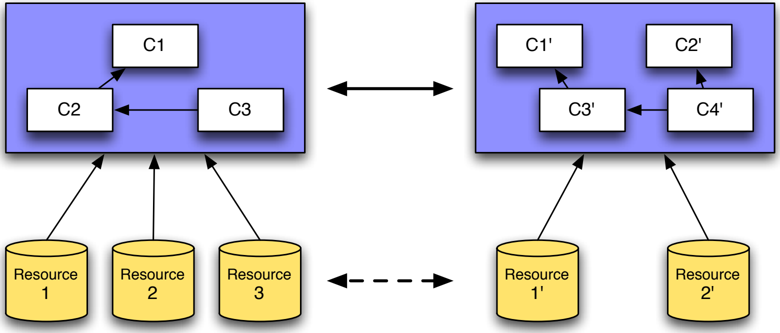
Ontology mediated access to data

Desiderata: achieve **logical transparency** in access to data:

- **Hide** to the user where and how data are stored.
- Present to the user a **conceptual view** of the data.
- Use a **semantically rich formalism** for the conceptual view.

This setting is similar to the one of Data Integration. The difference is that here the ontology provides a rich conceptual description of the data managed by the system.

Ontologies at the core of cooperation



The cooperation between systems is done at the level of the conceptualization.

Outline of Part 1

Issues in ontology-based information management

1. Choice of the formalisms to adopt
2. Efficiency and scalability
3. Tool support

Issue 1: Formalisms to adopt

1. Which is the right ontology language?
   - many proposals have been made
   - differ in expressive power and in complexity of inference
2. Which languages should we use for querying?
   - requirements for querying are different from those for modeling
3. How do we connect the ontology to available information sources?
   - mismatch between information in an ontology and data in a data source

In this course:

- We present and discuss variants of ontology languages, and study their logical and computational properties.
- We study the problem of querying data through ontologies.
- We discuss problems and solutions related to the impedance mismatch between ontologies and data sources.

Issue 2: Efficiency and scalability

- How can we handle large ontologies?
  - We have to take into account the **tradeoff** between **expressive power** and **complexity** of inference.
- How can we cope with large amounts of data?
  - What may be good for large ontologies, may not be good enough for large amounts of data.
- Can we handle multiple data sources and/or multiple ontologies?

In this course:

- We discuss in depth the above mentioned **tradeoff**.
- We will also pay attention to the aspects related to **data management**.
- We do not deal with the problem of integrating multiple information sources. See the course on *Information Integration*.

Issue 3: Tools

- According to the principle that "there is no meaning without a language with a formal semantics", the formal semantics becomes the solid basis for dealing with ontologies.
- Hence every kind of access to an ontology (to extract information, to modify it, etc.), requires to **fully** take into account its semantics.
- We need tools that perform reasoning over the ontology that is **sound and complete** wrt the semantics.
- The tools have to be as "efficient" as possible.

In this course:

- We discuss the requirements, the principles, and the theoretical foundations for ontology inference tools.
- We also present and use a tool for querying data sources through ontologies that has been built according to those principles.

Outline of Part 1

1. Introduction to ontologies
2. Ontology languages
   - Elements of an ontology language
   - Intensional and extensional level of an ontology language
   - Ontologies vs. other formalisms
3. UML class diagrams as FOL ontologies
4. References

Outline of Part 1

Elements of an ontology language

- **Syntax**
  - Alphabet
  - Languages constructs
  - Sentences to assert knowledge
- **Semantics**
  - Formal meaning
- **Pragmatics**
  - Intended meaning
  - Usage

Static vs. dynamic aspects

The aspects of the domain of interest that can be modeled by an ontology language can be classified into:

- **Static aspects**
  - Are related to the structuring of the domain of interest.
  - Supported by virtually all languages.
- Dynamic aspects
  - Are related to how the elements of the domain of interest evolve over time.
  - Supported only by some languages, and only partially (cf. services).

Before delving into the dynamic aspects, we need a good understanding of the static ones.

In this course we concentrate essentially on the static aspects.

Outline of Part 1

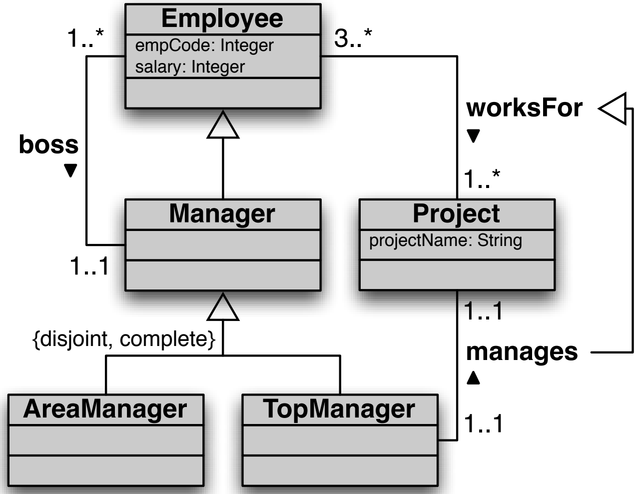
Intensional level of an ontology language

An ontology language for expressing the intensional level usually includes:

- Concepts
- Properties of concepts
- Relationships between concepts, and their properties
- Axioms
- Queries

Ontologies are typically **rendered as diagrams** (e.g., Semantic Networks, Entity-Relationship schemas, UML Class Diagrams).

Example: ontology rendered as UML Class Diagram



Concepts

Def.: Concept

Is an element of an ontology that denotes a collection of instances (e.g., the set of "employees").

We distinguish between:

- Intensional definition:
  specification of name, properties, relations, ...
- Extensional definition:
  specification of the instances

Concepts are also called **classes**, **entity types**, **frames**.

Properties

Def.: Property

Is an element of an ontology that qualifies another element (e.g., a concept or a relationship).

Property definition (intensional and extensional):

- Name
- Type: may be either
  - atomic (integer, real, string, enumerated, ...), or
    e.g., eye-color → { blu, brown, green, grey }
  - structured (date, set, list, ...)
    e.g., date → day/month/year
- The definition may also specify a default value.

Properties are also called **attributes**, **features**, **slots**, **data properties**.

Relationships

Def.: Relationship

Is an element of an ontology that expresses an association among concepts.

We distinguish between:

- Intensional definition:
  specification of involved concepts
  e.g., worksFor is defined on Employee and Project
- Extensional definition:
  specification of the instances of the relationship, called facts
  e.g., worksFor(domenico, tones)

Relationships are also called **associations**, **relationship types**, **roles**, **object properties**.

Axioms

Def.: Axiom

Is a logical formula that expresses at the intensional level a condition that must be satisified by the elements at the extensional level.

Different kinds of axioms/conditions:

- subclass relationships, e.g., $\text{Manager} \sqsubseteq \text{Employee}$
- equivalences, e.g., $\text{Manager} \equiv \text{AreaManager} \sqcup \text{TopManager}$
- disjointness, e.g., $\text{AreaManager} \sqcap \text{TopManager} \equiv \bot$
- (cardinality) restrictions,
  e.g., each Employee worksFor at least 3 Project
- ...

Axioms are also called **assertions**.
A special kind of axioms are **definitions**.

Extensional level of an ontology language

At the extensional level we have individuals and facts:

- An **instance** represents an individual (or object) in the extension of a concept.
  e.g., domenico is an instance of Employee
- A **fact** represents a relationship holding between instances.
  e.g., worksFor(domenico, tones)

Outline of Part 1

1. Introduction to ontologies
2. Ontology languages
   - Elements of an ontology language
   - Intensional and extensional level of an ontology language
   - Ontologies vs. other formalisms
3. UML class diagrams as FOL ontologies
4. References

Comparison with other formalisms

- Ontology languages vs. knowledge representation languages:

  Ontologies **are** knowledge representation schemas.

- Ontology vs. logic:

  Logic is **the** tool for assigning semantics to ontology languages.

- Ontology languages vs. conceptual data models:

  Conceptual schemas **are** special ontologies, suited for conceptualizing a single logical model (database).

- Ontology languages vs. programming languages:

  Class definitions **are** special ontologies, suited for conceptualizing a single structure for computation.

Classification of ontology languages

- Graph-based
  - Semantic networks
  - Conceptual graphs
  - **UML class diagrams**, Entity-Relationship diagrams
- Frame based
  - Frame Systems
  - OKBC, XOL
- Logic based
  - **Description Logics** (e.g., $\mathcal{SHOIQ}$, $\mathcal{DLR}$, **DL-Lite** , OWL, ...)
  - Rules (e.g., RuleML, LP/Prolog, F-Logic)
  - First Order Logic (e.g., KIF)
  - Non-classical logics (e.g., non-monotonic, probabilistic)

Outline of Part 1

1. Introduction to ontologies
2. Ontology languages
3. UML class diagrams as FOL ontologies
    - Approaches to conceptual modeling
    - Formalizing UML class diagram in FOL
    - Reasoning on UML class diagrams
4. References

Outline of Part 1

1. Introduction to ontologies
2. Ontology languages
3. UML class diagrams as FOL ontologies
   - Approaches to conceptual modeling
   - Formalizing UML class diagram in FOL
   - Reasoning on UML class diagrams
4. References

Modeling the domain of interest

We aim at obtaining a description of the data of interest in **semantic terms**.

One can proceed as follows:

1. Represent the domain of interest as a **conceptual schema**, similar to those used at design time to design a database.
2. Formalize the conceptual schema as a **logical theory**, namely the **ontology**.
3. 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 is described by a text in natural language.

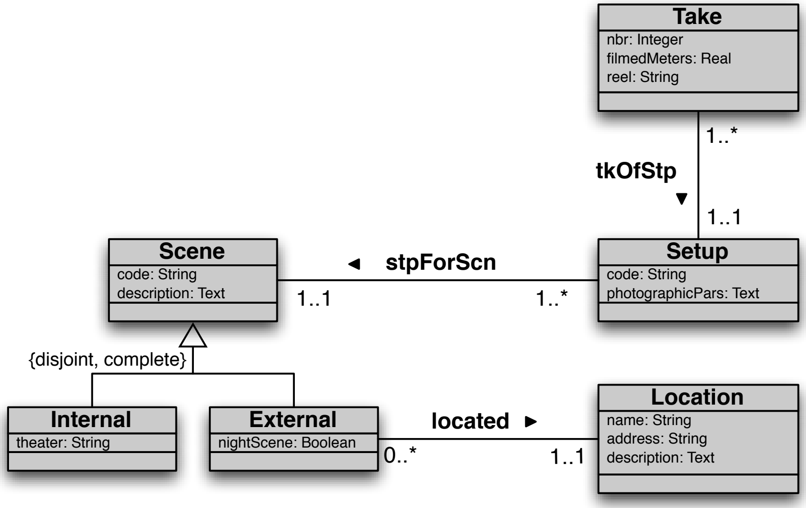
Every scene is filmed from different positions (at least one), each of this is called a setup. Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related 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: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x,y)$, $tkOfStp(x,y)$, $located(x,y)$, $\ldots$.

Axioms:

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

$\forall x,y.\ code_{Setup}(x,y) \rightarrow Setup(x) \land String(y)$
$\forall x,y.\ photographicPars(x,y) \rightarrow Setup(x) \land Text(y)$

$\forall x,y.\ nbr(x,y) \rightarrow Take(x) \land Integer(y)$
$\forall x,y.\ filmedMeters(x,y) \rightarrow Take(x) \land Real(y)$
$\forall x,y.\ reel(x,y) \rightarrow Take(x) \land String(y)$

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

$\forall x,y.\ nightScene(x,y) \rightarrow External(x) \land Boolean(y)$

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

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

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

$\forall x,y.\ stpForScn(x,y) \rightarrow Setup(x) \land Scene(y)$
$\forall x,y.\ tkOfStp(x,y) \rightarrow Take(x) \land Setup(y)$
$\forall x,y.\ located(x,y) \rightarrow External(x) \land Location(y)$

$\forall x.\ Setup(x) \rightarrow (1 \leq \sharp\{y \mid stpForScn(x,y)\} \leq 1)$
$\forall y.\ Scene(y) \rightarrow (1 \leq \sharp\{x \mid stpForScn(x,y))\}$
$\forall x.\ Take(x) \rightarrow (1 \leq \sharp\{y \mid tkOfStp(x,y)\} \leq 1)$
$\forall x.\ Setup(y) \rightarrow (1 \leq \sharp\{x \mid tkOfStp(x,y)\})$
$\forall x.\ External(x) \rightarrow (1 \leq \sharp\{y \mid located(x,y)\} \leq 1)$
$\ldots$

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 bad point, in fact.*

Solution 3: Use both!!!

Note these two approaches seem to be orthogonal, but in fact they can 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:

- Inherited 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'd)

Important:

The logical theories that are obtained from conceptual modeling diagrams are of a specific form.

- Their expressiveness is limited (or better, well-disciplined).
- One can exploit the particular form of the logical theory to simplify reasoning.
- The aim is getting:
  - decidability, and
  - reasoning procedures that match the intrinsic computational complexity of reasoning over the conceptual modeling diagrams.

Conceptual models vs. logic

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

We will use:

- as conceptual modeling diagrams: **UML Class Diagrams**.
  Note: we could equivalently use Entity-Relationship Diagrams instead of UML.
- as logic: **First-Order Logic** to formally capture semantics and reasoning.

Outline of Part 1

The Unified Modeling Language (UML)

The **Unified Modeling Language (UML)** was developed in 1994 by unifying and integrating the most prominent object-oriented modeling approaches:

- 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, …
- UML 1.4.2 is industrial standard ISO/IEC 19501.
- Current version: 2.3 (May 2010): http://www.omg.org/spec/UML/
- 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 (2nd ed., 2005)
- http://www.omg.org/ → UML
- http://www.uml.org/

UML Class Diagrams

In this course we deal only 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 software).

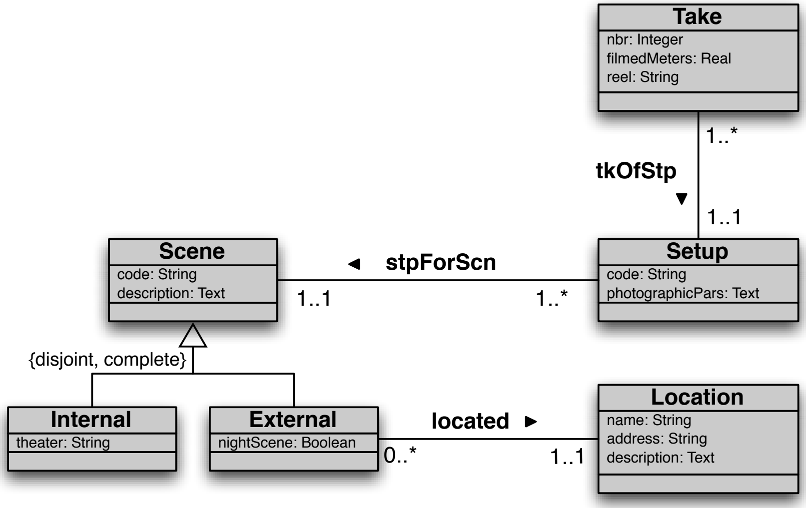
Note: This is exactly the goal of all conceptual modeling formalism, such as Entity-Relationship Diagrams (standard in Database design) or Ontologies.

UML Class Diagrams (cont'd)

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

- objects grouped into **classes**;
- **associations**, representing relationships between classes;
- **attributes**, representing simple properties of the instances of classes;
  Note: here we do not deal with "operations".
- **sub-classing**, i.e., ISA and generalization relationships.

Example of a UML Class Diagram



Use of UML Class Diagrams

UML Class Diagrams are used in various phases of a software design:

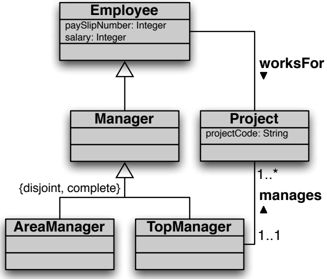
1. During the so-called analysis, where an abstract precise view of the domain of interest needs to be developed.
   $\leadsto$ the so-called "**conceptual perspective**".
2. During software development, to maintain an abstract view of the software to be developed.
   $\leadsto$ the so-called "implementation perspective".

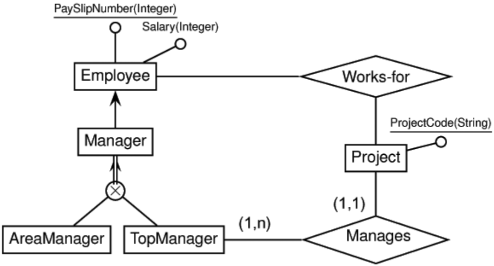
In this course we focus on 1!

UML Class Diagrams and ER Schemas

UML class diagrams (when used for the conceptual perspective) closely resemble Entity-Relationship (ER) Diagrams.

Example of UML vs. ER:





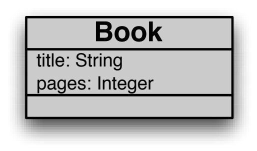
Classes in UML

A **class** in UML models a set of objects (its "instances") that share certain common properties, such as 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



– the name of the class is 'Book'
– the class has two properties (attributes)

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 of our class Book:

$$\{\text{book}_a, \text{book}_b, \text{book}_c, \text{book}_d, \text{book}_e\}$$
$$\{\text{book}_\alpha, \text{book}_\beta\}$$
$$\{\text{book}_1, \text{book}_2, \text{book}_3, \ldots, \text{book}_{500}, \ldots\}$$

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

Classes in UML: formalization

A class represents a set of objects. ... But which set? We don't actually know.

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

We represent a class as a **FOL unary predicate**!

Example

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

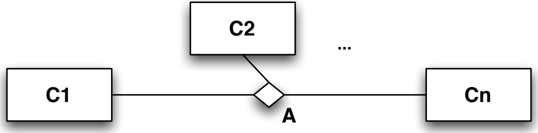
Associations

An **association** in UML models a relationship between two or more classes.

- At the instance level, an association is a relation between the instances of two or more classes.
- Associations model properties of classes that are non-local, in the sense that they involve other classes.
- An association between n classes is a property of each of these classes.

Example

Associations: formalization



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

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

Example

$$\forall x_1, x_2. \mathit{writtenBy}(x_1, x_2) \rightarrow \mathit{Book}(x_1) \wedge \mathit{Author}(x_2)$$

Associations: multiplicity

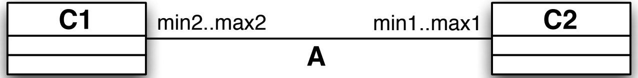
On binary associations, we can place **multiplicity constraints**, i.e., a minimal and maximal number of tuples in which every object participates as first (second) component.

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 of multiplicities



Multiplicities of binary associations are easily expressible in FOL:

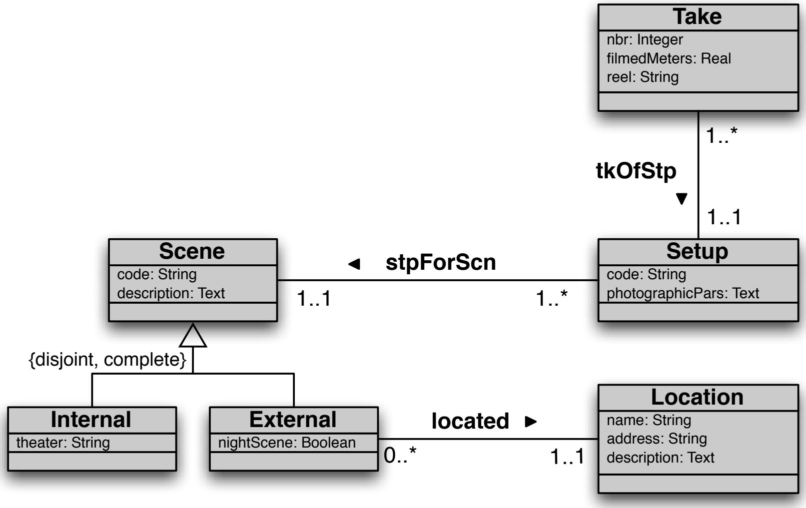
$$\forall x_1. C_1(x_1) \rightarrow (min_1 \leq \sharp\{x_2 \mid A(x_1, x_2)\} \leq max_1)$$
$$\forall x_2. C_2(x_2) \rightarrow (min_2 \leq \sharp\{x_1 \mid A(x_1, x_2)\} \leq max_2)$$

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 the cardinality of the set of possible values for y.

In our example ...



In our example ...

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x,y)$, $tkOfStp(x,y)$, $located(x,y)$, ...

Axioms:

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

$\forall x, y.\ code_{Setup}(x,y) \rightarrow Setup(x) \land String(y)$
$\forall x, y.\ photographicPars(x,y) \rightarrow Setup(x) \land Text(y)$

$\forall x, y.\ nbr(x,y) \rightarrow Take(x) \land Integer(y)$
$\forall x, y.\ filmedMeters(x,y) \rightarrow Take(x) \land Real(y)$
$\forall x, y.\ reel(x,y) \rightarrow Take(x) \land String(y)$

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

$\forall x, y.\ nightScene(x,y) \rightarrow External(x) \land Boolean(y)$

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

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

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

$\forall x, y.\ stpForScn(x,y) \rightarrow Setup(x) \land Scene(y)$
$\forall x, y.\ tkOfStp(x,y) \rightarrow Take(x) \land Setup(y)$
$\forall x, y.\ located(x,y) \rightarrow External(x) \land Location(y)$

$\forall x.\ Setup(x) \rightarrow (1 \leq \sharp\{y \mid stpForScn(x,y)\} \leq 1)$
$\forall y.\ Scene(y) \rightarrow (1 \leq \sharp\{x \mid stpForScn(x,y))\}$
$\forall x.\ Take(x) \rightarrow (1 \leq \sharp\{y \mid tkOfStp(x,y)\} \leq 1)$
$\forall x.\ Setup(y) \rightarrow (1 \leq \sharp\{x \mid tkOfStp(x,y)\})$
$\forall x.\ External(x) \rightarrow (1 \leq \sharp\{y \mid located(x,y)\} \leq 1)$
. . .

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 constraint
- $1..*$: $\forall x. C_1(x) \rightarrow \exists y. A(x, y)$
- $0..1$: $\forall x. C_1(x) \rightarrow \forall y, y'. A(x, y) \wedge A(x, y') \rightarrow y = y'$
  (or simply $\forall x, y, y'. A(x, y) \wedge A(x, y') \rightarrow y = y'$)
- $1..1$: $(\forall x. C_1(x) \rightarrow \exists y. A(x, y)) \wedge (\forall x, y, y'. A(x, y) \wedge A(x, y') \rightarrow y = y')$

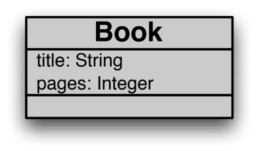
Attributes

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),
- a type (a collection of possible values),
- and possibly a multiplicity.

Example



– The name of one of the attributes is 'title'.
– Its type is 'String'.

Attributes as functions

Attributes (without explicit multiplicity) are:

- mandatory (must have at least a value), and
- single-valued (can have at most one value).

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

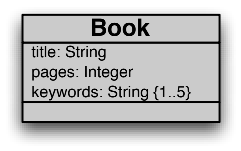
Example

$book_3$ has as value for the attribute 'title' the String: `"The little digital video book"`.

Attributes with multiplicity

More generally attributes may have an explicit **multiplicity** (similar to that of associations).

Example



– The attribute 'title' has an implicit multiplicity of $1..1$.
– The attribute 'keywords' has an explicit multiplicity of $1..5$.

Note: When the multiplicity is not specified, then it is assumed to be $1..1$.

Attributes: formalization

Since attributes may have a multiplicity different from $1..1$, they are better formalized as binary predicates, with suitable assertions representing types and multiplicity.

Given an attribute att of a class C with type T and multiplicity $i..j$, we capture it in FOL as a **binary predicate** $att_C(x, y)$ with the following assertions:

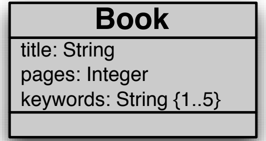
- An assertion for the attribute **type**:

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

- An assertion for the **multiplicity**:

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

Attributes: example of formalization

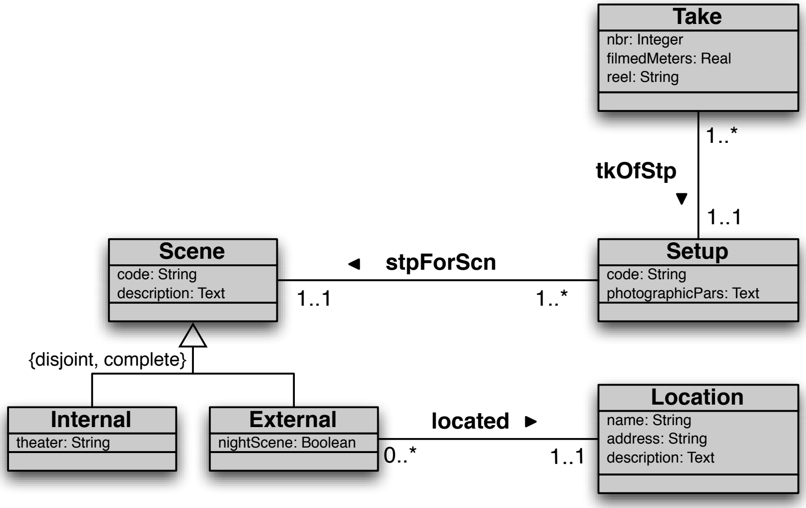


$$\forall x, y.\, \mathit{title}_B(x, y) \rightarrow \mathit{Book}(x) \wedge \mathit{String}(y)$$
$$\forall x.\, \mathit{Book}(x) \rightarrow (1 \leq \sharp\{y \mid \mathit{title}_B(x, y)\} \leq 1)$$

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

$$\forall x, y.\, \mathit{keywords}_B(x, y) \rightarrow \mathit{Book}(x) \wedge \mathit{String}(y)$$
$$\forall x.\, \mathit{Book}(x) \rightarrow (1 \leq \sharp\{y \mid \mathit{keywords}_B(x, y)\} \leq 5)$$

In our example ...



In our example ...

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x,y)$, $tkOfStp(x,y)$, $located(x,y)$, ...

Axioms:

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

$\forall x,y.\ code_{Setup}(x,y) \rightarrow Setup(x) \land String(y)$
$\forall x,y.\ photographicPars(x,y) \rightarrow Setup(x) \land Text(y)$

$\forall x,y.\ nbr(x,y) \rightarrow Take(x) \land Integer(y)$
$\forall x,y.\ filmedMeters(x,y) \rightarrow Take(x) \land Real(y)$
$\forall x,y.\ reel(x,y) \rightarrow Take(x) \land String(y)$

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

$\forall x,y.\ nightScene(x,y) \rightarrow External(x) \land Boolean(y)$

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

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

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

$\forall x,y.\ stpForScn(x,y) \rightarrow Setup(x) \land Scene(y)$
$\forall x,y.\ tkOfStp(x,y) \rightarrow Take(x) \land Setup(y)$
$\forall x,y.\ located(x,y) \rightarrow External(x) \land Location(y)$

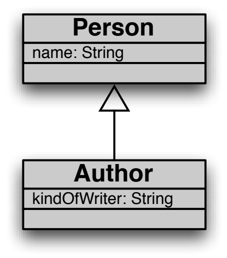
$\forall x.\ Setup(x) \rightarrow (1 \leq \sharp\{y \mid stpForScn(x,y)\} \leq 1)$
$\forall y.\ Scene(y) \rightarrow (1 \leq \sharp\{x \mid stpForScn(x,y))\}$
$\forall x.\ Take(x) \rightarrow (1 \leq \sharp\{y \mid tkOfStp(x,y)\} \leq 1)$
$\forall x.\ Setup(y) \rightarrow (1 \leq \sharp\{x \mid tkOfStp(x,y)\})$
$\forall x.\ External(x) \rightarrow (1 \leq \sharp\{y \mid located(x,y)\} \leq 1)$
...

ISA and 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

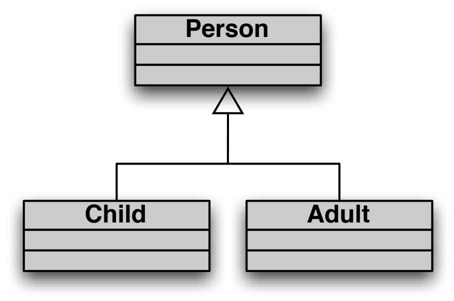


The attibute 'name' is inherited by 'Author'.

Generalizations

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

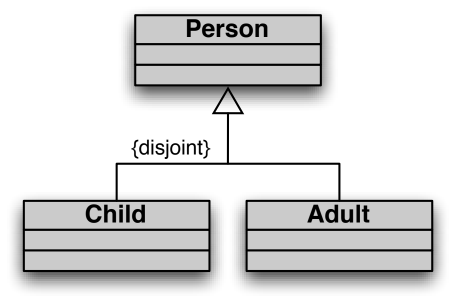
Example



Generalizations with constraints

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

Example

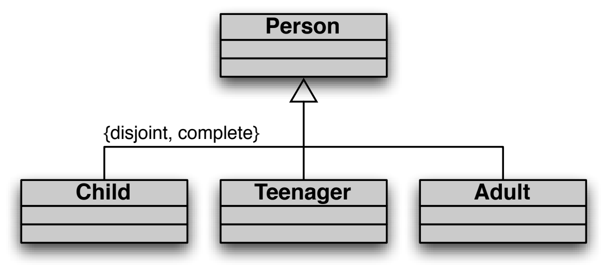


Generalizations with constraints (cont'd)

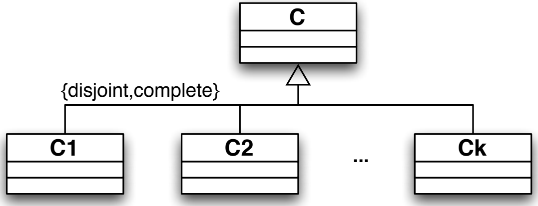
Most notable and used constraints:

- **Disjointness**, which 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"), which asserts that every instance of the superclass is also an instance of at least one of the subclasses.

Example



Generalizations: formalization

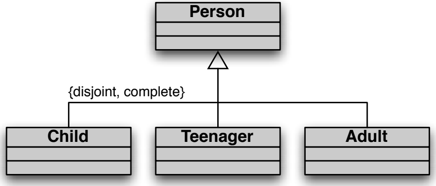


ISA: $\forall x. C_i(x) \rightarrow C(x)$, for $1 \leq i \leq k$

Disjointness: $\forall x. C_i(x) \rightarrow \neg C_j(x)$, for $1 \leq i < j \leq k$

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

Generalizations: example of formalization

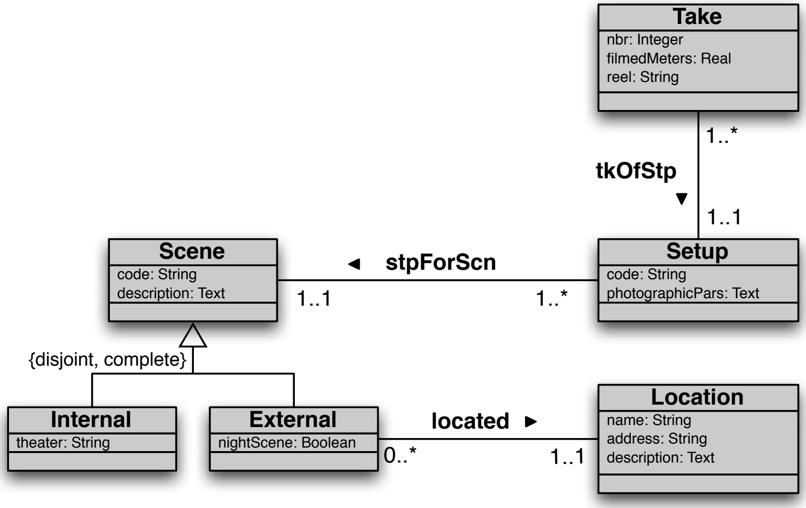


$$\forall x.\ Child(x) \rightarrow Person(x)$$
$$\forall x.\ Teenager(x) \rightarrow Person(x)$$
$$\forall x.\ Adult(x) \rightarrow Person(x)$$

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

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

In our example ...



In our example ...

Alphabet: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x,y)$, $tkOfStp(x,y)$, $located(x,y)$, ...

Axioms:

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

$\forall x,y.\, code_{Setup}(x,y) \rightarrow Setup(x) \land String(y)$
$\forall x,y.\, photographicPars(x,y) \rightarrow Setup(x) \land Text(y)$

$\forall x,y.\, nbr(x,y) \rightarrow Take(x) \land Integer(y)$
$\forall x,y.\, filmedMeters(x,y) \rightarrow Take(x) \land Real(y)$
$\forall x,y.\, reel(x,y) \rightarrow Take(x) \land String(y)$

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

$\forall x,y.\, nightScene(x,y) \rightarrow External(x) \land Boolean(y)$

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

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

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

$\forall x,y.\, stpForScn(x,y) \rightarrow Setup(x) \land Scene(y)$
$\forall x,y.\, tkOfStp(x,y) \rightarrow Take(x) \land Setup(y)$
$\forall x,y.\, located(x,y) \rightarrow External(x) \land Location(y)$

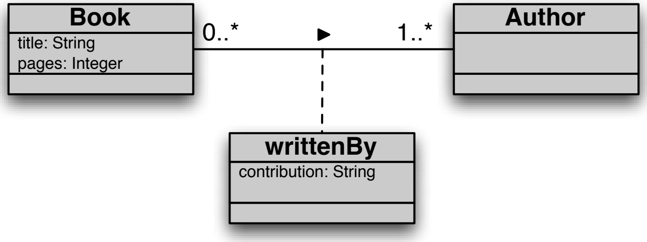
$\forall x.\, Setup(x) \rightarrow (1 \leq \sharp\{y \mid stpForScn(x,y)\} \leq 1)$
$\forall y.\, Scene(y) \rightarrow (1 \leq \sharp\{x \mid stpForScn(x,y))\}$
$\forall x.\, Take(x) \rightarrow (1 \leq \sharp\{y \mid tkOfStp(x,y)\} \leq 1)$
$\forall x.\, Setup(y) \rightarrow (1 \leq \sharp\{x \mid tkOfStp(x,y)\})$
$\forall x.\, External(x) \rightarrow (1 \leq \sharp\{y \mid located(x,y)\} \leq 1)$
...

Association classes

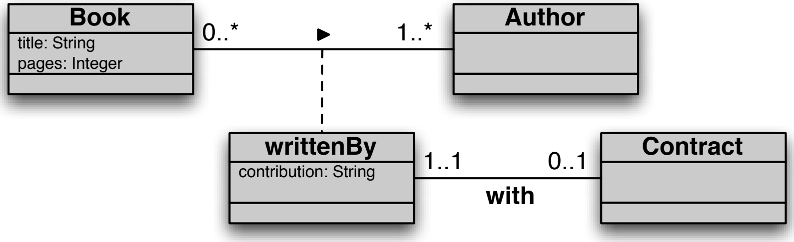
Sometimes we may want to assert properties of associations. In UML to do so we resort to **association classes**:

- 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).

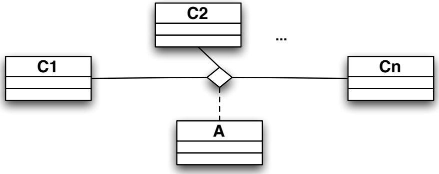
Association class – Example



Association class – Example (cont'd)



Association classes: formalization

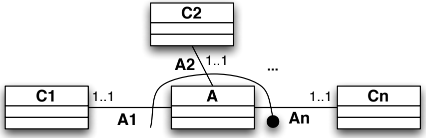


The process of putting in correspondence objects of a class (the association class) with tuples in an association is formally described as **reification**.

That is:

- We introduce a unary predicate A for the association class A.
- We introduce n new binary predicates $A_1, \ldots, A_n$, one for each of the components of the association.
- We introduce suitable assertions so that objects in the extension of the unary-predicate A are in bijection with tuples in the n-ary association A.

Association classes: formalization (cont'd)



FOL Assertions are needed for stating a bijection between instances of the association class and instances of the association:

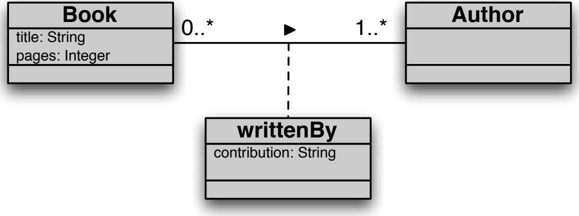
$$\forall x, y.\, A_i(x,y) \rightarrow A(x) \land C_i(y), \qquad \text{for } i \in \{1,\ldots,n\}$$

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

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

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

Association classes: example of formalization



$$\forall x, y.\, wb_1(x,y) \rightarrow writtenBy(x) \wedge Book(y)$$
$$\forall x, y.\, wb_2(x,y) \rightarrow writtenBy(x) \wedge Author(y)$$

$$\forall x.\, writtenBy(x) \rightarrow \exists y.\, wb_1(x,y)$$
$$\forall x.\, writtenBy(x) \rightarrow \exists y.\, wb_2(x,y)$$
$$\forall x, y, y'.\, wb_1(x,y) \wedge wb_1(x,y') \rightarrow y = y'$$
$$\forall x, y, y'.\, wb_2(x,y) \wedge wb_2(x,y') \rightarrow y = y'$$

$$\forall x, x', y_1, y_2.\, wb_1(x,y_1) \wedge wb_1(x',y_1) \wedge wb_2(x,y_2) \wedge wb_2(x',y_2) \;\rightarrow\; x = x'$$

Outline of Part 1

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 corresponding to the UML Class Diagram, and $C(x)$ the predicate corresponding to a class C of the diagram.

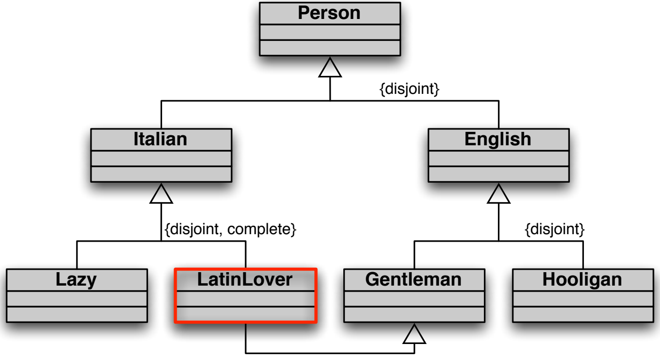
Then C **is consistent** iff

$$\Gamma \not\models \forall x. C(x) \rightarrow \mathit{false}$$

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

Note: Corresponding FOL reasoning task: satisfiability.

Class consistency: 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 corresponding to the UML Class Diagram.

Then, **the diagram is consistent** iff

Γ is satisfiable

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

Note: Corresponding 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.

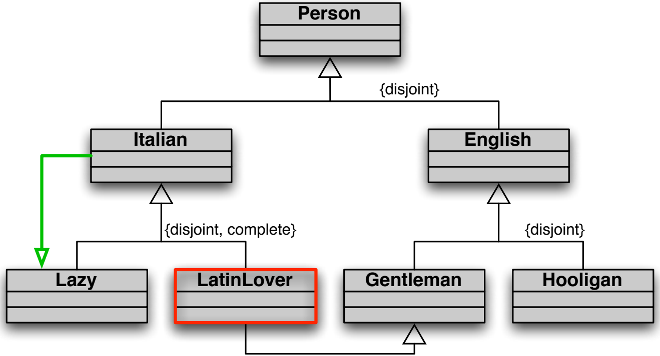
Let Γ be the set of FOL assertions corresponding to the UML Class Diagram, and $C_1(x)$, $C_2(x)$ the predicates corresponding to the classes C_1, and C_2 of the diagram.

Then C_1 **is subsumed by** C_2 iff

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

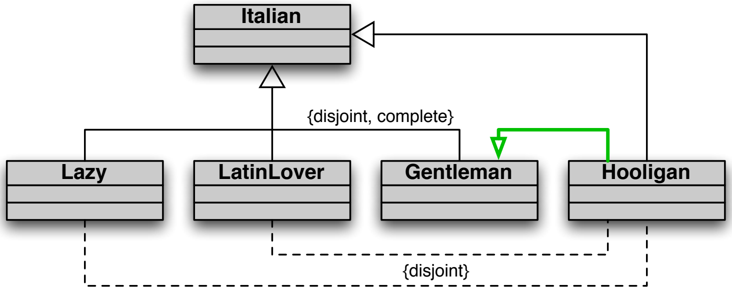
Note: Corresponding FOL reasoning task: logical implication.

Class subsumption: example



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

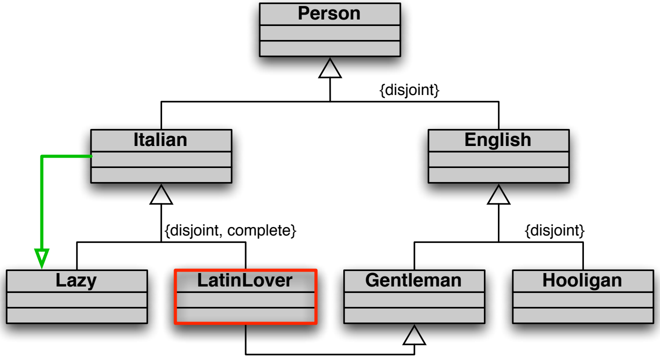
Class subsumption: another example (by E. Franconi)



$$\Gamma \models \forall x.\, \mathit{ItalianProf}(x) \rightarrow \mathit{LatinLover}(x)$$

Note: this is an example of reasoning by cases.

Class equivalence: 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 corresponding to the UML Class Diagram, and $\mathcal{P}$ (the formalization in FOL of) the property of interest

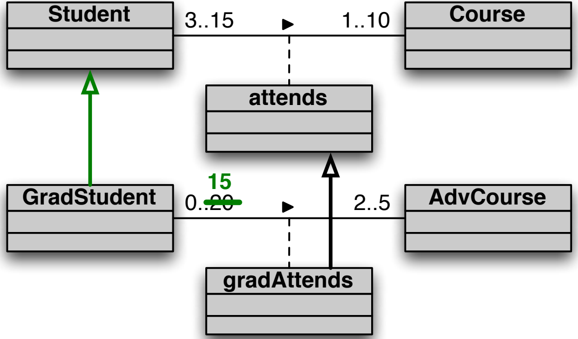
Then $\mathcal{P}$ is an implicit consequence iff

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

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

Note: Corresponding FOL reasoning task: logical implication.

Implicit consequences: example

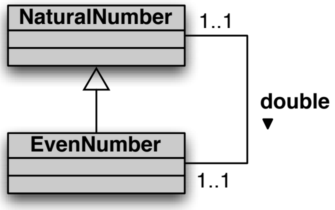


$\Gamma \models \forall x.\, AdvCourse(x_2) \rightarrow \sharp\{x_1 \mid gradAttends(x_1, x_2)\} \leq 15$

$\Gamma \models \forall x.\, GradStudent(x) \rightarrow Student(x)$

$\Gamma \not\models \forall x.\, AdvCourse(x) \rightarrow Course(x)$

Unrestricted vs. finite model reasoning



- Due to the multiplicities, the classes $NaturalNumber$ and $EvenNumber$ are in bijection.
  As a consequence, in every instantiation of the diagram, "the classes $NaturalNumber$ and $EvenNumber$ contain the same number of objects".
- Due to the ISA relationship, every instance of $EvenNumber$ is also an instance of $NaturalNumber$, i.e., we have that

$$\Gamma \models \forall x.\, EvenNumber(x) \rightarrow NaturalNumber(x)$$

Unrestricted vs. finite model reasoning (cont'd)

Question: Does also the reverse implication hold, i.e.,

$$\Gamma \models \forall x.\, NaturalNumber(x) \rightarrow EvenNumber(x) \quad ?$$

- if the domain is infinite, the implication does not hold.
- If the domain is finite, the implication does hold.

Finite model reasoning: means reasoning only with respect to models with a finite domain.

- Finite model reasoning is interesting for standard databases.
- The previous example shows that in UML Class Diagrams, finite model reasoning is different form unrestricted model reasoning.

Questions

In the above examples reasoning could be easily carried out on intuitive grounds. However, two questions come up.

1. Can we develop sound, complete, and **terminating** procedures for reasoning on UML Class Diagrams?

- We cannot do so by directly relying on FOL!
- But we can use specialized logics with better computational properties. A form of such specialized logics are **Description Logics**.

2. How hard is it to reason on UML Class Diagrams in general?

- What is the worst-case situation?
- Can we single out **interesting fragments** on which to reason efficiently?

We will address also **answering queries** over such diagrams, which is in general a more complicated task than satisfiability or subsumption.

Note: all what we have said holds for Entity-Relationship Diagrams as well!

References I

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors.

The Description Logic Handbook: Theory, Implementation and Applications.

Cambridge University Press, 2003.