Conceptual Modelling: An Introduction

Πληροφοριακά Συστήματα Διαδικτύου

Information Modeling

- Information modeling means “models of information”

  (NOT “models made by information”)

- We are interested in models of information about the real world, or *somebody’s conception* of the real world.

Information Modeling in Computer Science and Engineering

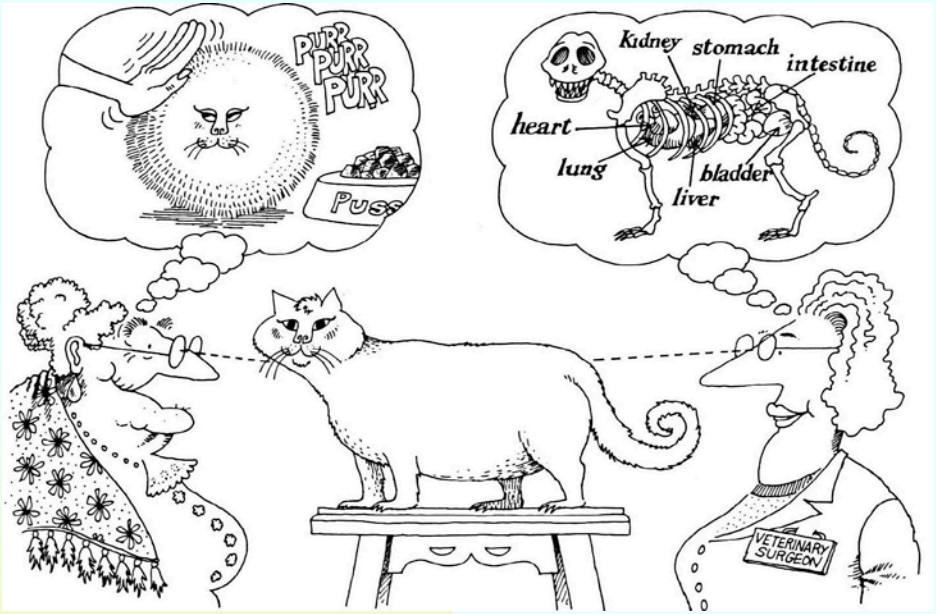
Information modeling is practiced and researched in several areas within Computer Science and Engineering, including:

- **Databases** -- data modeling, using semantic data models to build ***databases***;
- **Software Engineering** -- requirements modeling, using diagrammatic (structured, OO) techniques to build ***requirement specifications***; software process modeling using finite state machines, statecharts, rules, Petri nets to build ***process models***;
- **Artificial Intelligence** -- knowledge representation using logic-based notations, description logics, semantic networks, frames, etc. to build ***knowledge bases***;
- ...

Types of Information Models

- ***Physical models*** use machine-oriented terms (e.g., records, fields, strings, variable names, B-trees,...) to model an application.
  - ***...conflicting representational and efficiency concerns!***
- ***Logical models*** offer mathematical abstractions (e.g., arrays, lists, sets, relations) for modeling purposes, hiding the implementation details from the user (i.e the relational model in Databases).
  - ***...but logical symbol structures are flat and unintuitive.... can can't describe complex situations!***
- ***Conceptual models*** use application-oriented terms and use application-oriented terms and organize information on the basis of principles derived from Cognitive Science, such as generalization, aggregation and classification.
  - ***conceptual symbol structures model directly and naturally an application***

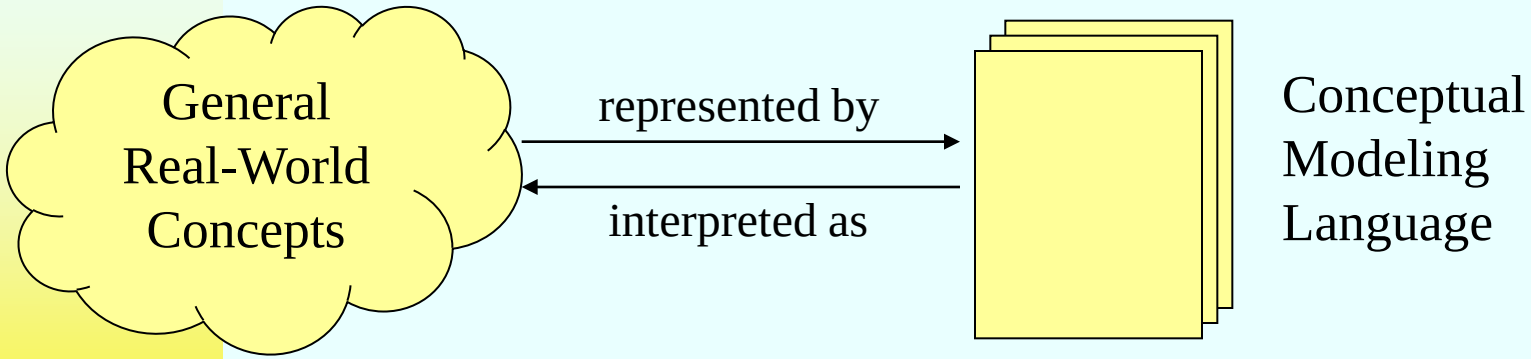
Research on Conceptual Modeling



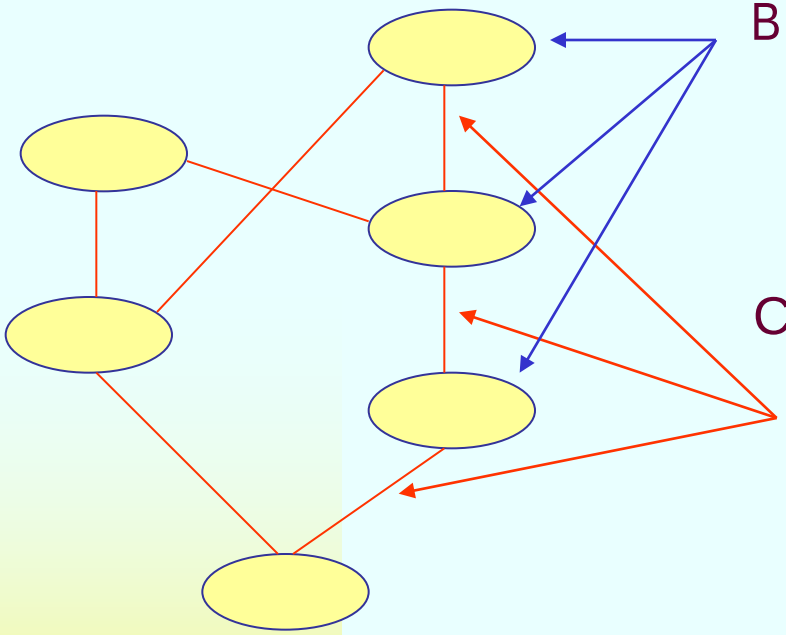
- **Goal**: improve models and tools for representing information and processes
- **Why**: narrow the gap between concepts in the real world and their representation in conceptual models
- **How**: identify and formalize powerful modeling primitives, like generic relationships, for more accurate and intuitive descriptions of real-world concepts

Conceptual Modeling

- "Conceptual Modeling is the activity of formally describing some aspects of physical and social world around us for the purposes of understanding and communication" *(Mylopoulos)*
- "Conceptual models offer abstract views on certain aspects of the real-world (descriptive view)" *(Yourdon)*



Characterizing Conceptual Models



Basic building blocks – Terms
descriptions, concepts, ...

Composition rules – Abstractions
abstraction mechanisms
semantic relationships
structuring mechanisms,...

Analysis and management tools

Terms, Abstraction, and Tools

Characterization of conceptual models by looking at the basic building blocks, the structuring mechanisms, and the tools they offer for building an information base:

- **Primitive Terms** - concepts built into a conceptual model, used to model an application (e.g. **Entity, Activity**, **Goal**, **Time**, **Space**,...)
- **Abstraction Mechanisms** (also called **Semantic Relationships**) -- primitive mechanisms for structuring (e.g. **Generalization, Aggregation, Classification, Materialization**,...)
- **Tools and Analysis techniques Tools** -- for creating, updating, searching, validating and managing an information base

Entities

- These represent classes of objects that have properties in common and an autonomous existence.
  - *City, Department, Employee, Purchase and Sale* are examples of entities for a commercial organization.
- An instance of an entity is an object in the class represented by the entity.
  - *Stockholm, Helsinki*, are examples of instances of the entity *City*, and the employees *Peterson* and *Johanson* are examples of instances of the *Employee* entity.

Relationships

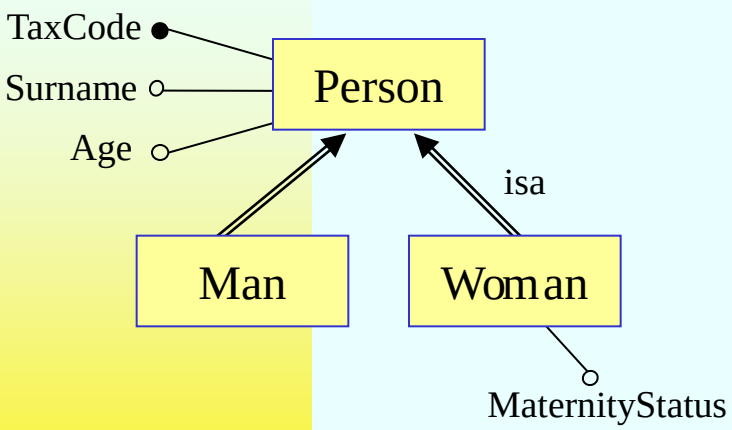
- They represent logical links between two or more entities.
  - *Residence* is an example of a relationship that can exist between the entities *City* and *Employee*; *Exam* is an example of a relationship that can exist between the entities *Student* and *Course*.
- An instance of a relationship is an n-tuple made up of instances of entities, one for each of the entities involved.
  - The pair (Johanssen,Stockholm), or the pair (Peterson,Oslo), are examples of instances in the relationship *Residence*.

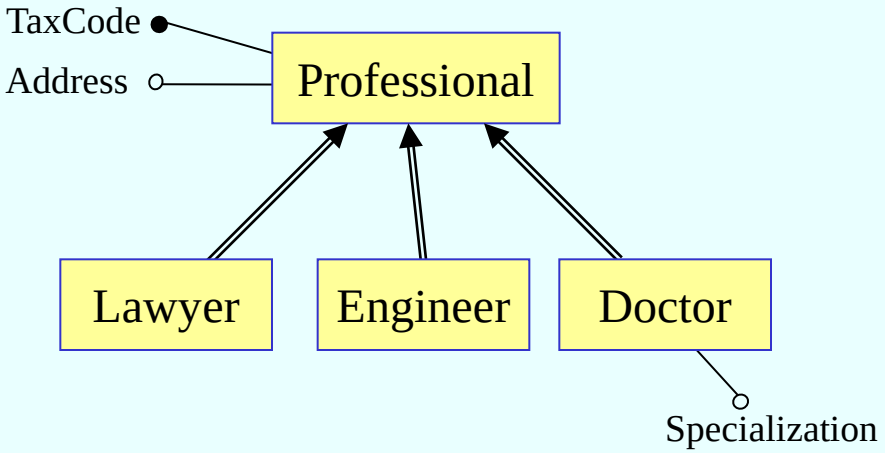
Attributes

- These describe the elementary properties of entities or relationships.
  - For example, *Surname*, *Salary* and *Age* are possible attributes of the *Employee* entity, while *Date* and *Mark* are possible attributes for the relationship *Exam* between *Student* and *Course*.
- An attribute associates with each instance of an entity (or relationship) a value belonging to a set known as the ***domain*** of the attribute.
- The domain contains the admissible values for the attribute.

Generalizations

- These represent logical links between an entity E, known as ***parent*** entity, and one or more entities E1,...,En called ***child*** entities, of which E is more general, in the sense that they are a particular case.
- In this situation we say that E is a ***generalization*** of E1,...,En and that the entities E1,...,En are ***specializations*** (noted isa) of E.





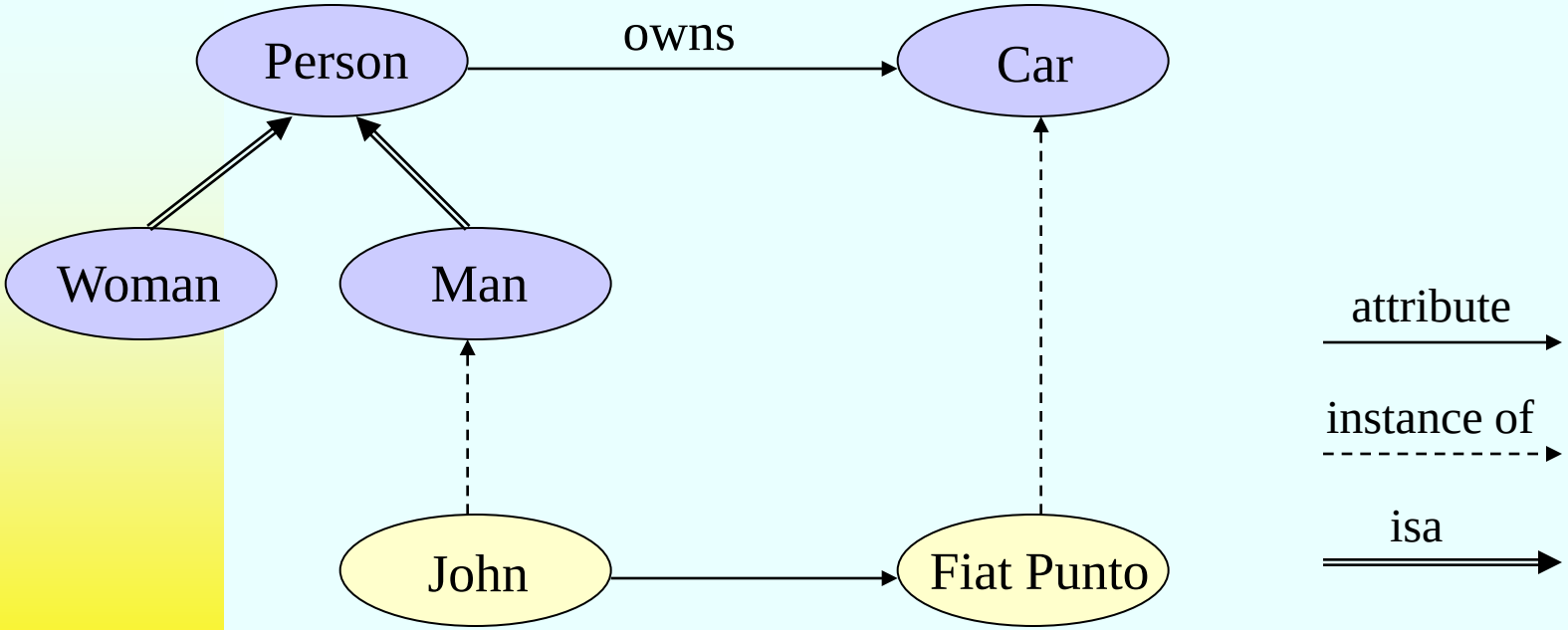
Properties of Generalization

- Every instance of a child entity is also an instance of the parent entity.
- Every property of the parent entity (attribute, relationship or other generalization) is also a property of a child entity. This property of generalizations is known as ***inheritance***.

Simple Example

Objects are structured along three main hierarchies:

- the classification hierarchy
- the generalization/specialization hierarchy
- the aggregation(attribute) hierarchy



Several Generic Relationships

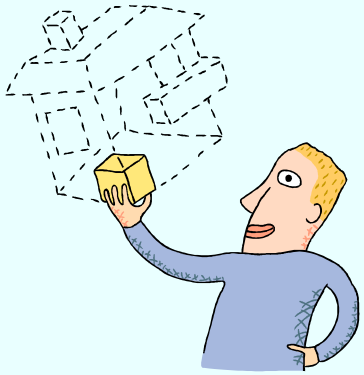
- **Classification**: an instance to its class (e.g., John and person)
- **Generalization**: a superclass to subclasses (e.g., person and employee)
- **Aggregation**: composites (e.g., car) formed from components (e.g., body and engine)
- **Materialization**: a class of categories (e.g., models of cars) and a class of more concrete objects (e.g., individual cars)
- **Ownership**: an owner class (e.g., persons) and a property owned (e.g., cars)
- **Grouping**: a member class (e.g., players in a team) to a grouping class (e.g., teams)
- **Viewpoint**: partial information about a class from a particular standpoint
- **Generation**: new output entities emerging from input entities
- **Versioning**: an object class and its time-varying versions
- **Role**: an object class (e.g., persons) and a role class (e.g., employees), describing dynamic states for the object class

Semantics of Generic Relationships

- **Class and instance semantics**: The semantics of generic relationships concerns both classes and instances of these classes. Must deal with both the class level and the instance level in a coordinated manner. (E.g., for generalization: a class can have several superclasses and several subclasses; an instance of a class is also an instance of all its superclasses)
- **Cardinality**: constrains the number of objects related by a relationship.
- **Composition**: A class plays several roles of the same generic relationship R in several specific relationships based on R.
- **Transitivity**: can follow from composition. E.g., Person ← Student ← GraduateStudent imply Person ← GraduateStudent

Example
(from architectural design)

- *hasGeometry*: relates a symbolic object and a geometric object; each symbolic object has at most one associated geometric object
- *hasPart*: a fundamental relationship for design objects in general (e.g., a floor is composed of a set of rooms, a room has a set of walls and at least one entrance)
- *adjacentTo*: between two objects not related by has part and whose distance is less than a specific threshold
- *connectedTo*: similar to adjacent to, but the two objects have overlapping volumes
- *hasRole* (e.g., a door, a window may function as fire exit)
- *isBlueprintFor*: relates specifications of different levels of abstraction



Conceptual Design

Design choices:

- Should a concept be modeled as an entity or an attribute?
- Should a concept be modeled as an entity or a relationship?
- Identifying relationships: Binary or ternary? Aggregation?

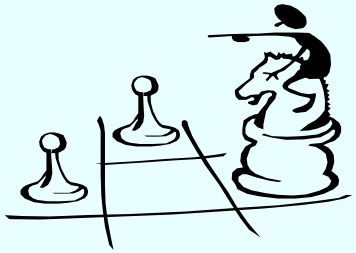
Some Rules of Thumb

- If a concept has significant properties and/or describes classes of objects with an autonomous existence, it is appropriate to represent it as an entity.
- If a concept has a simple structure, and has no relevant properties associated with it, it is convenient to represent it with an attribute of another concept to which it refers.
- If a concept provides a logical link between two (or more) entities, it is convenient to represent it with a relationship.
- If one or more concepts are particular cases of another concept, it is convenient to represent them in terms of a generalization relationship.

Example

Consider the **address** of a person. Is it an entity, relationship, or attribute?

- Consider **address** for a telephone company database, which has to keep track of how many and what type of phones are available in any one household, who lives there (there may be several phone bills going to the same address) etc. for this case, address is probably best treated as an entity.
- Or, consider an employee database, where for each employee you maintain personal information, such as her address. Here address is best represented as an attribute.
- Or, consider a police database where we want to keep track of a person's whereabouts, including her address (i.e., address from Date1 to Date2, address from Date2 to Date3, etc.) Here, address is treated best as a relationship.



Conceptual Modeling Strategies

- The design of a conceptual schema for a given set of requirements is an engineering process and, as such, can use design strategies from other disciplines:
  - Top-down
  - Bottom-up
  - Middle-out
  - Mixed

Top-Down Strategy

- The conceptual schema is produced by means of a series of successive refinements, starting from an initial schema that describes all the requirements by means of a few highly abstract concepts.
- The schema is then gradually expanded by using appropriate transformations that add detail.
- Moving from one level to the next involves modifications of the schema using a basic set of top-down transformations (i.e. from one entity to a generalization, from one relationship to an entity with relationships, adding attributes etc.)

Bottom-Up Strategy

- The initial specification is decomposed iteratively into finer-grain specifications, until a specification becomes atomic, i.e., consists of simple elements.
- Each atomic specification is then represented by a simple conceptual schema.
- The schemas obtained from atomic specifications are then integrated into a final conceptual schema.
- The final schema is obtained by means of a set of bottom-up transformations (i.e. generation of an entity, relationship, generalization, aggregation of attributes on an entity etc.)

Middle-Out Strategy

- This strategy can be regarded as a special case of the bottom-up strategy.
- It begins with the identification of only a few important concepts and, based on these, the design proceeds, spreading outward ‘radially’.
- First the concepts nearest to the initial concepts are represented, and we then move towards those further away by means of ‘navigation’ through the specification.

Mixed Strategy

- Here the designer decomposes the requirements into a number of components, as in the bottom-up strategy, but not to as fine-grain a level.
- Designer also defines a ***skeleton schema*** containing the main concepts of the application. This skeleton schema gives a unified view of the whole design and helps the integration of schemas developed separately.
- Then the designer examines separately these main concepts and can proceed with gradual refinements (following a top-down strategy) or extending them with concepts that are not yet represented (following a bottom-up strategy).

Qualities for a Conceptual Schema

- **Correctness**. Conceptual schema uses correctly the constructs made available by the conceptual model. As with programming languages, the errors can be ***syntactic*** or ***semantic***.
- **Completeness**. Conceptual schema represents all data requirements and allows for the execution of all the operations included in the operational requirements.
- **Readability**. Conceptual schema represents the requirements in a way that is natural and easy to understand. Therefore, the schema must be self-explanatory; for example, by choosing suitable names for concepts.
- **Minimality**. Schema avoids *redundancies*, e.g., data that can be derived from other data.

A Comprehensive Method for Conceptual Design

1. **Analysis of requirements**
   (a) Construct a glossary of terms.
   (b) Analyze the requirements and eliminate all ambiguities.
   (c) Arrange the requirements in groups.
2. **Basic step**
   Identify the most relevant concepts and represent them in a skeleton schema.
3. **Decomposition step** (to be used if appropriate or necessary). Decompose the requirements with reference to the concepts present in the skeleton schema.

A Comprehensive Method for Conceptual Design

4. **Iterative step**

   (a) Refine concepts in the schema, based on the requirements.

   (b) Add new concepts to the schema to describe any parts of the requirements not yet represented.

5. **Integration step**

   Integrate the various subschemas into a general schema with reference to the skeleton schema.

6. **Quality analysis**

   Verify the correctness, completeness, minimality and readability of the schema.

What is Metamodeling?

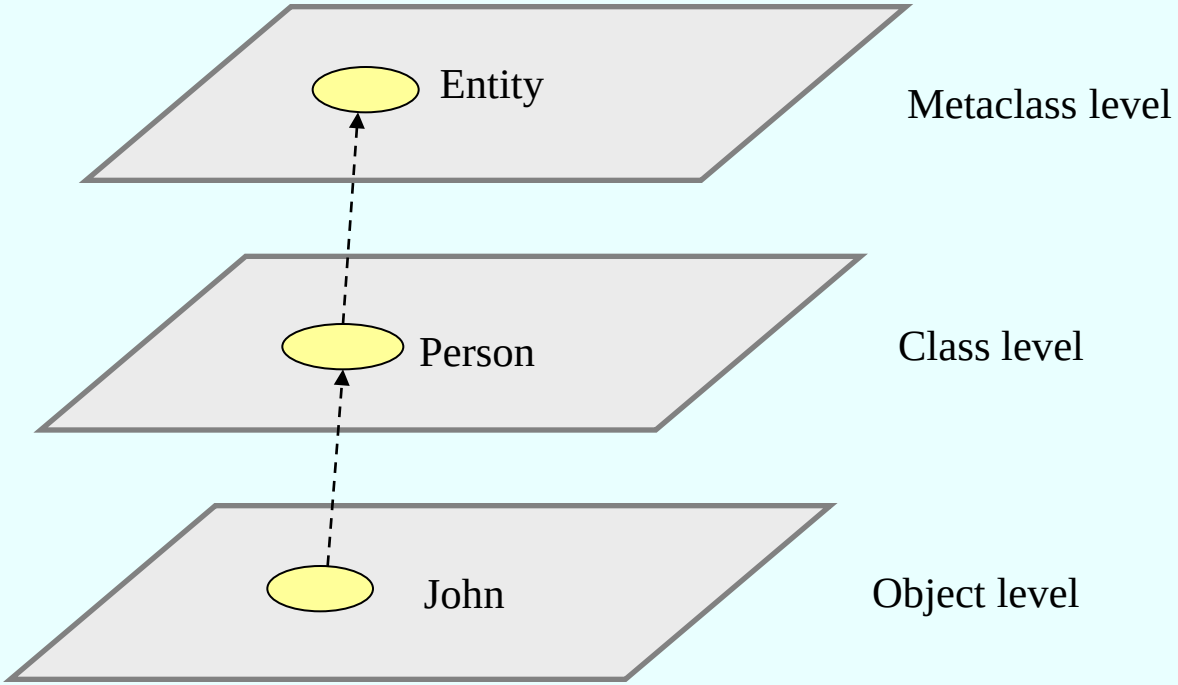
- "Meta" means literally "after" in Greek.
- In Computer Science, the term is used heavily and with several different meanings:
  - In Databases, metadata means "data about data" and refer to things such as a data dictionary, a repository, or other descriptions of the contents and structure of a data source;
  - In Conceptual Modeling, metamodel is a model of a data model, e.g., an ER model of the relational model, or an ER model of the ER model.

Metamodelling

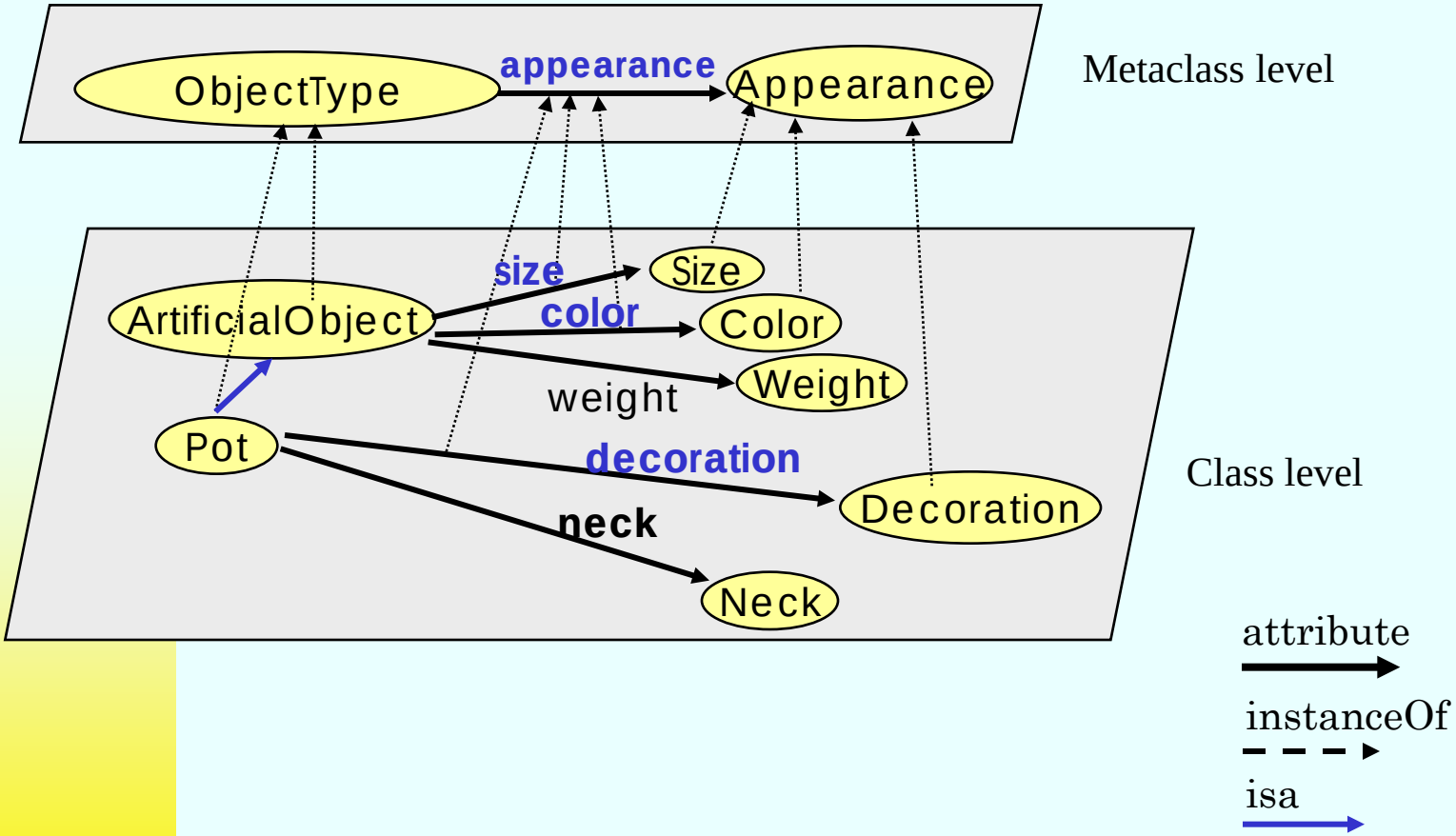
- Data are modelled by metadata (“schemata”, “classes”,...) which are parts of the metamodel; these units are instances of metadata which are parts of a metamodel, etc.

- We’d like to have metamodels which are self-descriptive to an arbitrary level of self-description.

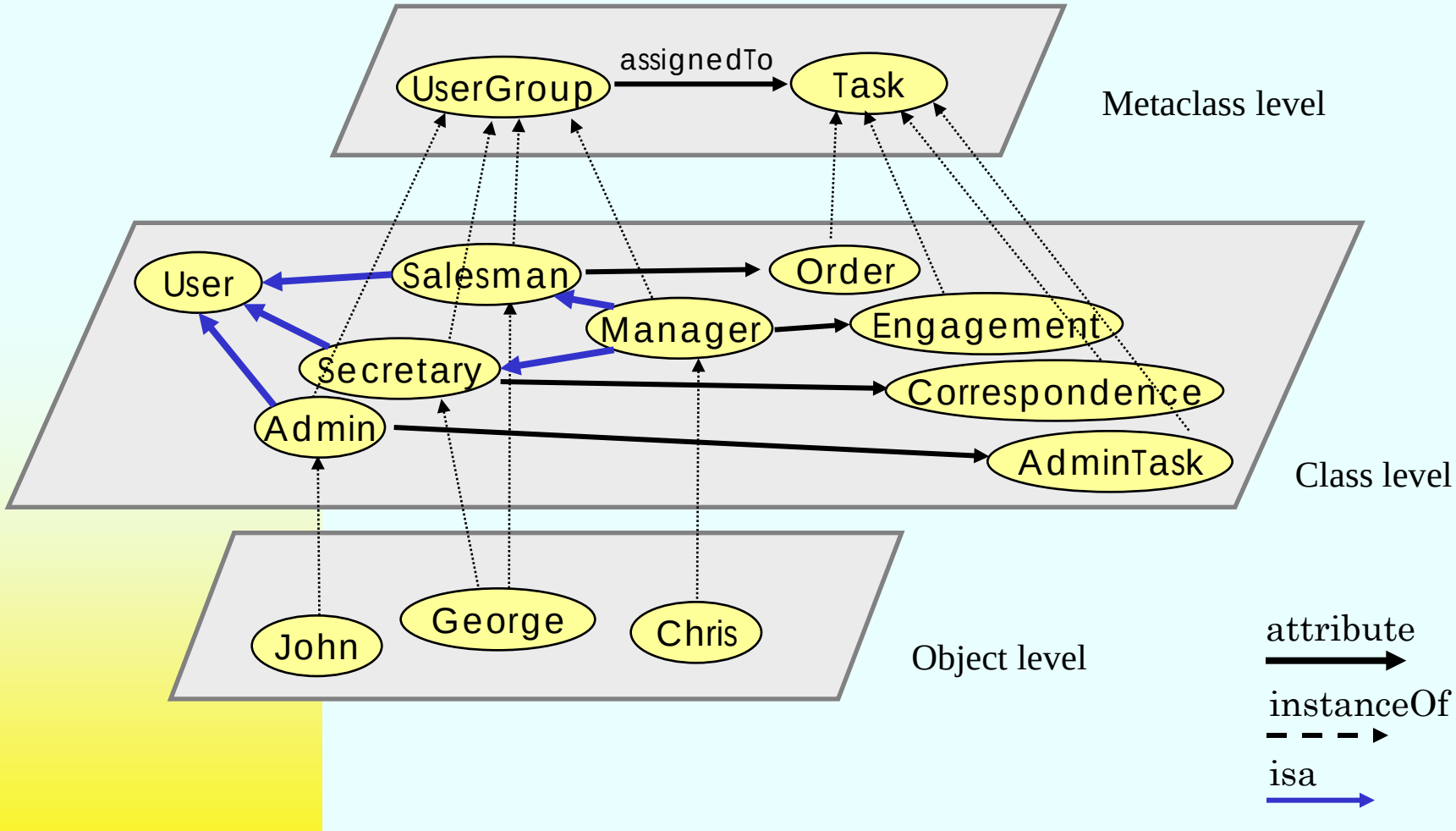
Structural Reflection



Example 1


ObjectType
appearance
Appearance
Metaclass level
ArtificialObject
size
Size
color
Color
Weight
weight
Pot
decoration
Decoration
neck
Neck
Class level
attribute
instanceOf
isa

Example 2


assignedTo
UserGroup
Task
Metaclass level
User
Salesman
Order
Manager
Engagement
Secretary
Correspondence
Admin
AdminTask
Class level
John
George
Chris
Object level
attribute
instanceOf
isa

References

- P. Chen, P., *The Entity-Relationship Model: Towards a Unified View of Data*, *ACM Transactions on Database Systems*, 1976.
- Loucopoulos, P. and Zicari, R., (eds.) *Conceptual Modeling, Databases and CASE: An Integrated View of Information System Development*, Wiley, 1992.
- Atzeni, P., Ceri, S., Paraboschi, S., and Torlone, R.,. *Database Systems*, McGraw-Hill, 1999.
- [Peckham95] J. Peckham, B. MacKellar, and M. Doherty. *Data model for extensible support of explicit relationships in design databases*, Very Large Data Bases Journal, 4:157-191, 1995.
- [Winston87] Winston, M.E., Chaffin, R., and Herrmann, D., *A taxonomy of part-whole relations*, Cognitive Science, 11:417-444, 1987.