

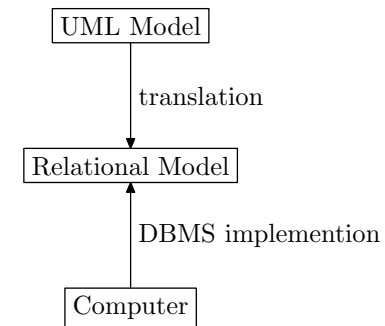
Lecture 9: Modelling and Translation

How to design a database

- What is a model?
- Why model?
- What is UML?
- Decisions
- Translation

See Elmasri and Navathe, Chapters 3 and 4, but especially 12.

Where are we now?



This weeks lectures and next weeks lab session are about the top box and the arrow coming out of it.

What is a Model?

A model of something

- resembles that thing
 - in a particular way, for a specific purpose
 - is smaller, cheaper, simpler
- can take that things place
 - so we can learn about the thing by studying the model
 - which is cheaper, faster, easier, possible

Example Models

Crash-Test Dummy

- resembles a human
- specifically, body movement in crashes
- shows us effects of crash without hurting anyone

Meteorological Model

- a mathematical structure and/or computer program
- resembles the weather and its progress
 - runs faster than reality
 - enables weather prediction

Descriptive and Prescriptive

- those two example models resemble something that exists
- but in software and databases we
 - make a model of something we want, which does not exist
 - create that thing from its model

In database design, we do both.

- the model resembles an aspect of the world that we want to record in the database
- we create a database schema from the model
- therefore the database resembles the world

UML

- In the early 1990's many competing "methodologists" proposed visual object oriented modelling languages
- Three of them, the "three amigos" got together and merged their languages to form UML
- Its now in version 2.1, under the control of the Object Management Group
- The goal is now for modelling to replace programming as the main software development activity.
(Model Driven Development or Engineering or Architecture)
- We are only interested in the most basic diagram types, the **class diagram** [E&N §3.8] and **object diagram**

Why Model?

Model purpose allows us to ignore irrelevant detail, and work at a "higher level of abstraction".

This applies to all phases of system lifecycle

- Requirements capture and analysis
- Documentation
(eg, the UML diagram of the labs database)
- Schema and code generation
- Test case generation and (even) formal verification
- Maintenance: modify model then retranslate

External Resources on UML

The course web page contains a couple of UML related links.

Brief overview of UML

<http://www.ibm.com/developerworks/rational/library/769.h>

A little more detail and advice on UML Class diagrams

<http://www.agilemodeling.com/artifacts/classDiagram.htm>
(controversy alert!)

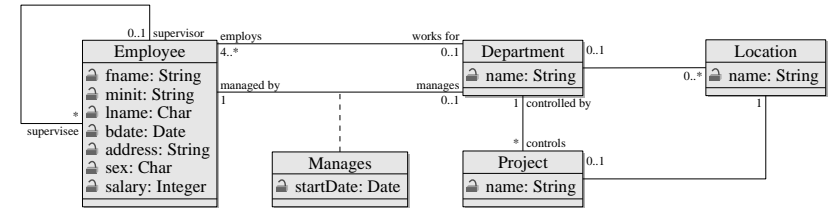
The current official definition of UML

<http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/>

Lab Scenario in UML

The database we used in the first lab is adapted from [E&N Figure 5.5, 5.6]. It is shown as a UML class diagram in [E&N Figure 3.16]. Here is an adapted (and corrected) version to match our lab material.

Lab Scenario in UML



UML and ER

The UML diagram [E&N Figure 3.16] modelling the employees scenario is given as an Entity-Relationship diagram in [E&N Figure 3.15]. We will learn mostly UML, but ER is still used, so read about that too.

- Entity-Relationship diagrams (Chen, 1976) are the origin of “conceptual modelling”.
- UML (mid-1990’s) class diagrams are essentially an object oriented form of Entity-Relationship diagrams.
- UML is probably too big and complicated for its own good, but we will only use a tiny subset that is mostly equivalent to Entity-Relationship diagrams and models.
- However, other parts of UML are useful in database design work. For example, Use-Cases for high level requirements, and Sequence Diagrams for modelling system interactions.
- I am biased, but I think the UML diagrams look much simpler and are easier to understand!

UML Class Diagrams

Even the class diagram part of UML is huge and has many many different elements.

However, it is a defined language, you can not just make up some notation and say “this means . . .”

It is also prudent to use the smallest part of the language you can. Less to learn, less to translate.

UML Elements - Class

The boxes represent classes. Like entities in ER, they represent a kind of thing that can exist in our scenario.

- the top compartment contains the class name
- the middle compartment contains the attributes (warning - same word, different context) it is possible to create domain-like custom types
- the bottom compartment contains operations, but we will not use it

UML Elements - Associations Continued

- Association ends can also have multiplicity constraints. These take the form (lower bound) .. (upper bound). If lower and upper bounds are the same, we don't repeat it.
 - In our example, each Employee can be employed by from 0 to 1 Departments. Each Department can employ from 4 to any number of Employees.
- Following ER terminology, we often talk about many-one and many-many associations

UML Elements - Associations

The lines joining classes are associations. The lines with a diamond at one end are called aggregations. These are also, perhaps incorrectly, considered to be a kind of association.

- Associations can have a name, written near the centre of the line. Our associations do not, but some in [E&N Figure 3.16] do.
- Associations can be **reflexive**. That is, they relate objects of the same class.
- The ends of the association can also have names. These are verb phrases, that make sense when used with the classes at the end.
 - For example, an Employee can **work for** a Department.
 - Note the order: class, far end, other class. ER diagrams do it backwards!

Association Classes

An Association Class is an Association with attributes (or operations or behaviour) of its own.

- In our example, the startDate does not belong to the Employee, nor the Department, but to the management relationship between the two.
- They can be “factored out” by replacing them with a normal class and 2 associations.

and many many more

Some other things you might see in UML Class diagrams.

- aggregation (strong and weak) - show a “part-of” relationship, probably best to use ordinary associations
- generalisation (like inheritance in object oriented programming)
- qualified associations

Reconceptualisation is Hard Work!

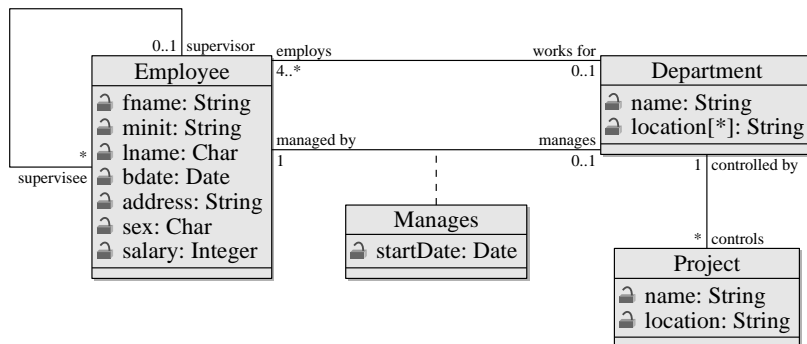
Activities like modelling, programming, formalisation (describing things using mathematical logic) are really hard, and often get stuck.

Why is that?

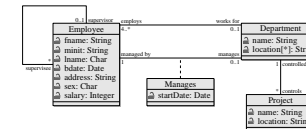
It's because we are trying to convert our ideas into a fixed system of metaconcepts (object, event, entity, relationship, ..).

There can be many ways to do it, no obvious criteria for making a choice. The ideas sometimes just don't fit!

Decisions Decisions



Location: Class (Entity) or Attribute?



- A revision of the example model, location is now an Attribute, not a Class
- Multiple valued attributes used (a UML feature considered harmful by some!)
- Choices like this can be controversial (and therefore time consuming)

Exercise

Modify the class model replacing the association class with an ordinary class and 2 associations.

Hint: take care with multiplicities.

Translation (Outline)

- For each class, create a relation schema with
 - 1 a relation attribute for each class attribute
 - 2 specify the primary key, as indicated in the diagram
- For each many-many association, create a relation schema with a foreign key for each association end
- For each many-one association, add a foreign key to the class at the many end

Exercise

Translate the modified UML class diagram from the previous exercise.

Note places where you must make a choice: what detail should be added to the procedure to make it fully automatic?