Data and Process Modelling

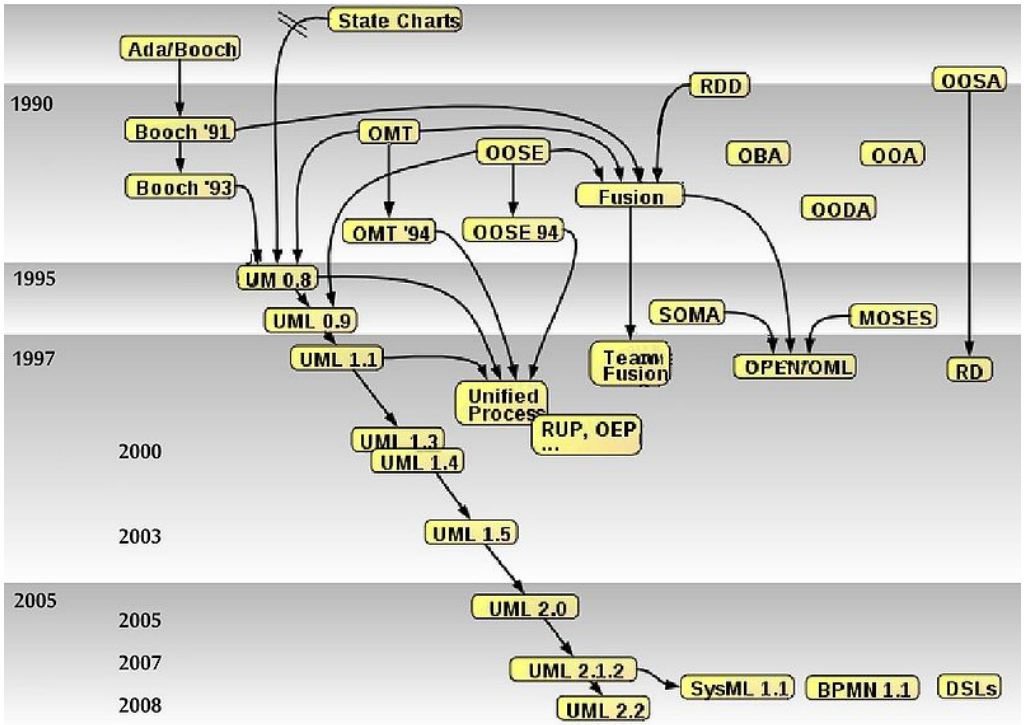
Lab 5. UML Classic Diagrams and ORM

Marco Montali

KRDB Research Centre for Knowledge and Data
Faculty of Computer Science
Free University of Bozen-Bolzano

A.Y. 2015/2016

UML: Modeling Standard for OO Software Engineering



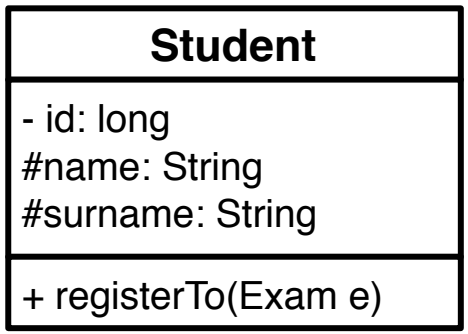
- Born from:
  - 3 amigos:
    - ★ Rumbaugh's Object-modeling technique;
    - ★ Booch's OO design;
    - ★ Jacobson's OO software engineering method.
  - Harel's state-charts.
- OMG standard since 1997.

- Family of notations:
  - Structure diagrams: class/object diagram, component, composite structure, deployment, package, profile.
  - Dynamic diagrams:
    - ★ Behavior: use case, state machine, activity.
    - ★ Interaction: communication, interaction overview, sequence, timing.

UML Class Diagrams

Used for object-oriented logical modelling, with incremental refinements.

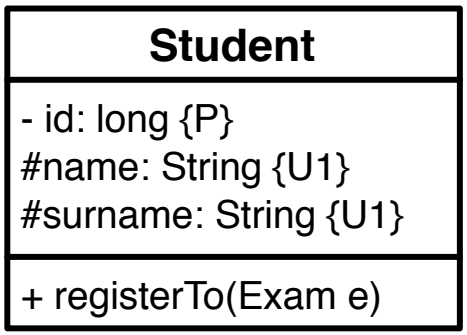
- First step: structural conceptual modelling
  $\rightsquigarrow$ alternative to ORM, with three main distinctions:
  - No preferred identification schemes.
  - Distinction between relationships and attributes.
  - ORM much more expressive than UML in terms of graphical constraints. UML complements the graphical notation with textual constraints (cf. OCL).
- Second step: logical-level information related to object-oriented software development.
  - Visibility attributes (+: public, –: private, #:protected, ∼: package).
  - Relationships navigation.
  - Link to behavioral aspects (operations).



Object Identification

An object is implicitly identified by an internal object identifier (e.g., memory address).

Preferred identification schemes and uniqueness constraints can be added as annotations (cf. vertical layout for database schemas).



Attributes

Properties relevant for a class.

Define relations that are "polarized" by the class (not always easy to decide how).

Multiplicity

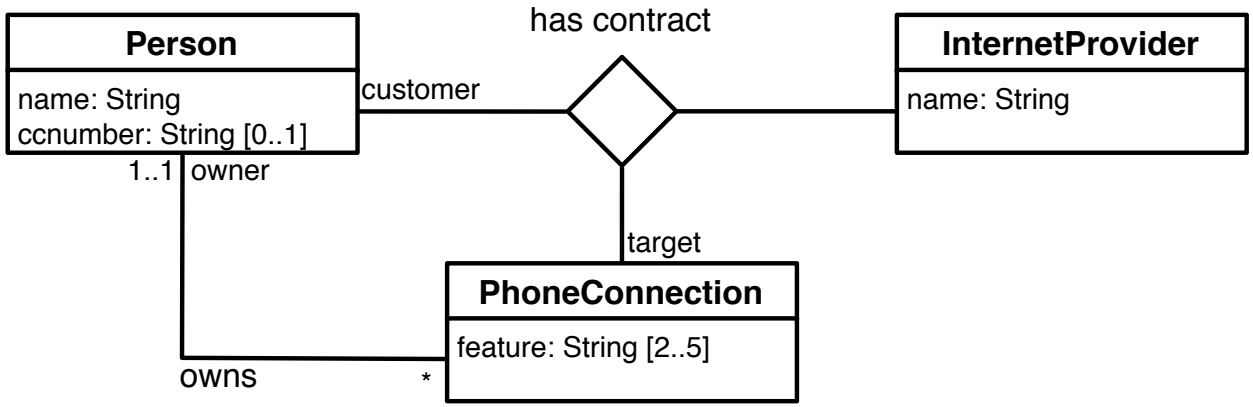
By default, attributes are mandatory and single valued. To override the default, the following annotations can be used:

- $0..1$: optional single valued.
- $0..*$ (or simply $*$): zero or more.
- $1..*$: one or more.
- $n..*$: at least n.
- $n..m$: at least n and at most m.

Associations

Fact types of arity ≥ 2.
Unary fact types encoded using boolean attributes.

- Binary associations: depicted as lines, possibly annotated with multiplicity constraints.
- Higher arity associations: depicted with a diamond, usually without multiplicity constraints (for the sake of readability).



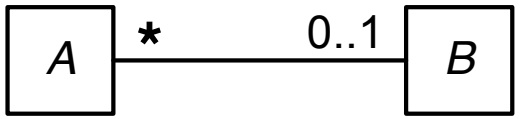
As in ORM, classes can have an indication about their roles.

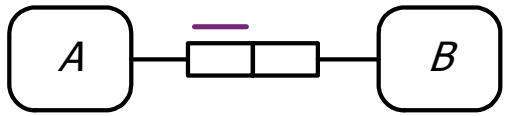
Multiplicities on Binary Associations

- Multiplicity constraints cover frequency constraints (including functionality) and optionality.
- Multiplicity constraints have a "look-across" semantics.
- By default, the multiplicity is $*$ (differently from attributes!).
- UML explicitly accounts for two kinds of part/whole relations: composition (black diamond), aggregation (white diamond).
  - See the discussion about mereology for the subtle issues that emerge for part/whole relations.

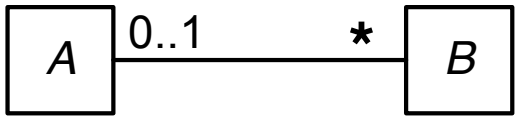
Multiplicities on Binary Associations: Main Cases

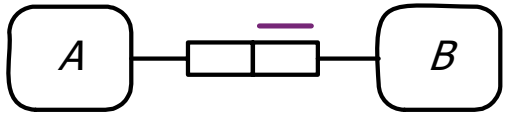
n:1
both roles optional



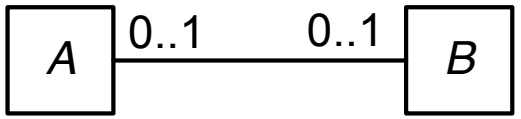


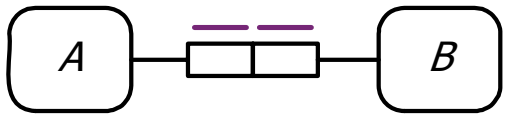
1:n:
both roles optional



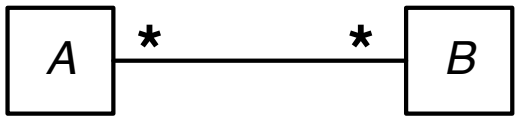


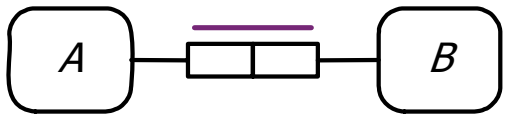
1:1
both roles optional



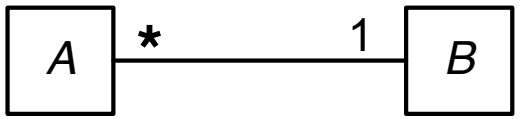


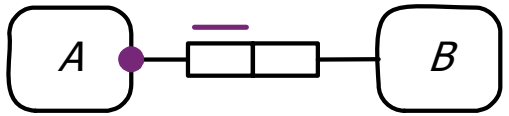
m:n
both roles optional



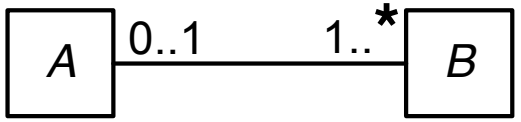


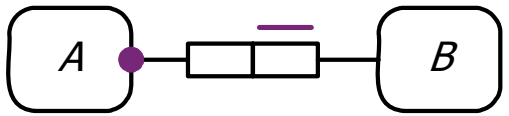
n:1
first role mandatory



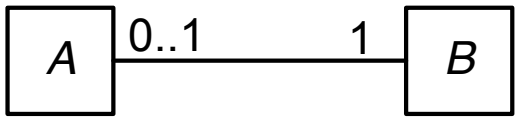


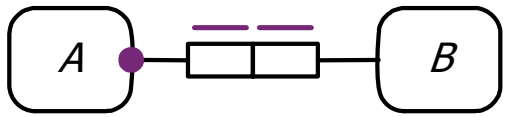
1:*n*
first role mandatory



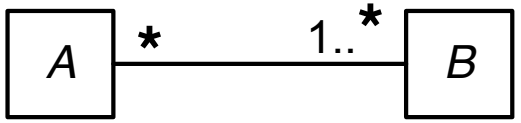


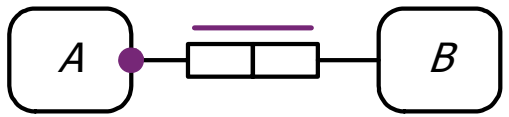
1:1
first role mandatory





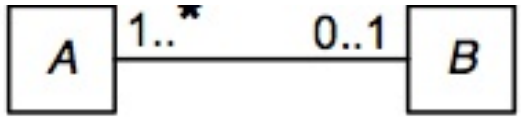
m:n
first role mandatory



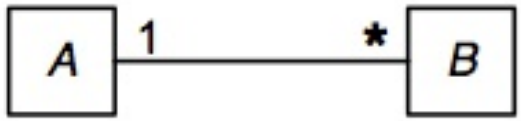


Multiplicities on Binary Associations: Main Cases

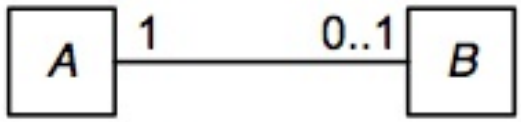
n:1
second role mandatory



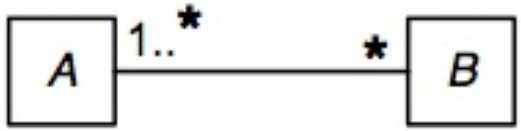
1:*n*:
second role mandatory



1:1
second role mandatory

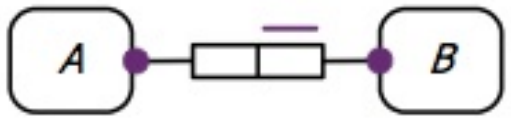


m:*n*
second role mandatory

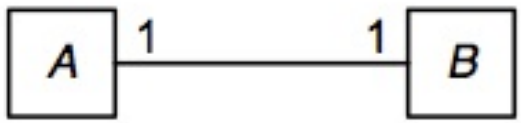


n:1
both roles mandatory

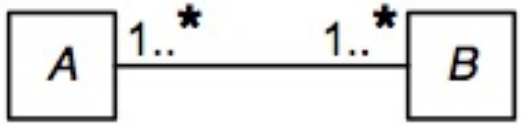
1:*n*
both roles mandatory



1:1
both roles mandatory

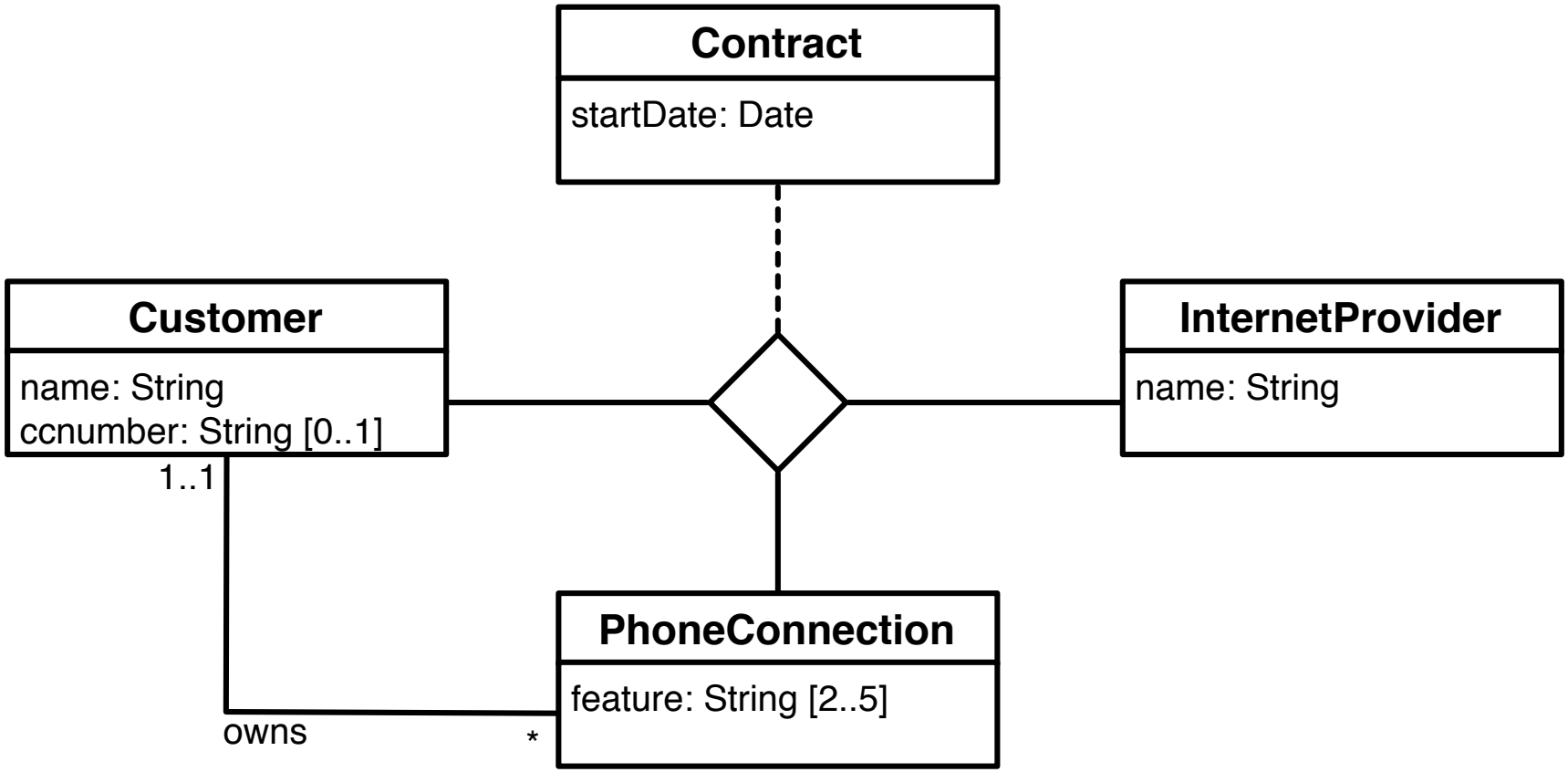


m:*n*
both roles mandatory

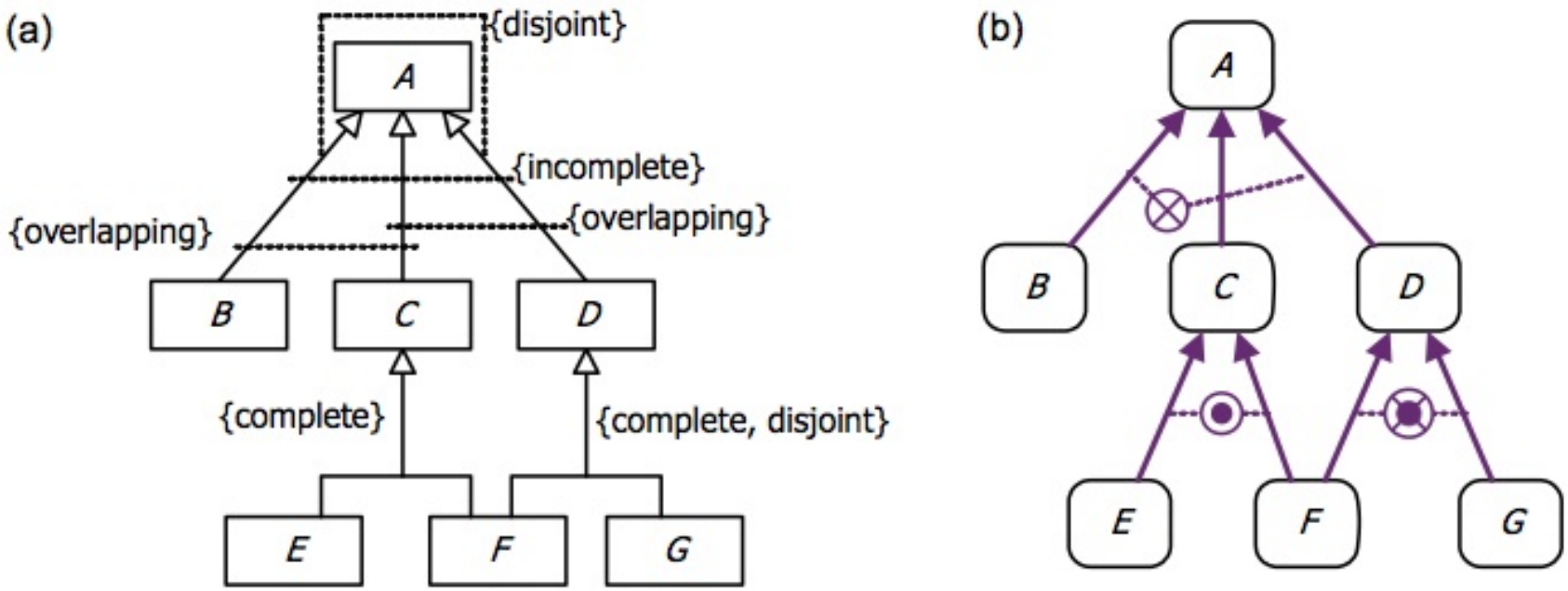


Objectification of Associations

Similar to ORM.



Subtyping



The name of an attribute used to discriminate between subclasses can be placed near the subtype arrow.

- An enumeration type can be introduced to define the cases. This corresponds to a value constraint in ORM.

From ORM to UML

Step	*Action*
1	Binarize any sets of exclusive unaries
2	Model selected object types as classes, and map a selection of their *n*:1 and 1:1 associations as attributes. To store facts about a value type, make it a class
3	Map remaining unary fact types to Boolean attributes or subclasses
4	Map *m:n* and *n*-ary fact types to associations or association classes. Map objectified associations to association classes
5	Map ORM constraints to UML graphic constraints, textual constraints, or notes
6	Map subtypes to subclasses, and if needed, subtype definitions to textual constraints
7	Map derived fact types to derived attributes/associations, and map semi-derived fact types to attributes/associations plus derivation rules