POLYTECH PARIS-SUD

2021

Cycle Ingénieur – 2ème année
Département Informatique

Verification and Validation

Part II : A Revision of the UML

Burkhart Wolff
Département Informatique
Université Paris-Saclay / LMF

Plan of the Chapter

- Introduction to the UML notation
- Syntax and semantics of class model elements and their visialization in diagrams
  - Class Invariants
  - Constraints
  - Operations
  - Pre- and Post-Conditions
- Syntax and semantics of state machines

Ultimate Goal:

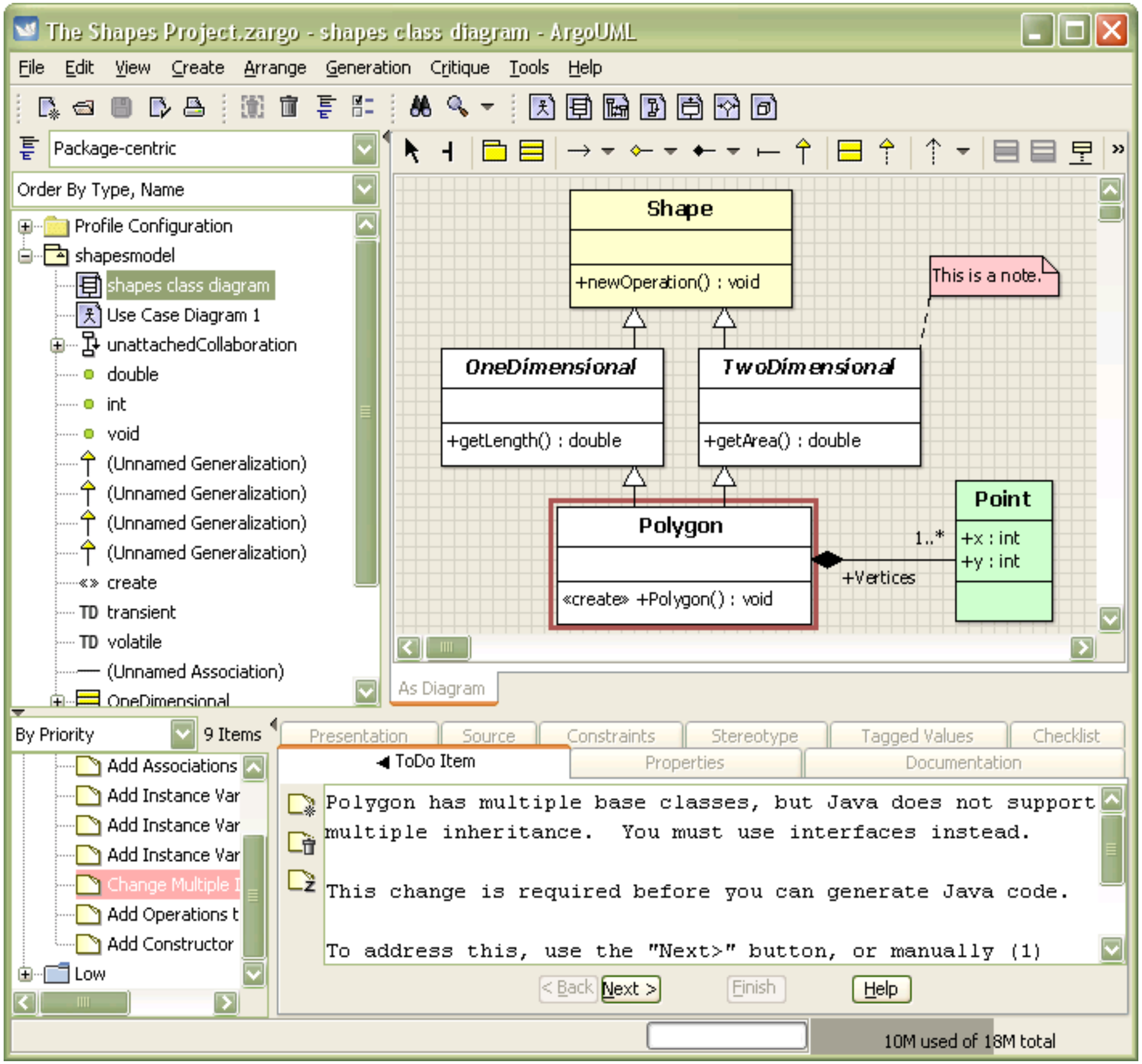
Specify system components for test and verification

The UML ...

- ... is the Unified Modeling Language
- ... is a normed data-structure, a „technical format“ of model-elements (that may contain other model-elements) with consistent naming for
  - various system descriptions
  - various code formats
- ... has various external representations
  - as XMI exchange format (tool-independent in theory ...)
  - as UML diagrams

The UML offers the advantage ...

- ... of being a basis for Integrated Development Environments
  (IDE's like ArgoUML, Poseidon, Rational Rose, ...)



```
Polygon has multiple base classes, but Java does not support
multiple inheritance.  You must use interfaces instead.

This change is required before you can generate Java code.

To address this, use the "Next>" button, or manually (1)
```

The UML offers the advantage ...

- ... of being a basis for Integrated Development Environments (IDE's like ArgoUML, Poseidon, Rational Rose, ...)
- ... to offer „object-oriented“ specifications
- ... to offer a formal, mathematical semantics (well, at least to parts of the UML)
- ... to be fairly widely used in industry, even if not always supported entirely
- ... is the basis for a whole software-engineering paradigm called Model-Driven Engineering (MDE).

The UML 2.0 Diagrams (for corresp. models)

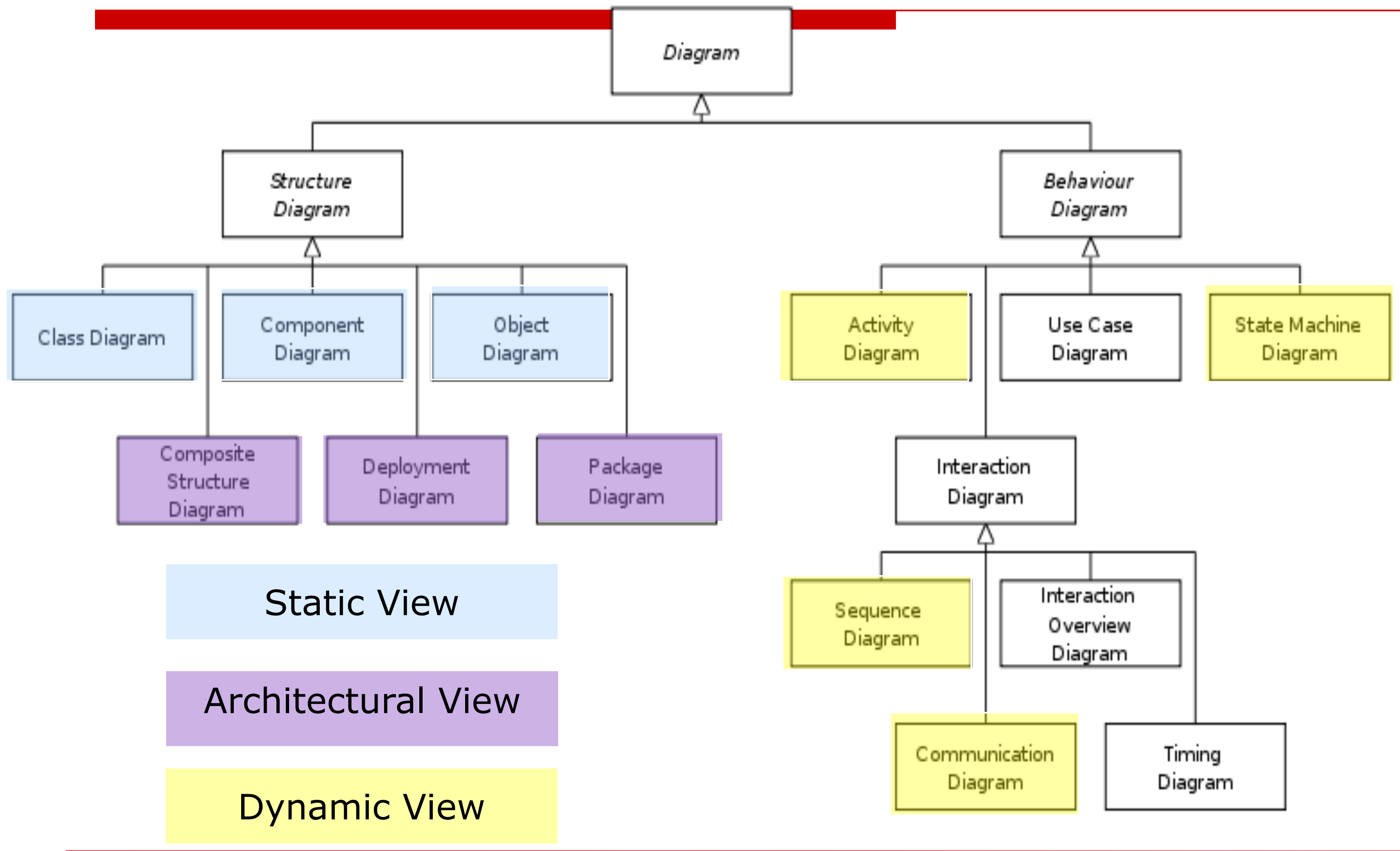
- UML, Version 1.1 : 9 types of diagrams

- UML, Version 2.0 adds

  4 more types of diagrams

  - structure composition
  - communication
  - packaging
  - temporal constraints (timing)

The UML 2.0 Diagrams (for corresp. models)



Principal UML diagram types (1)

- **Structure and Vizualization**
  - **Use Case Models and Use Case Diagrams**
  - **Sequence Models and Sequence Diagrams**
  - **State Machines and State Charts**
  - **Class Models and Class Diagrams**
  - **Object Graphs and Object Diagrams**

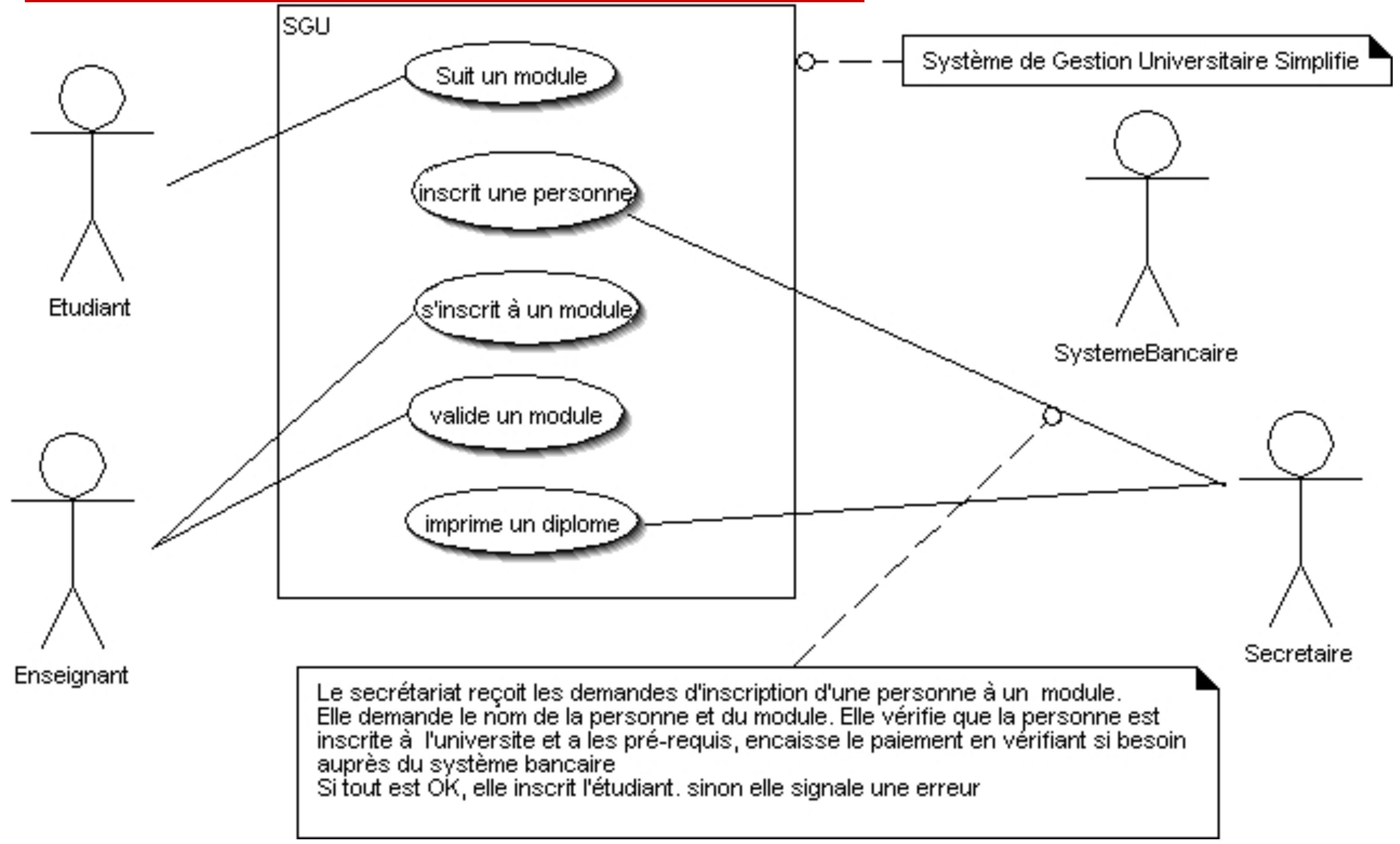
All these Model Elements are discribed in a UML-document itself, the „Meta-Object-Framework“ (MOF)

Principal UML diagram types (1)

- **Use Case Diagrams** („Diagrammes des cas d'utilisation") :
  models the system **operations by**
  - the **interactions** of the system with the external world (external agents communicating with the system seen as a black box.)
  - Just the priciple cases, the alternatives, the extensions

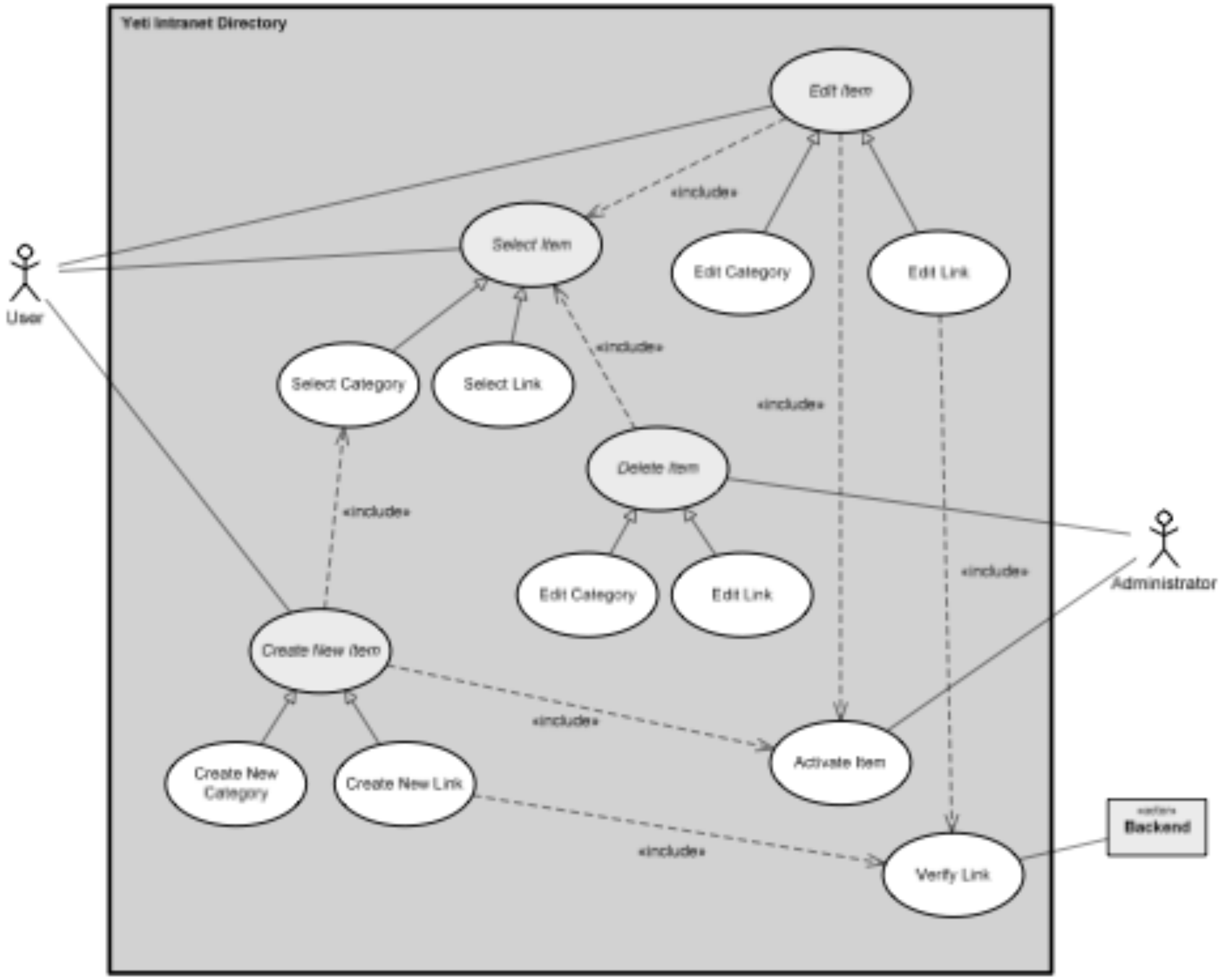
Emphasis on (top-level) functionality !

Example: Use Case Diagram (Conceptual)



Le secrétariat reçoit les demandes d'inscription d'une personne à un module.
Elle demande le nom de la personne et du module. Elle vérifie que la personne est inscrite à l'universite et a les pré-requis, encaisse le paiement en vérifiant si besoin auprès du système bancaire
Si tout est OK, elle inscrit l'étudiant. sinon elle signale une erreur

Example: Use Case Diagram (Design)



Summary: A «Use Case Diagram»

- A Use-Case Diagram
  - ... just represents the principal user-classes (**stake-holders**) of a system
  - ... and the top-level „activities“
  - ... is useful during conceptual modeling in requirement engineering
  - ... has no real semantics,
  - ... but is often used to configure templates
    - for interfaces
    - security settings

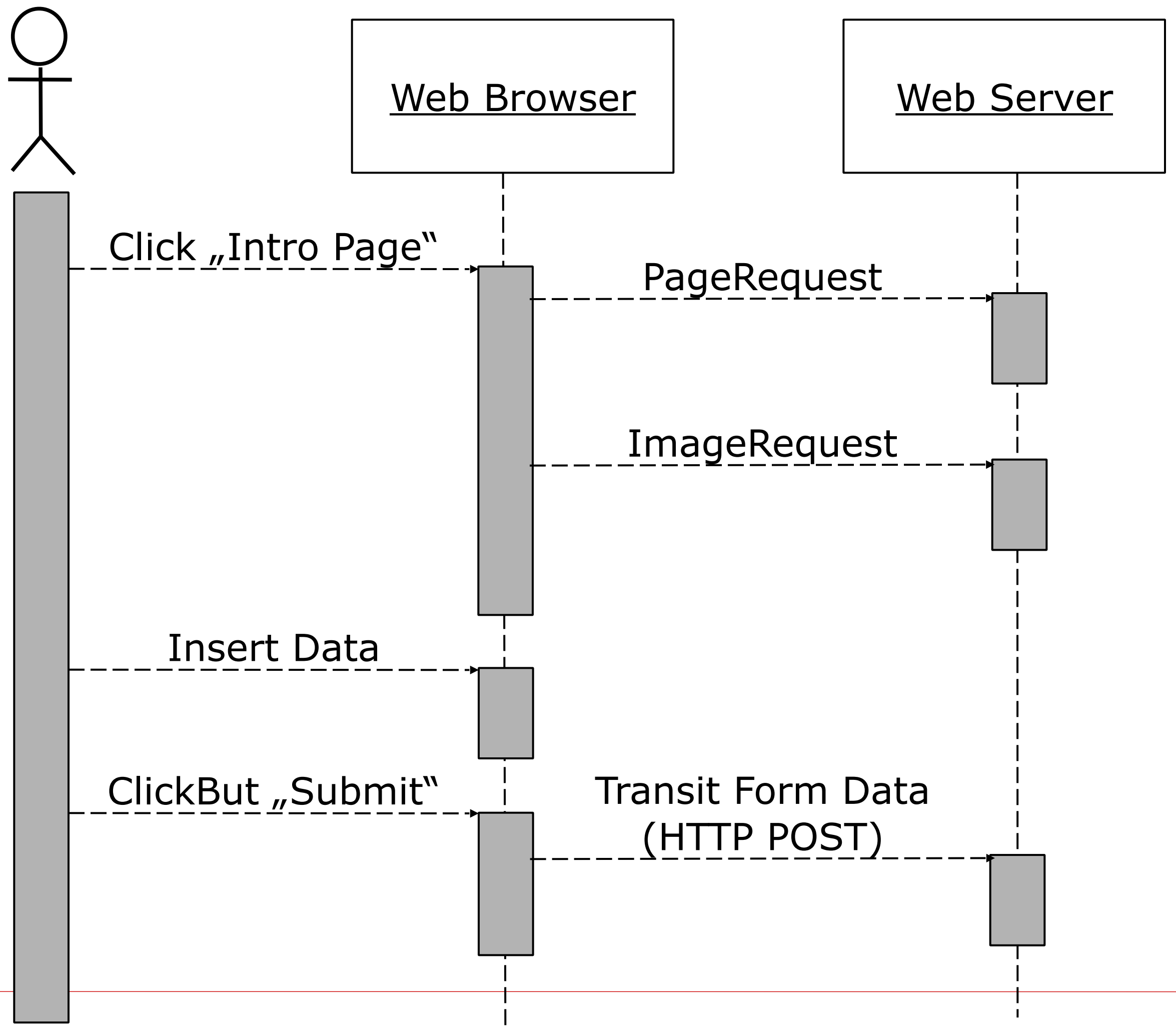
Principal UML diagram types (2)

- **Interaction Diagram** („Diagrammes d'interaction"):

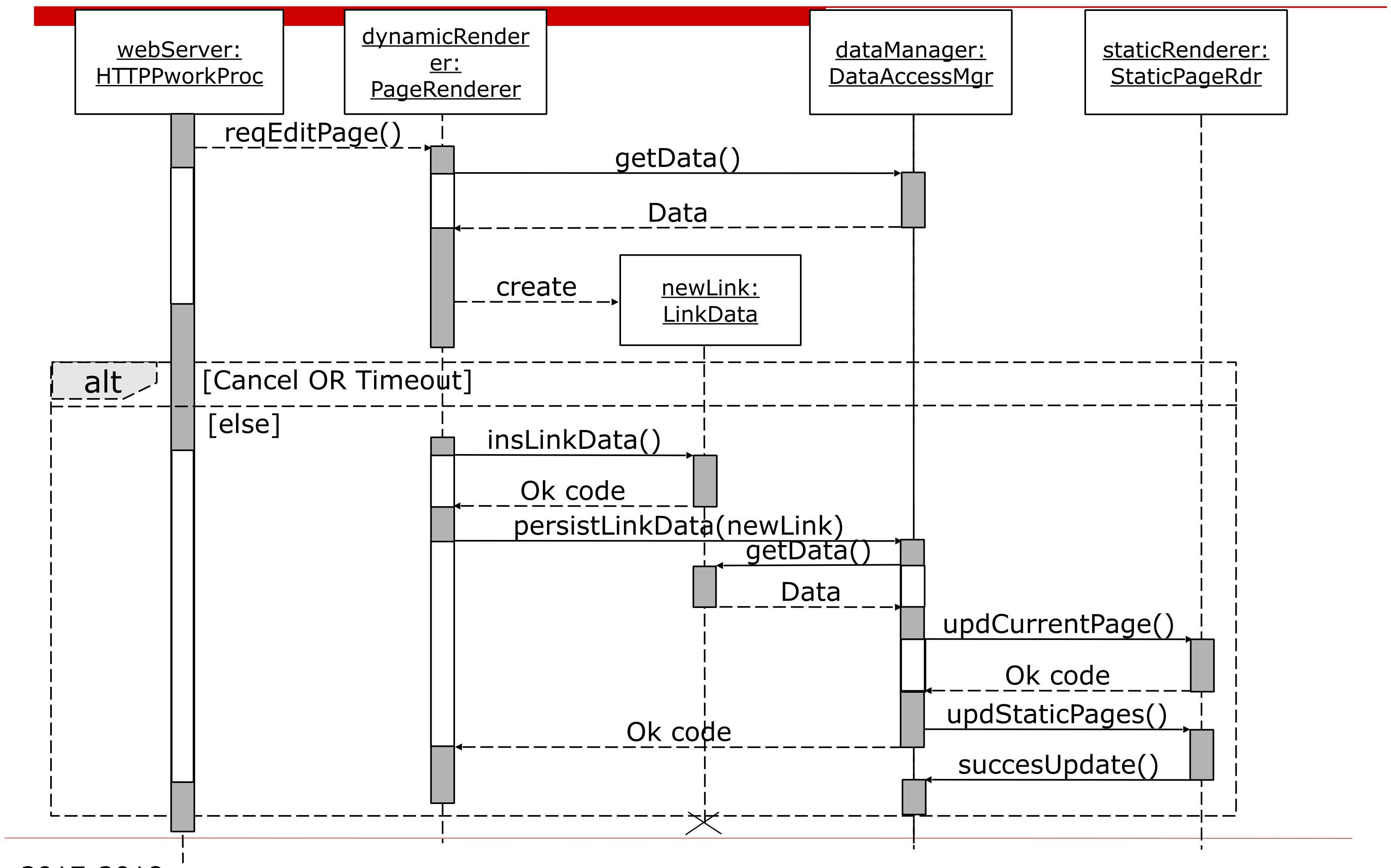
  the interacion between objects for realizing a functionality

  - **SequenceDiagram**: privileged temporal description of exchanges of events. Notions of utilization **scenarios**.
  - **Collaboration Diagram**: centered around objects and top-level collaborations of them.

Example: Sequence Diagram (high-level)



Example: Sequence Diagram (design-level)



Summary: Sequence Diagrams (a)

- Two types can be distinguished:
  - **Diagrams for requirements analysis**:
    description for use-case scenarios of the system, i.e.
    examples of the interactions of the system, i.e.
    of top-level behaviour. Good for error-cases.
  - **Diagrams for system or protocol design:**
    communications between different instances
    of operations; or events occuring in state machines.
    Processes can be created, and synchronous and
    asynchronous communications were modeled.
    Alternatives possible.

Summary: Sequence Diagrams (b)

- Two types can be distinguished:
  - **Semantics of Diagrams for requirements analysis:** none.
  - **Semantics of Diagrams for system design:** many ;-)

    Can be interpreted in Temporal Logic and therefore in automata in many ways ...

    Mostly depends what tools make out of it ;-(

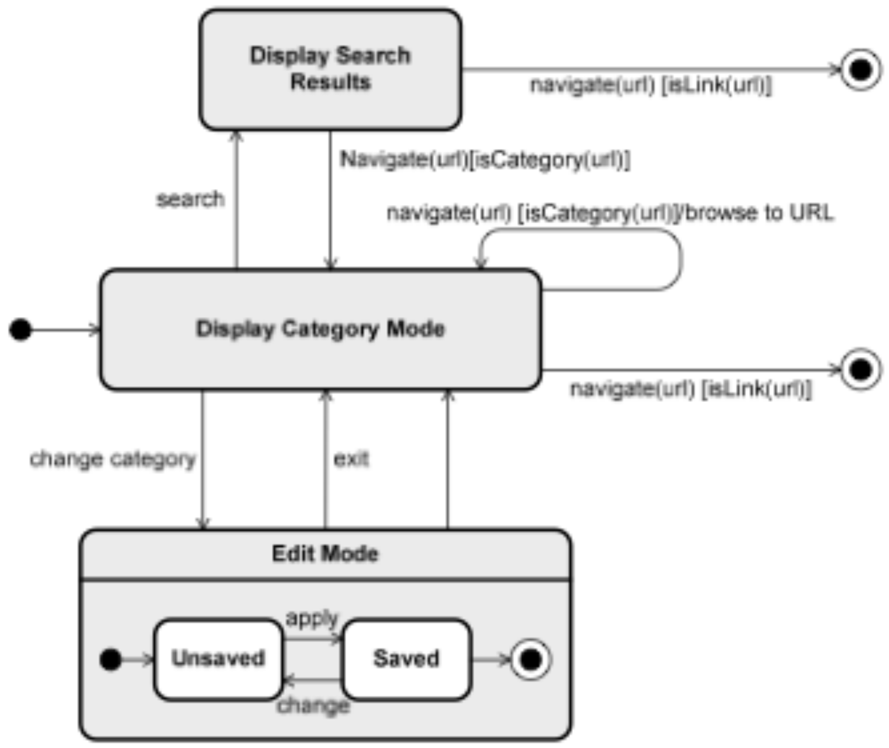
Principal UML diagram types (3)

- **State Charts** (ou « machine à états ») :

  a description of **behaviour** by (hierarchical) automata

  - interesting if an object reacts on events (asynchronous as well as synchronous) by the external environment
  - or if the internal state of an object leads to a somewhat interesting life-cycle of an object (transitions between well-characterized states of the object)

Example: **State Chart (design level)**


Display Search Results
navigate(url) [isLink(url)]
Navigate(url)[isCategory(url)]
search
navigate(url) [isCategory(url)]/browse to URL
Display Category Mode
navigate(url) [isLink(url)]
change category
exit
Edit Mode
apply
Unsaved
Saved
change

Summary: State Charts

- Two types can be distinguished:
  - **Semantics of Diagrams for requirements analysis:** many.
  - **Semantics of Diagrams for system design:** **many.**

  Can be interpreted in by automata, process calculi, Labelled Transition Systems (LTL) in several, reasonable ways (depends on context and application).

Principal UML diagram types (4)

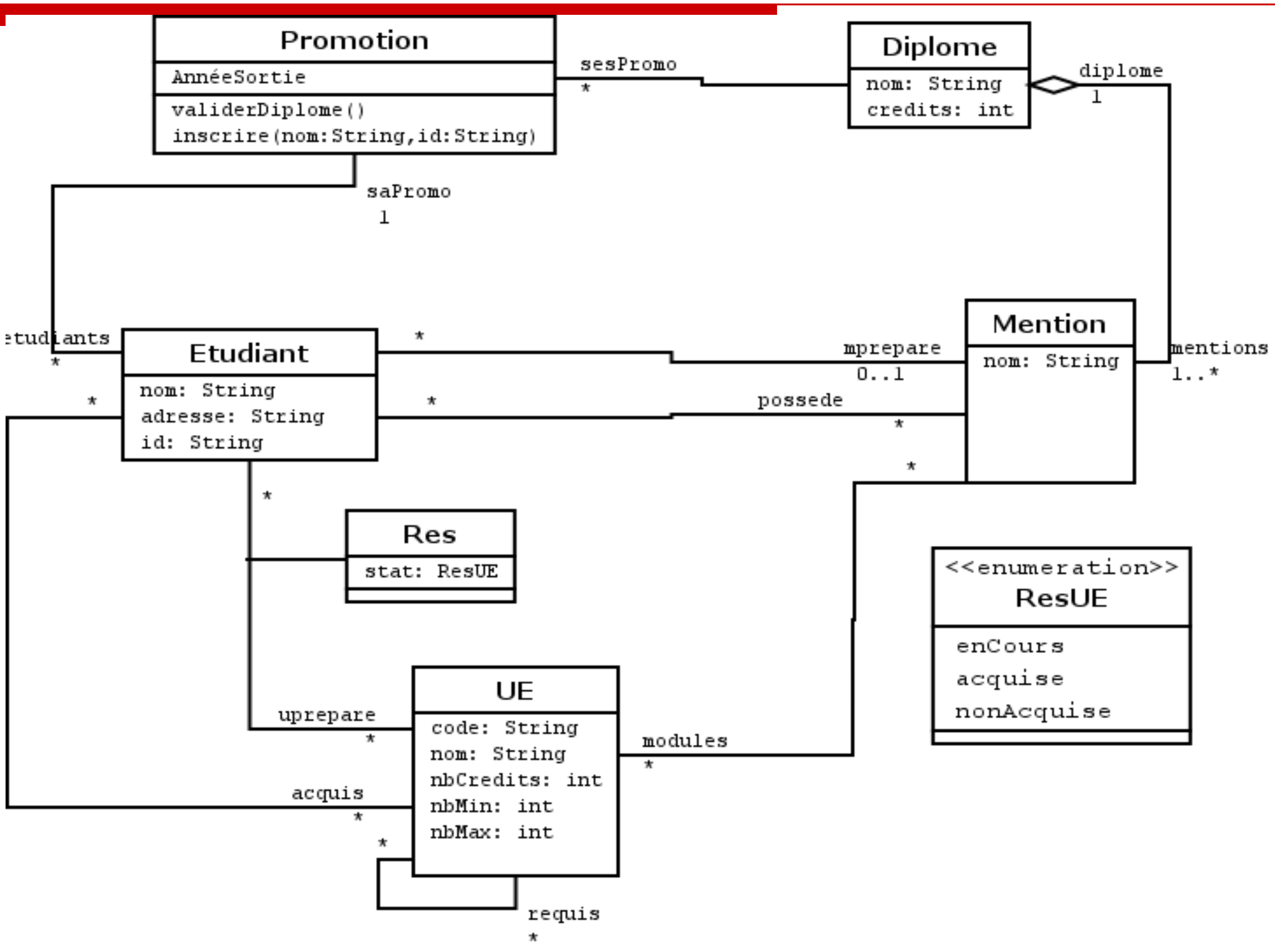
- **Class Diagrams** („Diagrammes de classes") :

  the static **structure** of the DATA of the system

  - the classes of interest to be represented in the system
  - the relations between classes
  - the attributes and the methods
  - the types, required/defined interfaces …

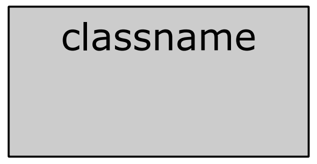
  can be used for top-level views as specific interfaces for local code ...

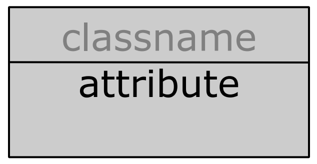
Example: A Class Diagram

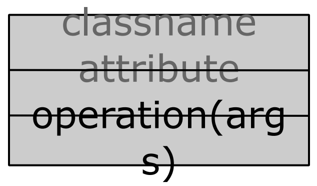


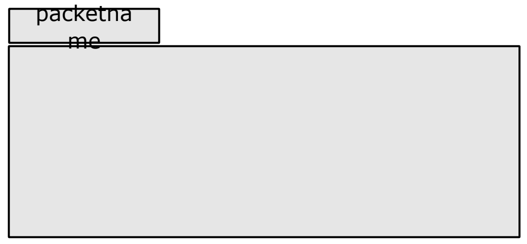
A propos Class Diagrams (1)

- Model-Elements
  - Class
  - Attributes
  - Operations (methods)
  - Packages (grouping mechanism for parts of a class model)



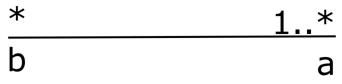




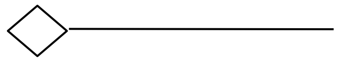


A propos Class Diagrams (2)

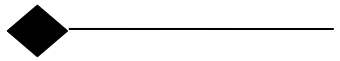
- Model-Elements
  - Association (with optional roles cardinalities)



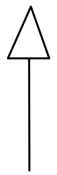
  - Aggregation (« has a » relationship with weak linkage)



  - Coposition (« has a » relationship with strong linkage)

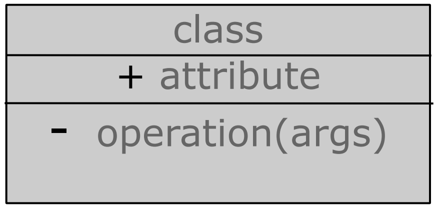


  - Specialization (modeling of a „is-a" relationship between classes)

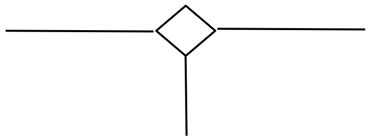


A propos Class Diagrams (3)

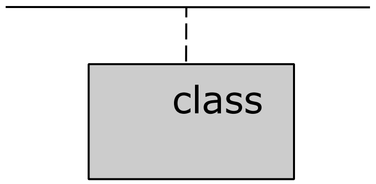
- Model-Elements
  - Visibilities
    (optional public
    and private, see more later)



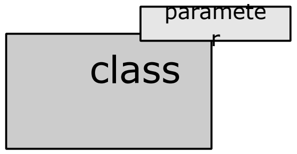
  - N-ary associations



  - Association Class



  - templates with parameter
    (usually classes)

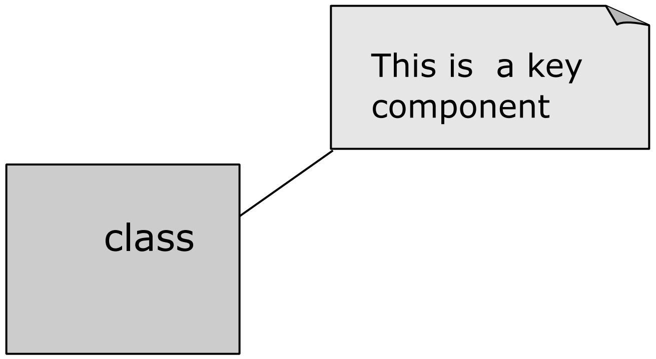


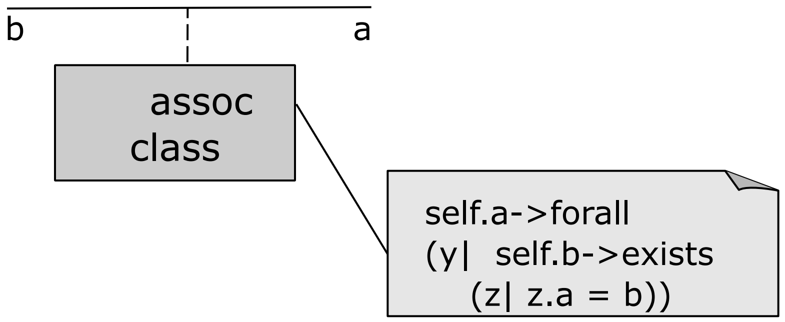
A propos Class Diagrams (4)

- Model-Elements
  - Annotations

    ... typically on classes

    ... can be informal text as well as OCL (see next part !)





A propos Class Diagrams (1)

- Semantics: Classes are:

  - types of objects
  - tuples „attributes" AND
    association ends (« roles »),
    which are collections (Set, Sequence, Bag) of
    references to other objects
  - objects may be linked via references
    to each other into a state called „object graph"
  - cardinalities, etc. are INVARIANTS in this state.

A propos Class Diagrams (2)

- Attributes
  - can have simple type (Integer, Boolean, String, Real) or primitive type (see Date example) only !
  - in diagrams, attributes may NOT have collection type (use therefore **associations)**
  - In a requirement analysis model, everything is **public** by default (we will refine this notion later)

A propos Class Diagrams (3)

- operations (in an analysis class diagram)
  - we will only distinguish operations linked to a use-case diagram
  - we will sometimes not even link them to a specific class - this will come later.

- operations (in an design class diagram)
  - a complete interface;
    can be compiled from a JAVA Interface !

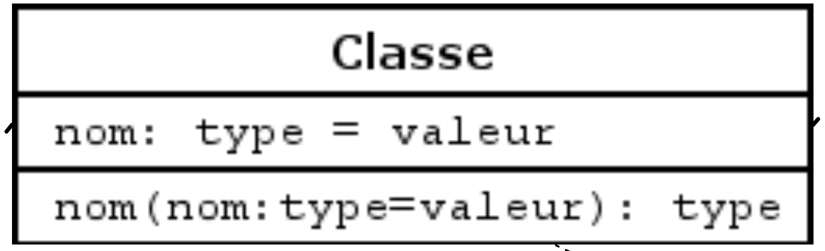
Class Diagrams in Requirements Analysis

The **static aspects** of a model were represented by

- **The class diagram**
  Classes with their attributes
  Class hierarchies via inheritance
  Relations between classes (associations + cardinalities)
  The „roles" at the association ends give an intuitive semantics
- The **invariants** make the description complete ...
  ce qui n'est pas exprimable directement dans le diagramme
  Plages de valeurs ou contraintes sur des attributs
  Contraintes complexes sur une association isolée
  Contraintes globales sur un ensemble d'attributs/associations
  Contraintes sur un ensemble d'instances des classes

More Specific Details in UML 2

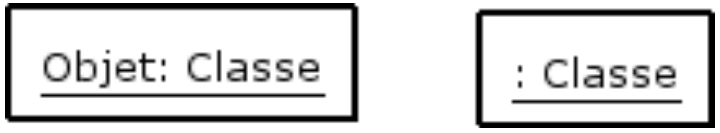
***Visibilities**:*
+: public
- : private
#: protected
/ : derived



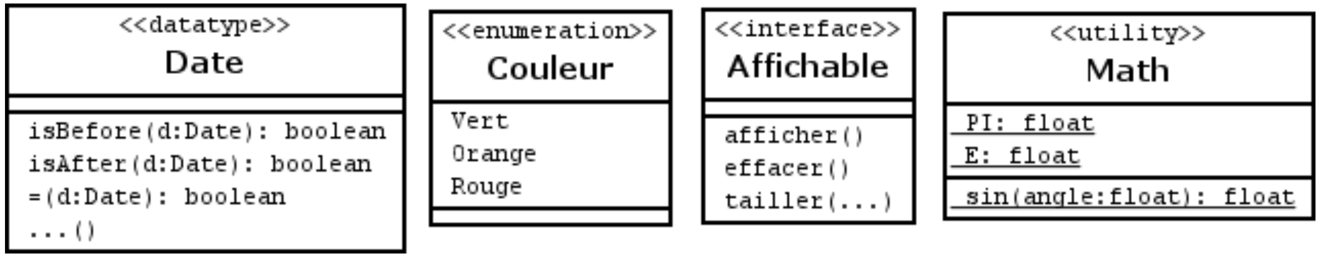
***Modifiers**:*
<u>static</u>
abstract

***Parameter modes**:*
in (par défaut)
out
in out

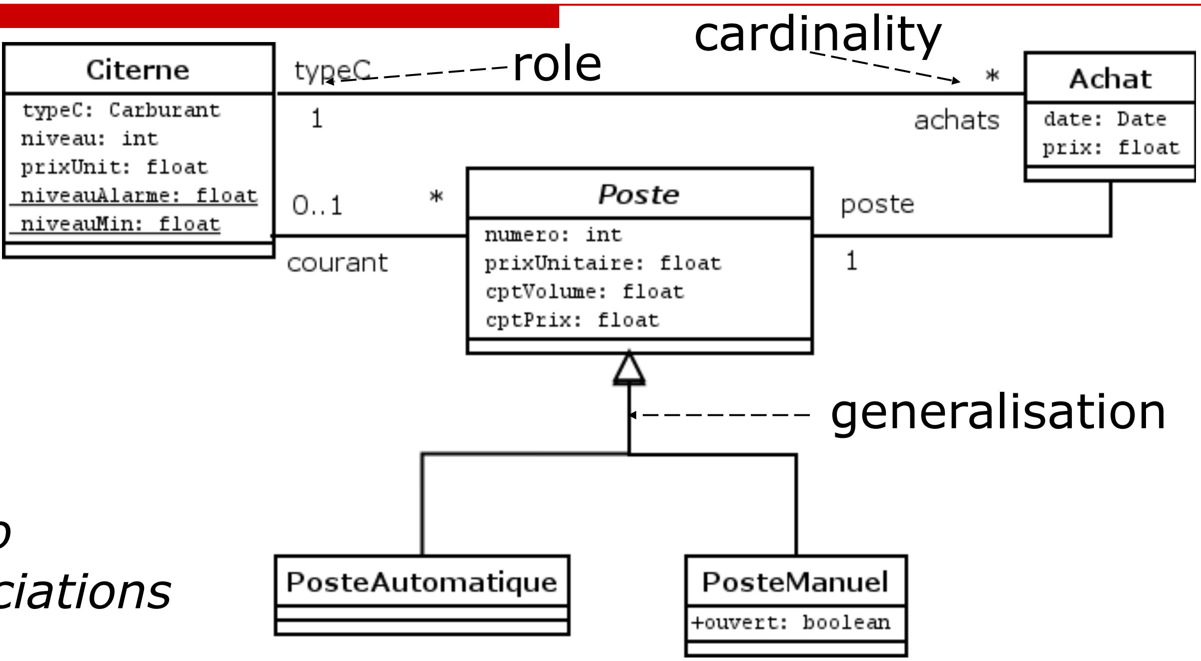
***Instances**:*



Stéréotypes:



More Specific Details in UML 2



The roles were used to navigate accross associations

for `a:Achat`, the OCL expr `a.poste` denotes an instance of `Poste`.

for `c:Citerne`, the OCL expr `c.achats` denotes an instance of `Achat`

for `p:Poste`, the OCL expr `p.courant` corresponds to a collection of 0 or 1 instances of `Citerne`.

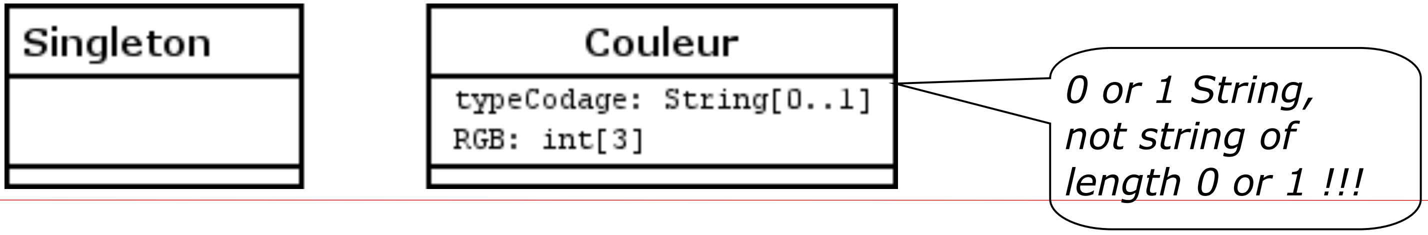
Le nom de classe peut servir de rôle par défaut (si pas d'ambiguïté)

More Specific Details in UML 2

Cardinalities in associations can be:

- 1, 2, or an integral number (no expression !)
- * (for « arbitrary », ...)
- an interval like 1..*, 0..1, 1..3, (**not** like 1..N)
  - on donnera systématiquement les cardinalités
  - Attention à la distinction: une instance (1), au plus une instance (0..1), une collection d'instances (* ou 1..*)

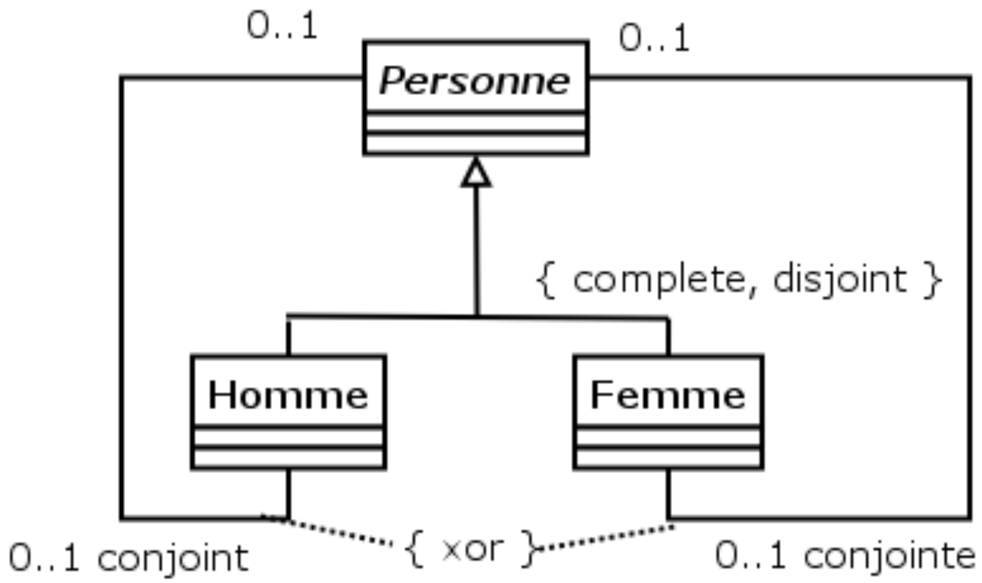
Multiplicities on attributs and classes can be:

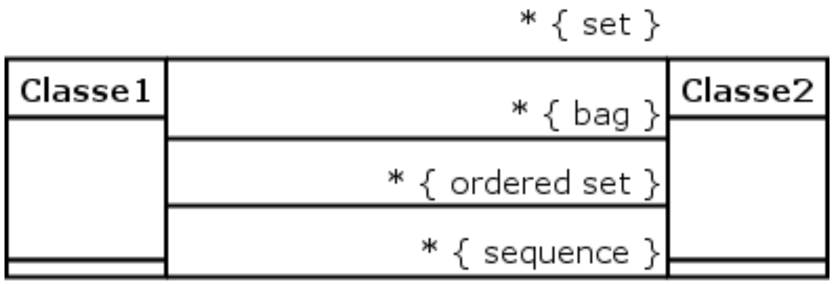


More Specific Details in UML 2

Contraints on associations

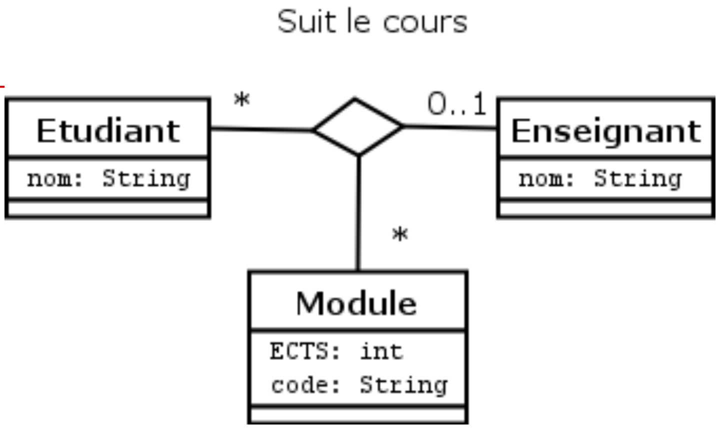
- For generalisation:
  - `complete, incomplete`
  - `disjoint, overlapping`
- Between associations
  - `xor`
- Collection Types may now also be specified !!!
  - `no duplicates, unordered`
  - `duplicates, unordered`
  - `no duplicates, ordered`
  - `duplicates, positioned`



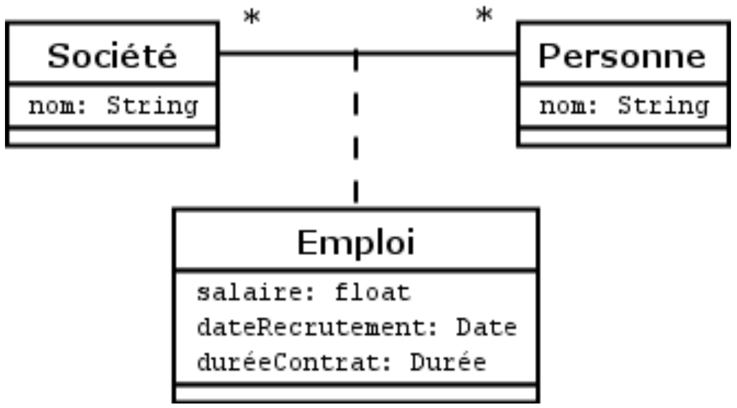


More Specific Details in UML 2

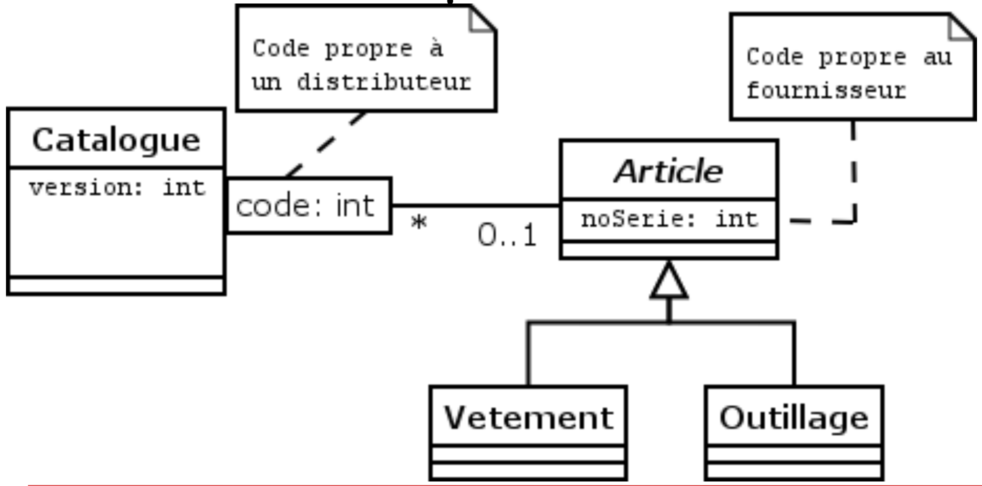
N-ary Associations



Association with attributes

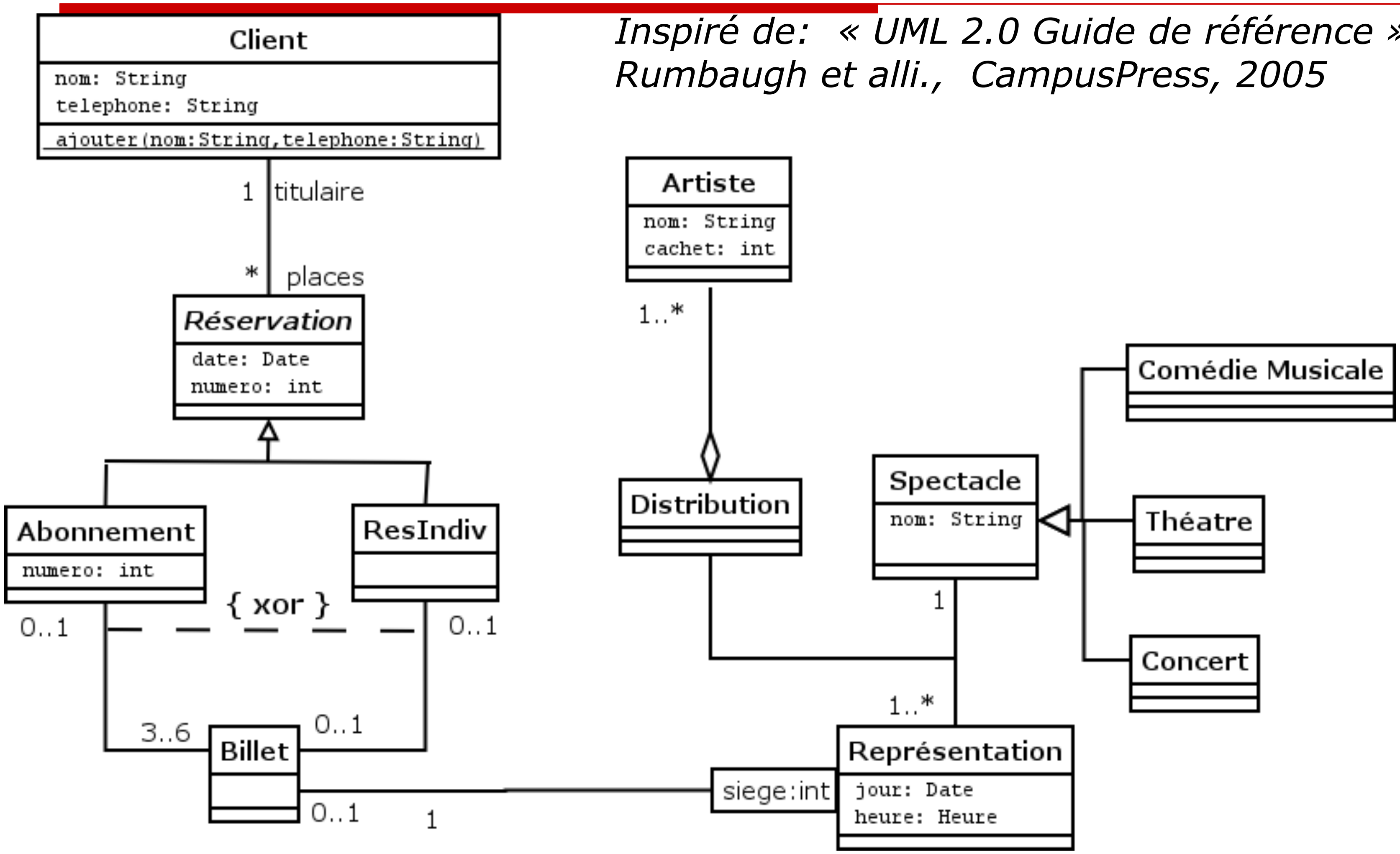


Association « qualified »



Putting all together …

Inspiré de: « UML 2.0 Guide de référence », Rumbaugh et alli., CampusPress, 2005



Principal UML diagram types (5)

- **Object Diagrams** („Diagrammes d'objects") :

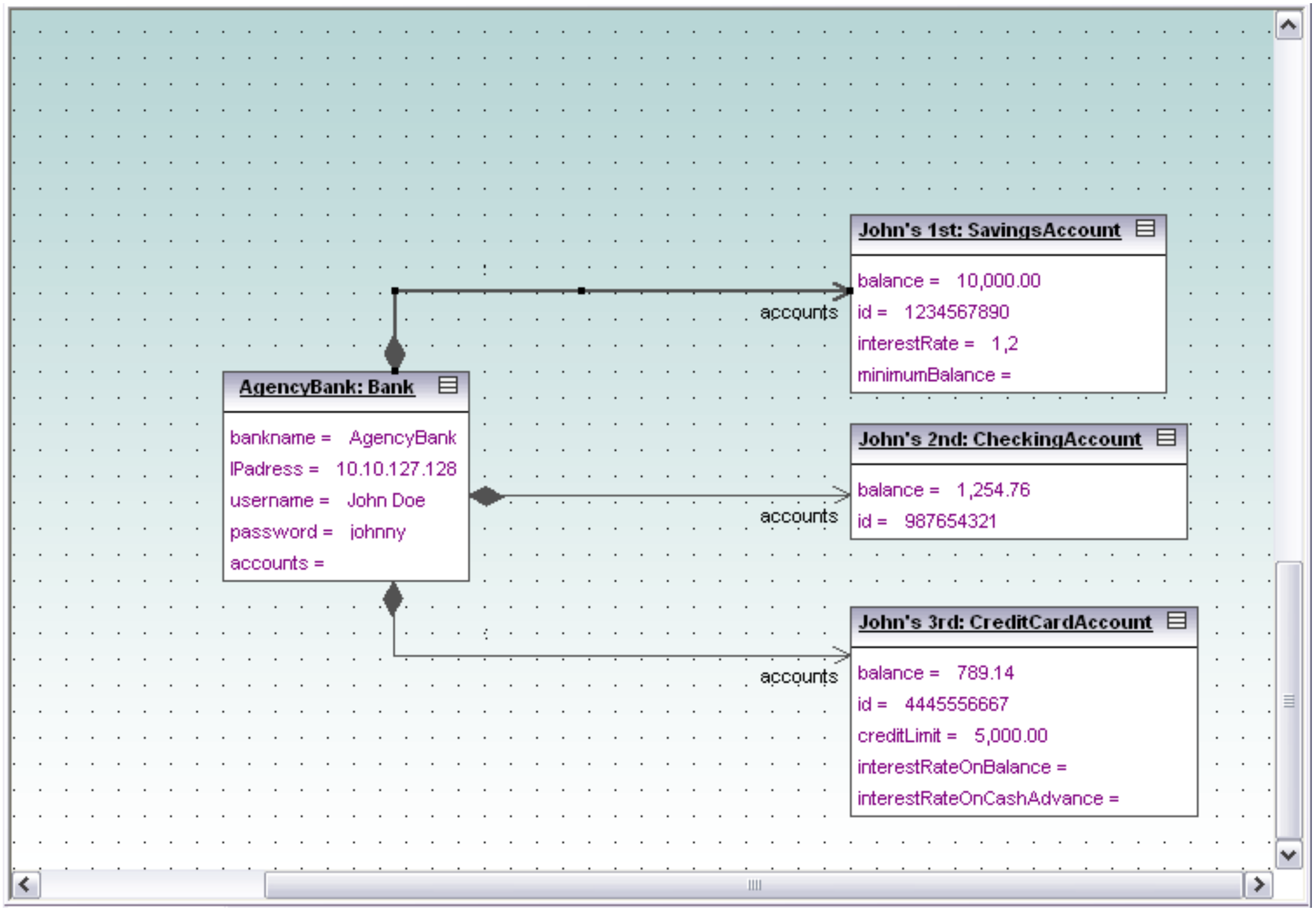
  denote a concrete system state,

- typically used in connection with a Class Diagram
  - attributes have concrete values
  - associations were replaced by directed arcs representing the links

  can be used for debugging purposes ...
  (semantics: fully clear).

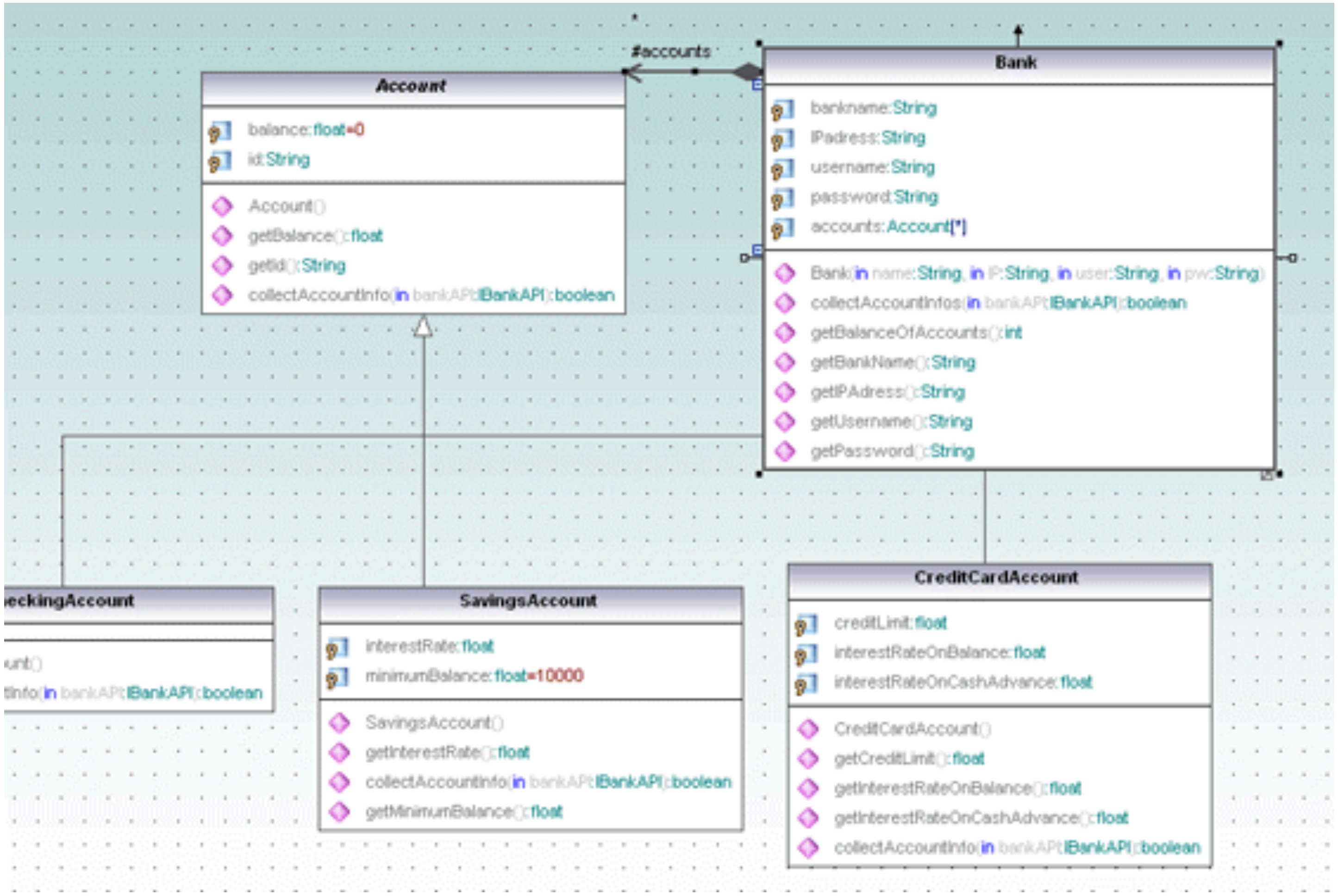
Example Object Diagram

- Corresp. Object Diagram



Example Object Diagram

- Class Diagram



*
#accounts
Account
balance:float=0
id:String
Account()
getBalance():float
getId():String
collectAccountInfo(in bankAPI:IBankAPI):boolean
Bank
bankname:String
IPadress:String
username:String
password:String
accounts:Account[*]
Bank(in name:String, in IP:String, in user:String, in pw:String)
collectAccountInfos(in bankAPI:IBankAPI):boolean
getBalanceOfAccounts():int
getBankName():String
getIPAdress():String
getUsername():String
getPassword():String
eckingAccount
unt()
tInfo(in bankAPI:IBankAPI):boolean
SavingsAccount
interestRate:float
minimumBalance:float=10000
SavingsAccount()
getInterestRate():float
collectAccountInfo(in bankAPI:IBankAPI):boolean
getMinimumBalance():float
CreditCardAccount
creditLimit:float
interestRateOnBalance:float
interestRateOnCashAdvance:float
CreditCardAccount()
getCreditLimit():float
getInterestRateOnBalance():float
getInterestRateOnCashAdvance():float
collectAccountInfo(in bankAPI:IBankAPI):boolean

Summary: Object Diagrams

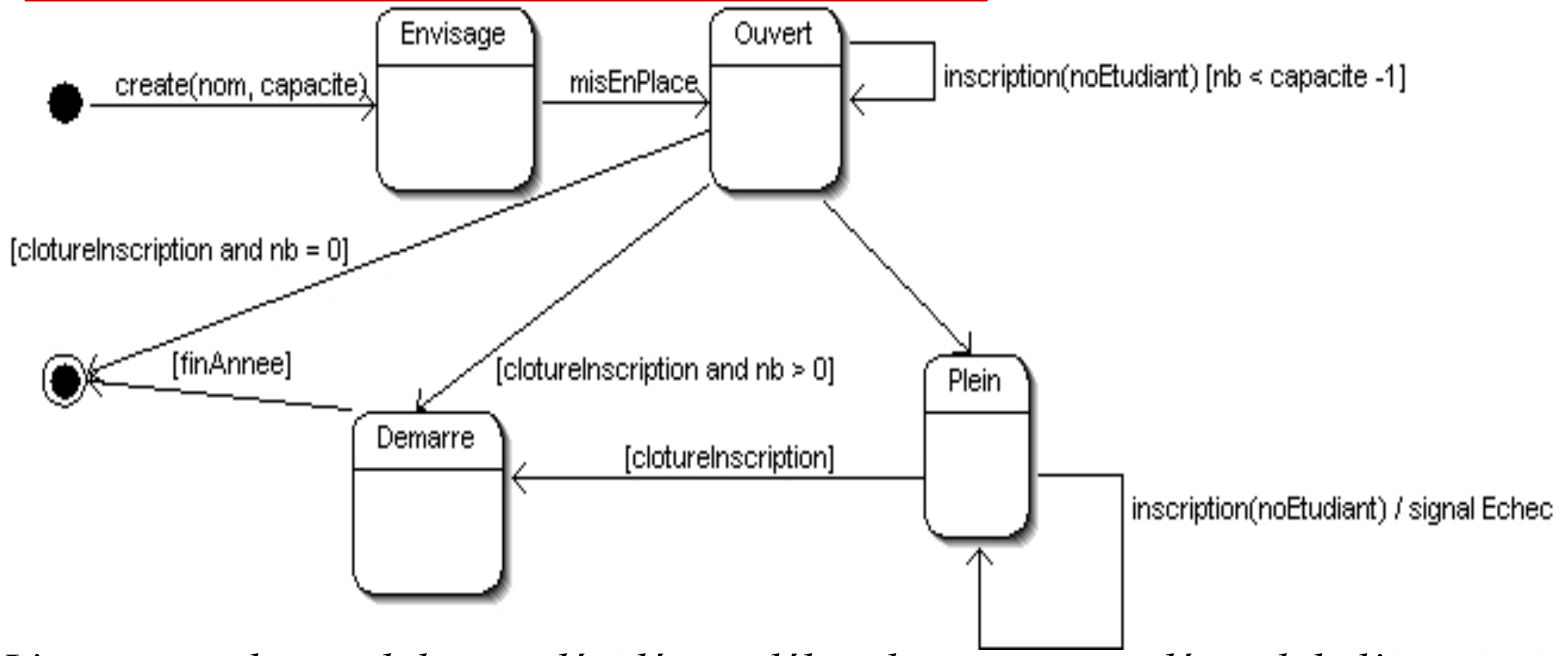
- Object Models denote a concrete State of a Class Model; Class Diagram denote (essentially) a Signature of the elements in the state, as well as the possible operations on them.

  Multiplicities and Cardinalities express INVARIANTS on (valid) Object Models to a given Class Model – with this respect, serves as Specification of States.

A propos Class Diagrams (3)

- Not all constraints on an object graph can be expressed by arrows so far:
  - The student numbers should ne distinct
  - A student can not acquire a module he has already finished
  - A module may not be part of the pre-requisites “pré-requis“
  - A student may only follow a module if he has acquired the necessary pre-requisites
  - A student can only follow modules offered at his „filière“
  - …

Example of a State Machine: a (teaching) module


Envisage
Ouvert
Plein
Demarre
create(nom, capacite)
misEnPlace
inscription(noEtudiant) [nb < capacite -1]
[clotureInscription and nb = 0]
[finAnnee]
[clotureInscription and nb > 0]
[clotureInscription]
inscription(noEtudiant) / signal Echec

« L'ouverture des modules est décidée en début de semestre et dépend de l'inscription effective d'étudiants. La capacité d'accueil est fixe et les inscriptions prises dans l'ordre d'arrivée. Aucune inscription n'est admise une fois le module démarré. »

This describes the life-cycle of an isolated module ... will we find this later on in the implementation the equivalent of the possible transitions ?

A propos Class Diagrams (3)

- Not all constraints on an object graph can be expressed by arrows so far:
  - ...
  - a student can only subscribe a module if he is targeting for a diplome
  - Il existe un facteur 3 au plus entre les nombres de crédits de deux U.E. d'une même mention (cas des Licences) ???

  ☞ we will need mechanisms to describe all this in the design phase !!! (Object Constraint Language, OCL.
  Instead: We use MOAL ...)