Introduction to Software Engineering

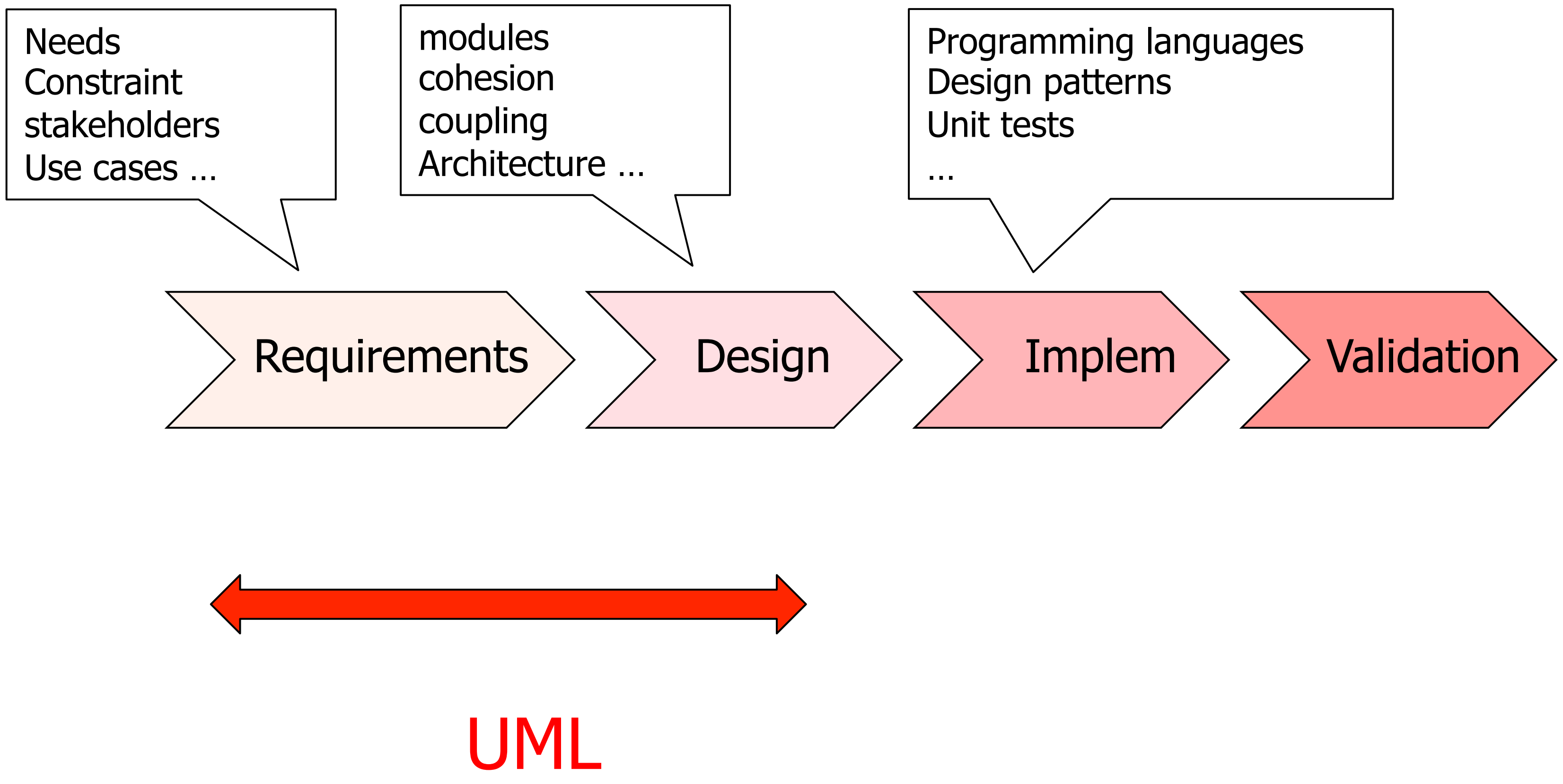
Introduction to UML

Philippe Lalanda

Philippe.lalanda@imag.fr

http://membres-liglab.imag.fr/lalanda/

Development activities - reminder



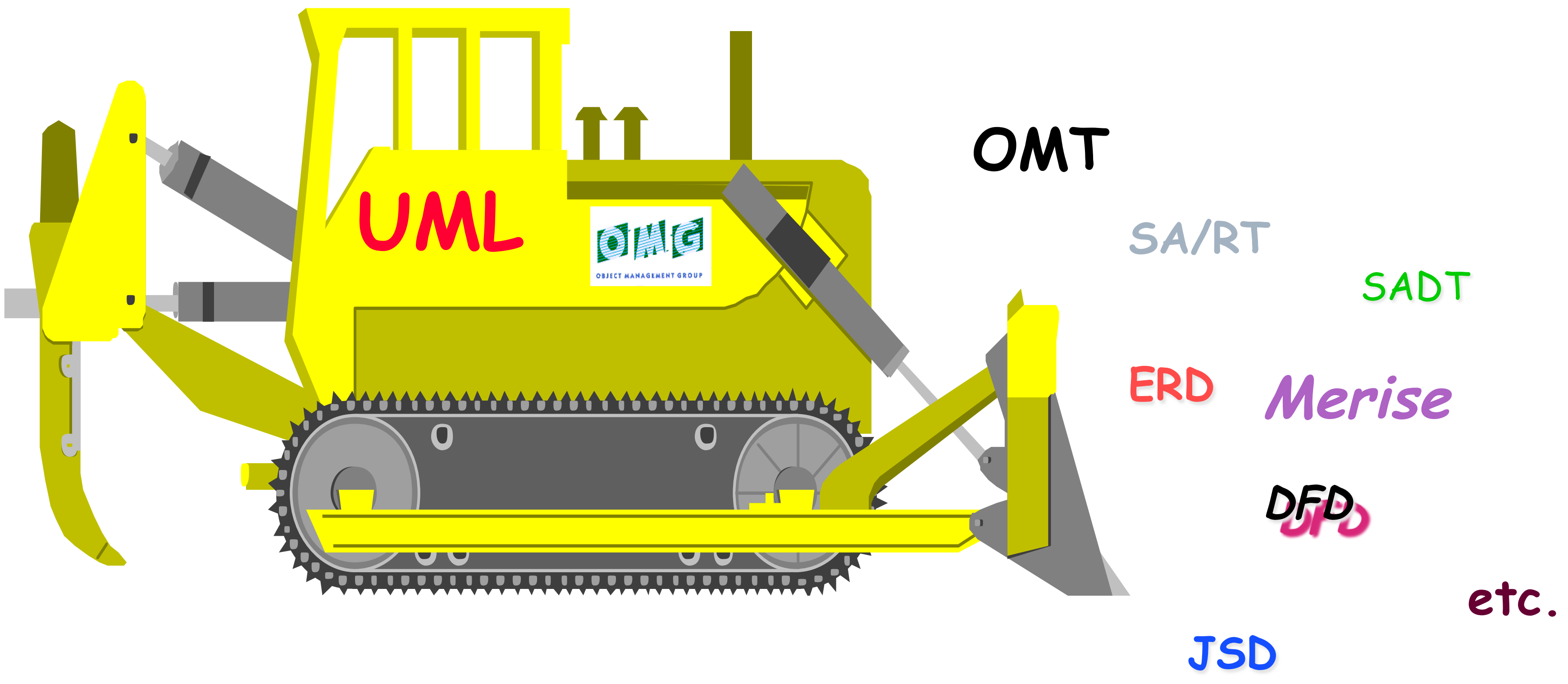
Outline

- UML presentation
- Basic concepts
- Advanced concepts
- Conclusion

Unified Modeling Language– from Favre/Parissis

- UML is a notation for OO analysis and design
- It is complemented by methods
  - The Rational Unified Process
  - The Unified Software Development Process
- … and tools
  - Rational Rose, Objecteering, Together J, ArgoUML, Poseidon, …

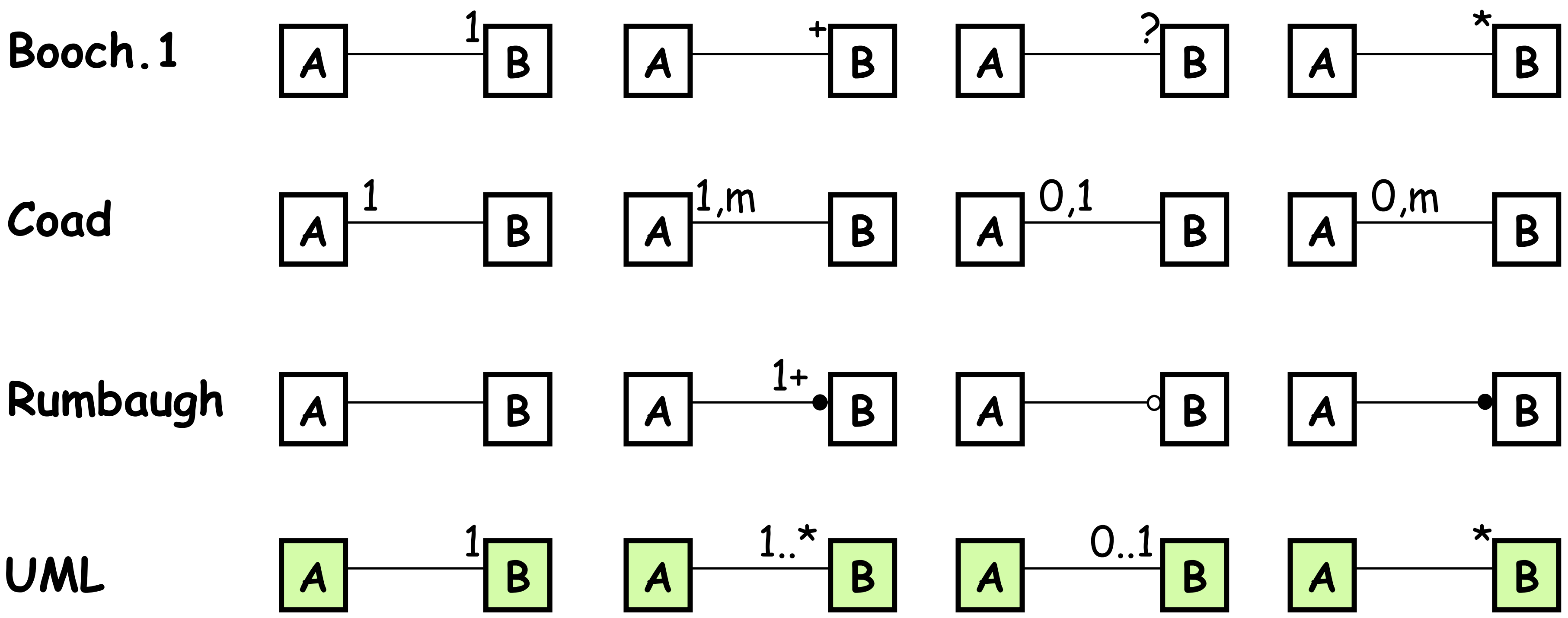
Unified Modeling Language – from Bezivin



Vocabulary unification – from Bezivin

Booch	Class	Uses	Inherits	Contains
Coad	Class/Object	Instance connection	Gen/Spec	Whole/Part
Jacobson	Object	Association Acquaintance	Inherits	ConsistsIn
Odell	Object/Type	Relation	Sub	Composition
Rumbaugh	Class	Association	Generalization	Aggregation
Shlaer/Mellor	Object	Relation	Sub	N/D
UML	Class	Association	Generalization	Aggregation

Notation unification – from Bezivin



A major stake – from Favre/Parissis

- Standardization was needed to stabilize and disseminate OO practices in the industry
- A difficult task
  - Many industrial lobbies and pressure groups
  - Very important stakes
  - Various interests and motivations
    - Tool vendors, consultants, industrial users, ...
- UML targets consensuality, not innovation

Consensus – from Favre/Parissis

- A minima
  - Intersection
  - Leads to simplicity (or impoverishment)
  - Good for users, not providers
  - Requires maturity
- A maxima
  - Union
  - Leads to complexity and instability
  - Hard for users
  - Allows providers differentiation
  - Easy solution when lack of maturity
- UML relies on *a maxima* consensus

UML impact – from Favre/Parissis

- *De facto* standard in the industry
- Adopted by tool vendors
- Integration in industrial development processes
- Used in production environment
  - Although coherence maintenance is rarely ensured
- Lots of jobs, required skill
- UML has won the OO modeling battle

UML tools – from Favre/Parissis

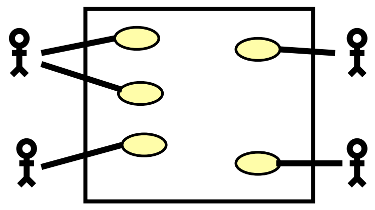
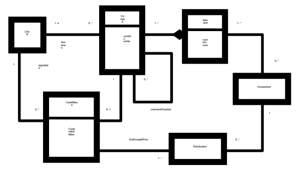
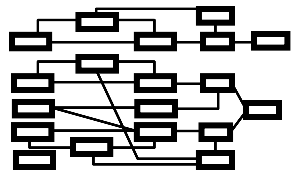
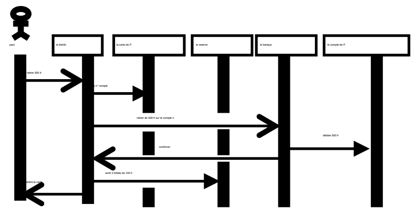
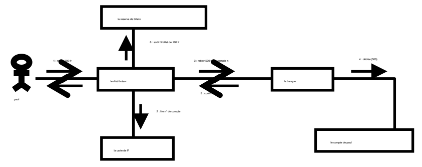
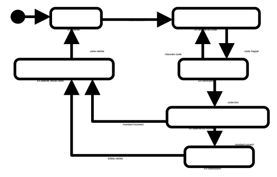
- Hundreds of UML tools
  - Modeling tools
  - Generation of code, documents, tests, …
  - Model transformation
- An important effort has been made to turn to UML
  - Hard (and costly) to go back

UML notation – from Favre/Parissis

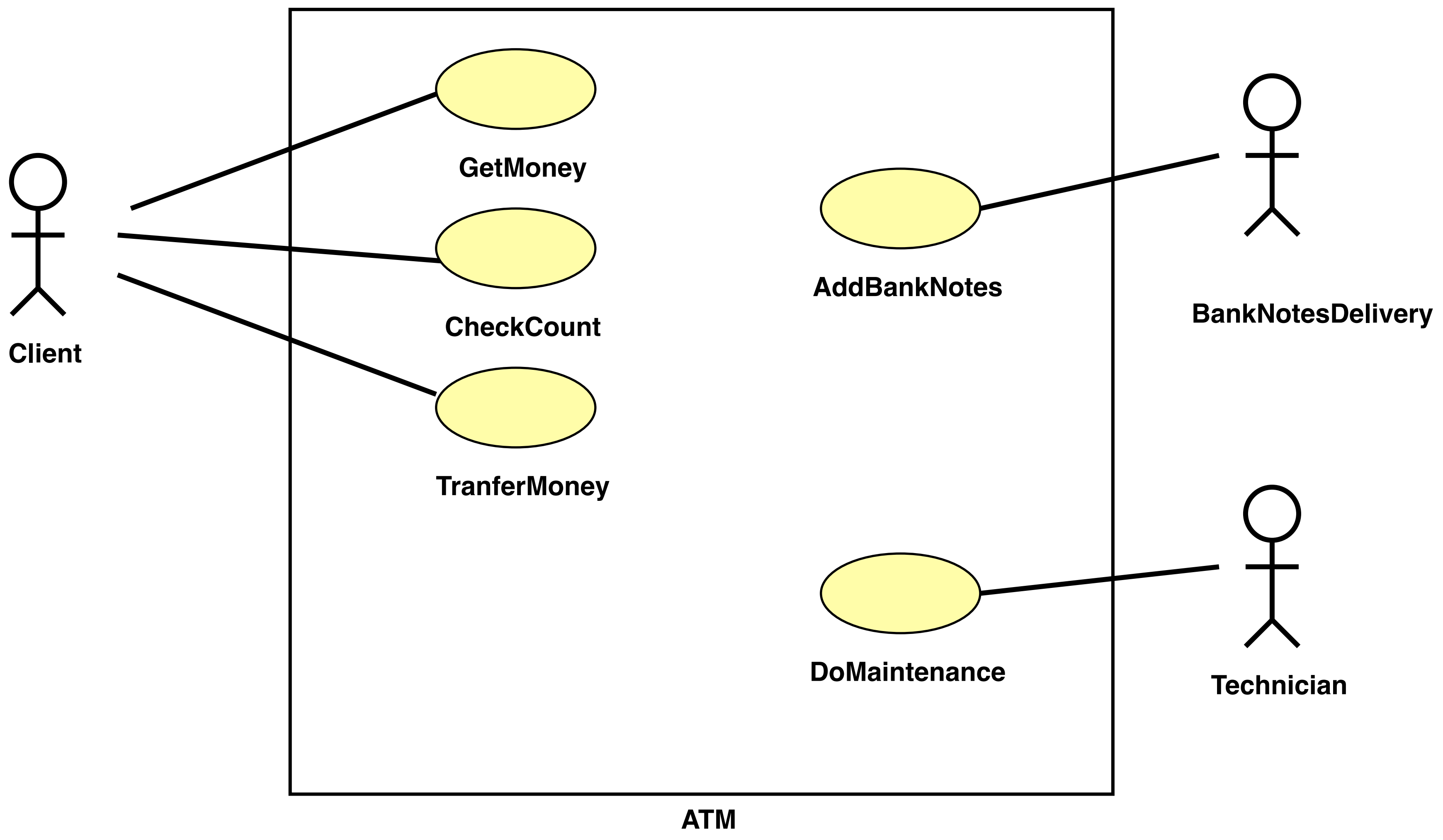
- Many notations, actually
  - Graphical and textual
- Notations are
  - precise (in a context)
  - Standard (not always respected)
  - General (not always appropriate)
  - Extension (through low level mechanisms)

UML notation – from Favre/Parissis

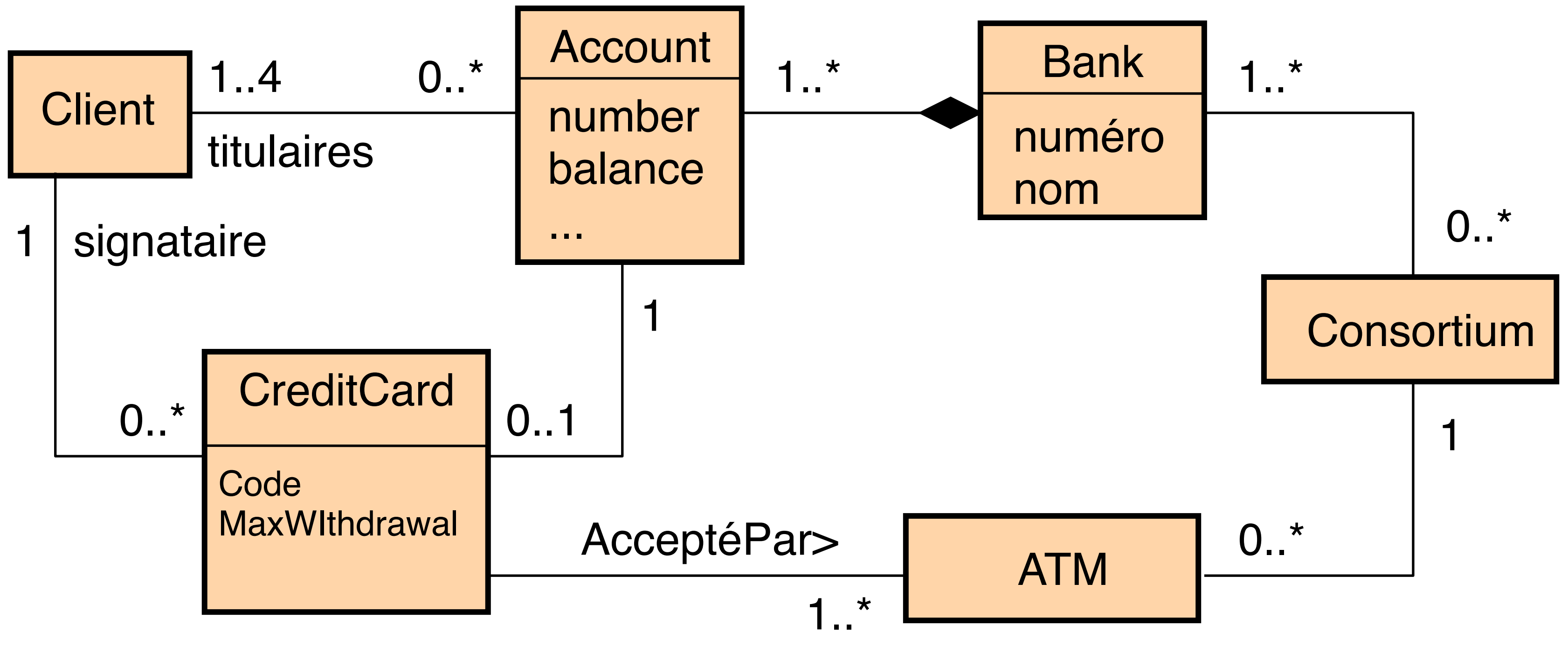
- Use case diagram
- Class diagram
- Object diagram
- Sequence diagram
- Collaboration diagram
- State diagram
- Activity diagram
- Component diagram
- Deployment diagram
- Constraint language
- Action language, ...

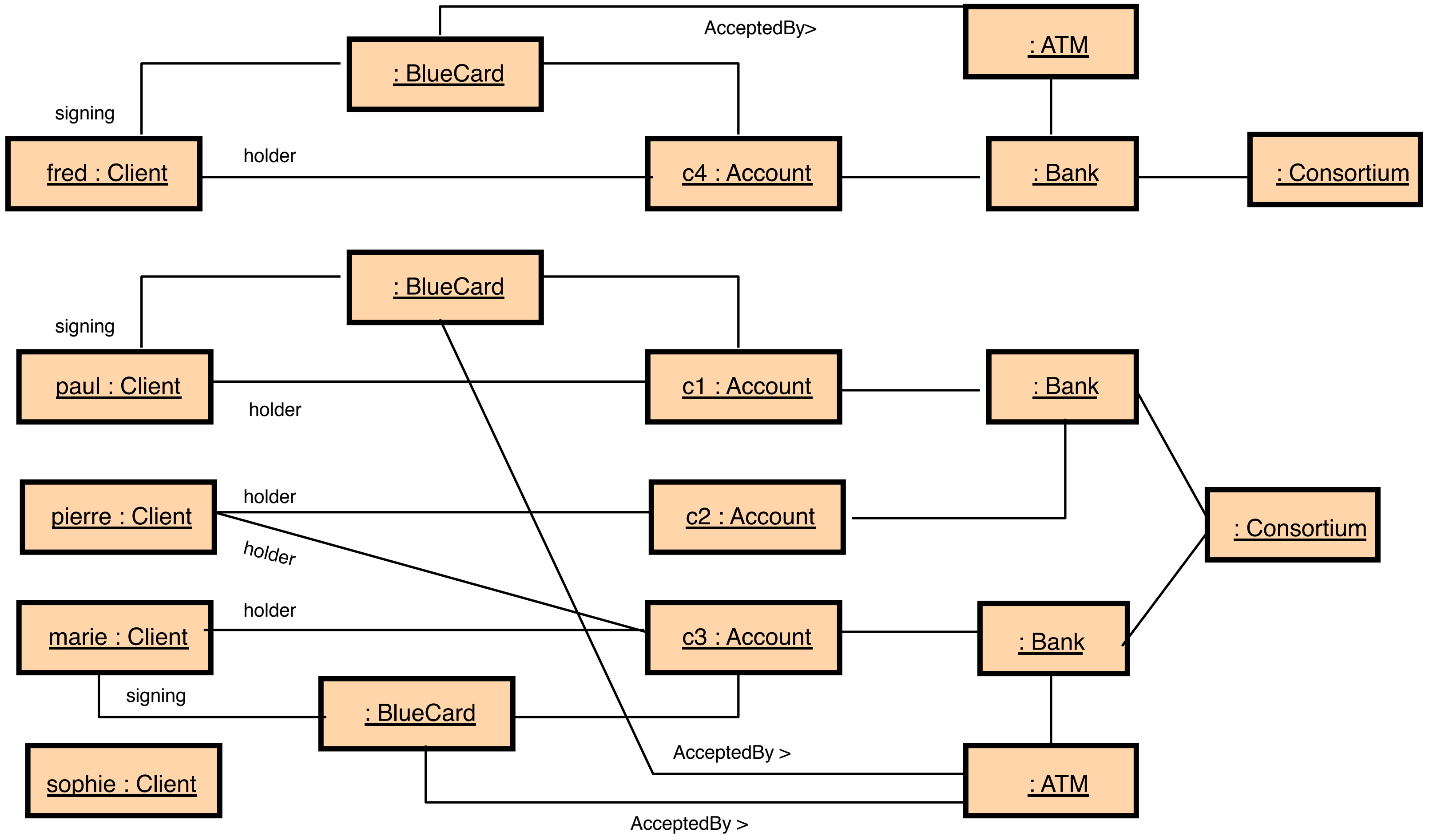
Use cases – from Favre/Parissis



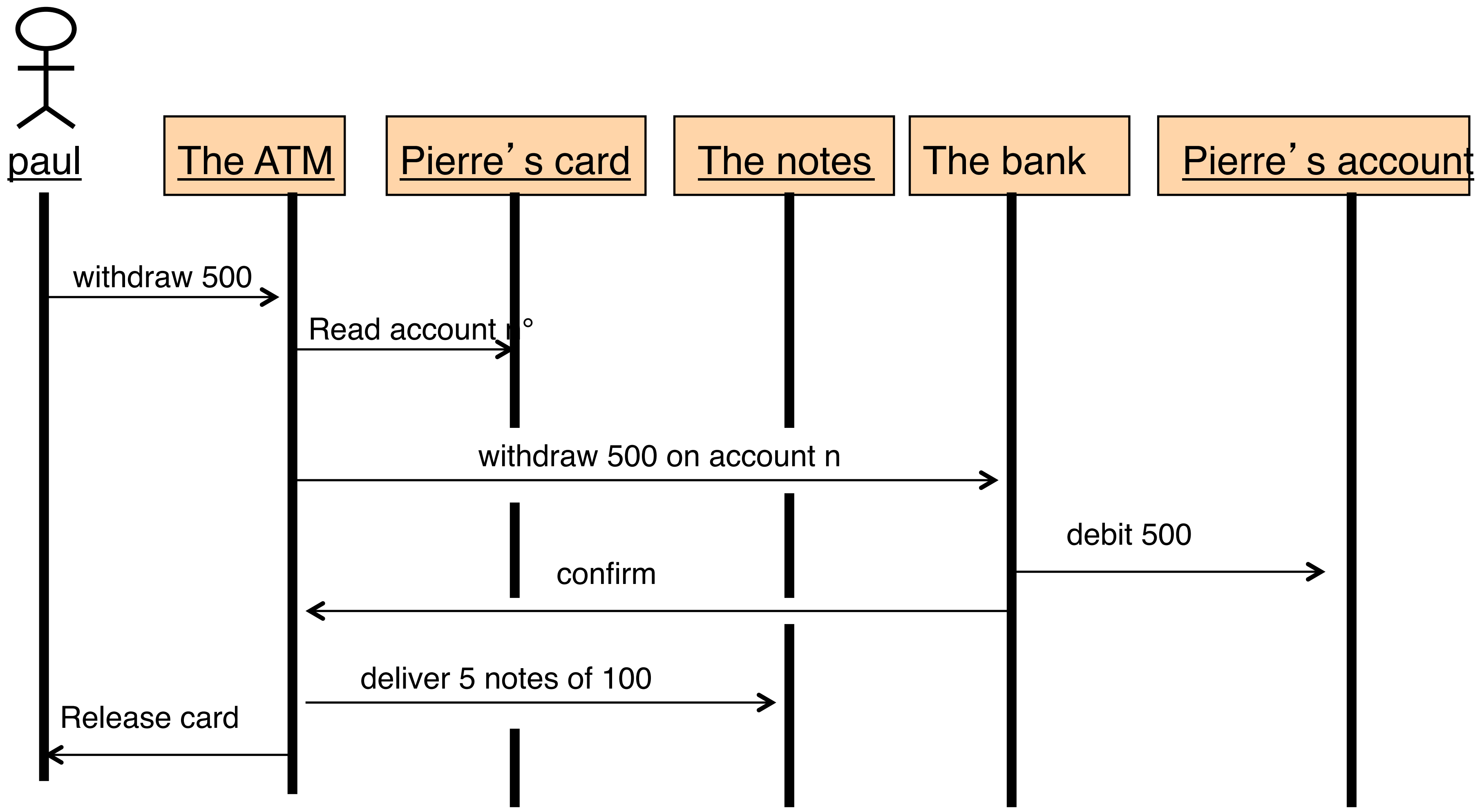
Class diagram – from Favre/Parissis



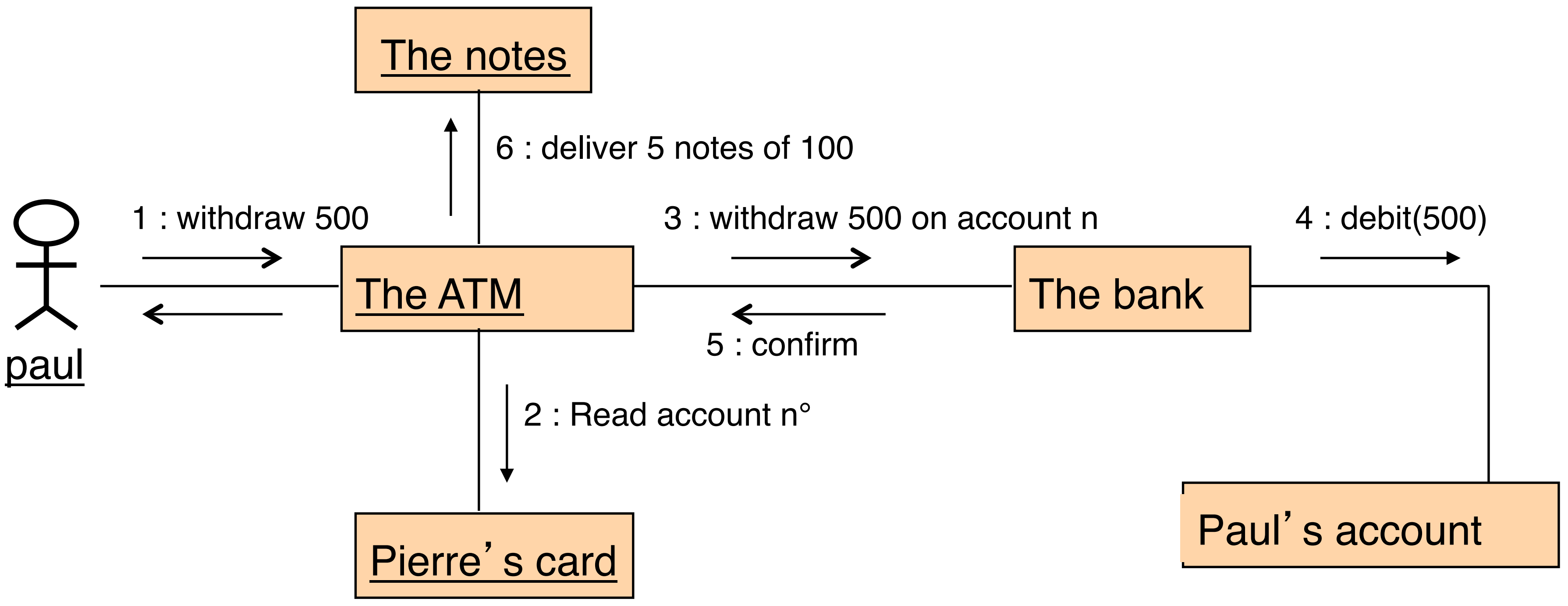
Object diagram – from Favre/Parissis



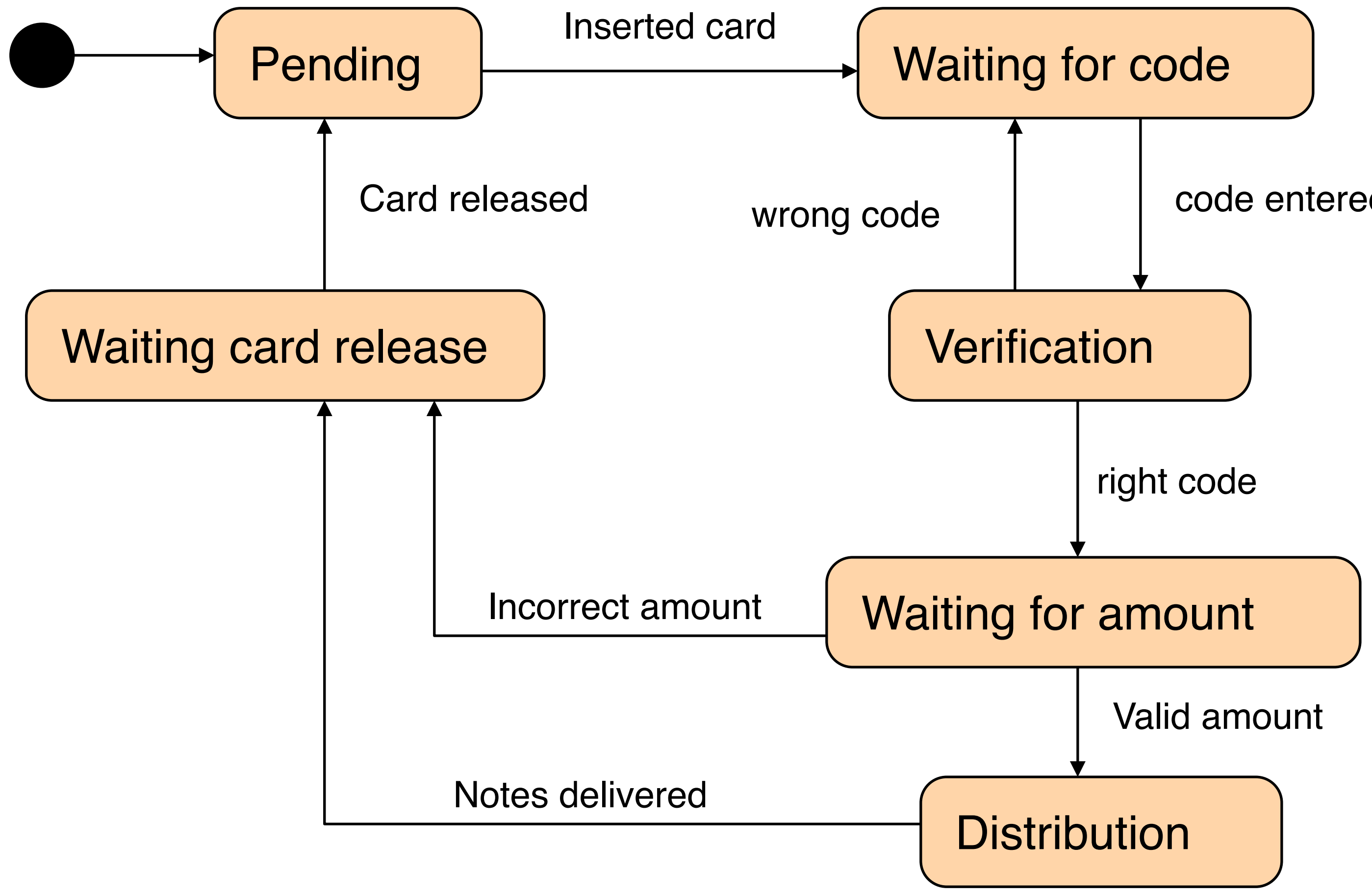
Sequence diagram – from Favre/Parissis



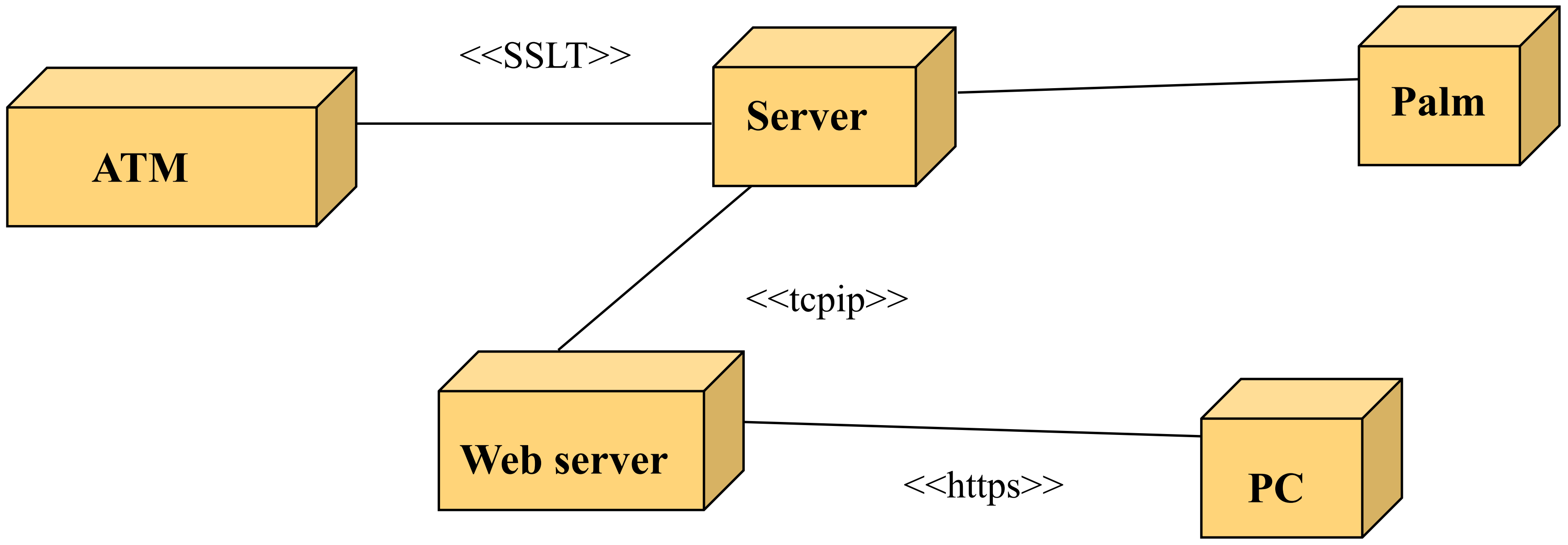
Collaboration diagram – from Favre/Parissis



State diagram – from Favre/Parissis



Deployment diagram – from Favre/Parissis



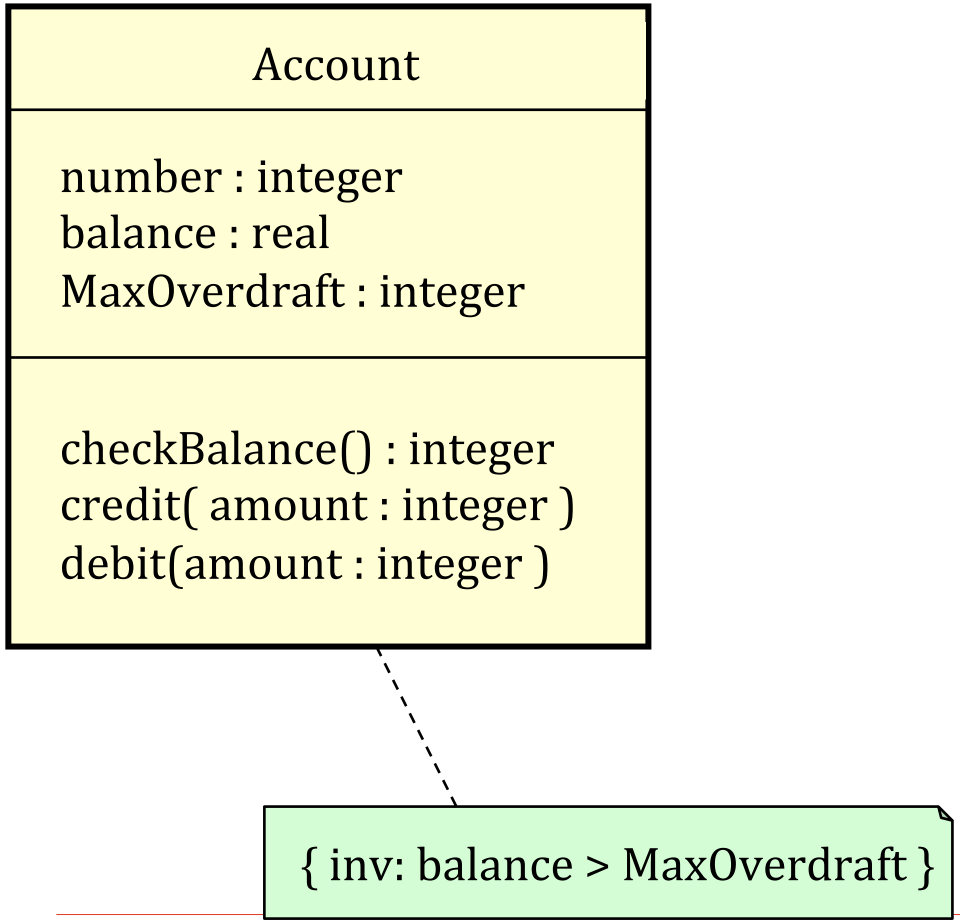
Outline

- UML presentation
- Basic concepts
- Advanced concepts
- Conclusion

UML impact – from Favre/Parissis

- UML is based on OO principles
  - Object and class
  - Links and association
  - Inheritance
  - Constraint
- UML defines notations to build diagrams manipulating these concepts
  - Class diagrams (model level)
  - Object diagrams (instance level)

Class notation – from Favre/Parissis



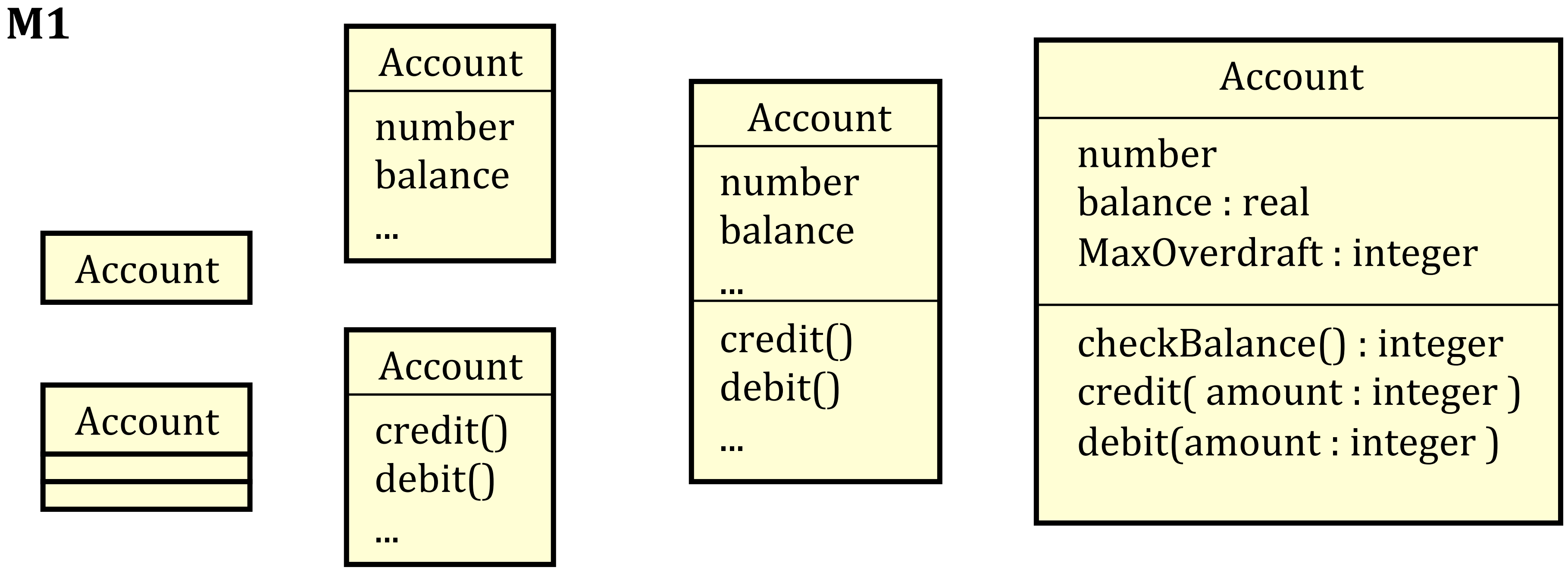
Class name

Attributes
- name
- type

Operations
- name
- parameter
- Result type

Constraint

Simplified notations – from Favre/Parissis

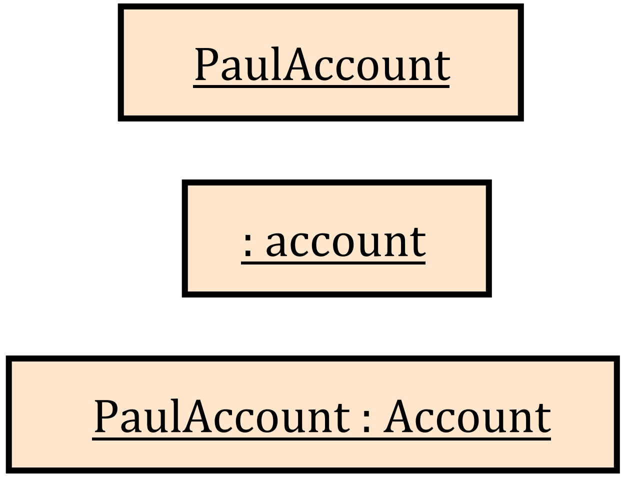


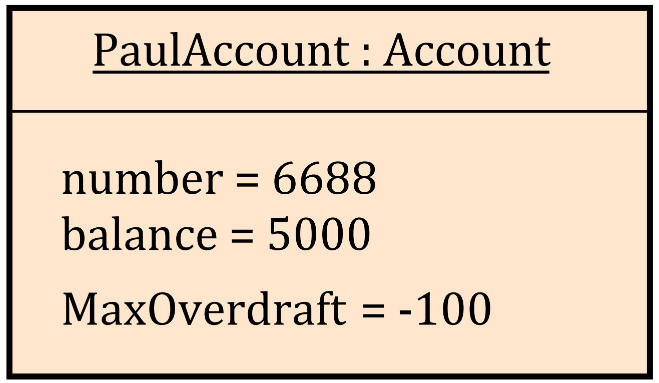
Style note:

- class names begin with a upper case
- attributes and method names begin with a lower case

Object notation – from Favre/Parissis

M0



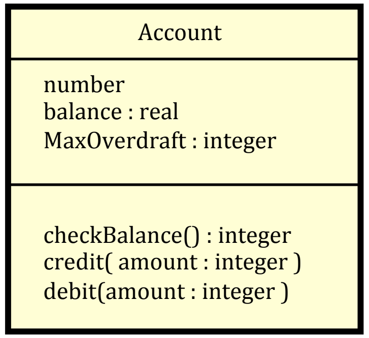


Convention :

- object names begin with a lower case and are underlined

Class vs. Object – from Favre/Parissis

A class specifies the structure and the behavior of a set of objects (of the same nature)



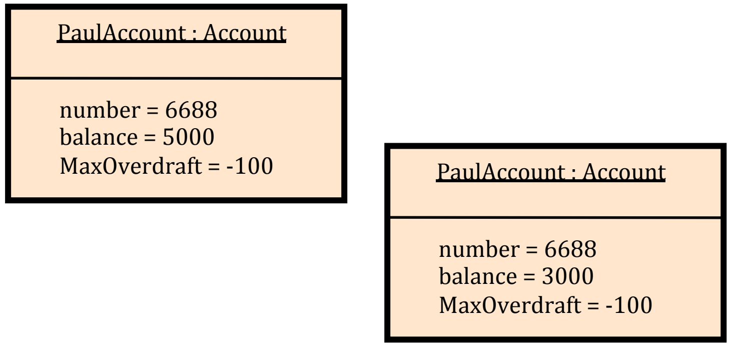
Class diagram

- A class structure is constant over time

M1

M0

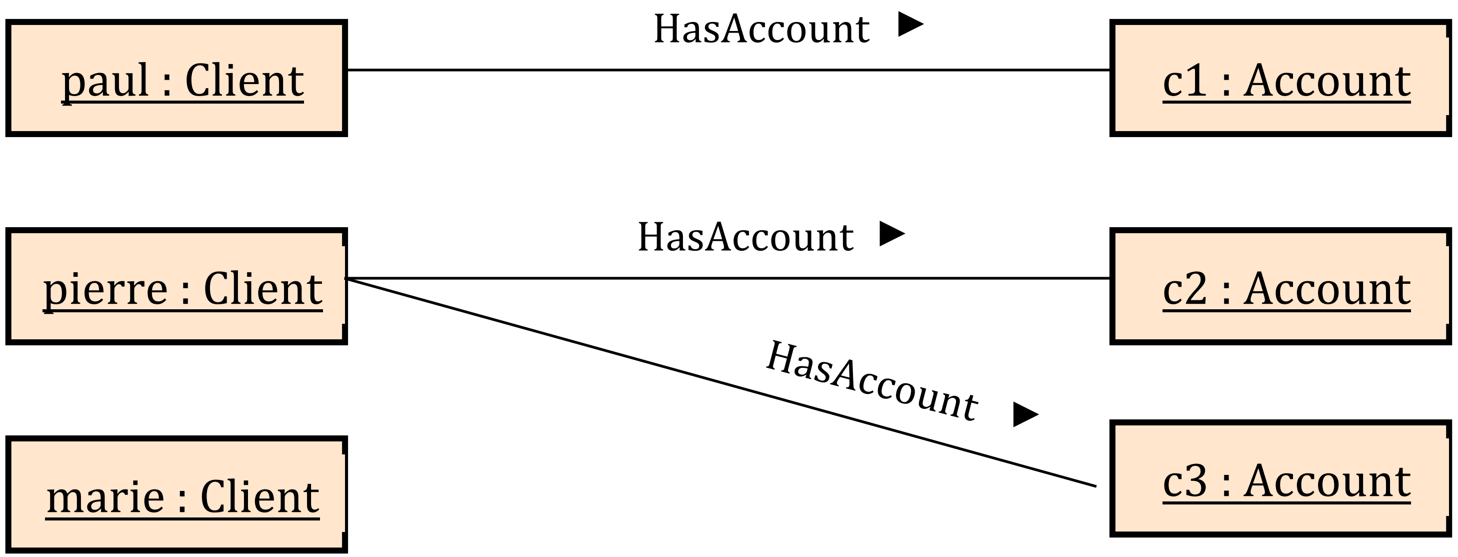
- Objects can be created and deleted at run time
- Attributes values can be changed



Object diagram

Links – from Favre/Parissis

A link specifies a connection between two objects

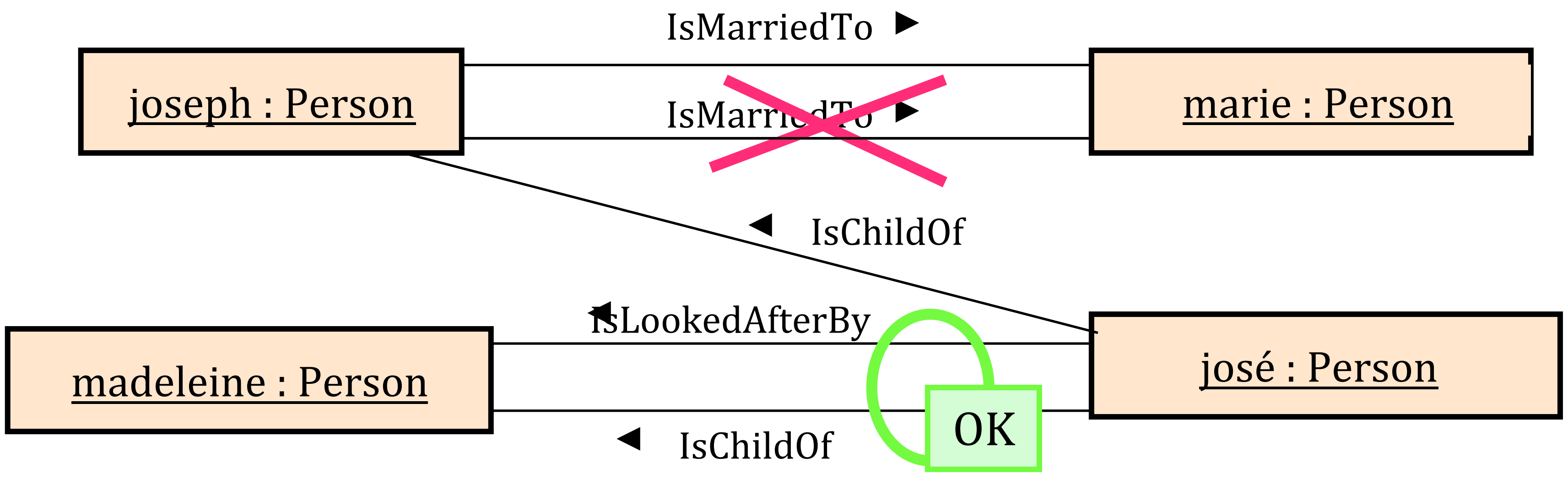


Style note:

- links names are verbal forms and begin with an uppercase
- ▶ arrow indicates how to read

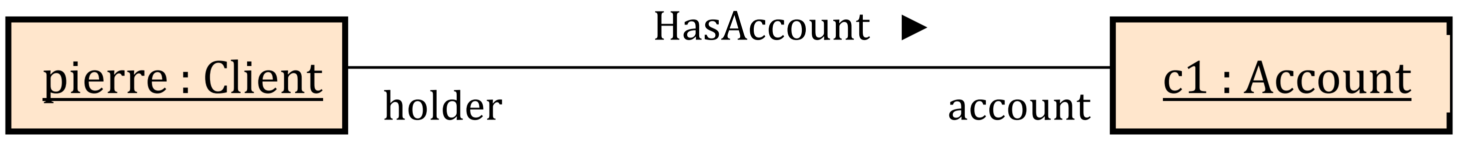
Constraint about links – from Favre/Parissis

- No more than one link of a given type between two objects



Role – from Favre/Parissis

- Linked objects play a different role



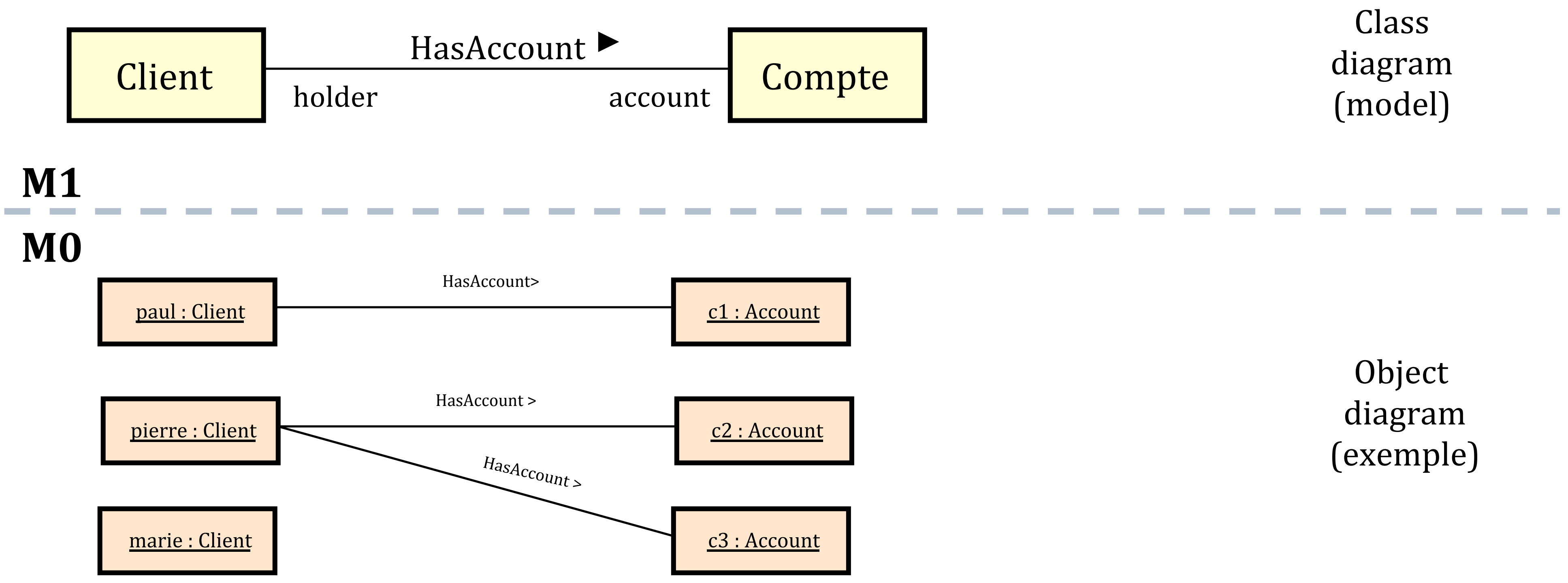
- pierre owns account c1
- c1 *plays the account role for* pierre
- pierre *plays the holder role for* c1

Style note:

- a role is expressed as a name
- by default, the role is the name of the class

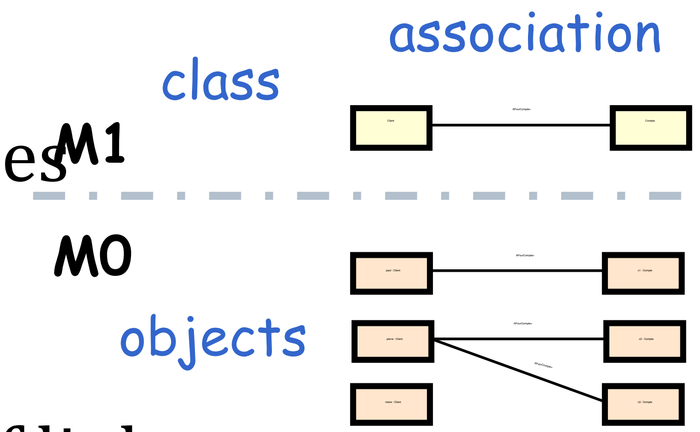
Associations – from Favre/Parissis

An **association** describes a set of links having a same « semantic »



Association vs. links – from Favre/Parissis

- A link relates two objects
- An association relates two classes

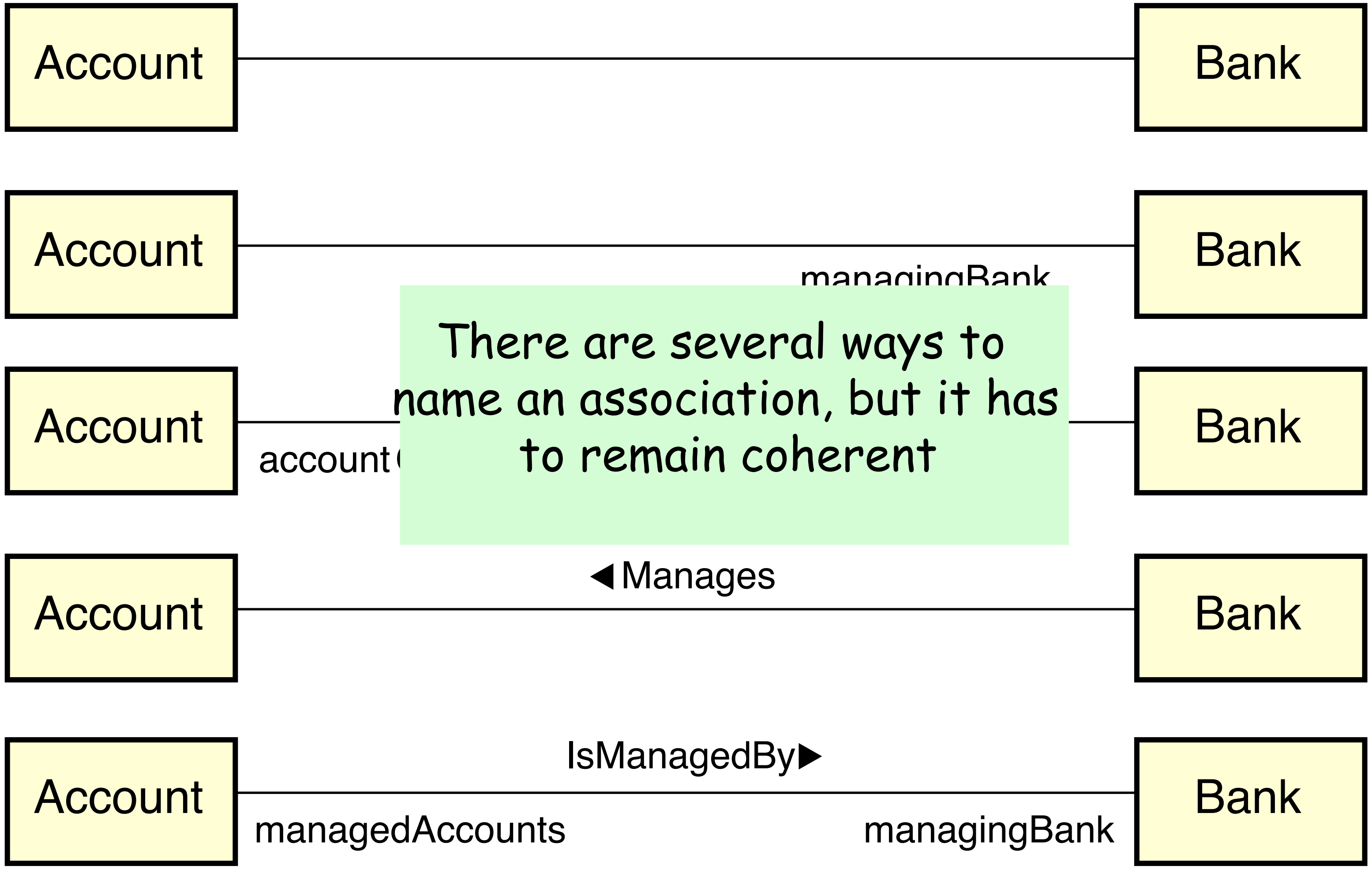


- A link is an association instance
- An association describes a set of links

- Links can be created and deleted at runtime, not associations

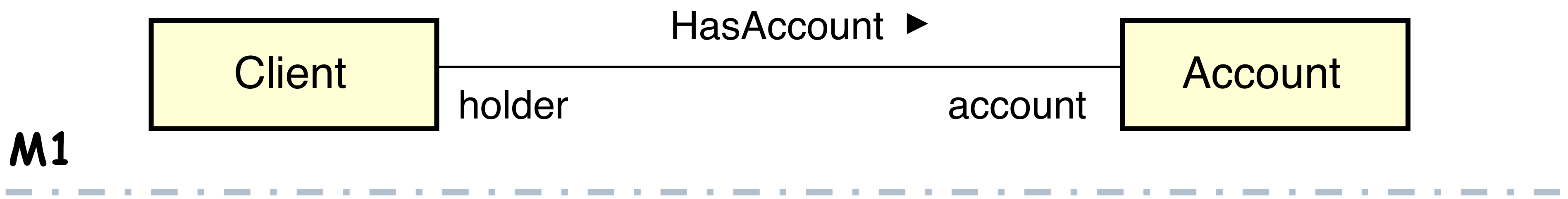
- Note: the term "relation" is not part of UML

Associations naming – from Favre/Parissis



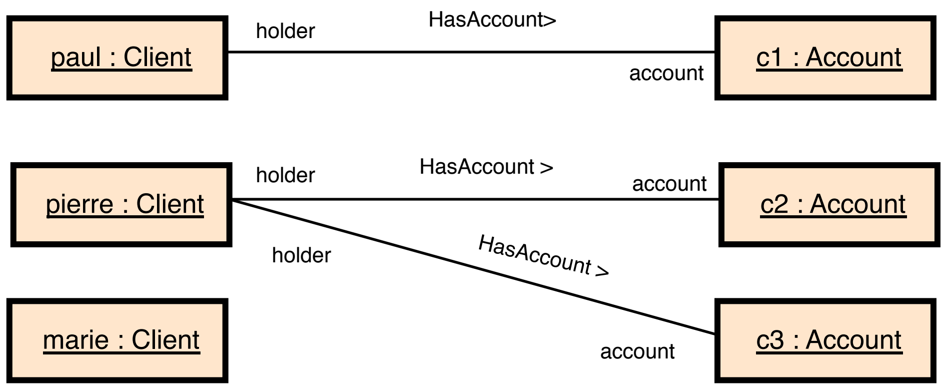
There are several ways to name an association, but it has to remain coherent

Roles and navigation – from Favre/Parissis



M0

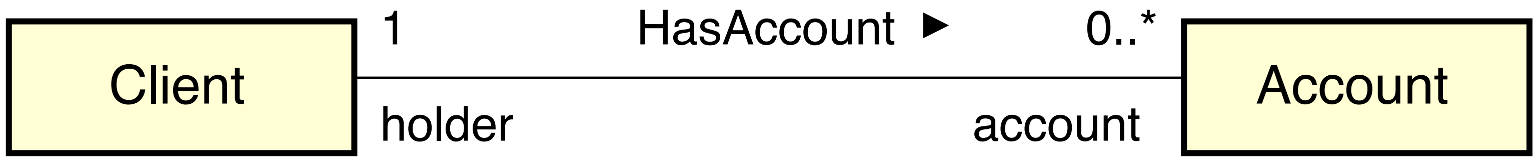
paul.account	= {c1}
pierre.account	= {c2,c3}
marie.account	= { }
c1.holder	= paul
c2.holder	= pierre
c3.holder	= pierre



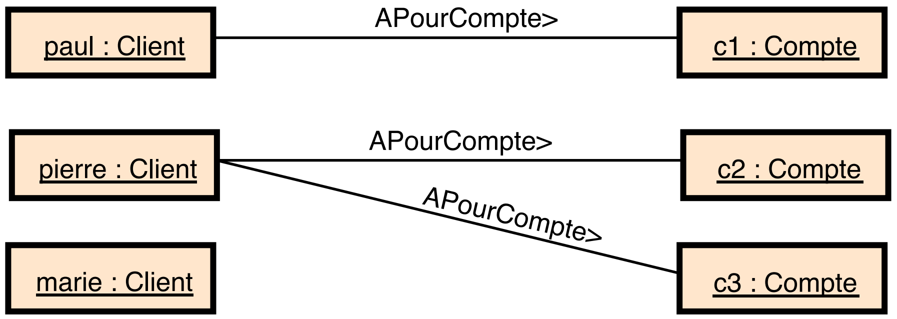
Name roles in priority : careful name selection ! (code generation)

Cardinality – from Favre/Parissis

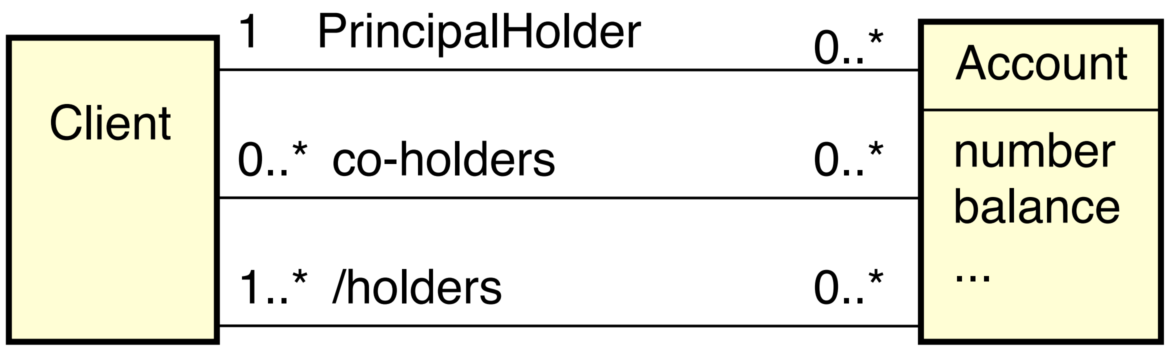
- Specify how many objects can be linked to a source object
  - Max and min cardinalities (C_{min}, C_{max})
  - Use of constants



« A client has 0 or several accounts »
« An account has always one and only one client »



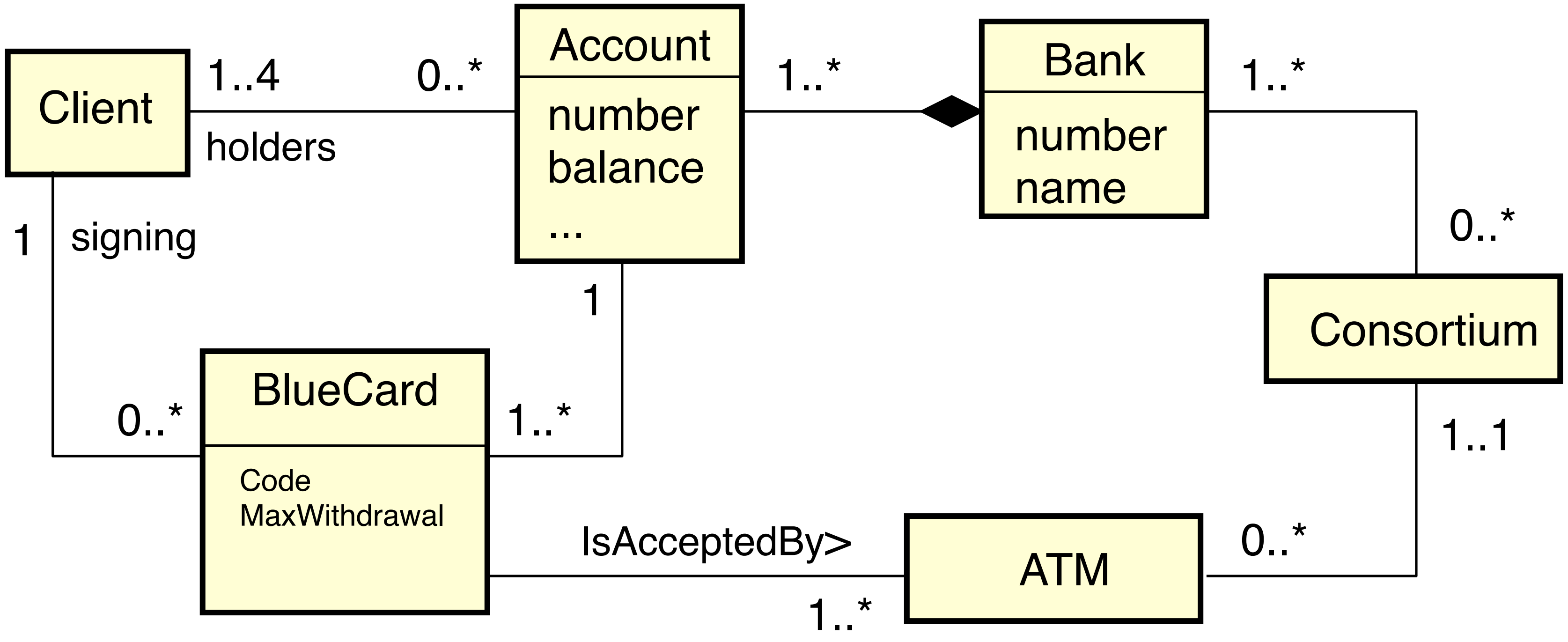
Constraints between associations –Favre/Parissis



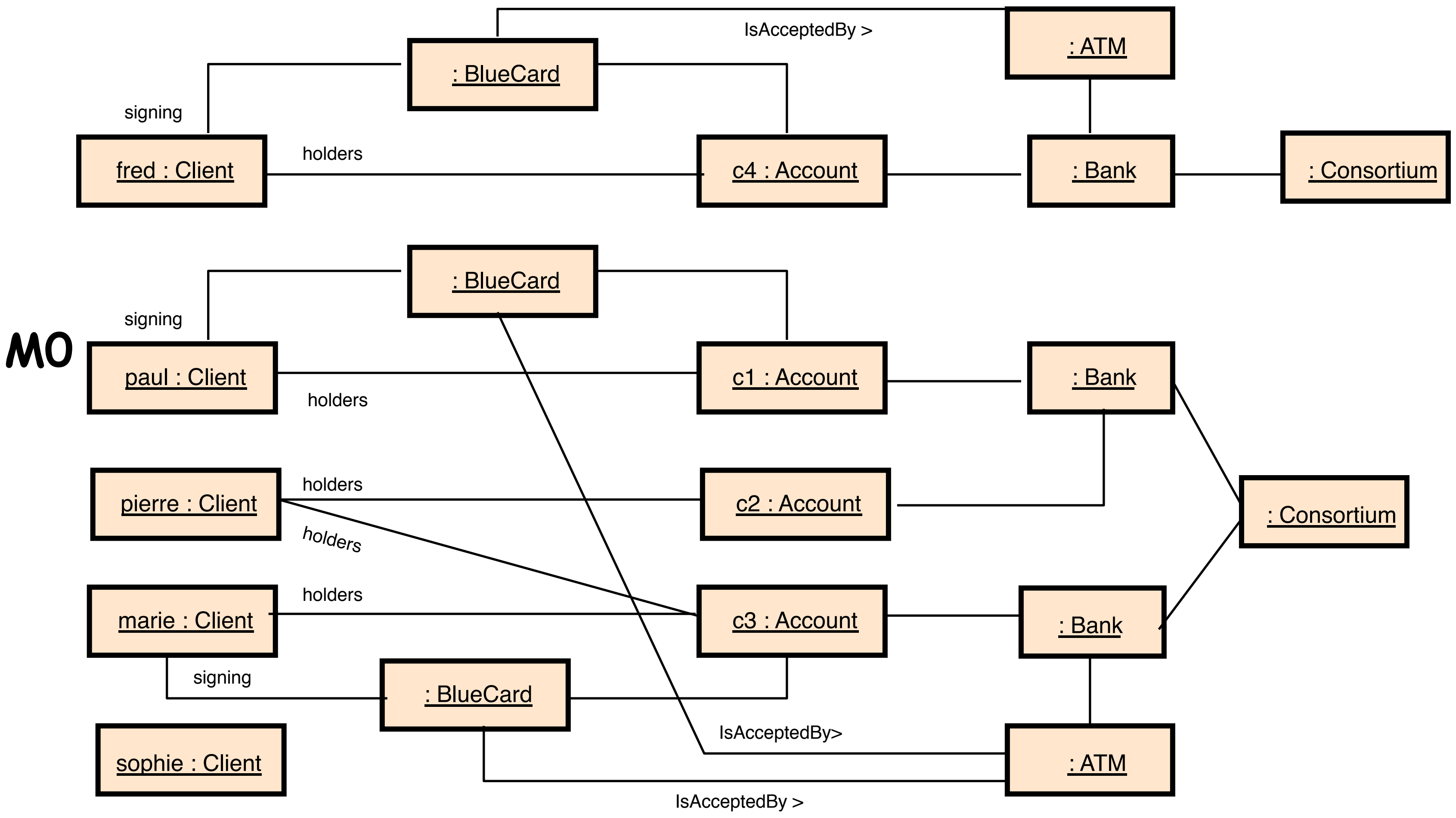
Cardinality are not enough to express all the constraints
→ additional constraints are described in Natural Language (or OCL)

(1) A client cannot be both principal holder and co-holder of a same account
(2) Holders of an account include the principal holder and, possibly, co-holders

Example of Class diagram – from Favre/Parissis

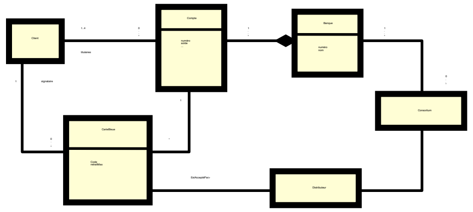


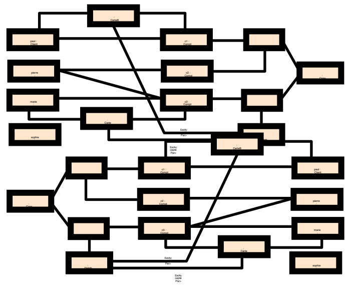
Example of object diagram – from Favre/Parissis



Class diagram / object diagram –Favre/Parissis

- A class diagram
  - Defines all the possible states
  - Constraints must be always met

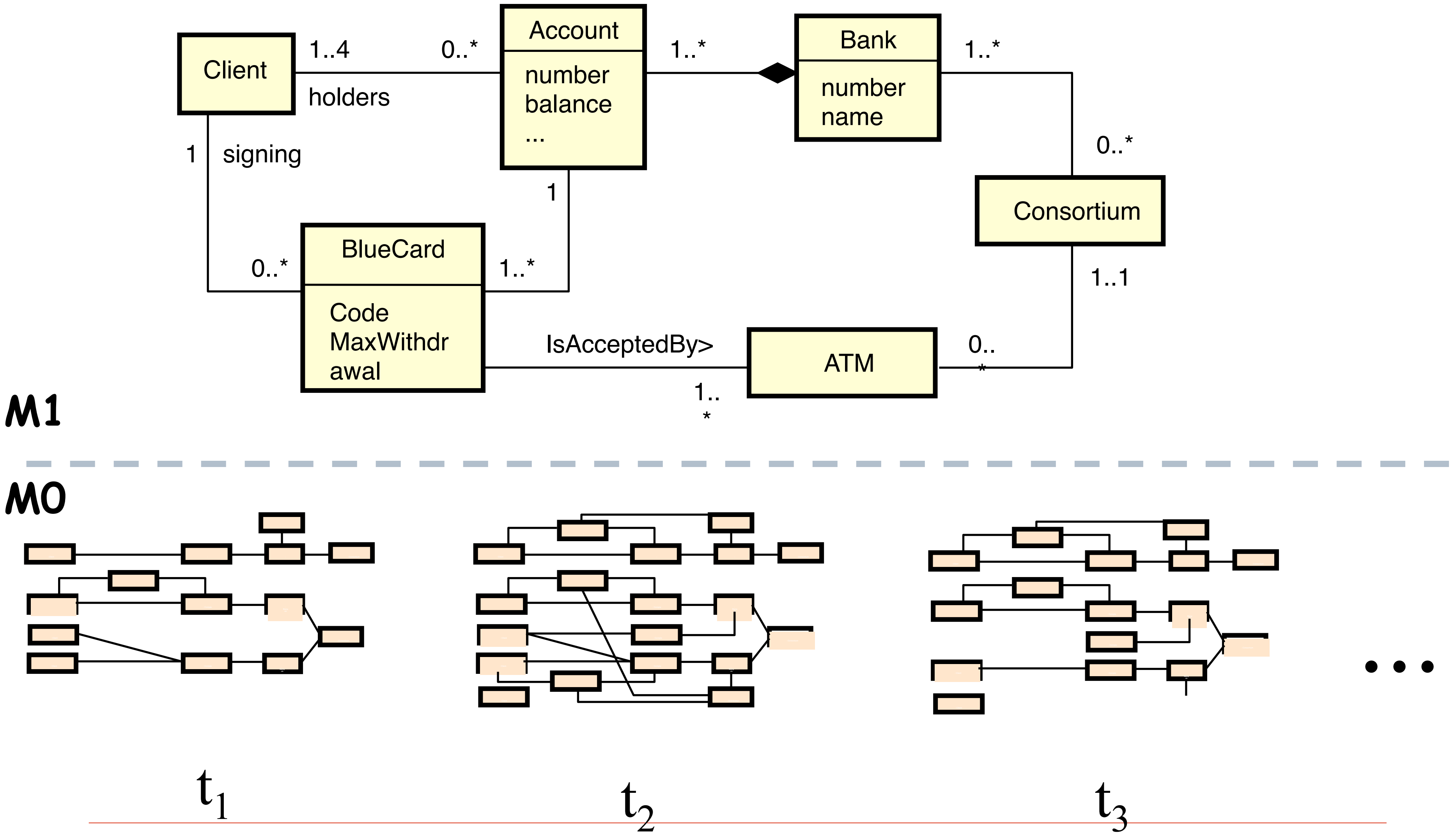




- An object diagram
  - Defines a possible state at a given time
  - Must be conformed to the class diagram

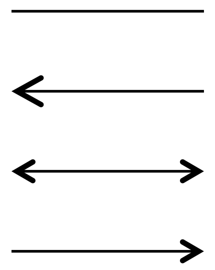
- Object diagrams can be used to
  - Exemplify a class diagram (explanation)
  - Validate a class diagram (“test” it)

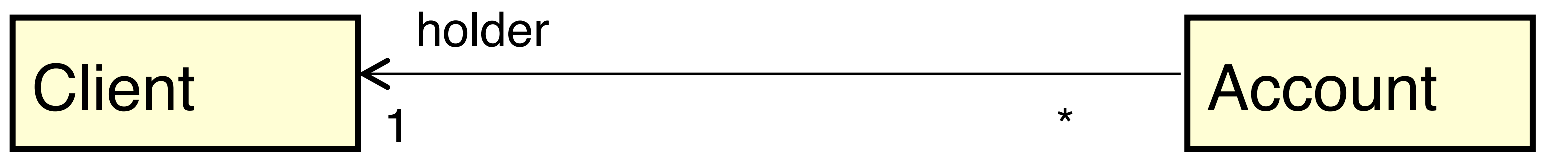
Class diagram vs. object diagram –Favre/ Parissis



Navigation

If the association is unidirectional
the navigation is only one way

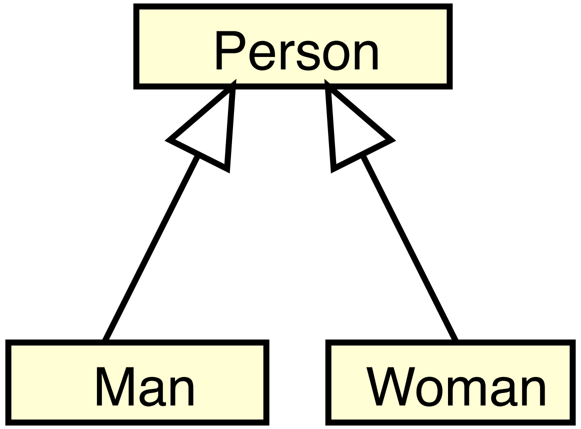




A priori useful only during design or implementation
If in doubt, don't put any direction !!!

Generalization / Specialization –Favre/Parissis

- A class can be the generalisation of other classes
- These classes are specialisation of this class

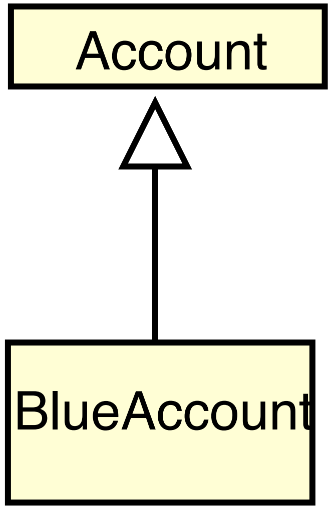


"Super class"

General case

« Sub class"

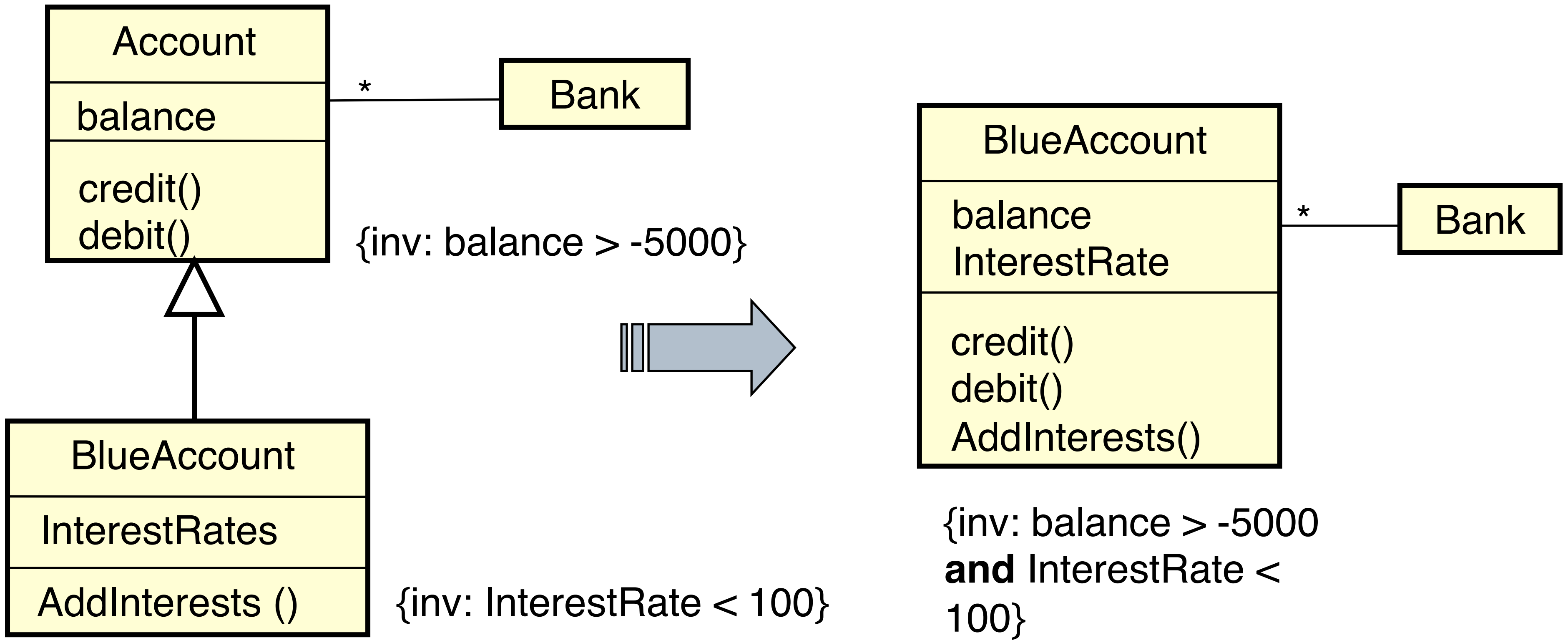
Specific case



Two interpretations (in UML) :
- inheritance relation
- sub-type relation

Inheritance –Favre/Parissis

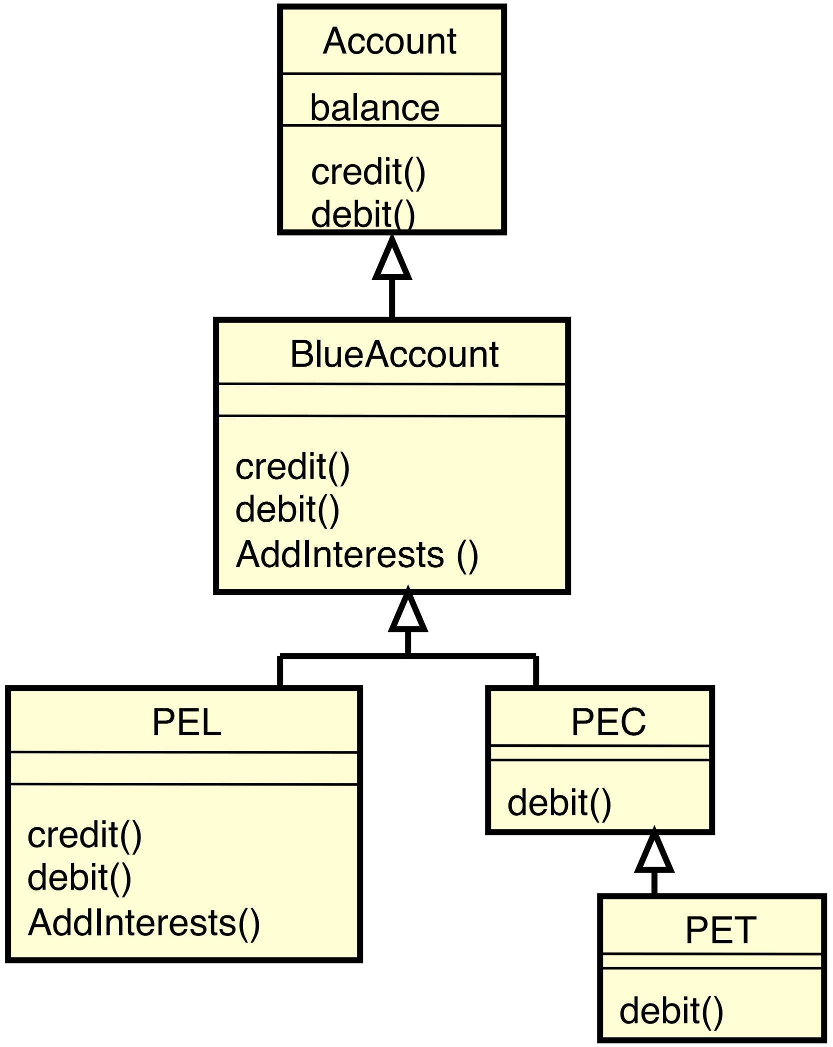
- Sub-classes inherit properties of super classes (attributes, methods, associations, constraints)



Inheritance and redefinition –Favre/Parissis

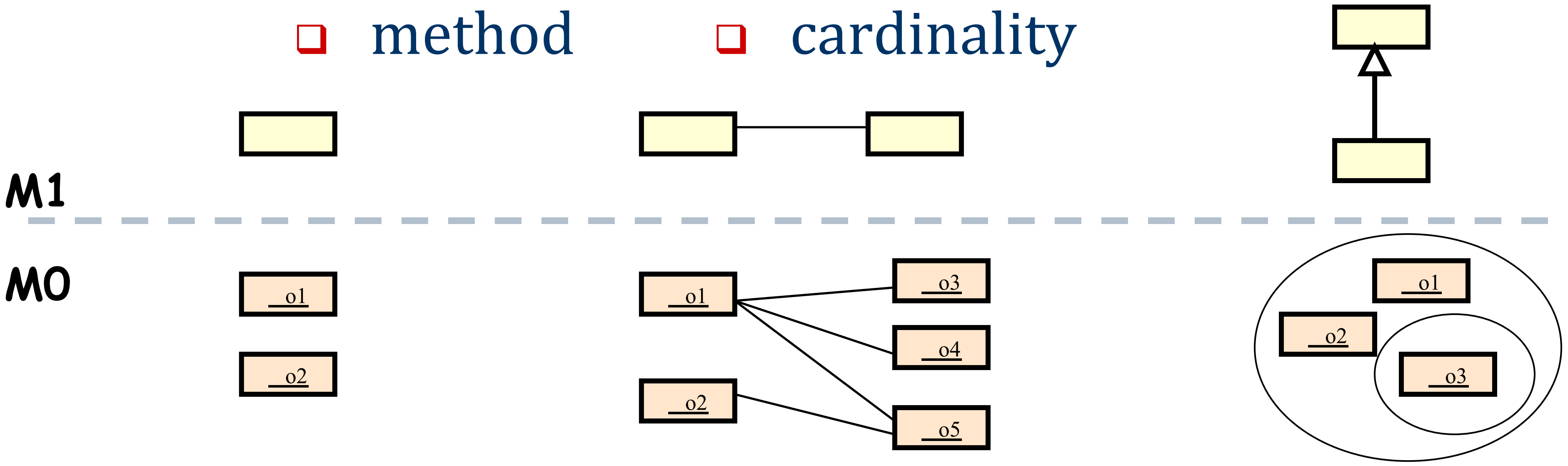
An opération can be redefined in sub classes

Allows the definition of specific methods to realize a same operation



Synthesis about base concepts –Favre/Parissis

- Class
  - attribute
  - method
- Association
  - role
  - cardinality
- Inheritance



- Object
- Link
- Inclusion

Outline

- UML presentation
- Basic concepts
- Advanced concepts
- Conclusion

+ # - Visibility –Favre/Parissis

- Restrain the access to model elements
- Control and avoid dependencies between classes and packages

+	public	visible
#	protected	visible in class / sub-classes
-	private	visible in class
~	package	visible in package

- Useful at design and implementation times
- Meaningless in an abstract model
- To be used only when necessary
- Semantics depends on the programming language

Attribute declaration – Favre/Parissis

[/] [*visibility*] *name* [: *type*] [*card order*] [= *initial-value*] [{ *props...* }]

```
age
+age
/age
- balance : Integer = 0
# age  : Integer [0..1]
# numsecu : Integer {frozen}
# keyWords  : String [*] {addOnly}
nbPerson : Integer
```

- Detail level should be adapted to the level of abstraction

Operation declaration – Favre/Parissis

[/] [*visibility*] *name* [**(** *params*)] [**:** *type*] [{ *props...* }]

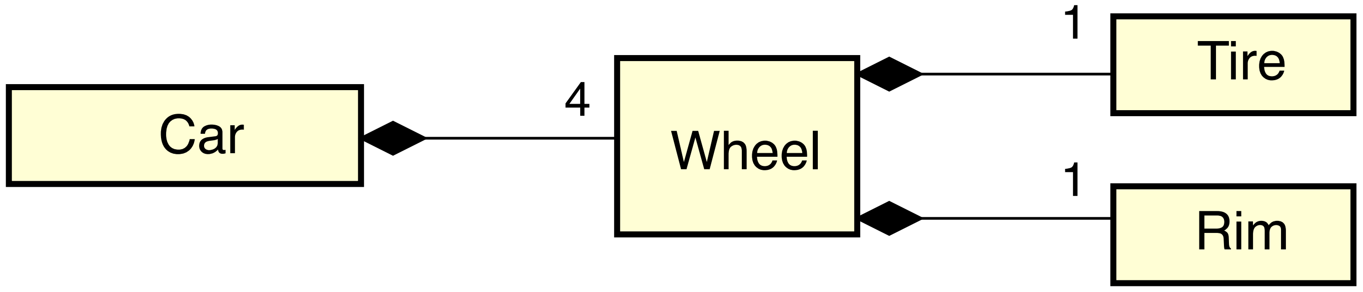
params := [*in* / *out*/ *inout*] *nom* [**:** type] [**=**defaut] [{ *props...* }]

```
/getAge()
+ getAge() : Integer
- updateAge( in date : Date ) : Boolean
# getName() : String [0..1]
+getAge() : Integer {isQuery}
+addProject() : { concurrency = sequential }
+addProject() : { concurrency = concurrent }
+main( in args : String [*] {ordered} )
```

- Detail level should be adapted to the level of abstraction

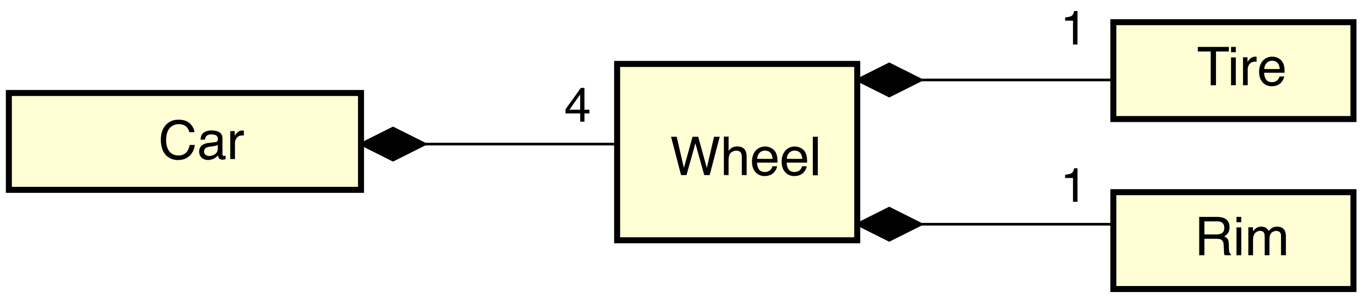
Composition –Favre/Parissis

- Intuitively: component/composite relationship
  - A specific association providing constraints related to the notion de component



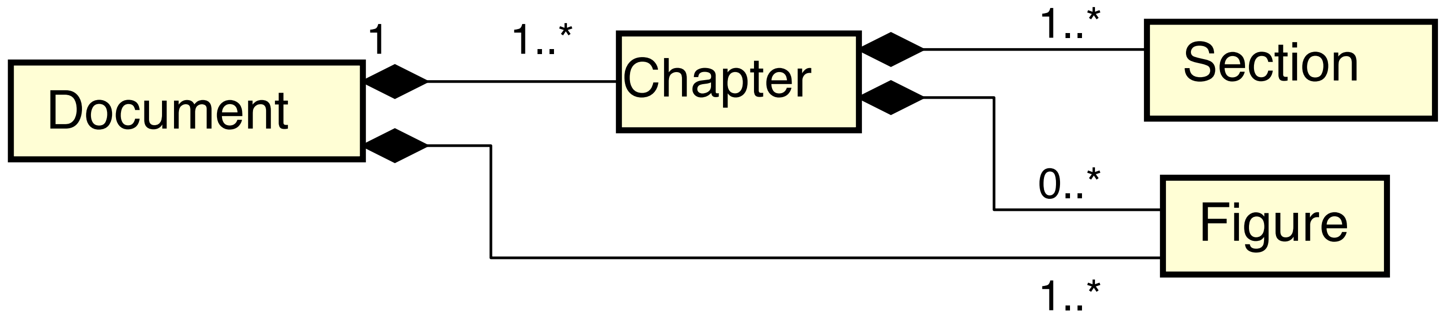
Composition –Favre/Parissis

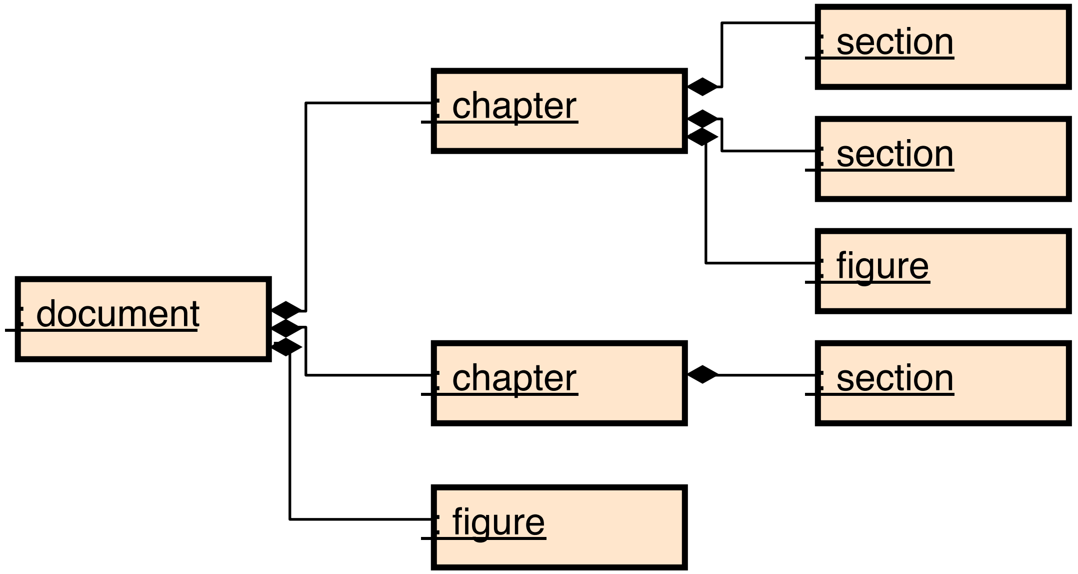
- Constraints
  - A component can only be in a single composite
  - A component cannot exist without its composite
  - When a composite is destroyed, its components are destroyed too



- Really depends on the situation (system) to be modeled
  - Car dealer vs. reseller parts

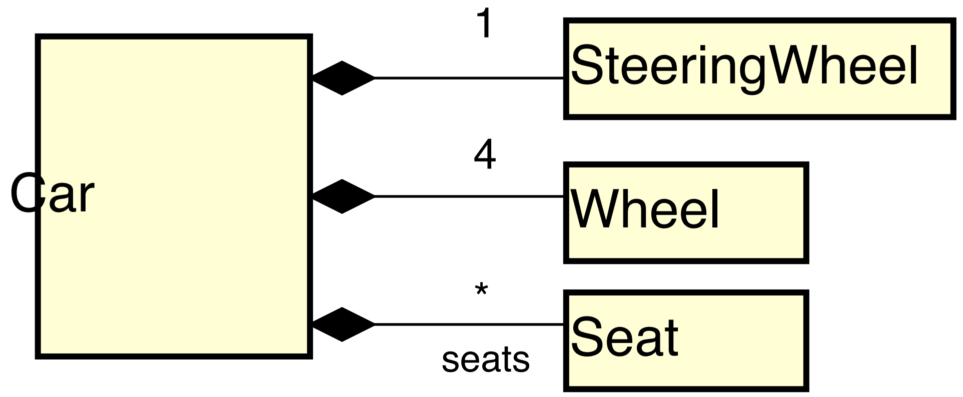
Composition – other example

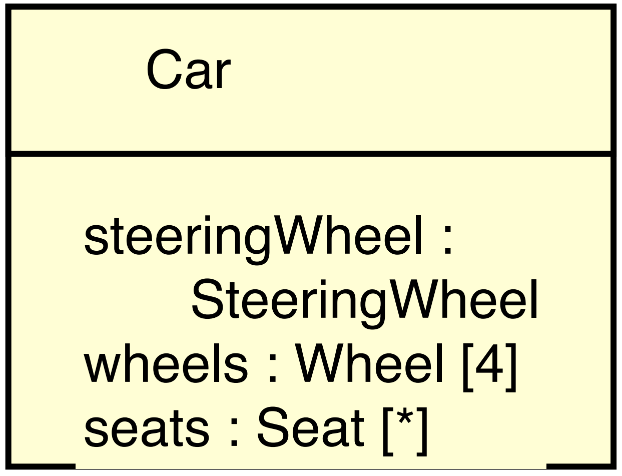


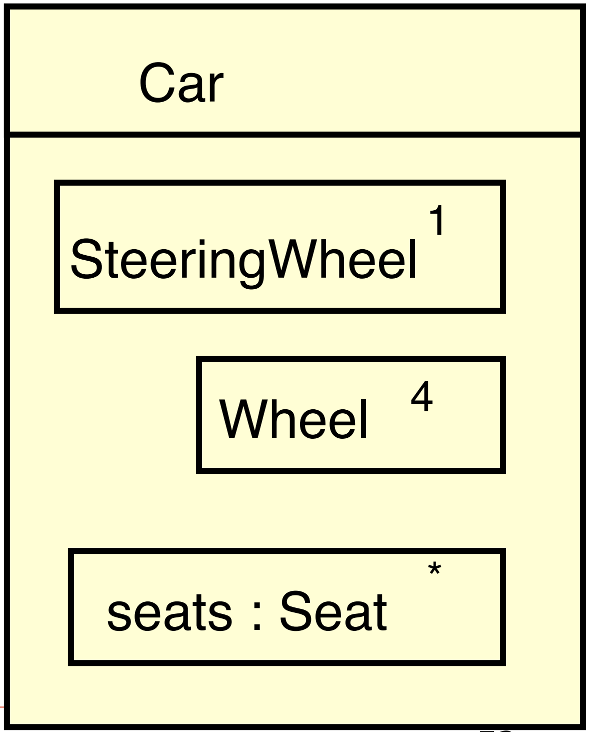


Constraint :
the components make up a tree

Composition – other notations

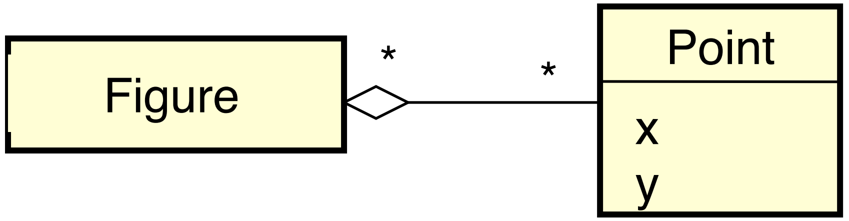


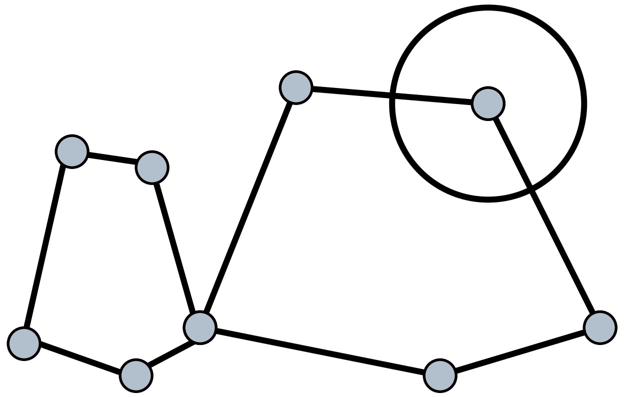




Aggregation –Favre/Parissis

- An association
  - With constraints characterizing the notion of membership

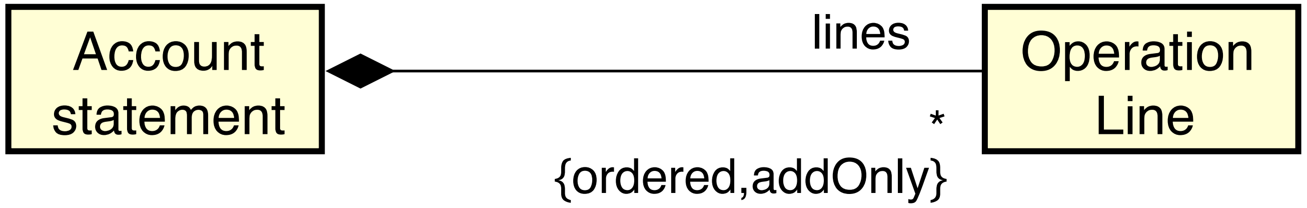




- Notes
  - Sharing is authorized
  - To use with cautious – suppressed in UML2.0

Predefined association constraints –Favre/ Parissis

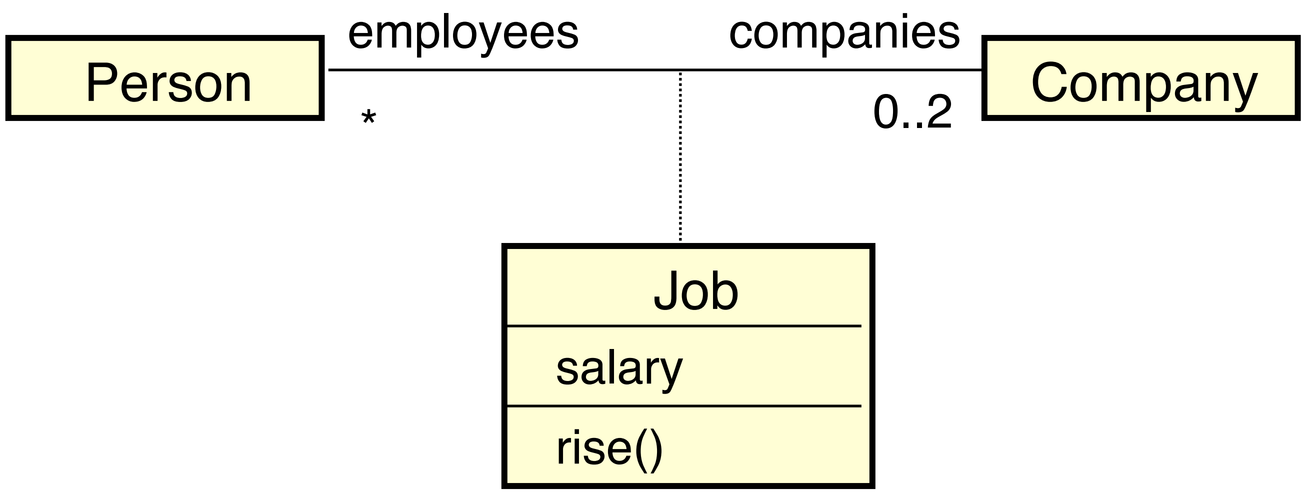
- For instance
  - { ordered } : collection elements are ordered
  - { nonUnique } : possible repetition (UML2.0)
  - { frozen } : fixed at creation, cannot be changed
  - { addOnly } : no element can be deleted



- More constraints can be defined

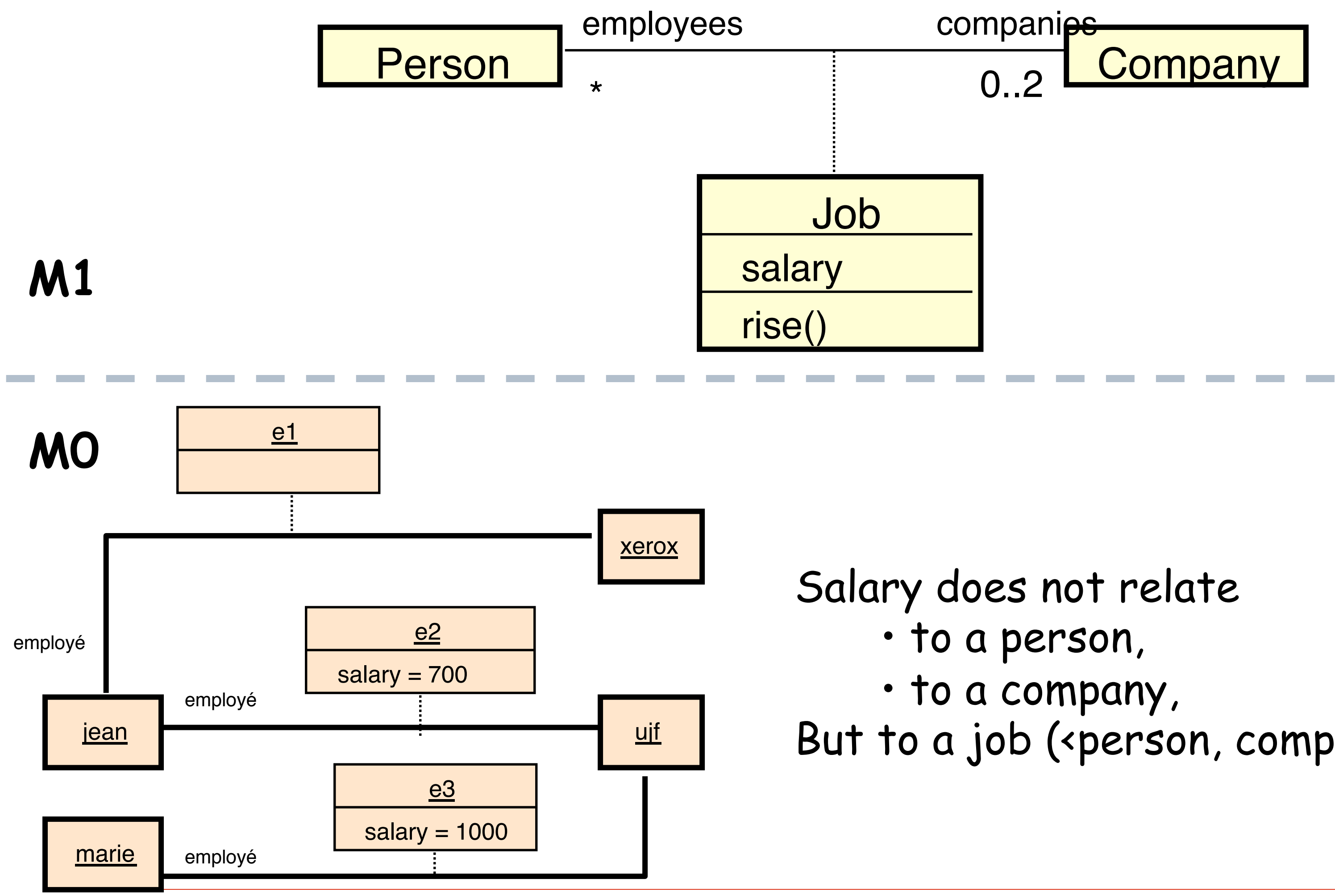
Associative classes –Favre/Parissis

- To associate attributes/methods to associations



- The name of the class is the name of the association

Associative classes –Favre/Parissis

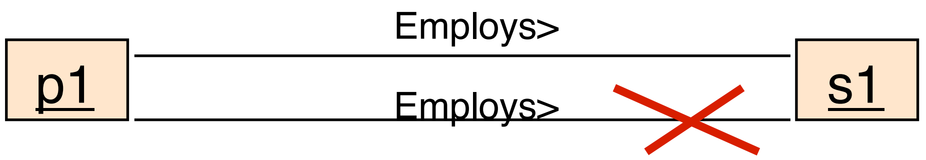


Salary does not relate
- to a person,
- to a company,

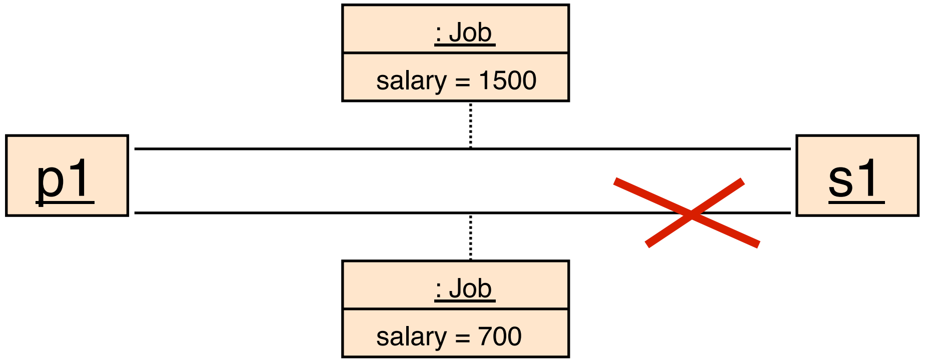
But to a job (<person, company> couple).

Associative classes –Favre/Parissis

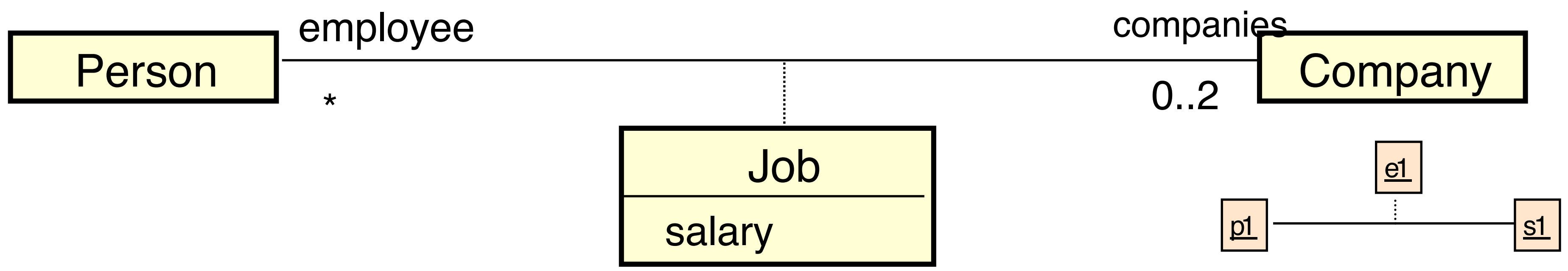
Reminder: No more than one link of a given type between two objects



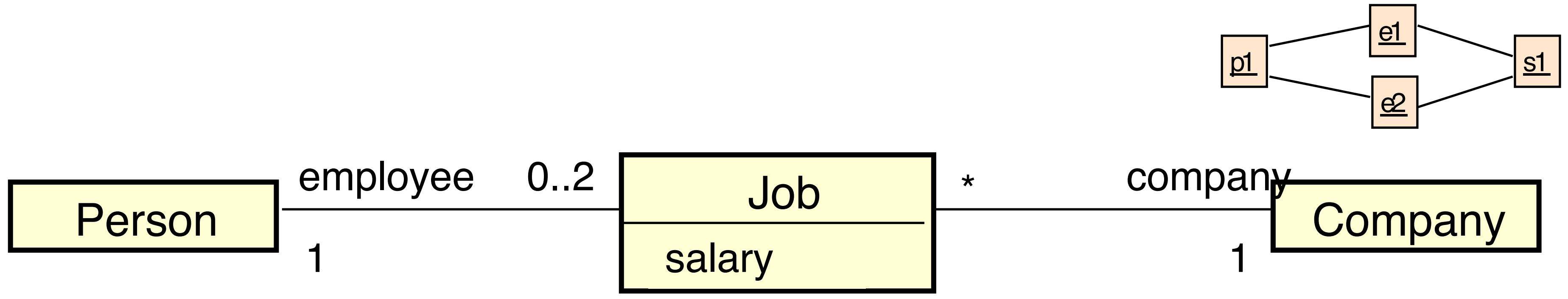
This is still valid for an associative class



Associative classes –Favre/Parissis



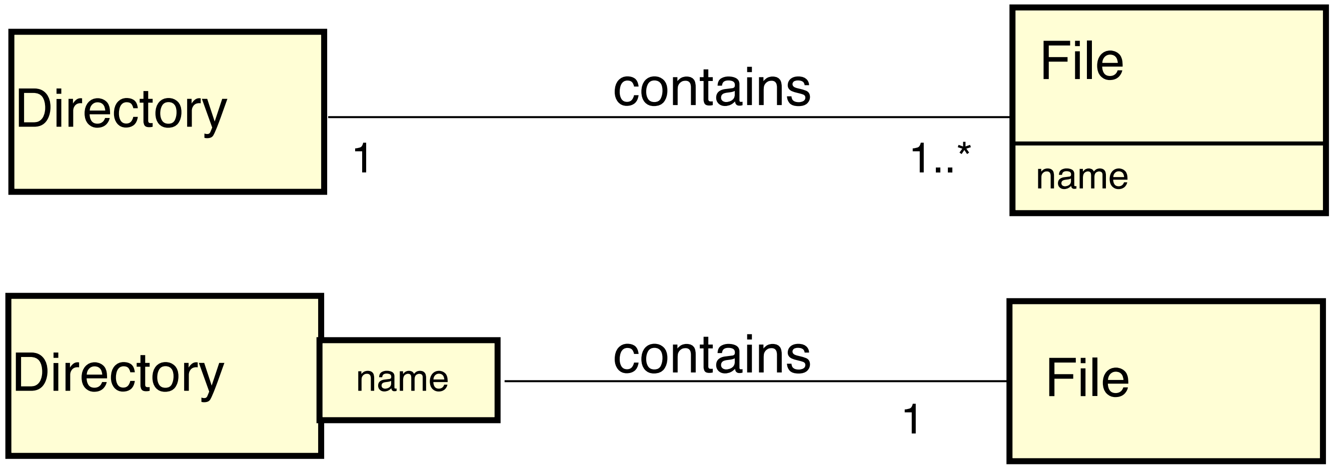
Here, a person may have two jobs, but not in the same company



Here, a person may have two jobs in the same company

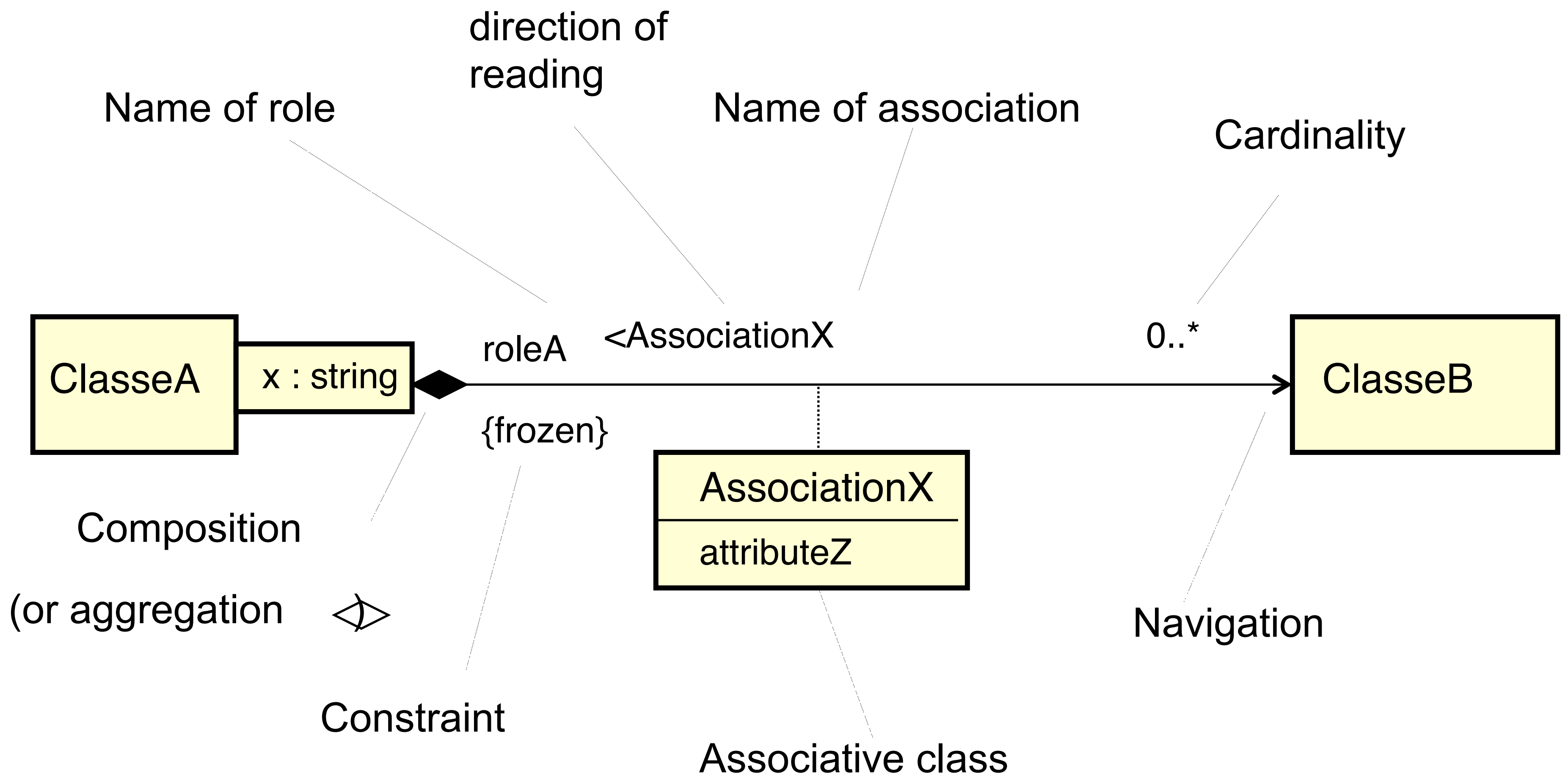
Qualified associations

A ***qualifier*** is an attribute or a set of attributes whose value is used to determine what are the instances associated with a given instance via an association.

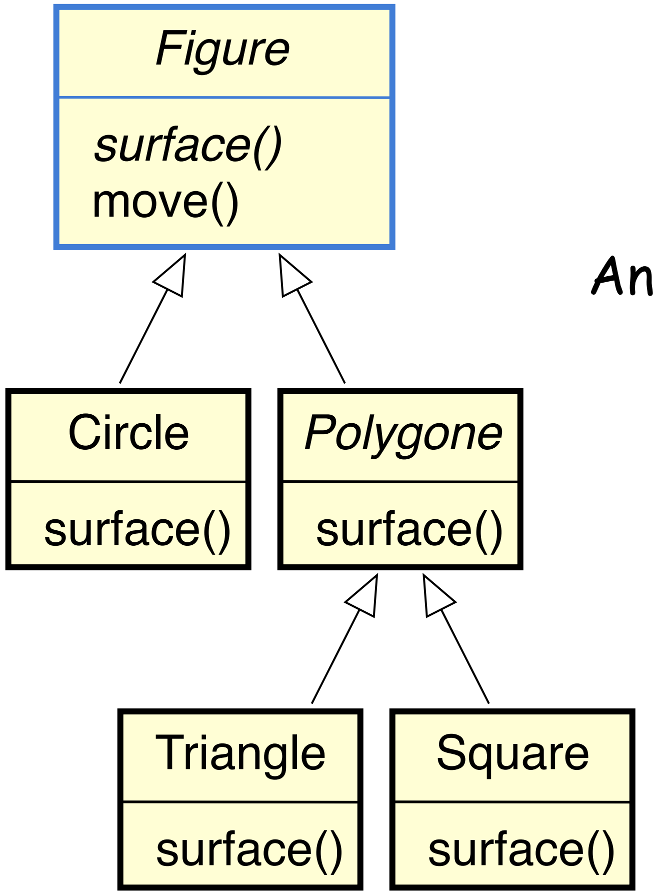


The attributes of the qualifier are attributes of the association.
The qualification reduces the multiplicity, usually to 1 (notion of key)

Synthesis on association



Abstract classes and methods –Favre/ Parissis

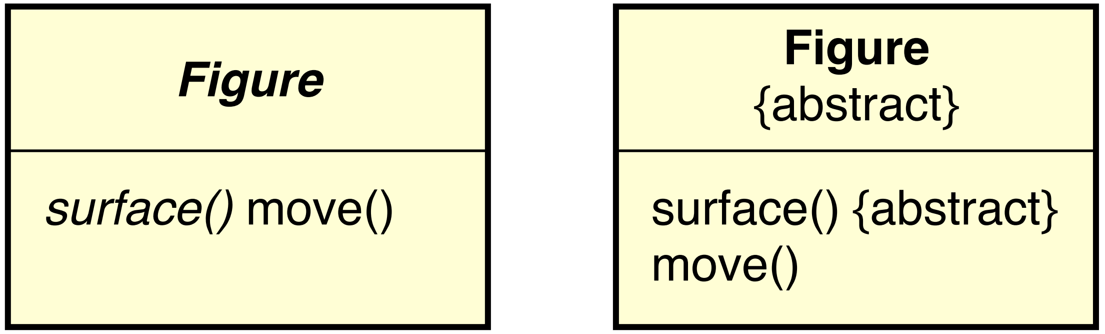


An abstract class

- cannot be instantiated
- allows the definition of an abstract behavior
- can contain abstract methods

An abstract method

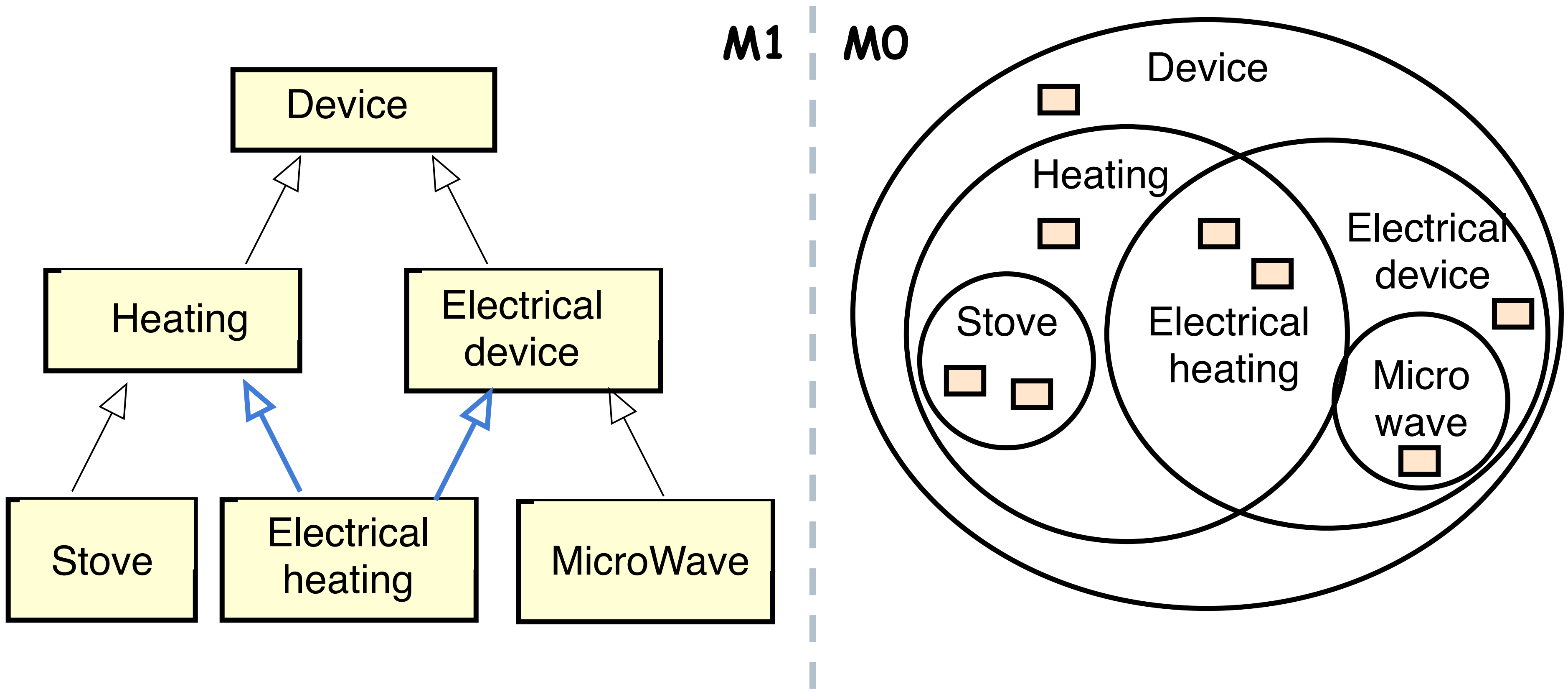
- must be defined in a sub-class
- belongs to an abstract class



Equivalente notions

Multiple inheritance –Favre/Parissis

A class can inherit from several super classes



Not allowed in some languages (for instance Java et C#)

UML inheritance – from Favre/Parissis

- Default hypothesis
  - A class can inherit from several super classes
  - An object is an instance of a single class
  - An object cannot change its class (from which it has been created)

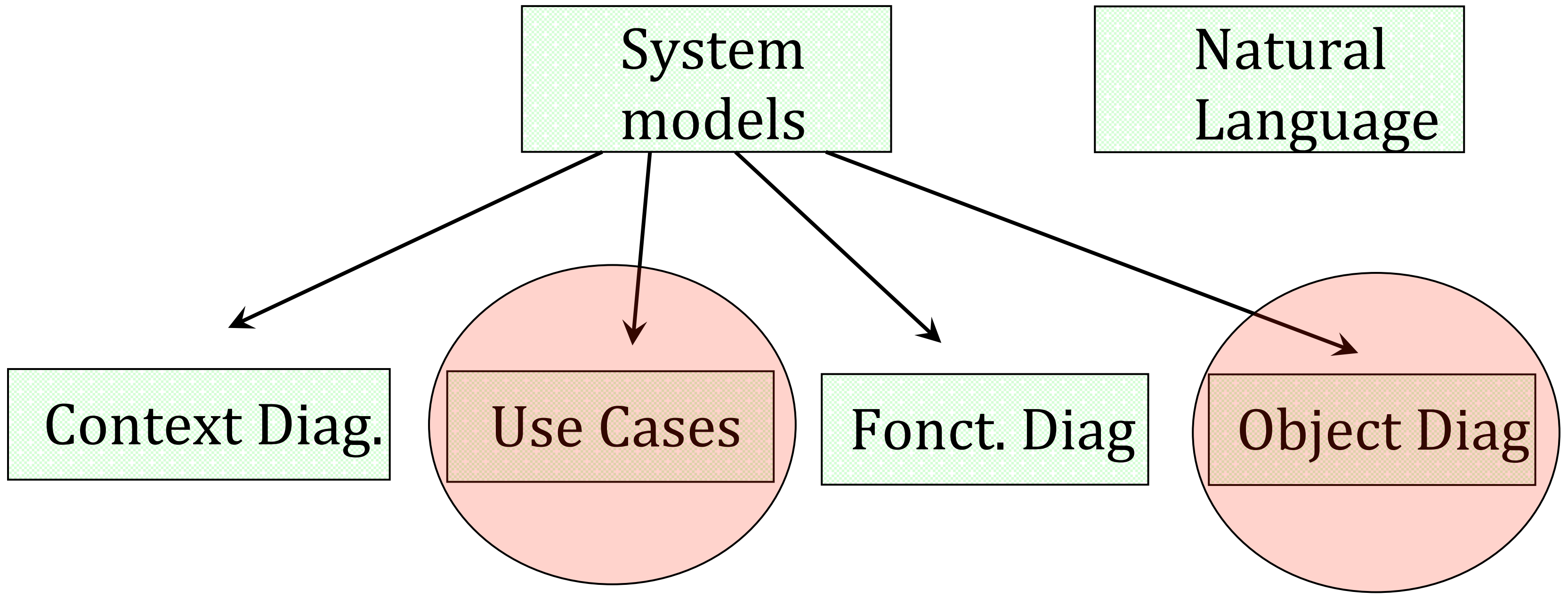
Outline

Conclusion – from Favre/Parissis

- UML is standard, popular but complex
- UML can be used during analysis and design
- Several extensions have been proposed
  - Specialization
- UML is here to last …

Reminder

- Requirement document



- Not enough !!!

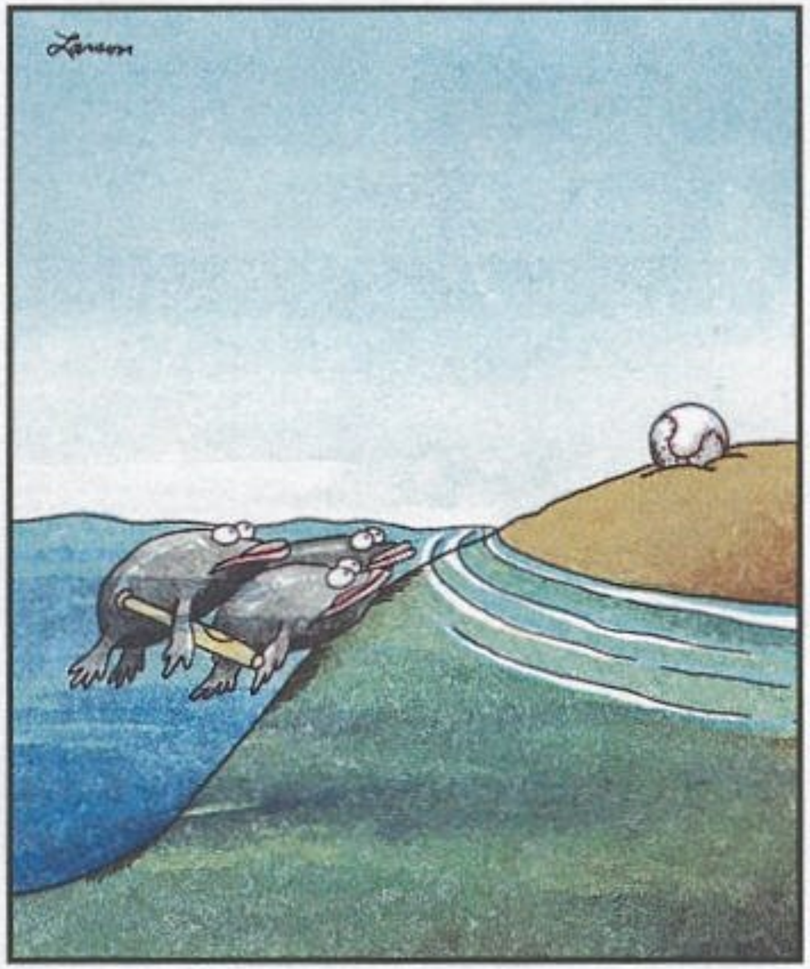
Conclusion

- UML is standard, popular but complex
- UML can be used during analysis and design
- Several extensions have been proposed
  - Specialization
- UML is here to last …

Conclusion

Model based development is immature.

It progresses ...



Great moments in evolution