INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Software engineering & UML modeling: UML

Frédérique Laforest

Main references

- http://www.uml.org/

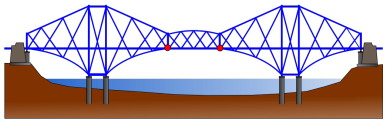
- Scott W. Ambler. *The Object Primer 3rd - Agile Model Driven Development with UML 2*, Cambridge University Press, 2004

  http://www.agilemodeling.com/

- Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2003

What is a model?

- Representation of an object of study
  - To communicate, design, document
- Can be generic
  - Valid for multiple instances
- A point of view on the object
  - Represents what's needed



Why should we make models?

3 complementary objectives:

- A mean to ease **discussions** on the studied system
  - System to be built, or existing system
  - Incomplete or false models help discussion
- Models provide **documentation** on an existing system
  - models must be correct even if incomplete
- A **detailed description** of future system (Model-driven development)
  - models must be correct AND complete

Which model for software?

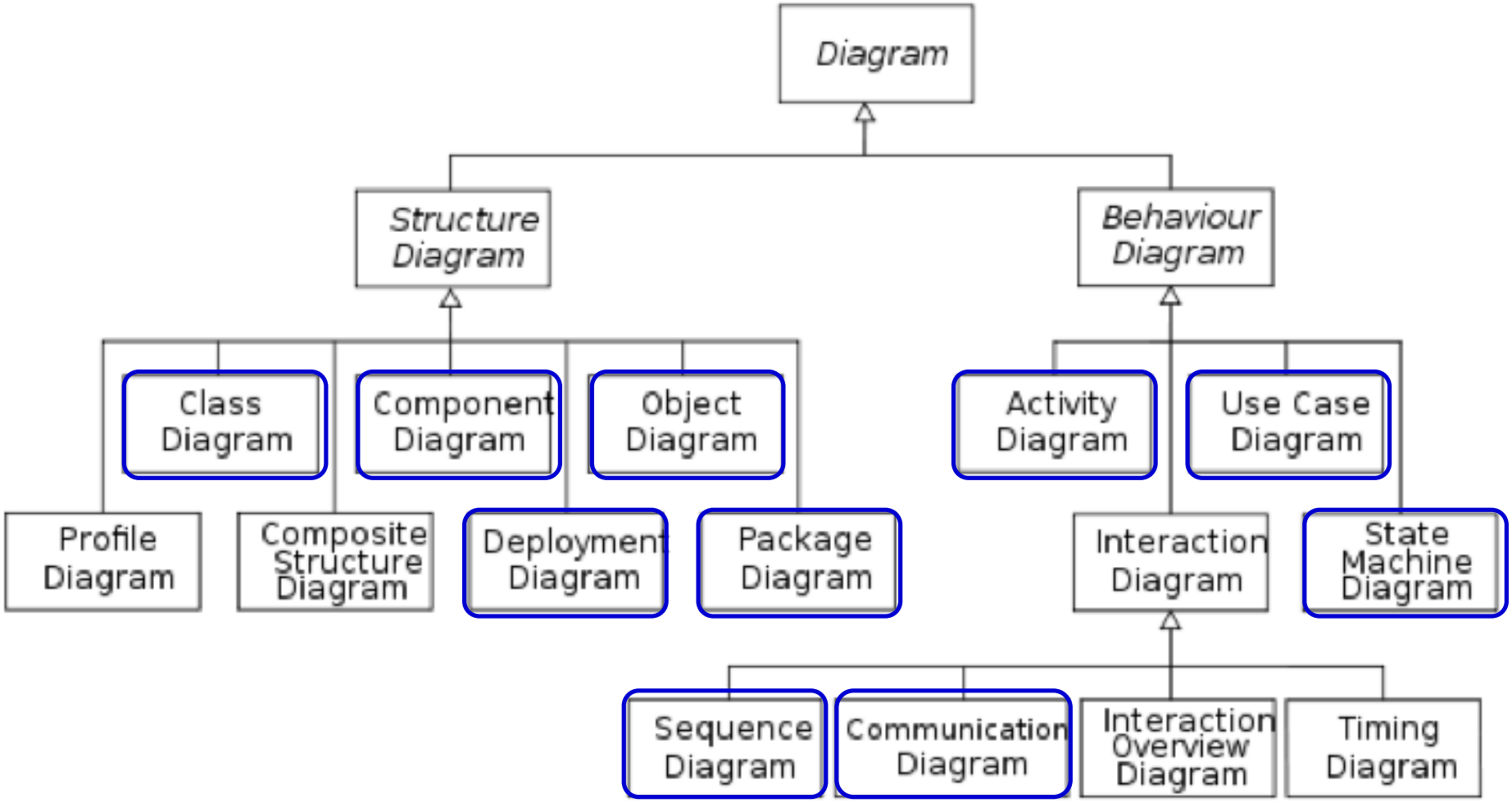
Grady Booch, James Rumbaugh, Ivar Jacobson

- Before UML
  - > 100 methods and graphical representations, having various advantages and drawbacks
- UML : Unified Modeling Language (v0 in1996)
  - A set of models : terms glossary and graphical representations
  - No advocated method on how to build these models
- UML is a standard from OMG
  - V2.5 since octobre 2012

UML 2 – the diagrams

- Graphical language for modeling
  - Mainly schemas …
  - But also associated description texts
- Based on object technology
- 14 types of diagrams
  - see hierarchy next page

UML 14 types of diagrams



UML - tools

- freeware
  - To be installed : ArgoUML, Papyrus, ***StarUML***
  - On line : ***UMLetino, draw.io***
  - …
- commercial
  - Rational Rose
  - …

A long list on
http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

The PrintAI app

A company has developed a new medical sensor. This sensor, put on a patient chest, provides a set of values corresponding to various attributes; the set of data is called a patient *print*. Of course, *prints* are sensitive data and should be managed carefully and follow the law on medical data.

The Health Department of our Government wants to provide doctors with a *print* analysis system, called PrintAI, that can help on diseases prevention. For that, a large set of prints is available, where each print is labelled with a set of diseases.

The doctor enters a print file in PrintAI, and the system gives rapidly the set of statistically possible diseases. On high risks, the doctor will prescribe complementary tests (out of the PrintAI system). **The objectives of the system are to reduce time to risk identification and to lower the number of inadequate complementary tests.**

PrintAI must also provide the list of diseases it knows.

The Health Department of our Government selected you for the development of PrintAI.

UML – Use Case Diagram

- Behavior diagram
- Identify the interaction opportunities between the system and actors (outside the system)
- List system functionalities
  - First objectives= help requirement engineering
- A diagram … and explanatory text!

UML – Use case diagram

- Aim : get functional requirements of the system
  - WHAT it will have to do, which services it will provide
  - But NOT HOW it will do
  - End user point of view: it shows **the reasons why** the end user will use the system

- Provides a schema AND textual descriptions of use cases

Use Case diagram example

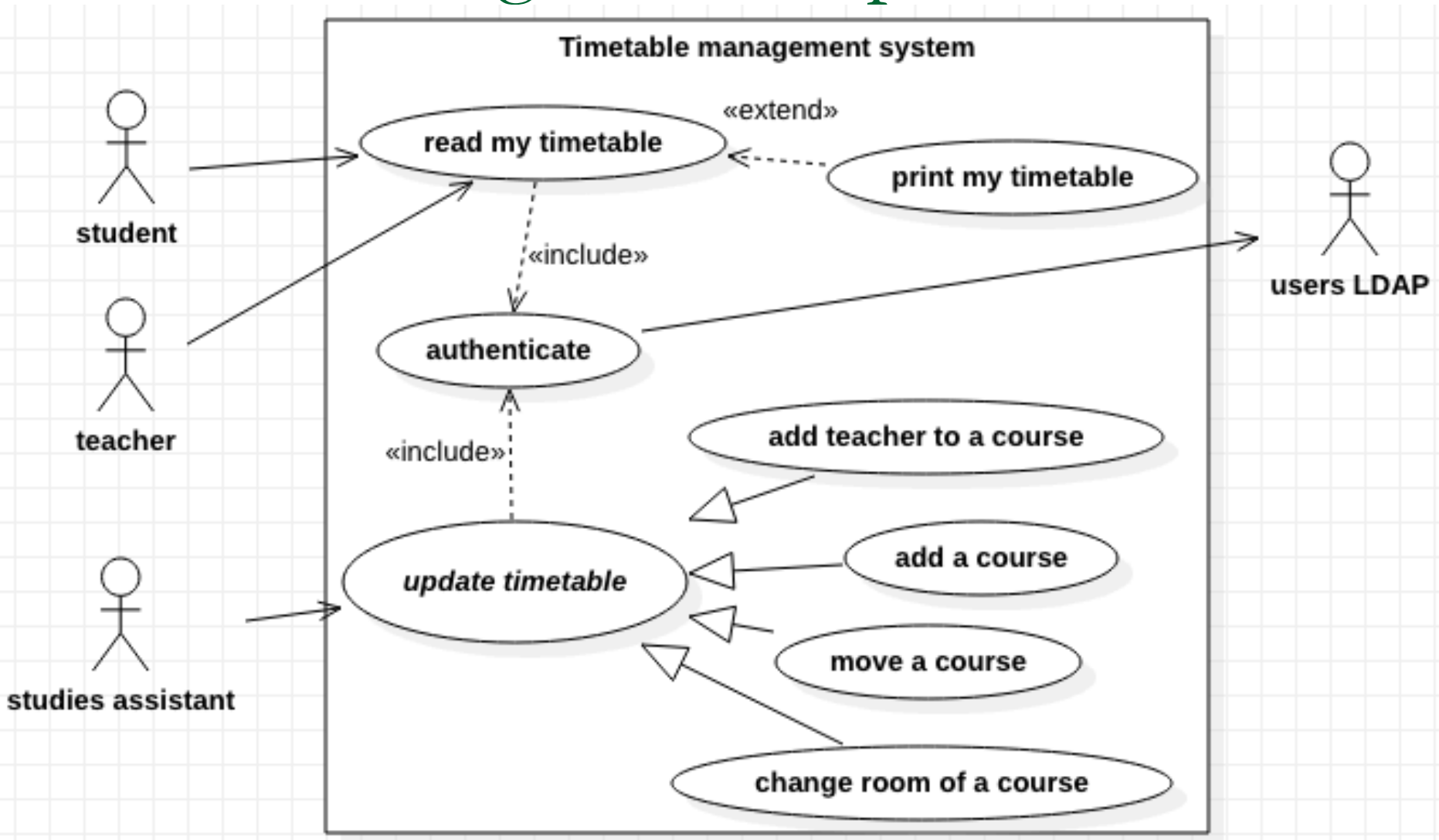


Diagram elements

- The system to be modeled
  - Draw the fontiers of the system
  - Drawn as a *rectangle*
- Actor
  - Person or other system in interaction with the modeled system
  - Drawn as a *stick man*, outside the rectangle
- Use case
  - task made by the system in interaction with an actor
  - Drawn as an *oval*, inside the rectangle
- Association actor-> case
  - Defines which kind of actor can use the system to do this use case
  - Drawn as an *arrow* going from actor to use case
- Association case -> actor
  - Defines which actor the system should appeal to to realise the use case
  - Drawn as an *arrow* from case to actor

Actor (we should say actor role)

- Entity (human or machine) outside the system
  - Allows to determine the limits of the system
- An actor plays a role regarding the system
  - The actor activates the system that triggers an action
  - The actor is called by the system to realise some duties
- Categories of actors
  - Main actors (main functions of the system)
  - Secondary actors (administration/maintenance)
  - External hardware
  - Other systems

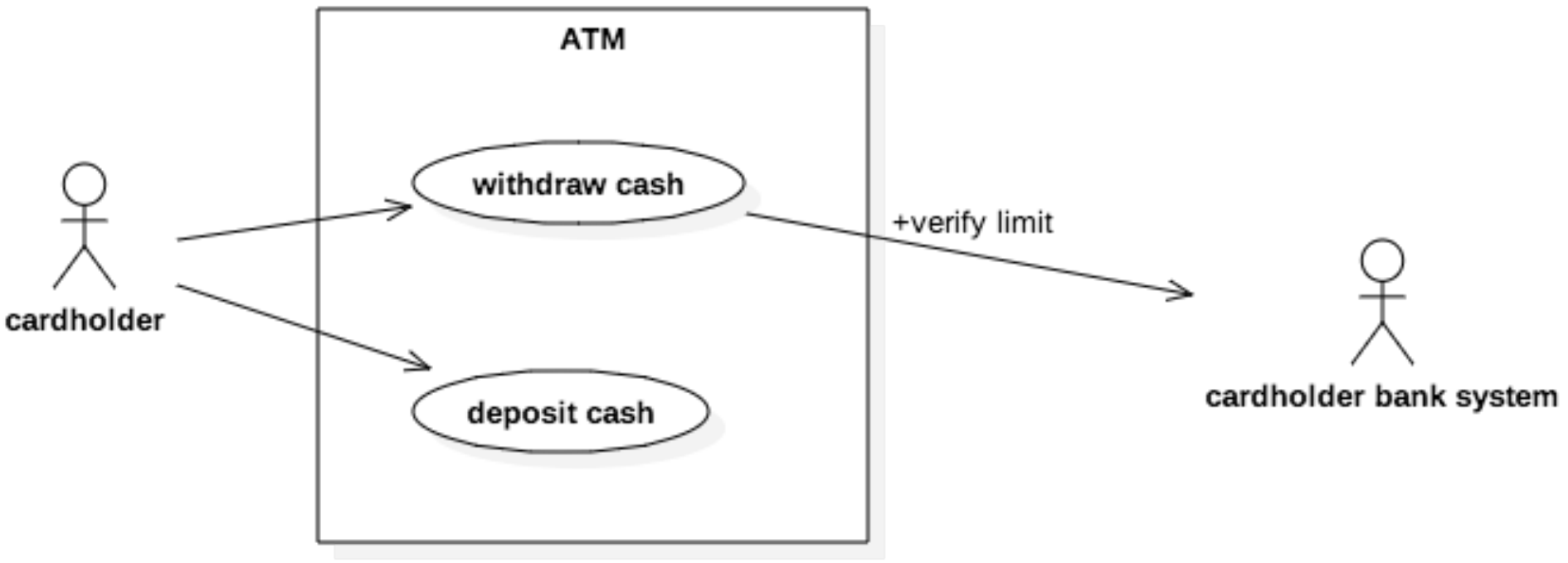
Text : An actor is described with few lines

Use case

- A specific functionality launched by an actor
  - Main word = a verb
- An ordered set of actions made by the system to fulfil the functionality.
  - Each use case can define different scenarios depending on context
- It produces a visible result for an actor
  - Examples : withdraw cash, answer an email

Description text of a use case : see later

Use Case example: system = ATM



Incomplete diagram!

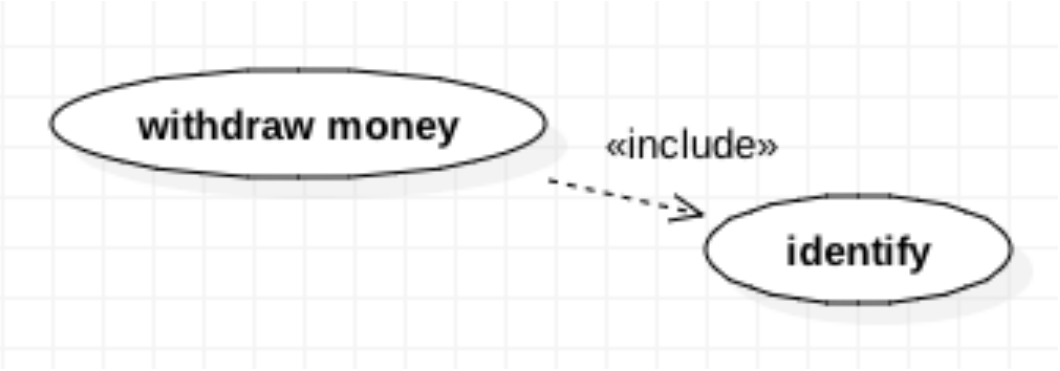
Use cases relationships

- Use cases can have relationships
- 3 types:
  - Include
  - Extend
  - Generalize

Include relationship

- Case A Includes Case B
  - B contains a subset of A scenario
  - Dashed arrow from A to B + stereotype <<include>>

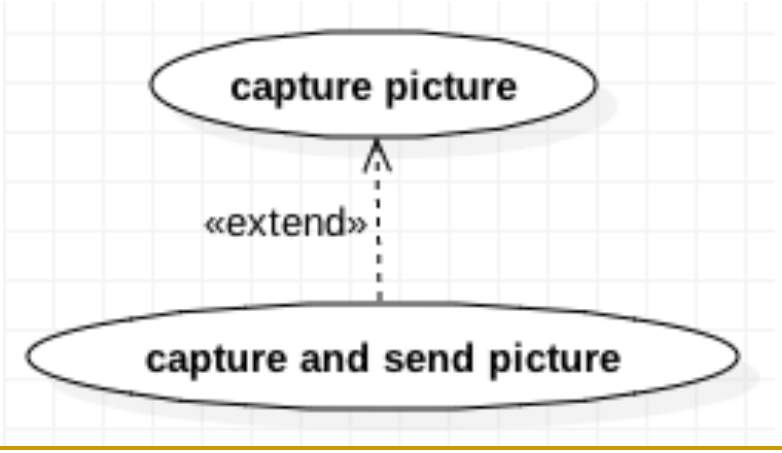
- Example : « Withdraw Money » includes « identify »



Extend relationship

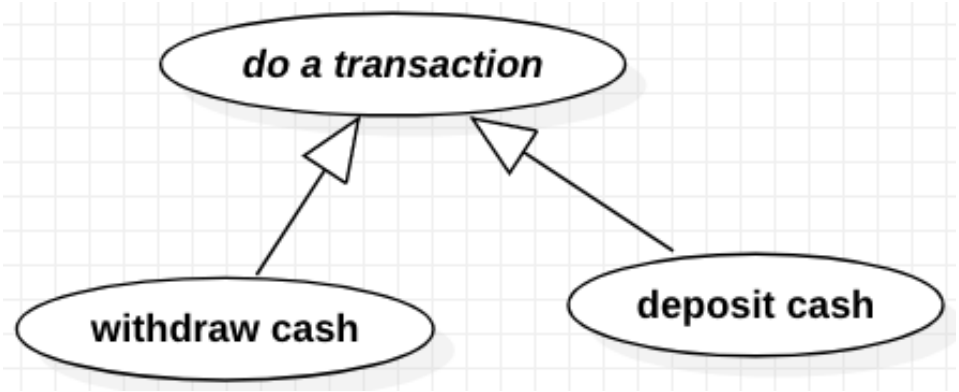
- Case A Extends case B
  - A enhances be B by adding further functionalities
  - Dashed arrow from A to B + stereotype <<extends>>

- Example : « Capture and send picture » extends « capture picture »



Generalize relationship

- Case B generalizes case A
  - Case A is a particular case of case B
  - Similar to inheritence to an abstract class
    - e.g. « do a transaction » is an abstract use case
  - Drawn as an arrow with a triangle end



Description of a use case

- A use case can be executed following different scenarios depending on the context.
- Scenario
  - Sequence of steps to fulfil a use case
  - All scenarios of the same use case have the same actor initiator
- Descriptions of scenarios are often reused for assessment tests by the client

Text describing a use case

- The text is structured as a set of required items
  - name, preconditions… see later
- Each item contains some short text
- The text provides
  - A nominal scenario = what happens when all goes well
  - Some extensions at some points in the nominal scenario
  - Failure points
  - It may also refer to other diagrams, GUI schemas, etc.

Text describing a use case

- Use case name
- Context of usage : for what reason this use case can be launched
- Systems considered
- Main actor
- Other actors
- Preconditions
- Success guaranties: state of the world if success
- Trigger: what makes the use case start, can be time
- Nominal scenario : description of steps
- Extensions : of the nominal scenario
- Additional information

Example – text description of use case « withdraw cash »

- Use case name : withdraw cash
- System considered: ATM
- Main actor : cardholder
- Other actors and role: cardholder bank system
- Precondition : enough money on cardholder account
- Success guaranties: cash delivered, cardholder account balance updated accordingly

Example – text description of use case « withdraw cash »

- Nominal scenario :
  1. Cardholder introduces card and code
  2. ATM validates card and code (other scenario included : identify)
  3. ATM provides the list of available use cases + quit option
  4. Cardholder chooses « cash withdrawal » and selects an amount
  5. ATM asks authorization from cardholder bank server
  6. ATM delivers card, money, recept
  7. ATM registers the transaction
  8. Back to step 3

Nominal scenario = idealistic scenario, with no error nor pb

Example – text description of use case « withdraw cash »

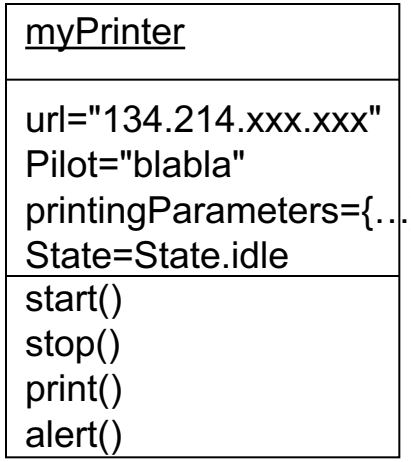
- Extensions
  - General breakdown: ATM cancels the transaction, informs the cardholder, delivers the card
  - Stolen card: ATM keeps the card
  - Account balance too low: ATM shows and messages, abords the transaction, delivers the card

Why texts with use case diagrams?

- ++ diagrams are simple to read => communication!
  - Useful to clarify fuzzy requirements
  - Understandable by the client
  - Prepare tests
- - diagrams can become unreadable
  - spaghettis
- - diagrams miss precision
  - The text adds information on the diagram

What is an Object?

- Object = State + Behaviour + Identity
- State
  - Set of values describing the state of an object
  - Each value is associated to a property called attribute
  - Values can be objects
- Behavior
  - Set of operations the object can do (methods)
  - The execution of a method is launched when the corresponding message is received
- Identity
  - Unique internal identifier, allows to distinguish 2 objects, even when they have the same state and behavior

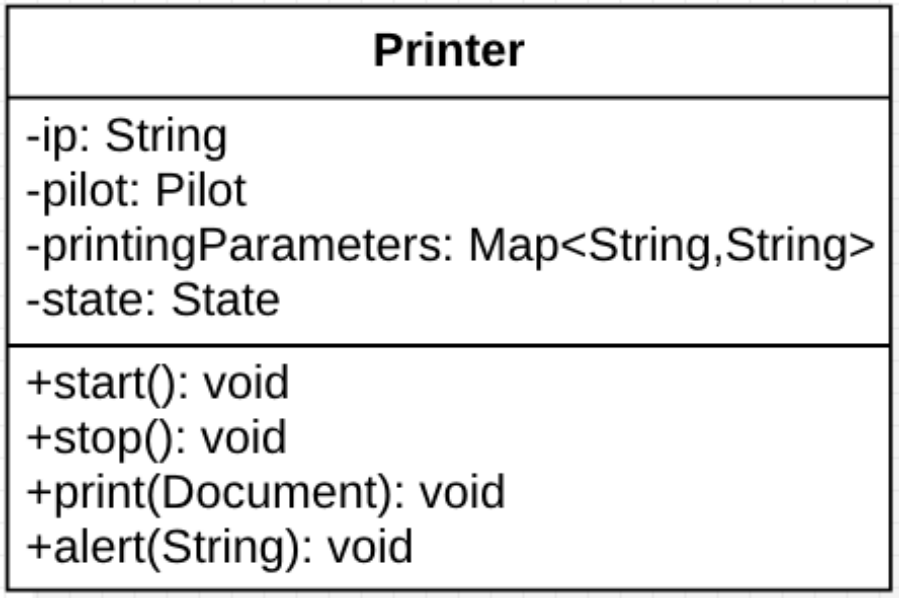


Objects and classes

- Objects that share the same structure and behaviour belong to the same class

- A class can be seen as a mould that allows to create objects
  - The created objects are instances of the class

What is a Class ?



- A class defines
  - **The Structure** of instances
    - = set of attributes
    - Fields that characterise an object (or its class)
  - **The Behavior** of instances
    - = set of methods
    - Operations that instances (or the class) can execute

NB : methods and attributes are called ***members*** of the class

  - A mechanism for **instanciation**
    - Creation of objects
    - Done by specific methods called constructors

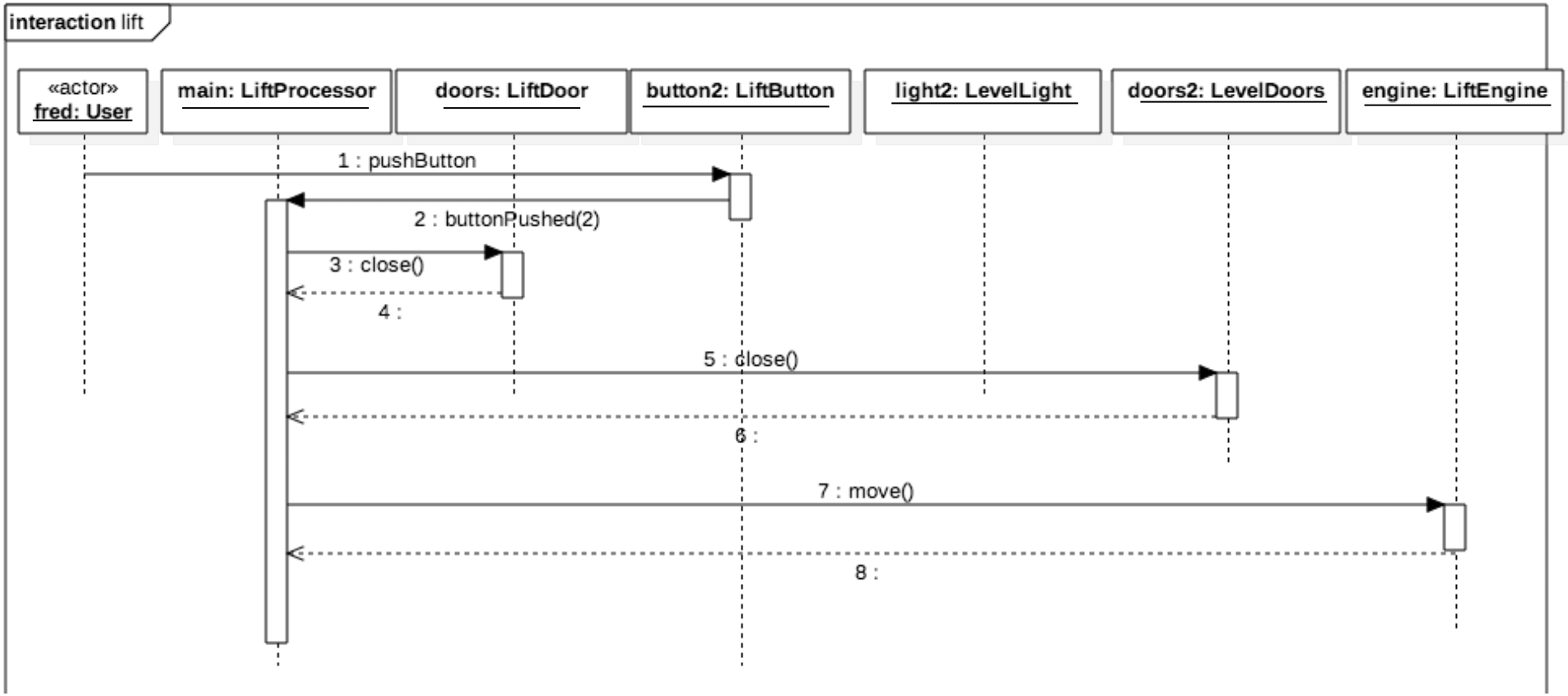
UML – Sequence Diagram

type Interaction

- **Sequential** representation of some processing and interaction of components of the system and/or actors

- This kind of diagram is very often used

First example

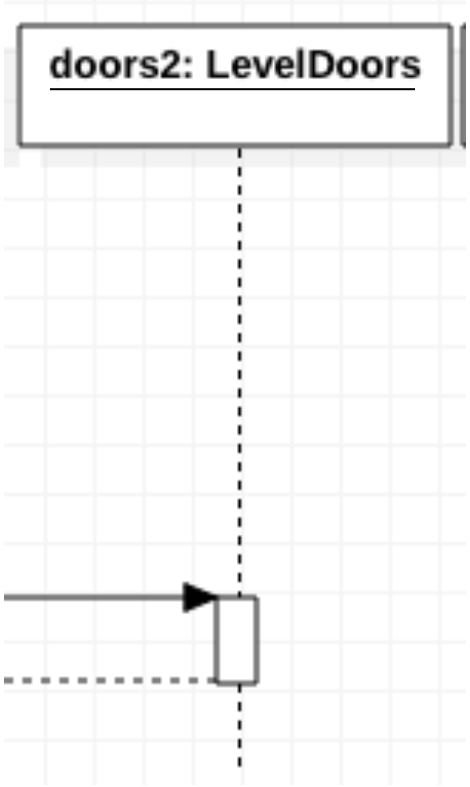


Main elements in a sequence diagram

- Participant
  - **actor** or **object** of the system participating in the realisation of the actions
  - Drawn as a *rectangle, name underlined (and/or :className)*
- Lifeline
  - Represents the objet in memory,
  - Drawn as a vertical dashed line, with rectangles when the object is active
- Message
  - Signal or method call or data sending
  - Drawn as an *arrow* (different types or lines and ends depending on the kind of message)

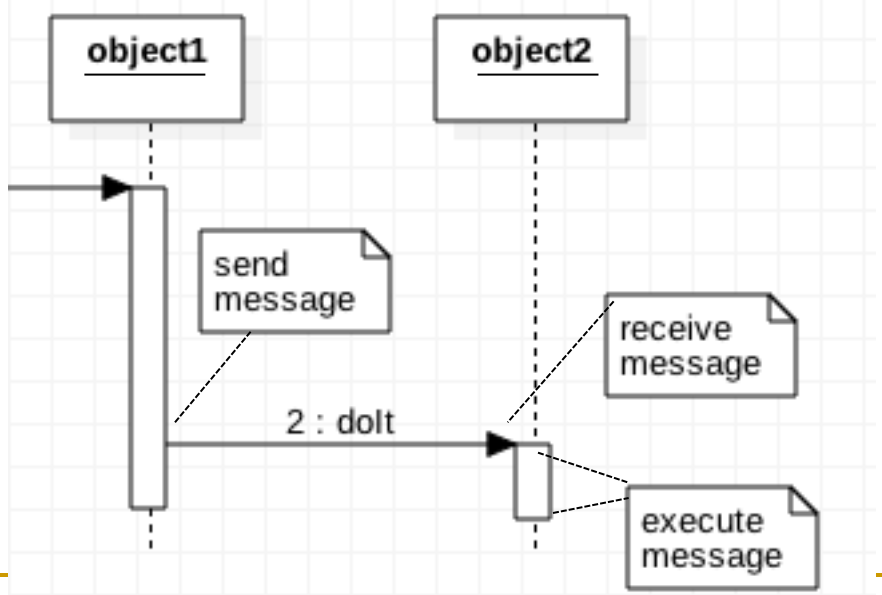
Participants and life lines

- 1 column = 1 objet
  - The rectangle gives the object name and type (class)
  - A vertical dashed line for its life line
  - If many objects of the same class play a role, should put one column by object



Message

- object1 sends the message doIt() to object2
  - object1 knows object2
  - object1 is active and sends a message to object2
  - doIt() is a functionality of object2
  - a rectangle on object2 lifeline shows it executes doIt()

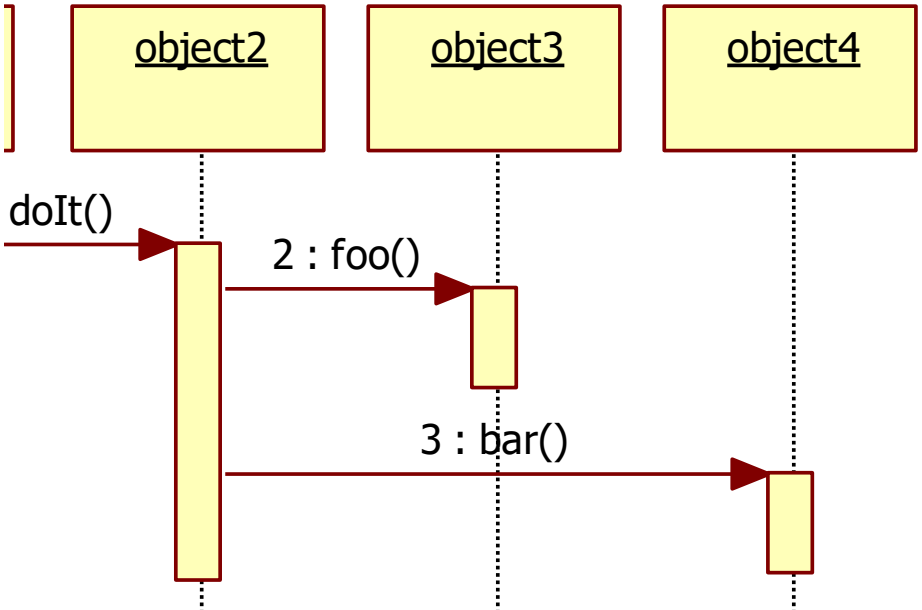


Differents types of messages

- Call
- Send
- Return
- Create
- Destroy

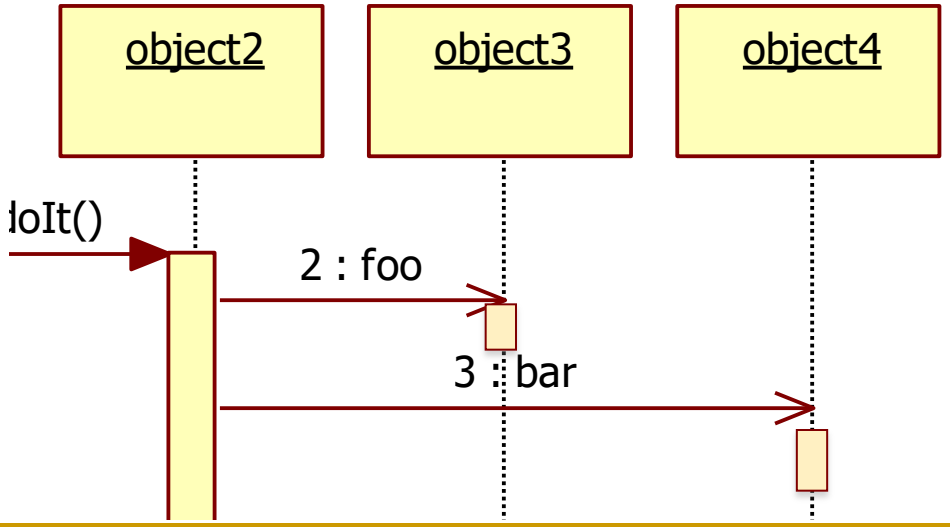
Message type Call

- Synchronous call
  - The caller is suspended during execution
  - Full line, filled triangle end



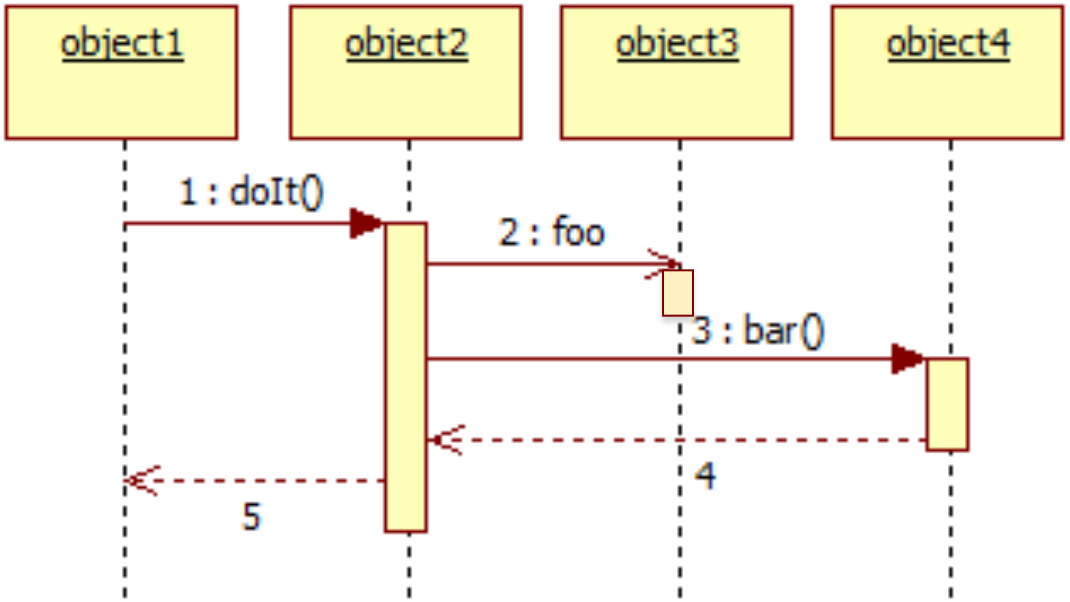
Message type Send

- Asynchronous call
  - The caller does not wait for an answer, it continues its activities
  - Full line, open arrow end



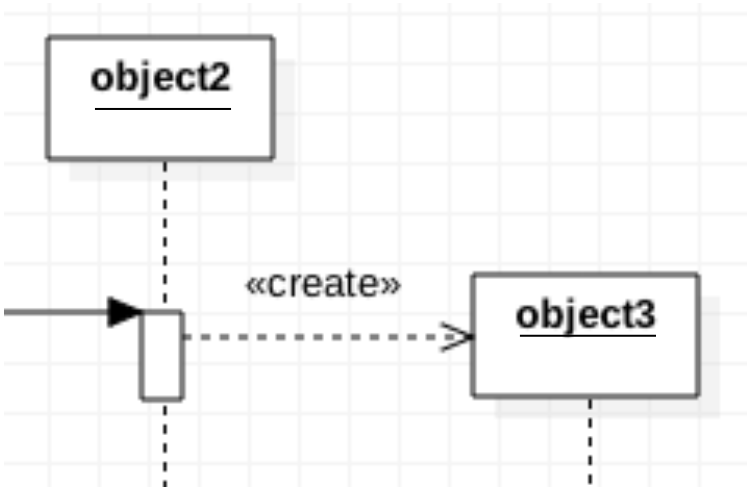
Message type Return

- Answer to a message (sync or async)
  - Dashed line, open arrow end



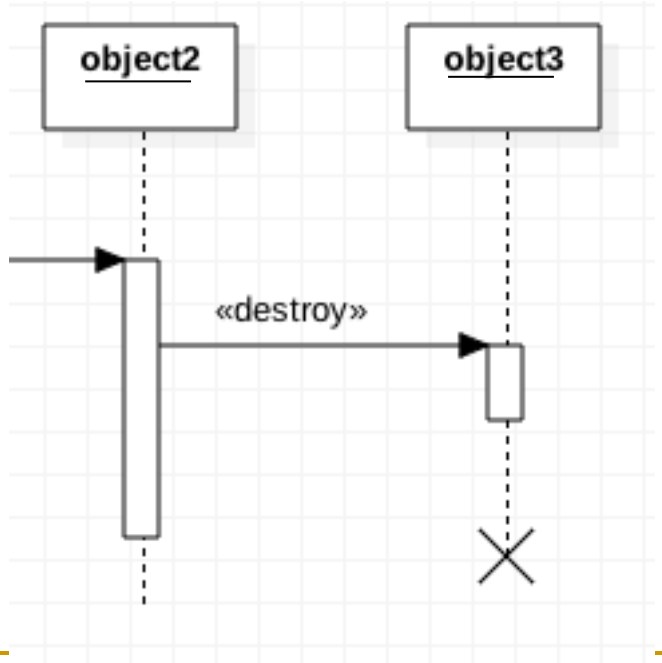
Message type Create

- Instanciation of an object
  - Dashed line, triangle end arriving on the created object rectangle, <<create>> prototype on the line



Message type Destroy

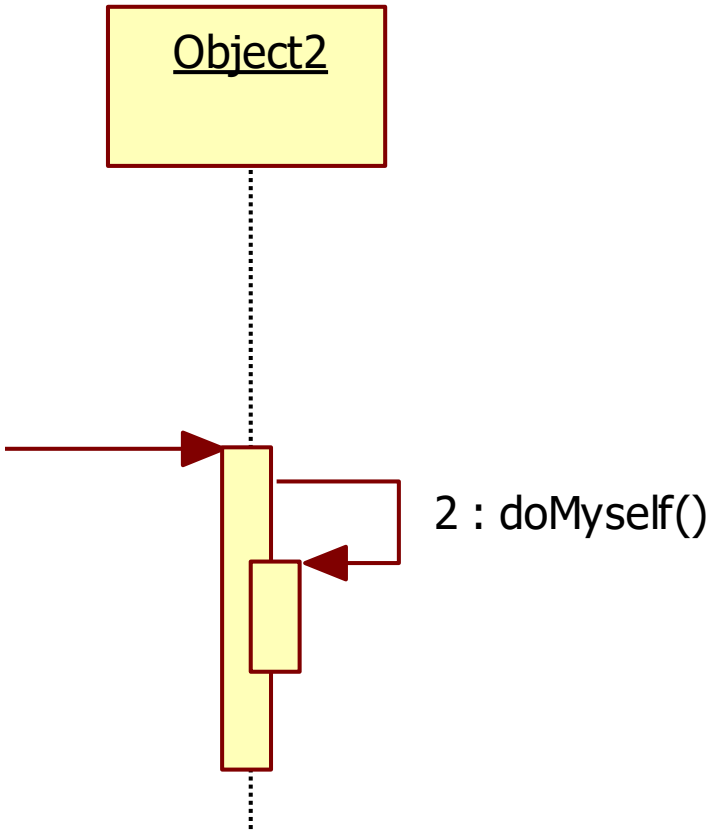
- Removes an object from memory
  - Full line, filled triangle end, cross on life line end, prototype « destroy »



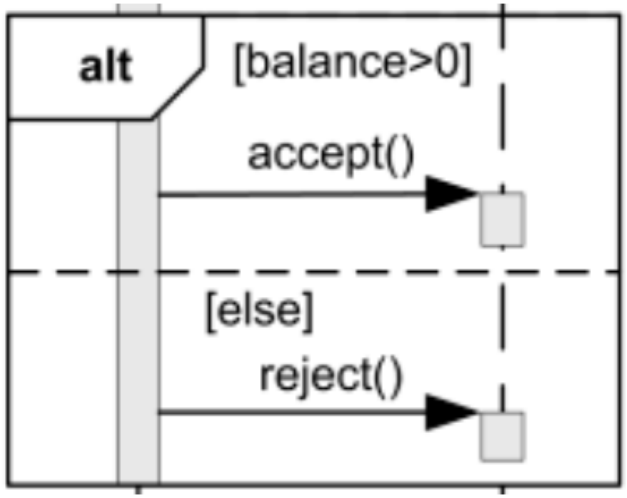
Other operators

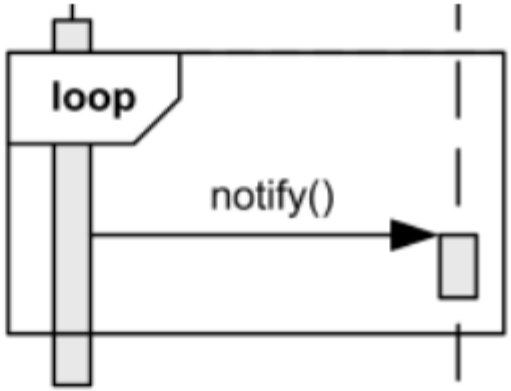
- Self messaging
- Choice and loop
  - alternative
  - option
  - loop
  - break
- parallelism
  - parallel
  - critical region
- Order of messages sending
  - strict sequencing
  - weak sequencing

Self messages

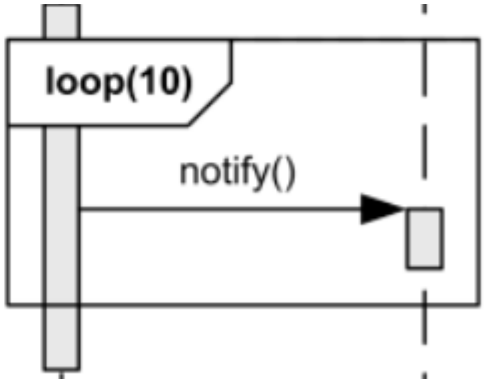


Choice and loops

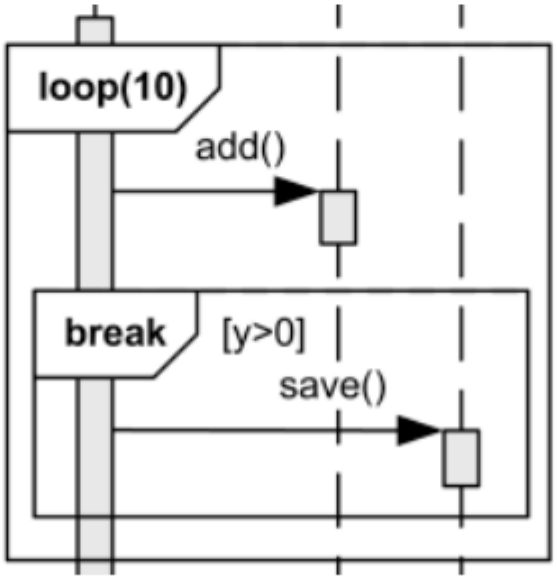




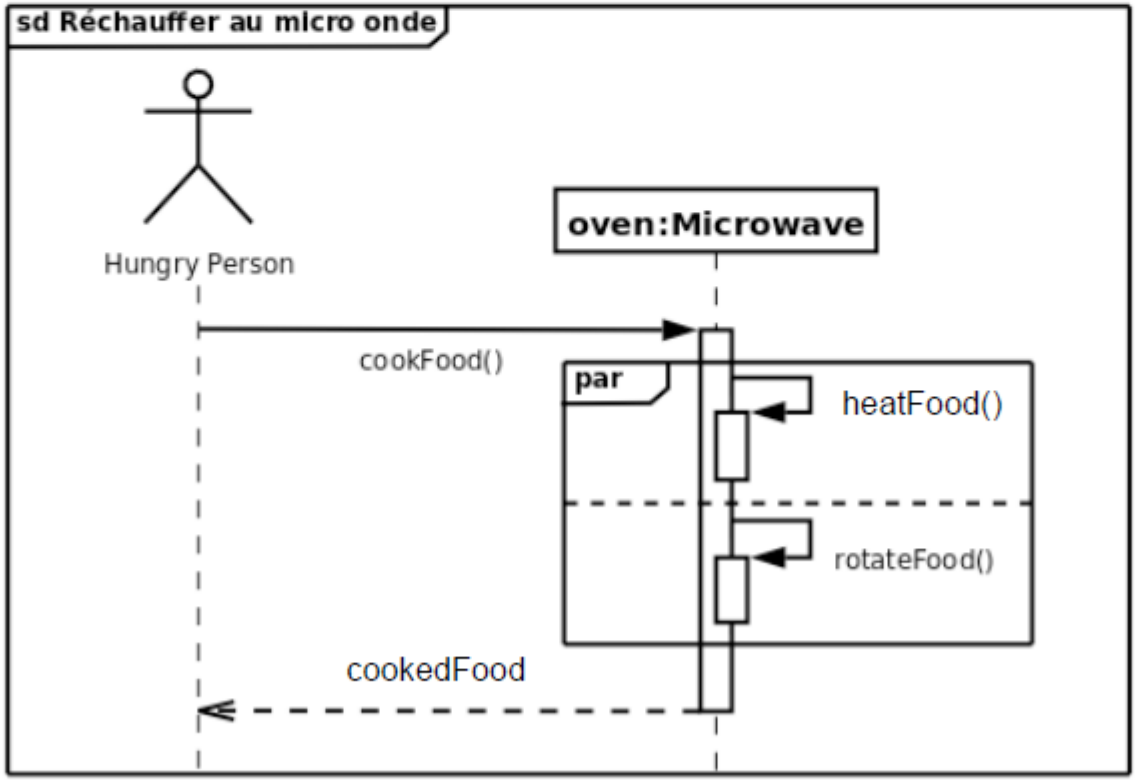
Possibly infinite loop

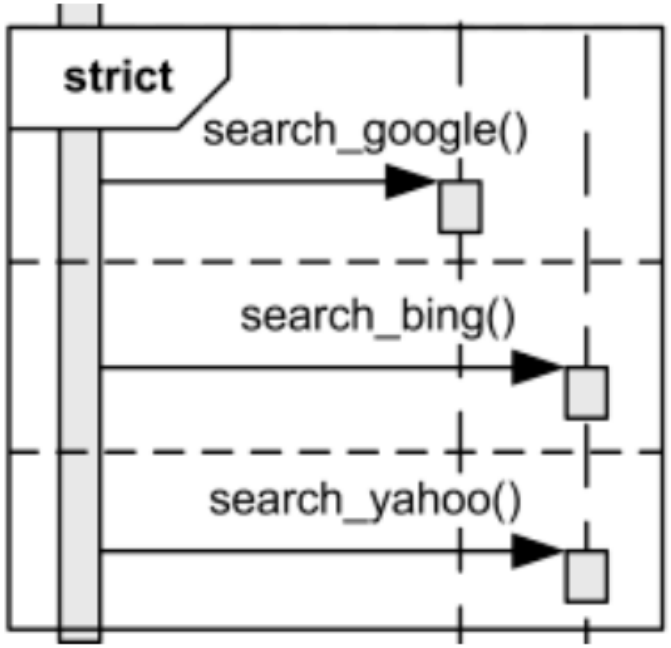


Loops 10 times

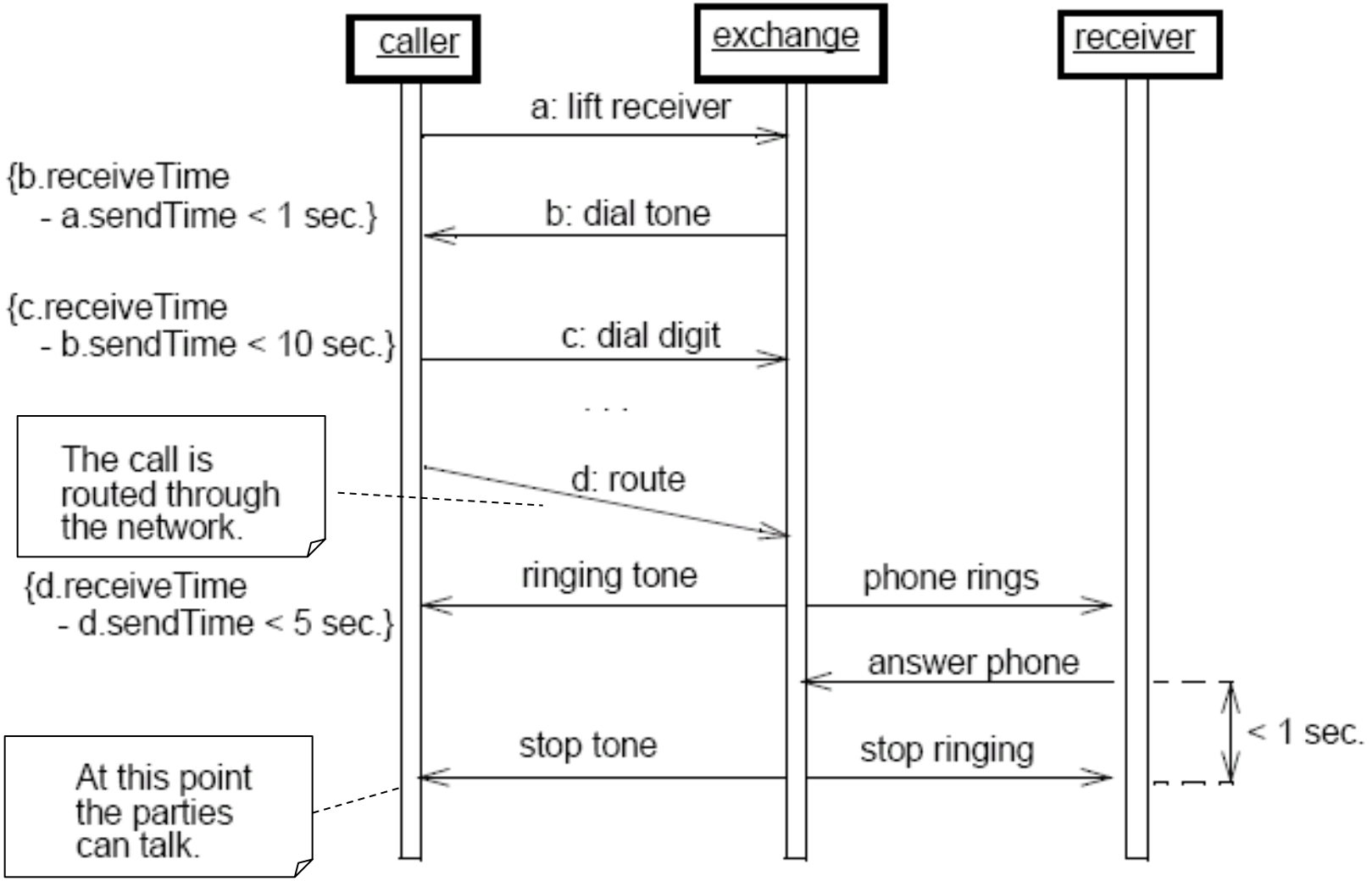


Parallel and strict sequence

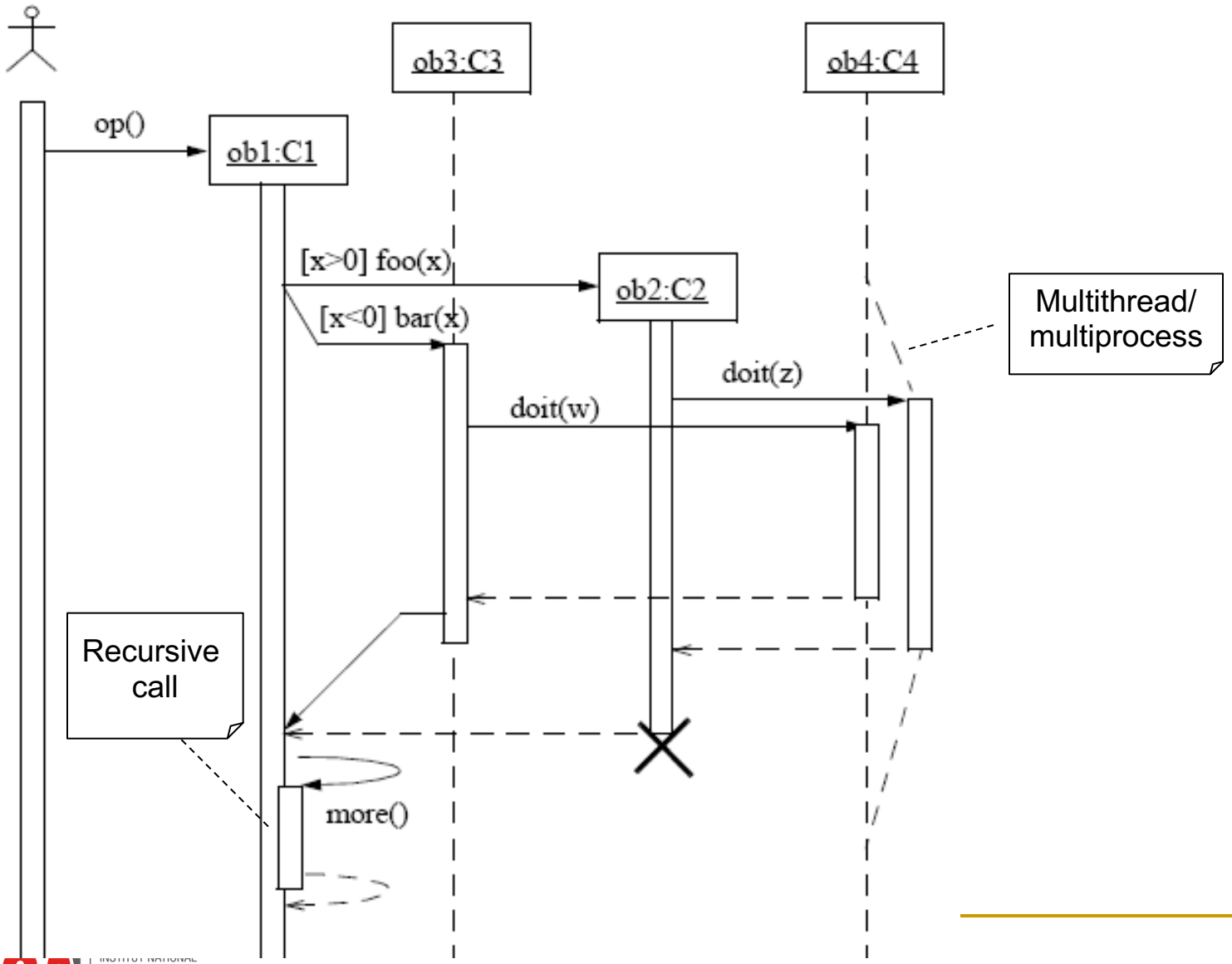




Other example with signals

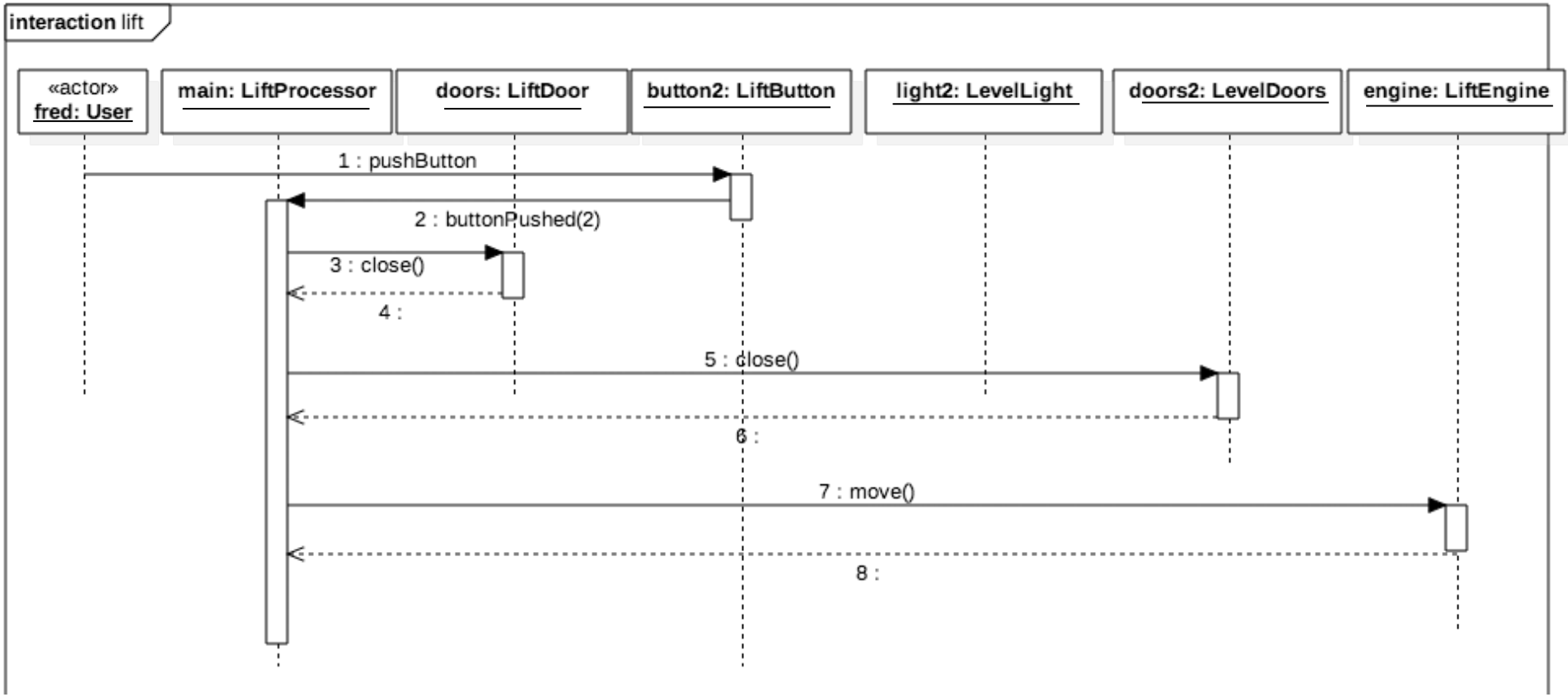


Other example again



From sequence diagram to code

- Example : method ButtonPushed in class LiftProcessor



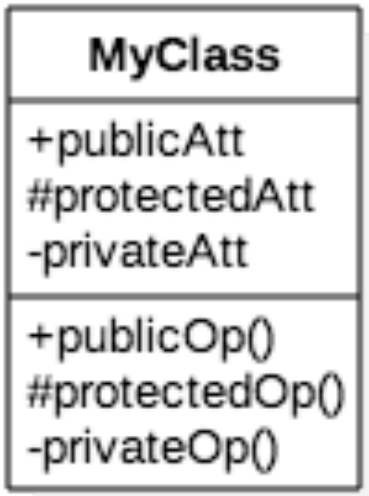
UML – Class diagram

- type Structure

- Elements in the class diagram
  - Classes and their properties, Relations between classes, Interfaces, Packages

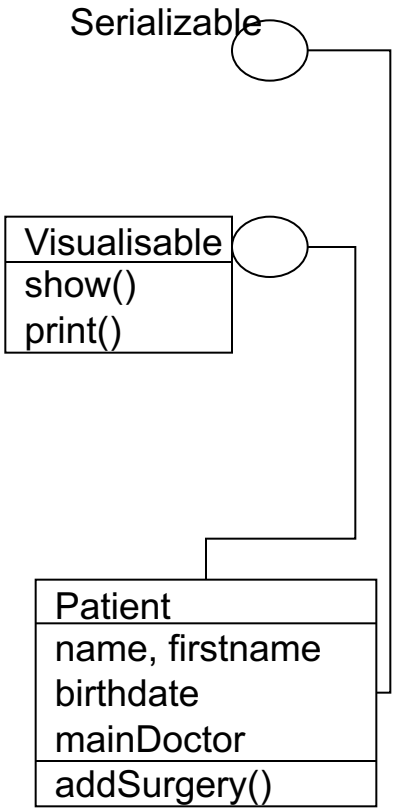
- Can be more or less detailed, depending on the project status
  - First diagrams are not detailed, and will evolve !!
  - Diagrams in the design reports of the project are very detailed

Visibility of classes members



- Public Member
  - Callable by any other object
  - Participates in the class description
- Private Member
  - Only objects of this class can use this member
  - Part of the implementation of the class
  - Useful to organize the class code
- Protected Member
  - Available only to the objects of classes in the same package or that extend this class

Interface



- Interfaces define ***standard signatures*** for methods
  - C++ equivalent : pure virtual classes
  - An interface has only methods signatures
  - An interface has NO attribute
- Usage : "contracts"
  - Classes that implement an interface "ensure" they conform to its methods declarations
- Examples
  - interface Stockable to manage persistance uniformly
    - Methods save(), restore()
  - interface Visualisable to manage presentation uniformly
    - Methods show(), print()
- A lot of interfaces are defined in Java libraries

Public class Patient implements Visualisable, Serializable{…}

Interface example: Iterator (Java 7)

Methods

Modifier and Type	Method and Description
`boolean`	**`hasNext()`** Returns `true` if the iteration has more elements.
`E`	**`next()`** Returns the next element in the iteration.
`void`	**`remove()`** Removes from the underlying collection the last element returned by this iterator (optional operation).

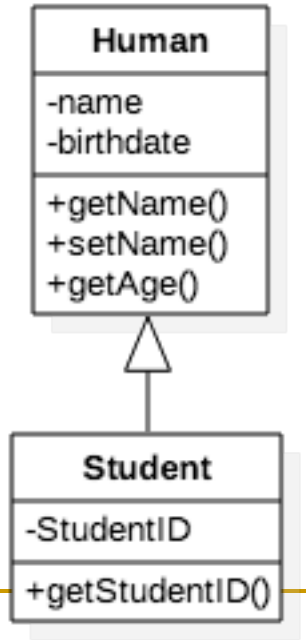
Other examples
Collection, Serializable, Component, EventListener, Key, Refreshable etc.

Relations between classes

- Generalization/specialization/inheritance
  - «is a kind of »
- Aggregation
  - « has »
- Other Associations
  - Any other kind of link between classes
  - Will be later transformed into composition, reference, classes…
- Aggregations and other associations can have cardinalities (# of elements)

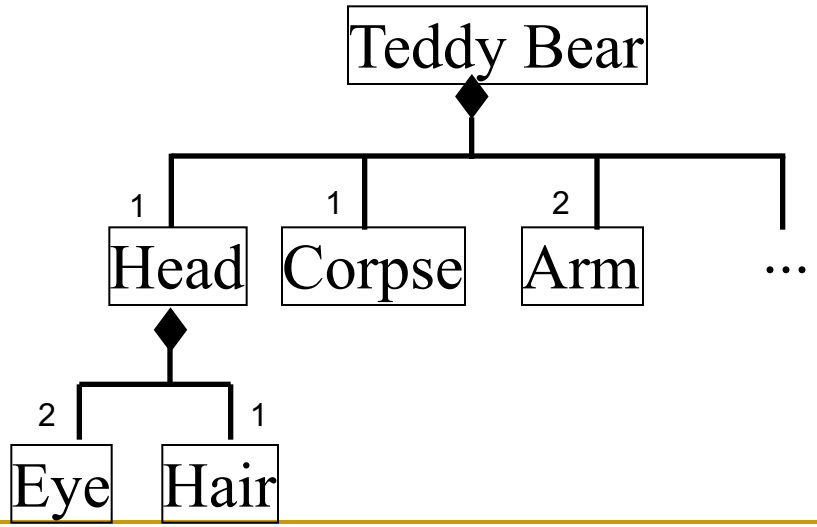
Generalization/specialization relation

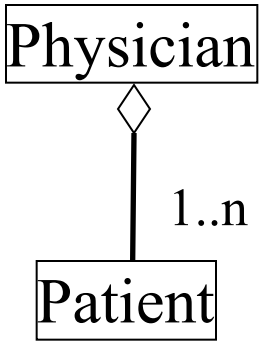
- When all objects of a class also belong to a more general class
  - Translation of « to be » (all Students are Humans)
  - The sub-class gets alls the members of the mother class plus its specific members
- Example and vocabulary:
  - Human *generalizes* Student
  - Student *specializes* Human
  - Student *is subclass of* Human



Aggregation relation

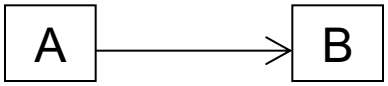
- When objects include other objects
  - Translation of « to have »relation
  - 2 kinds of aggregation relations:
    - *composition* : If I destroy the object, its components are also destroyed ♦
    - *reference* : If I destroy the object, the referenced objects are still alive ◊





Other relations between classes

- Navigable relations
  - Objects of class A will call methods on objects of class B



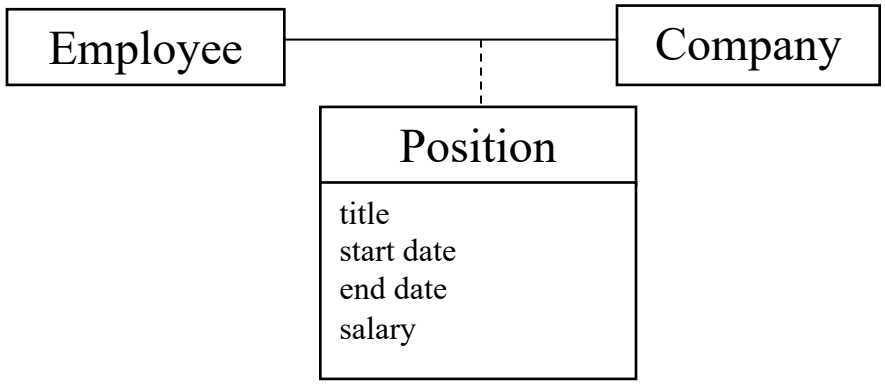
  - How is it translated in a programming language?

- We often use named binary relations

  - How is it translated in a programming language?

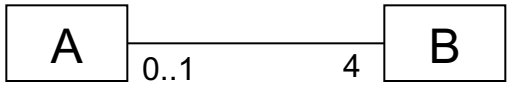
Complex relations as classes

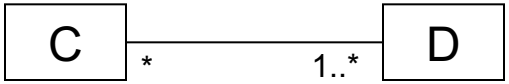
- More complex relations can be represented as classes

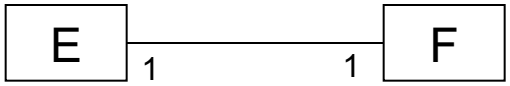


How is it translated in a programming language?

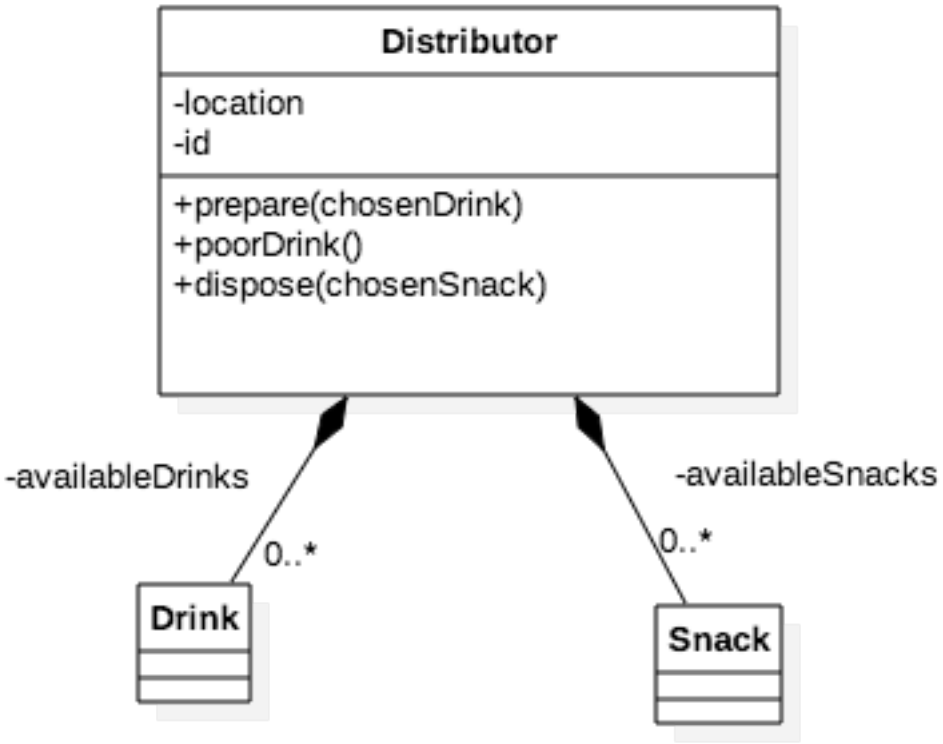
Cardinalities of relations







Class Diagram: Aggregation



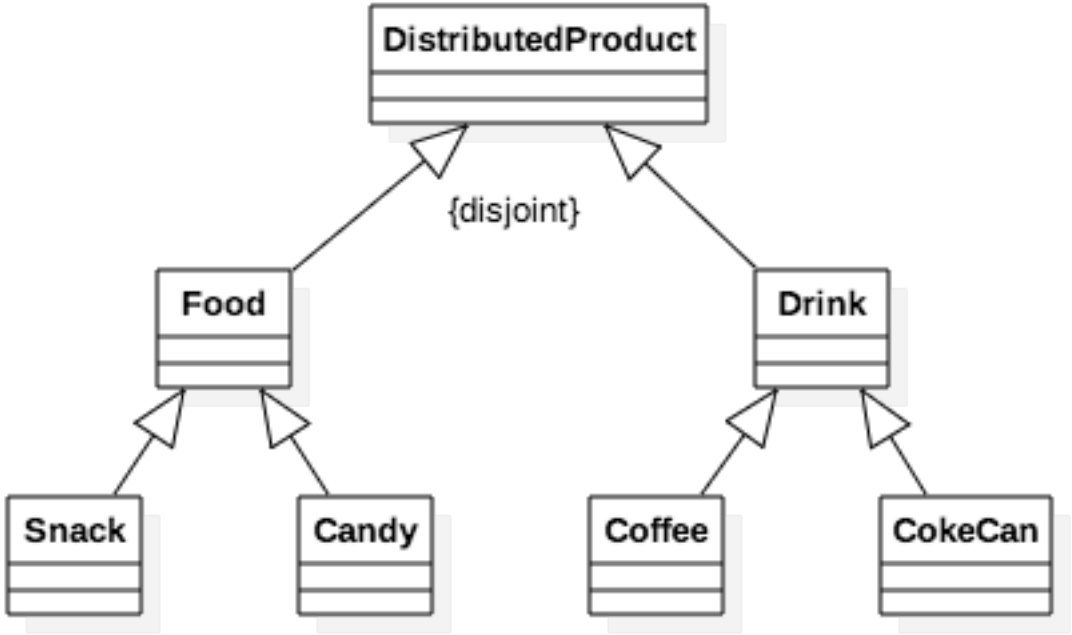
2 interchangeable versions

Distributor

-location
-id
-avalableDrinks: Collection<Drink>
-availableSnacks: Collection<Snack>

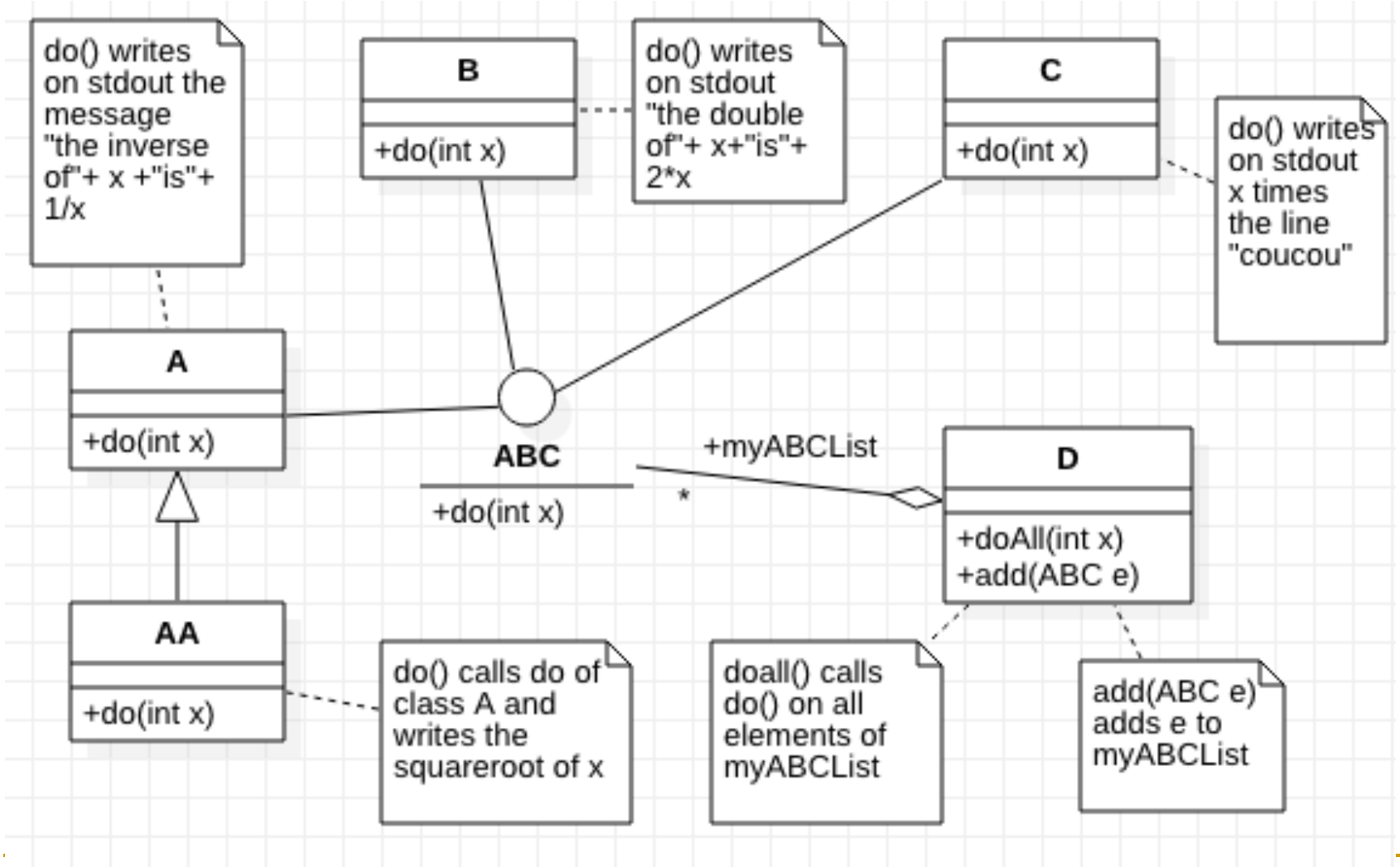
+prepare(chosenDrink)
+poorDrink()
+dispose(chosenSnack)

Class diagram : Generalization



Possible constraints : complete, incomplete, disjoint, overlapping

From class diagram to code

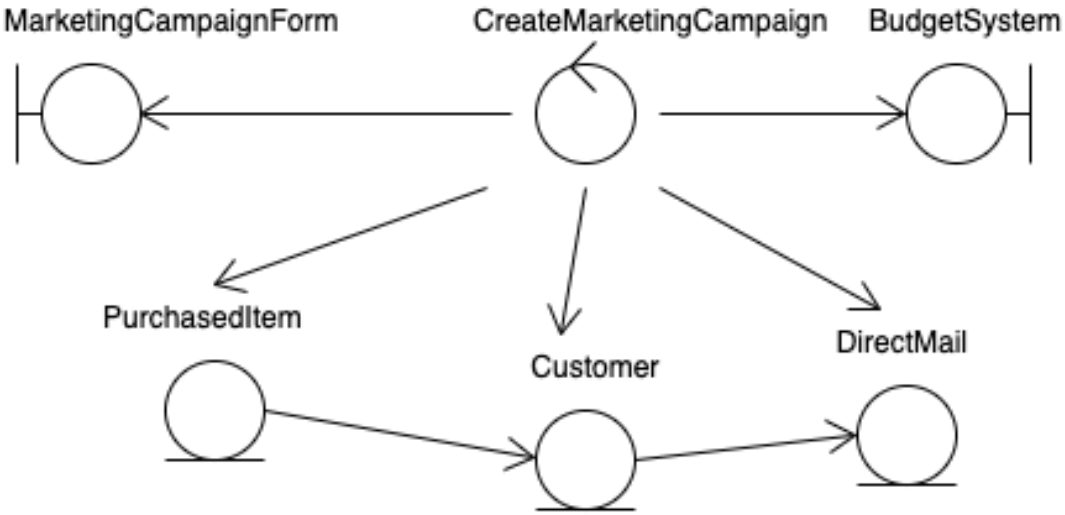


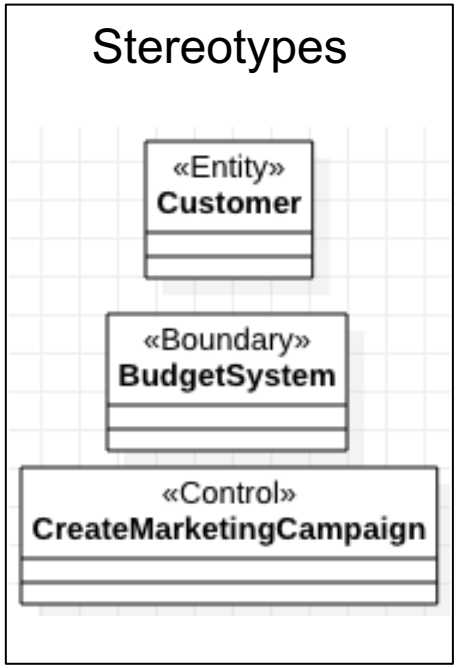
Different kinds of classes

- Classes can have different purposes in the application architecture, for example:
  - User interface
  - Data management
  - Functional services
  - Communication with other applications
  - Etc.

Classes : specific icons or stereotypes

- Some specific icons allow to better identify the role of classes in the application architecture.
- A graphical alternative is to use stereotypes
  - Stereotypes can be freely defined

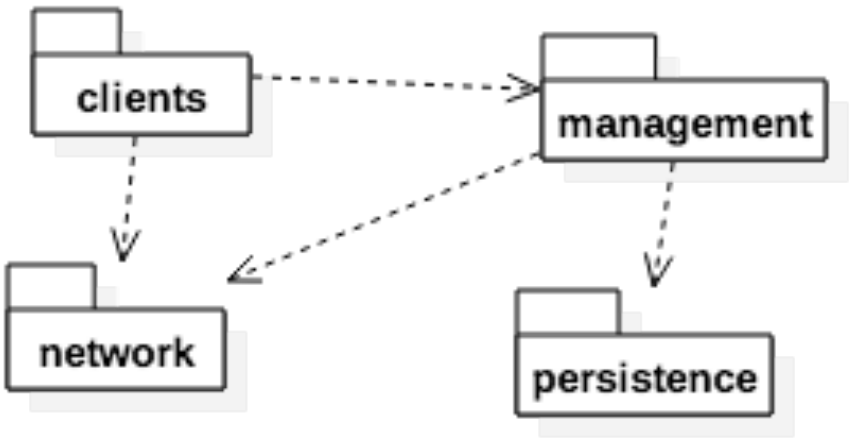




http://www.utm.mx/~caff/doc/OpenUPWeb/openup/guidances/guidelines/entity_control_boundary_pattern_C4047897.html

Package Diagram

- Package = logical container that groups and organizes elements
  - Clarity and organisation of code, task sharing
  - e.g. javax.swing, java.io
- Packages diagram= shows dependencies between packages

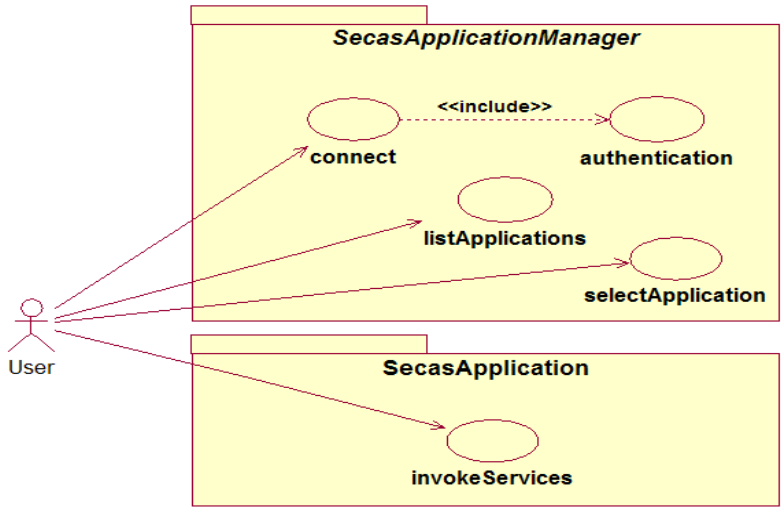


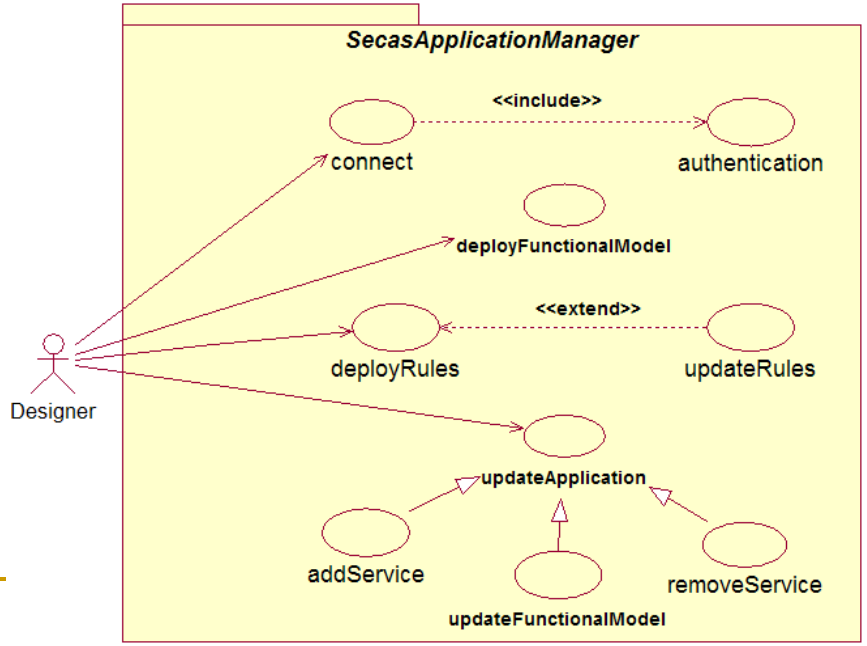
SECAS, an example

- SECAS : a context-aware system

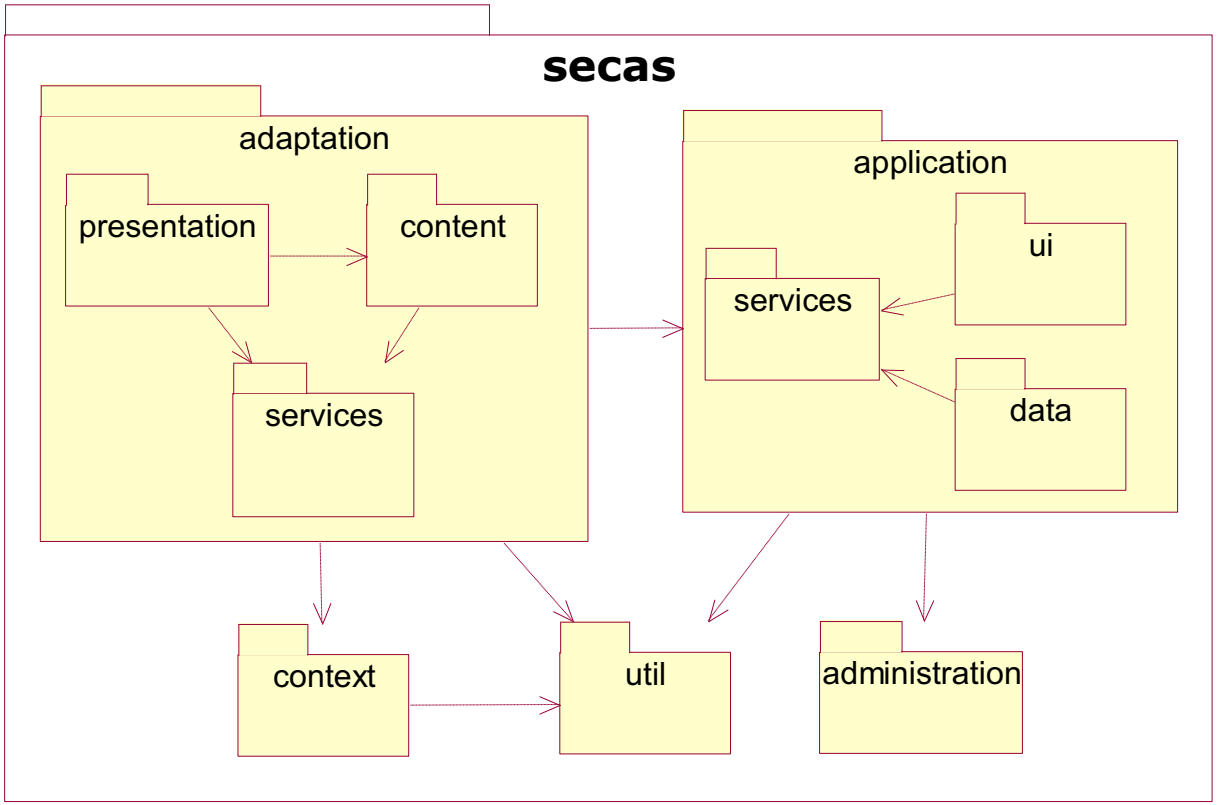
  http://theses.insa-lyon.fr/publication/2007ISAL0058/these.pdf

  - Adaptation to the end-user context
    - User interfaces, data, services

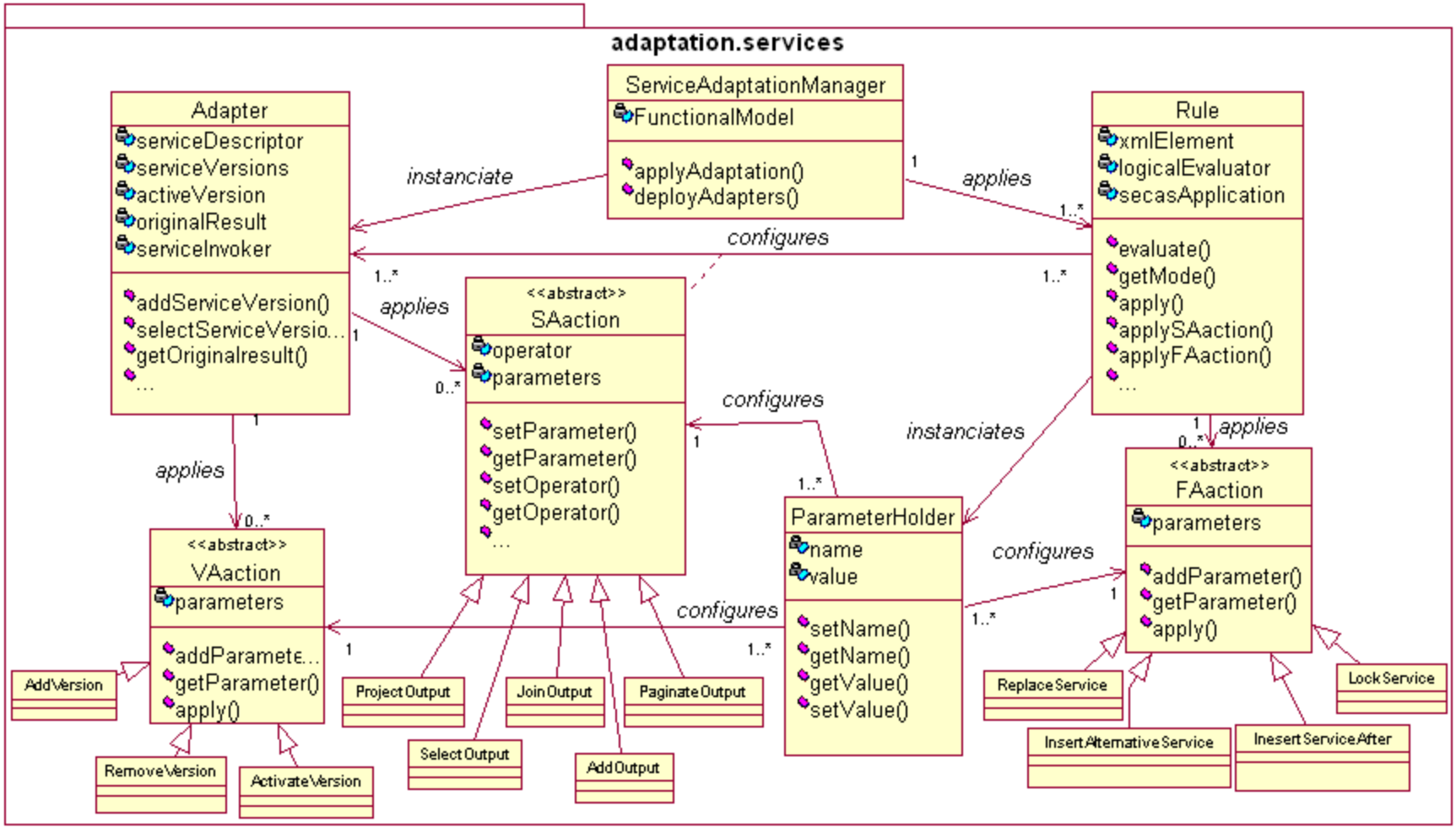




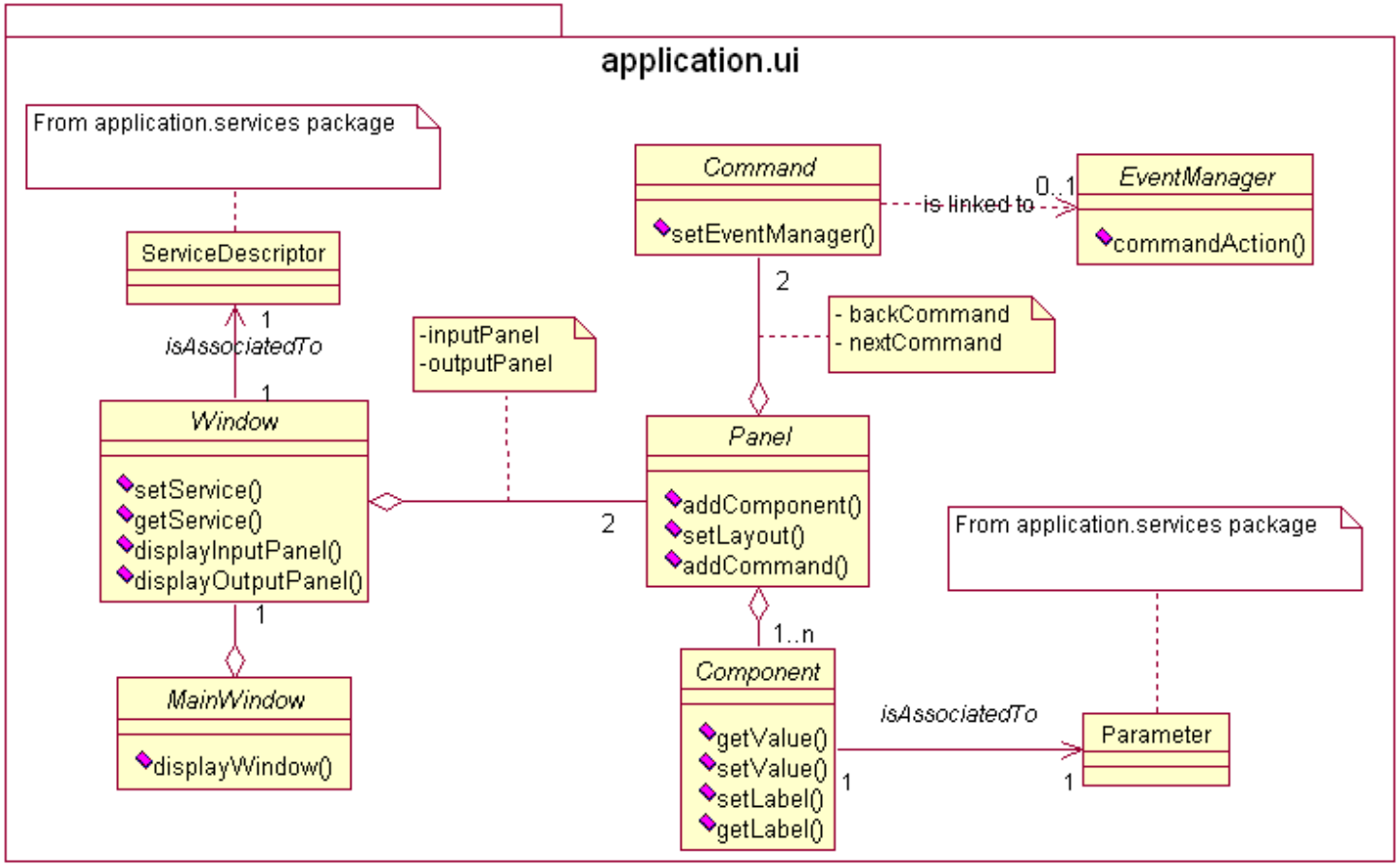
SECAS Package diagram



SECAS class diagram of adaptation.services package



SECAS class diagram of application.ui package



UML – Object diagram

type Structure

- Represents some classes instances (objects) at some instant t
- Mainly used to show example in complex cases

Object

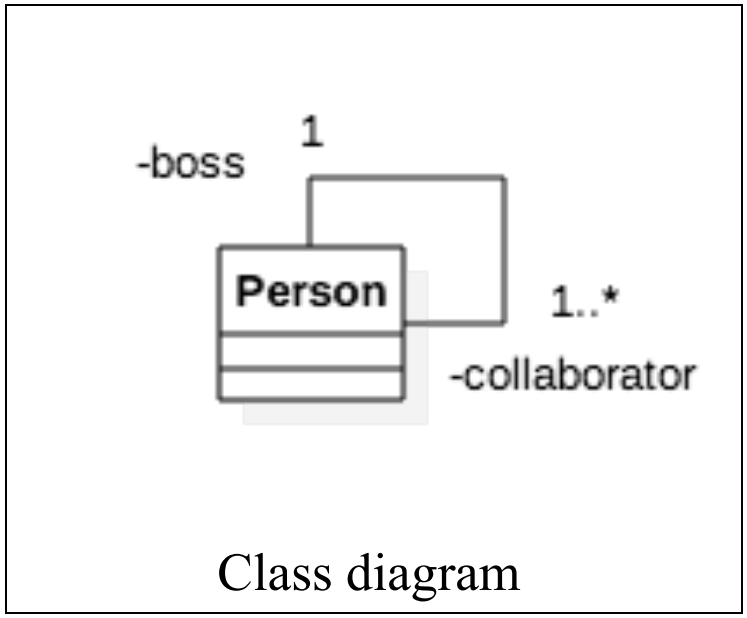
teachersPrinter:Printer

- An object is a class instance. It is defined by:
  - Identifier(oid)
    - System internals
  - State = attributes values
    - The class gives the list of attributes
    - The object has personal values for each attribute.
    - Values may change with time
  - Behavior = set of methods applicable on the object (defined in the class)
    - Methods often use attributes values

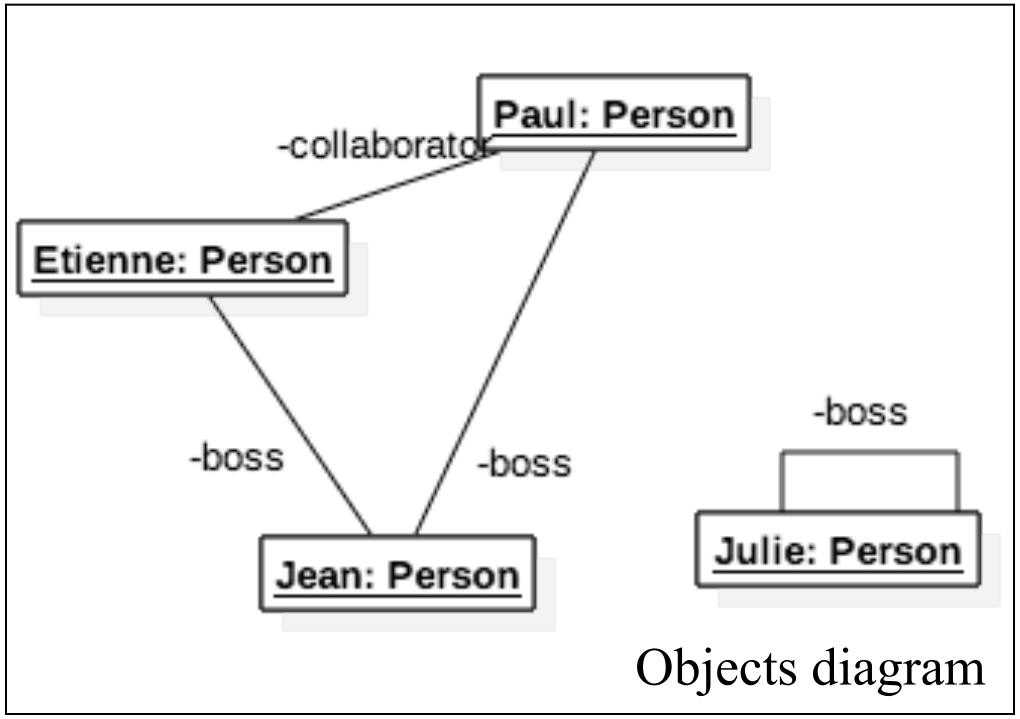
object teacherPrinter instance of class Printer
State: url=xxx.xxx.xxx.xxx; pilot=aficiov2.1;
status=idle;
printingOptions={bothSides, noStaple}
Behavior: start(), stop(), print(), alert()

UML : Objects diagram

- Looks like a class diagram BUT objects are underlined
  - Shows memory state at instant t



Class diagram



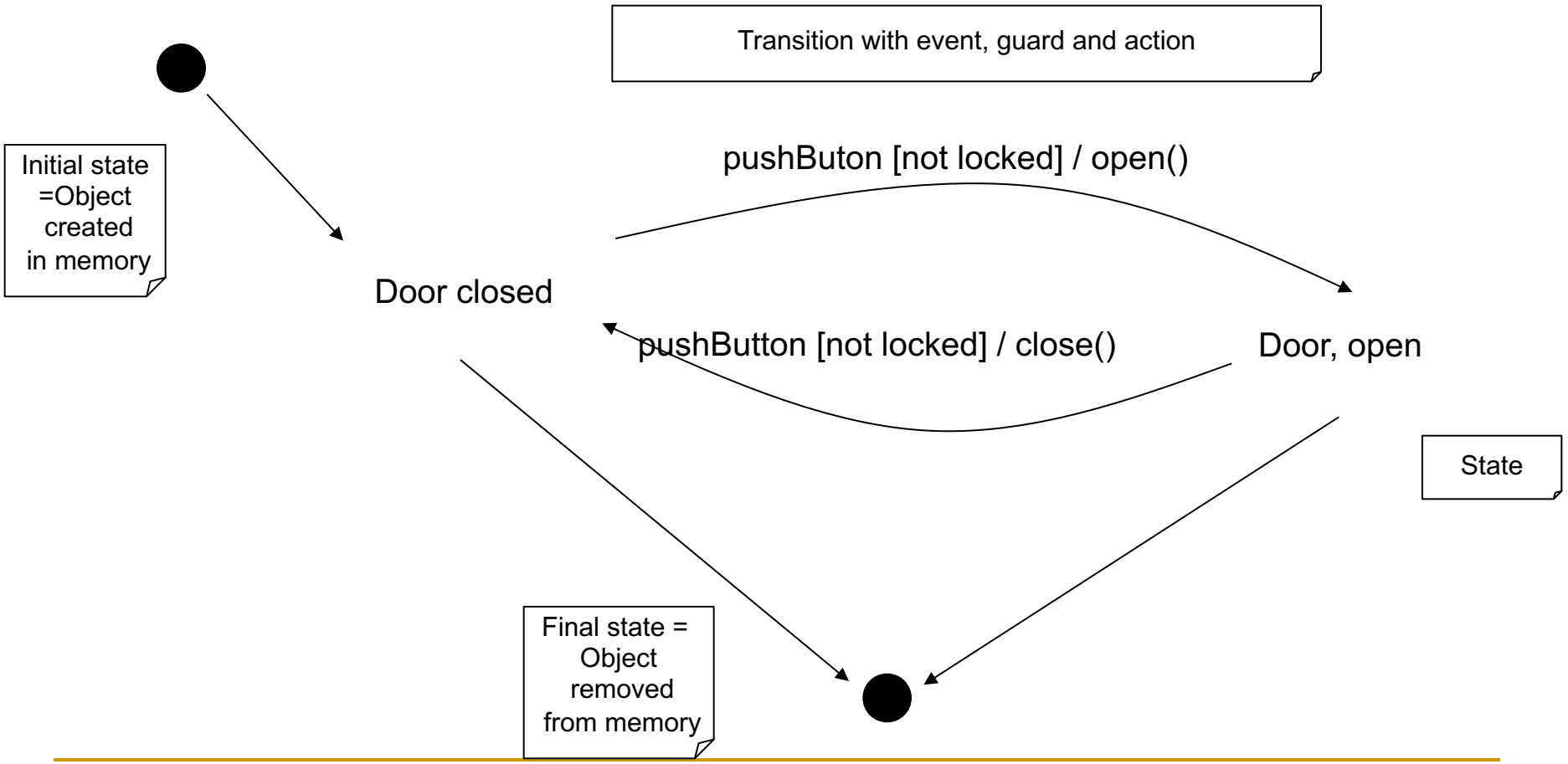
Objects diagram

UML – Statechart Diagram

type behavior

- Describes with a state machine the different states of ONE object, and the possible transitions from one state to the other
- Items in the diagram
  - States
  - Transitions fired by external events
  - Actions made by the object
  - Guards(required conditions)

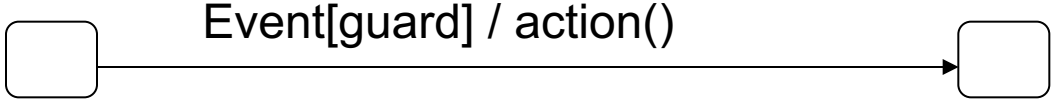
UML : Statechart Diagram example for objects of class Door



State

- A state
  - Is stable and may last
  - Is characterised by stable attributes values in the object
  - Example : state « door opened » for a door object, state « washing » for a dishwasher object

Transition



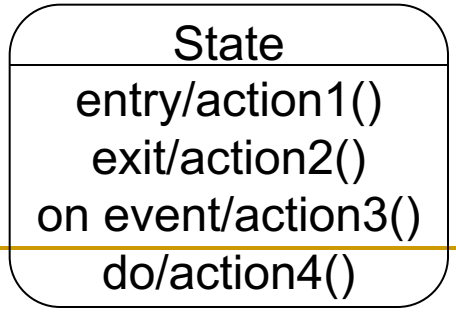
- A transition
  - Represents the *instantaneous* change of state
  - Fired by an external event arriving
  - Can be conditioned by guards
    - Boolean expression written in natural language and with brackets
  - Can launch an action
    - method made by the object

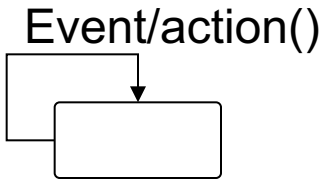
Action

- Corresponds to a methods the object can execute
- Can be executed with a transition

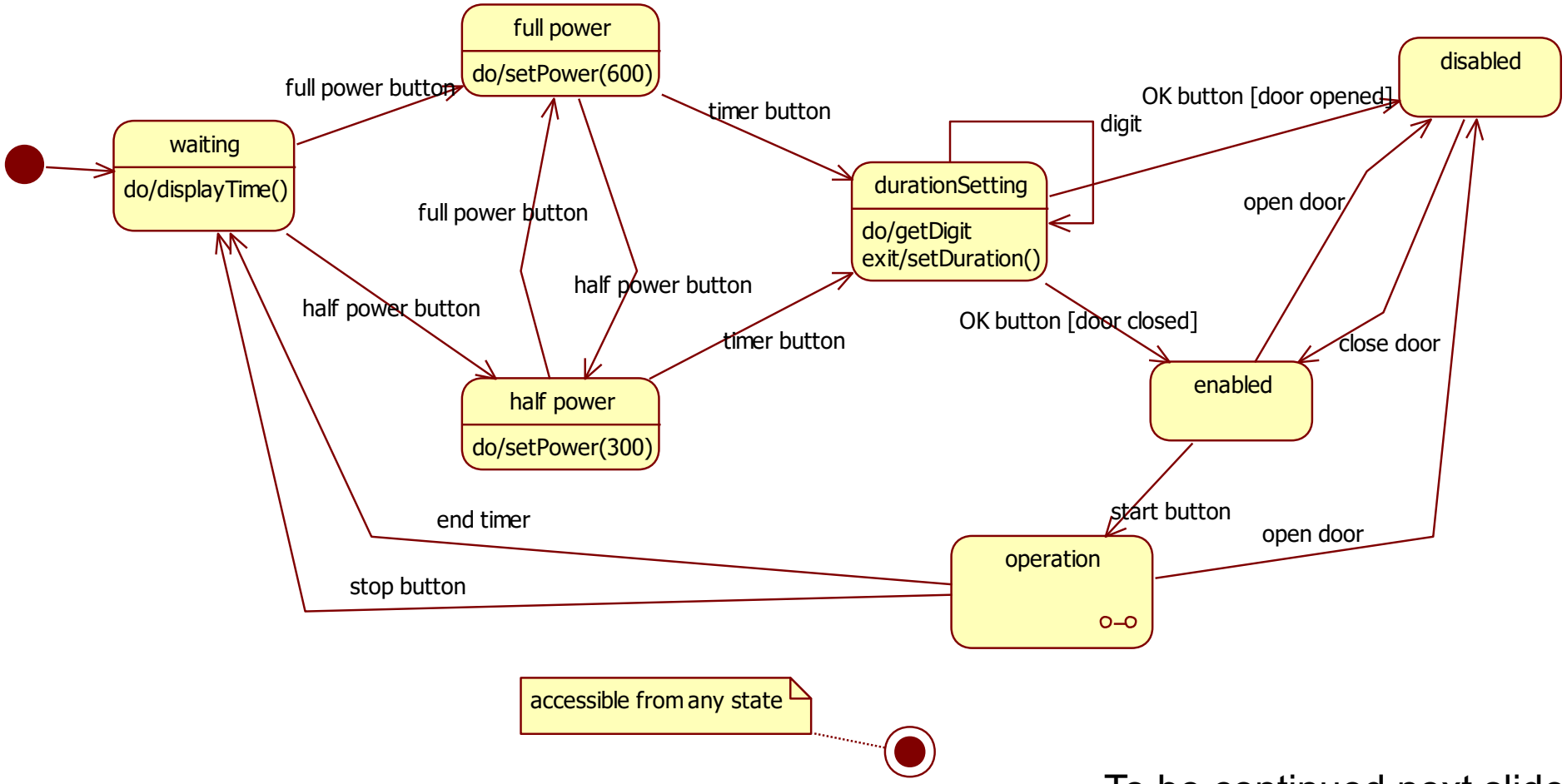
Event [guard] / action()

- Can be executed in a state
  - Types of actions
    - entry / *action* : executed when entering the state
    - exit / *action* : executed when leaving the state
    - on *event* / *action* : executed each time the event happens
    - do / *action* : action executed as long as remaining in this state



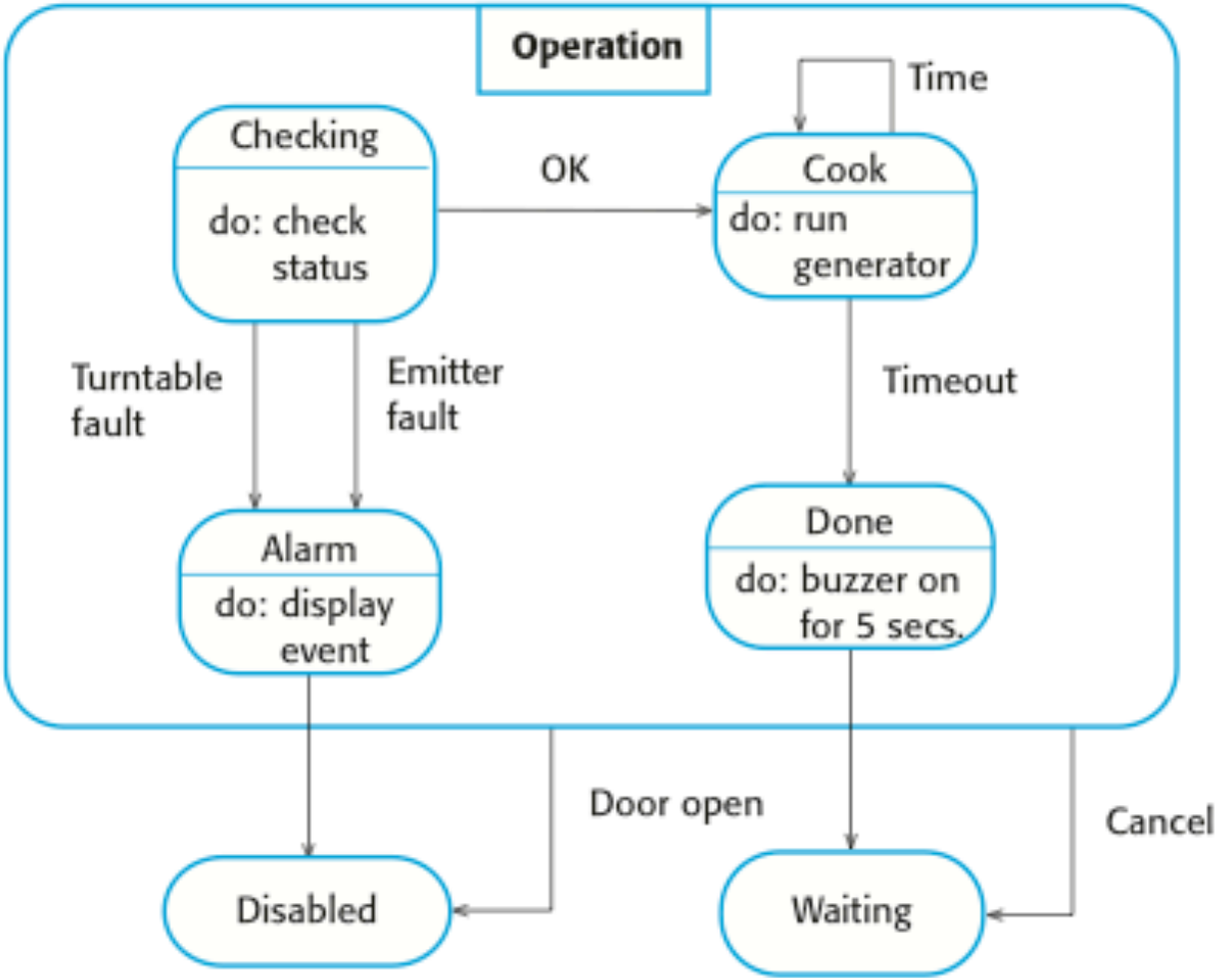


Statechart diagram for microWave Oven



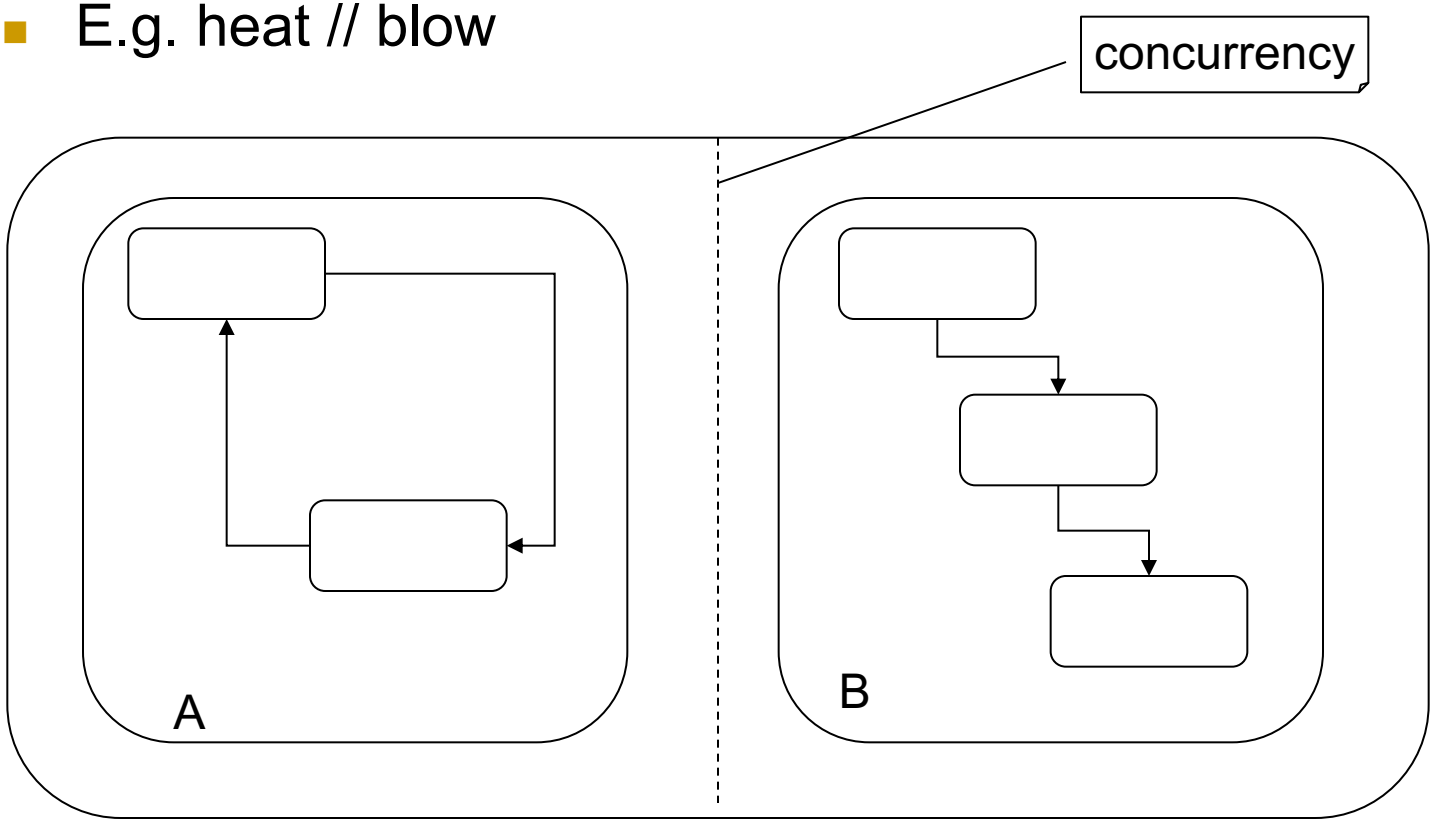
To be continued next slide

Super-State / Composite State



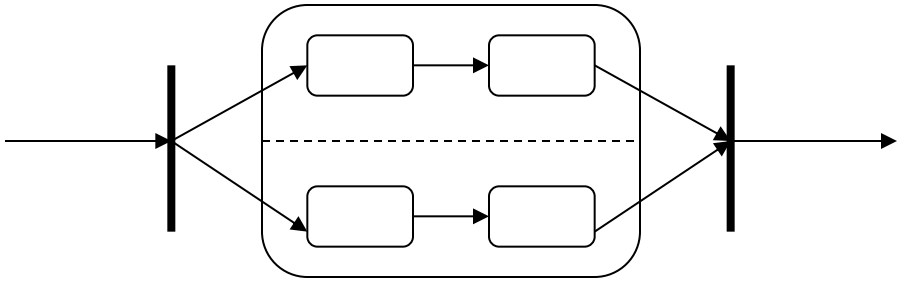
Concurrent States

- When an object can do two things at a time
  - E.g. heat // blow

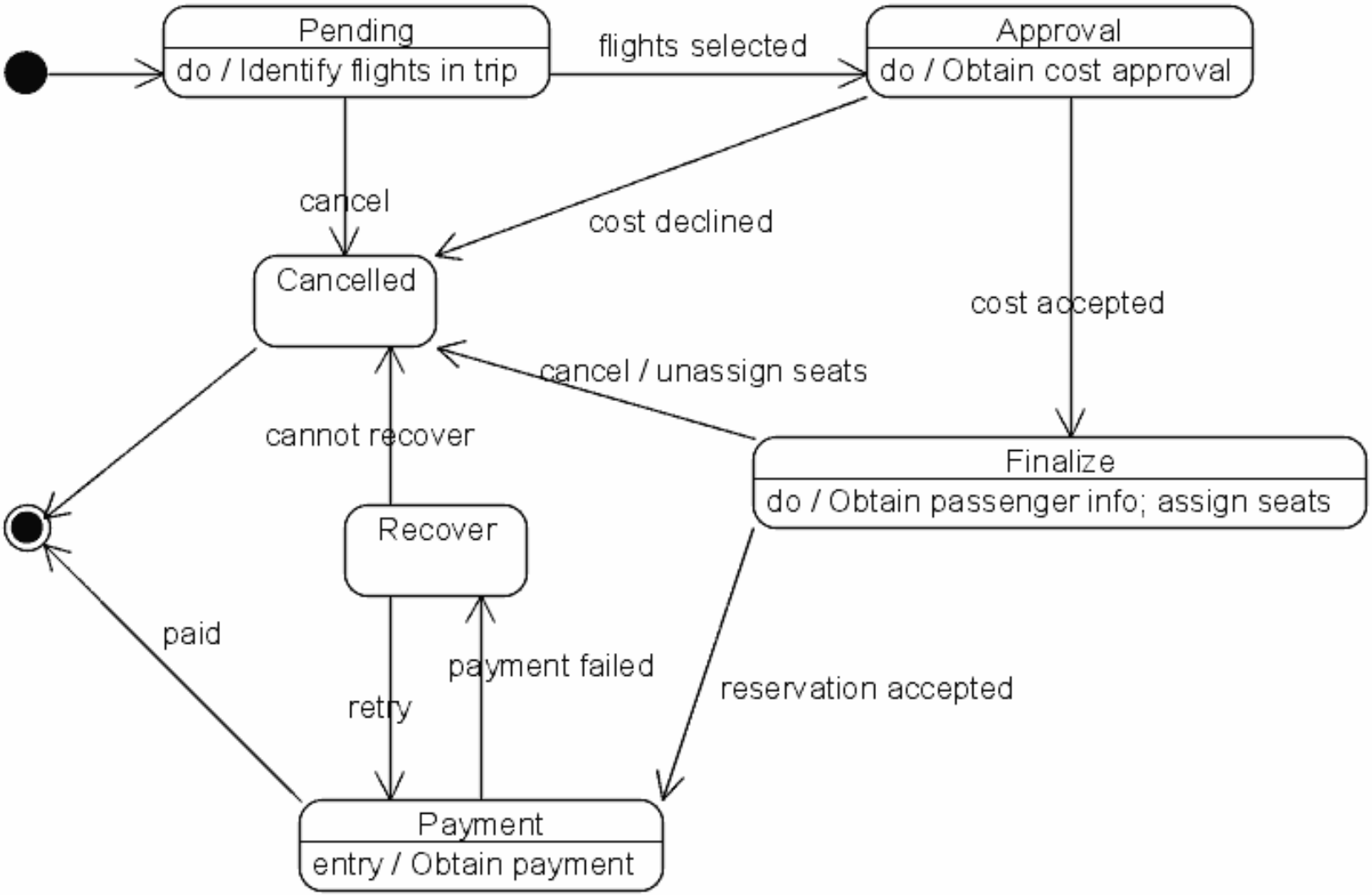


Synchronisation of states

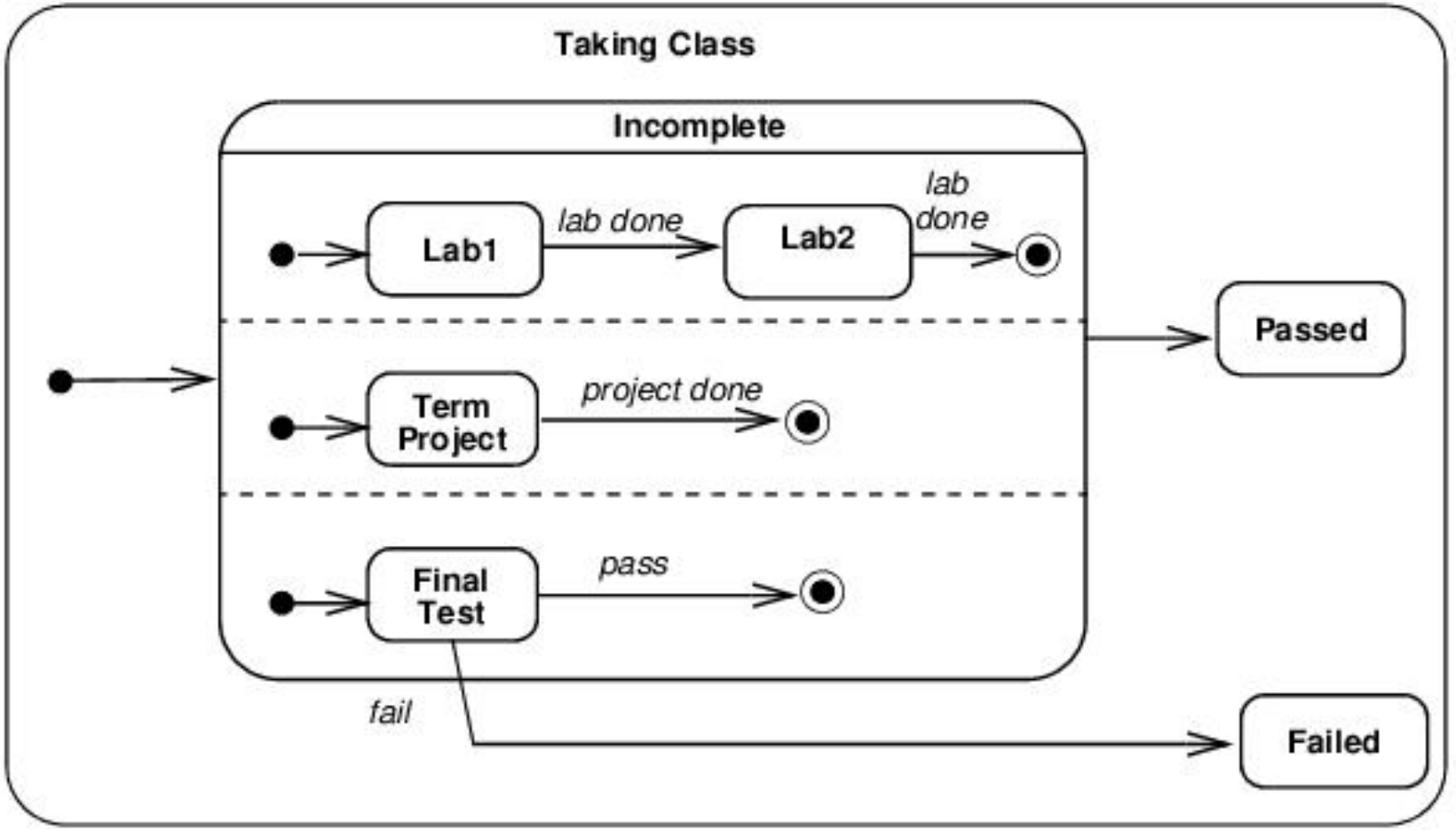
- The sync line is crossed when all required states are left (all arriving transitions are launched)
- Transitions that leave a sync line are automatic



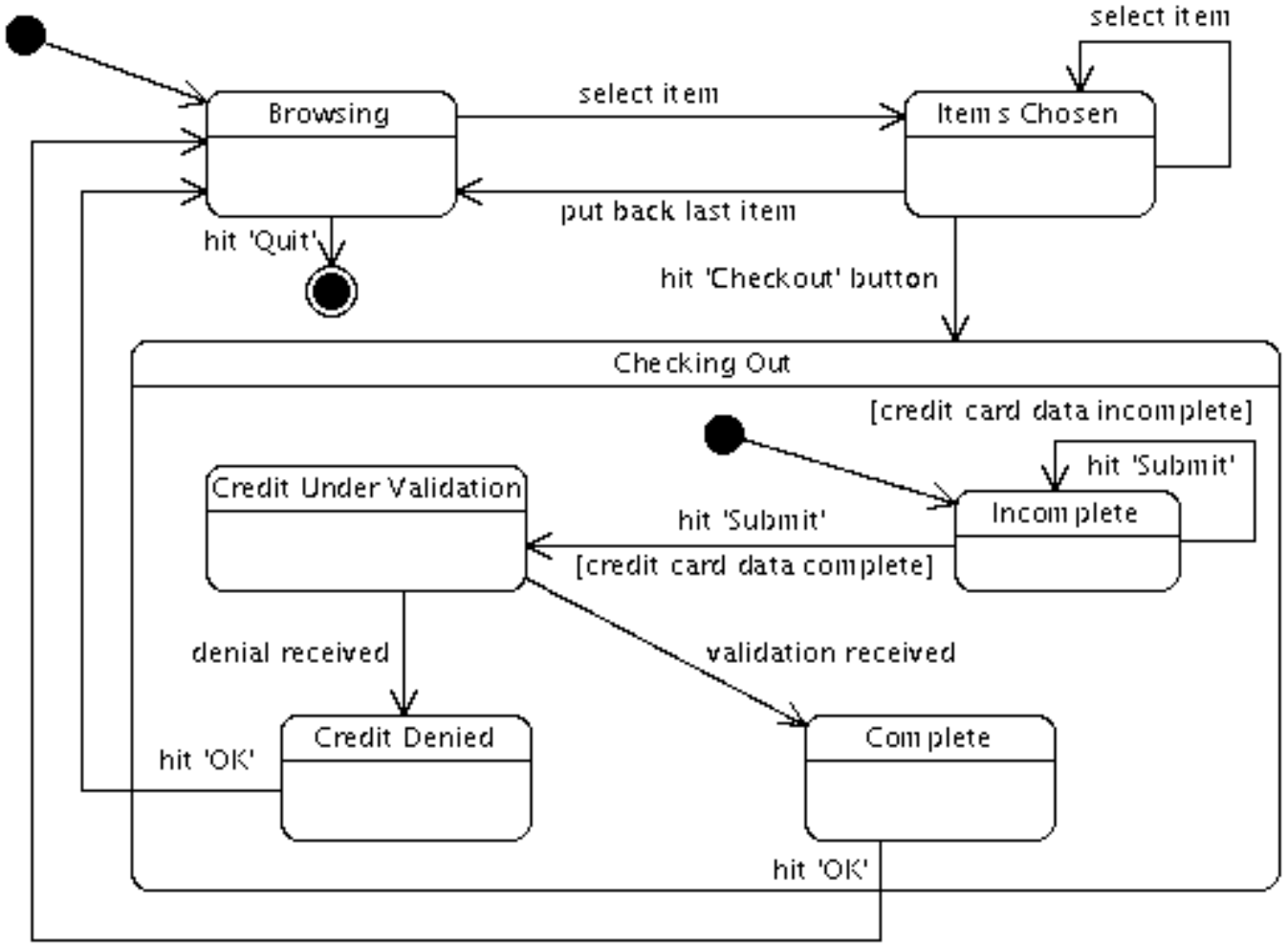
Statechart diagram for flight booking



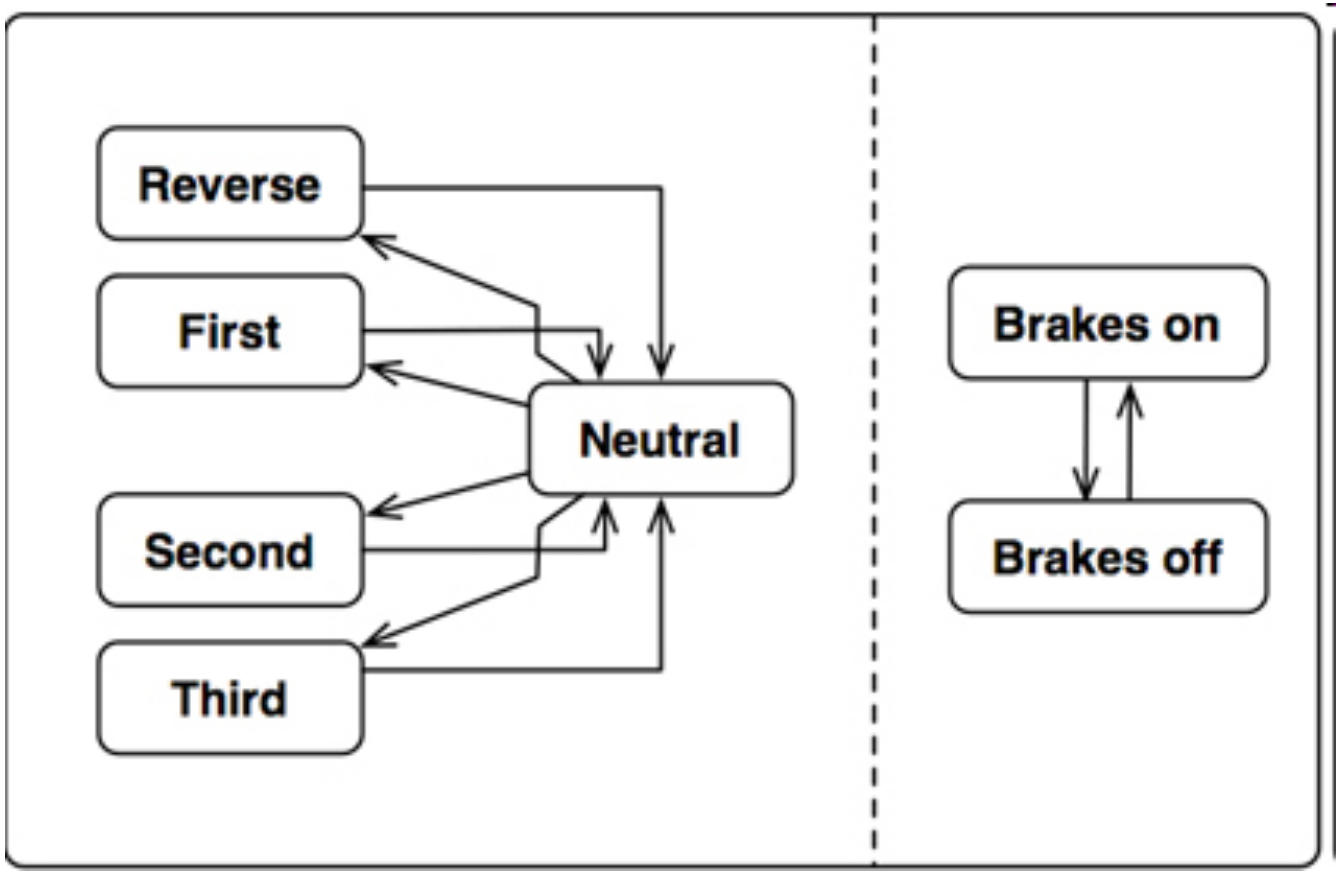
Statechart diagram for course validation



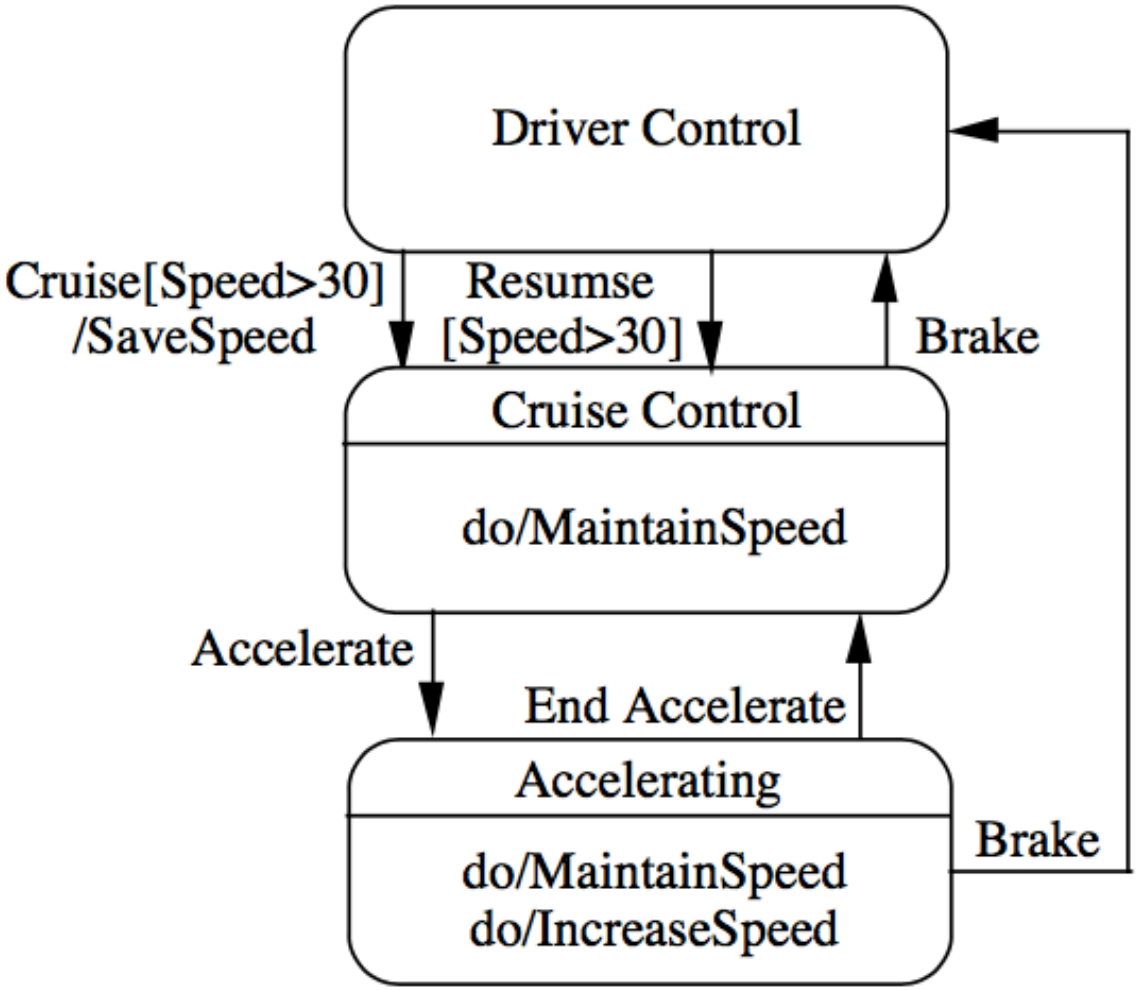
Statechart diagram for shopping cart



Part of statechart for car gears



Statechart for cruising



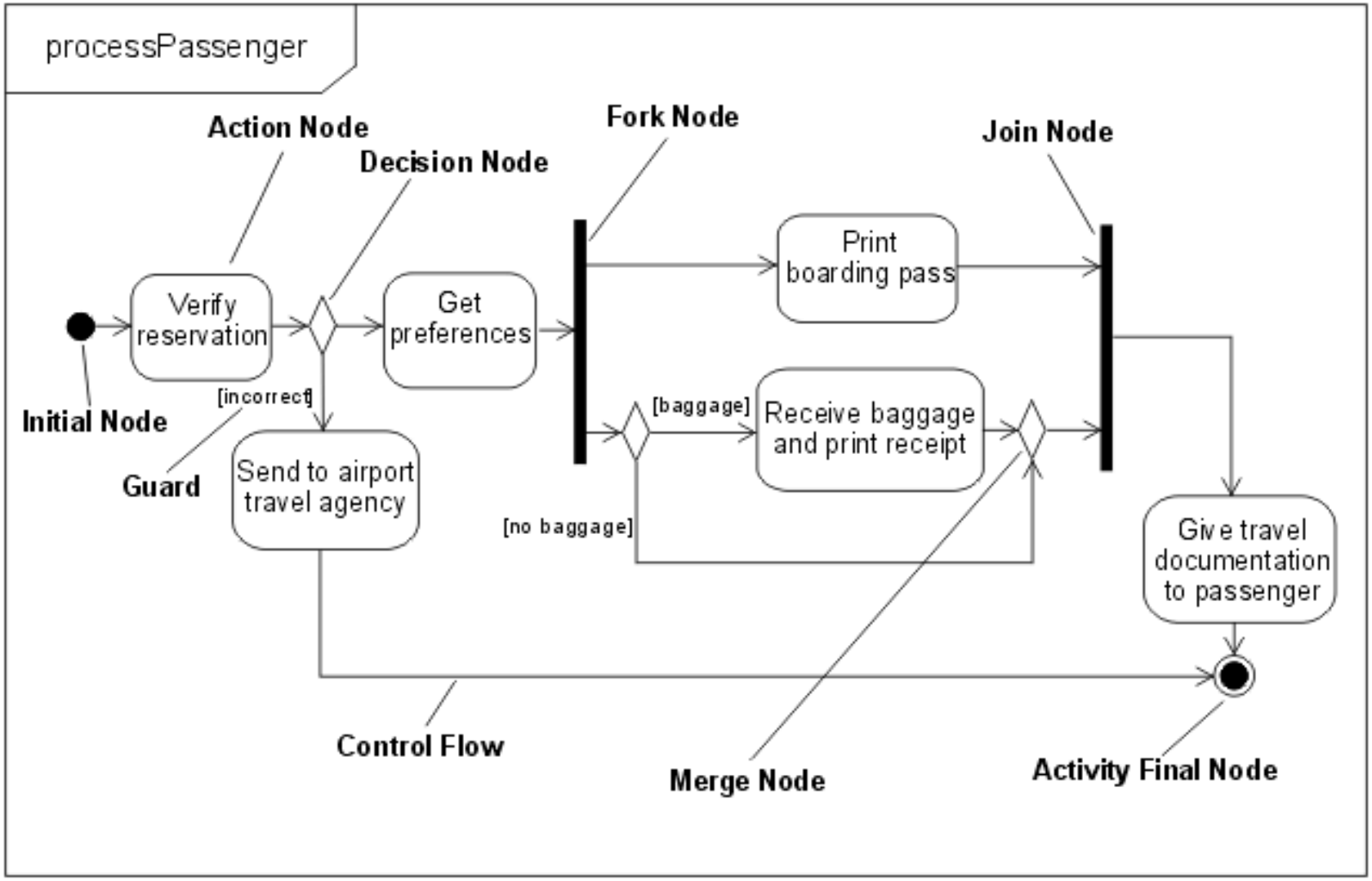
UML – Activity diagram

type Behavior

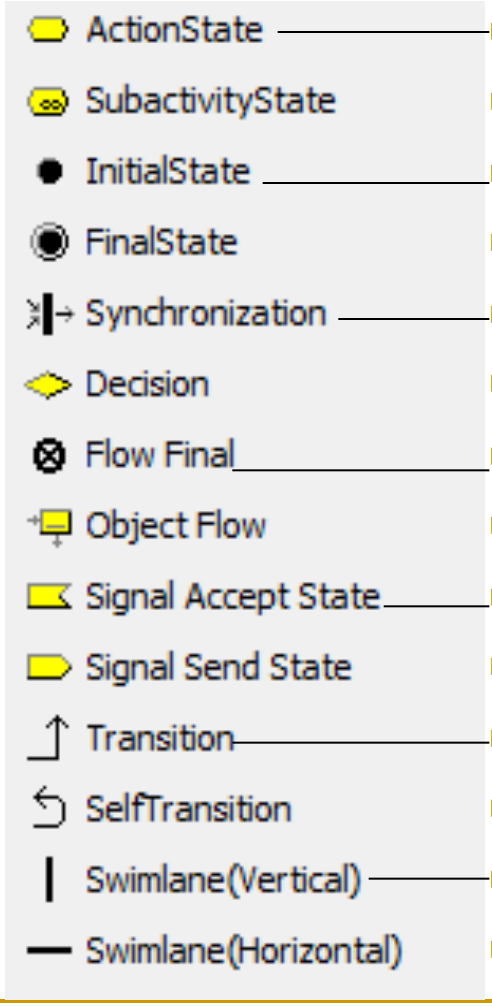
- Describes the behavior of a system or some components under the form of a stream/sequence of activities

Example

http://home.iscte-iul.pt/~hro/RUPSmallProjects/core.base_rup/guidances/supportingmaterials/differences_between_uml_1_x_and_uml_2_0_CA70F2E6.html



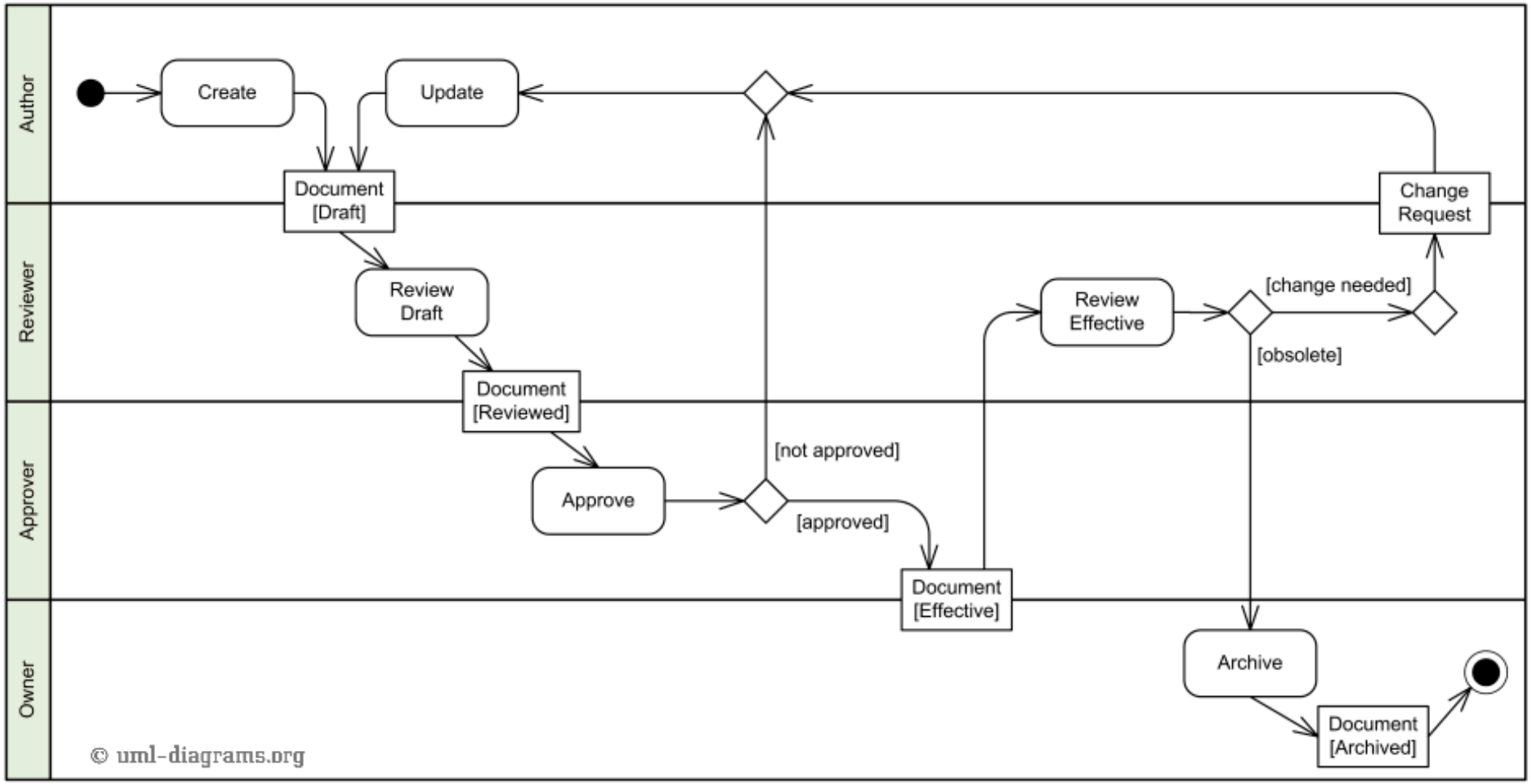
Main elements in an activity diagram



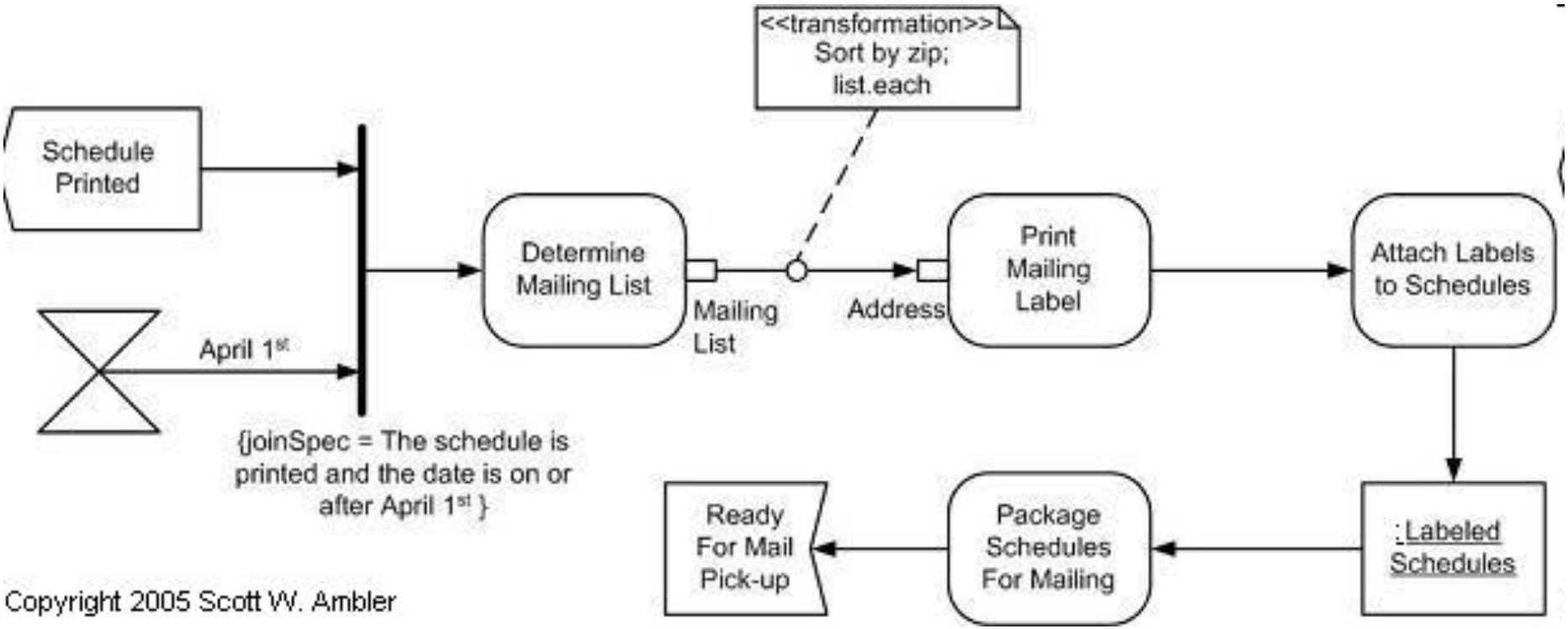
- Processing unit
- Composed activity
- Starting point (only one)
- End point(may be many)
- Synchronizes transitions
- The transitions that follow depend on conditions
- End of data processing, data are not used anymore
- Objectthat describes produced data
- Reception of a signal from outside of the system
- Send a signal
- From an action to the other
- loop
- Draws a border between actors
- Draws a border between actors

Partitions (swimlanes)

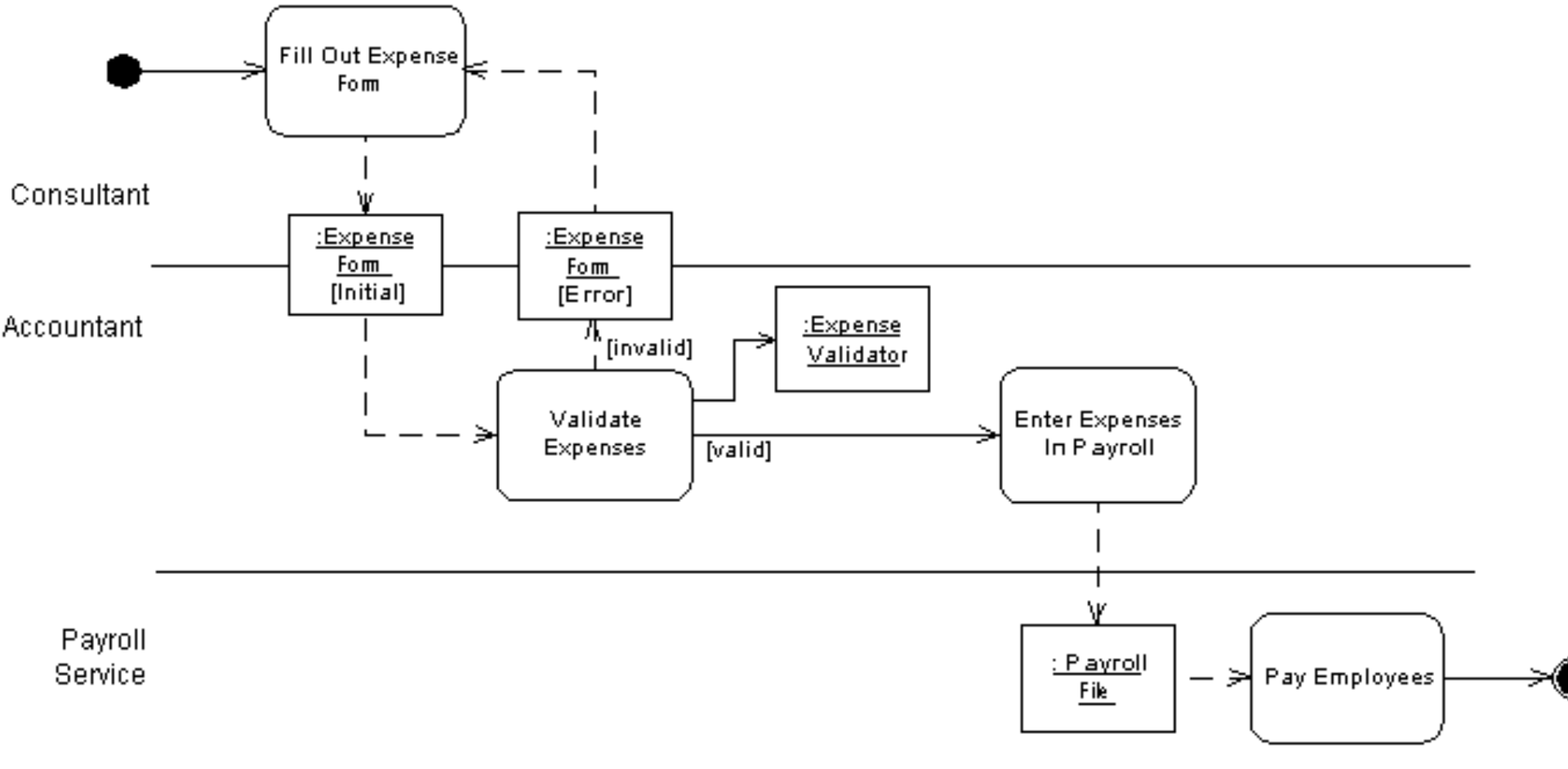
http://www.uml-diagrams.org/document-management-uml-activity-diagram-example.html



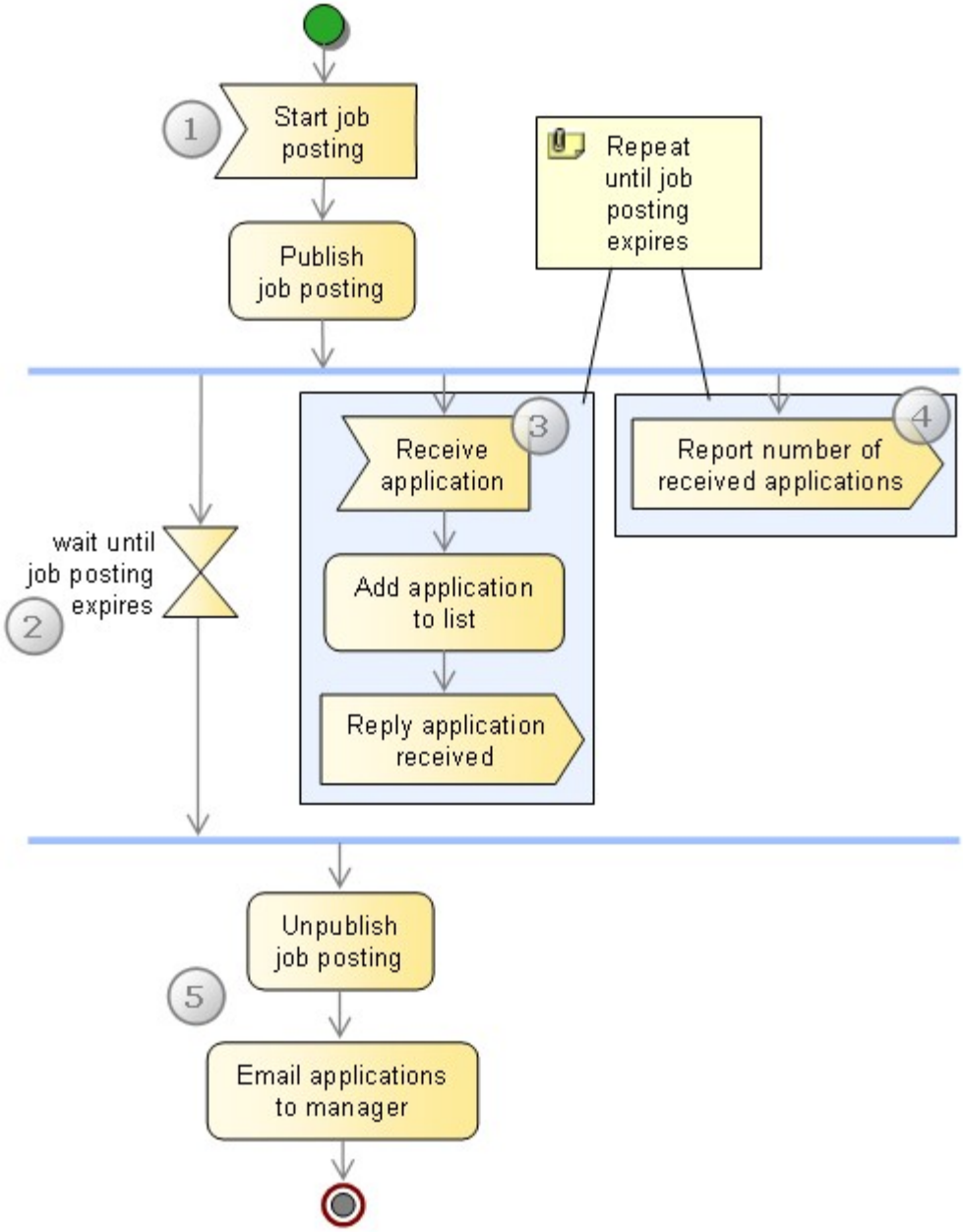
Activity Diagram and data transformation



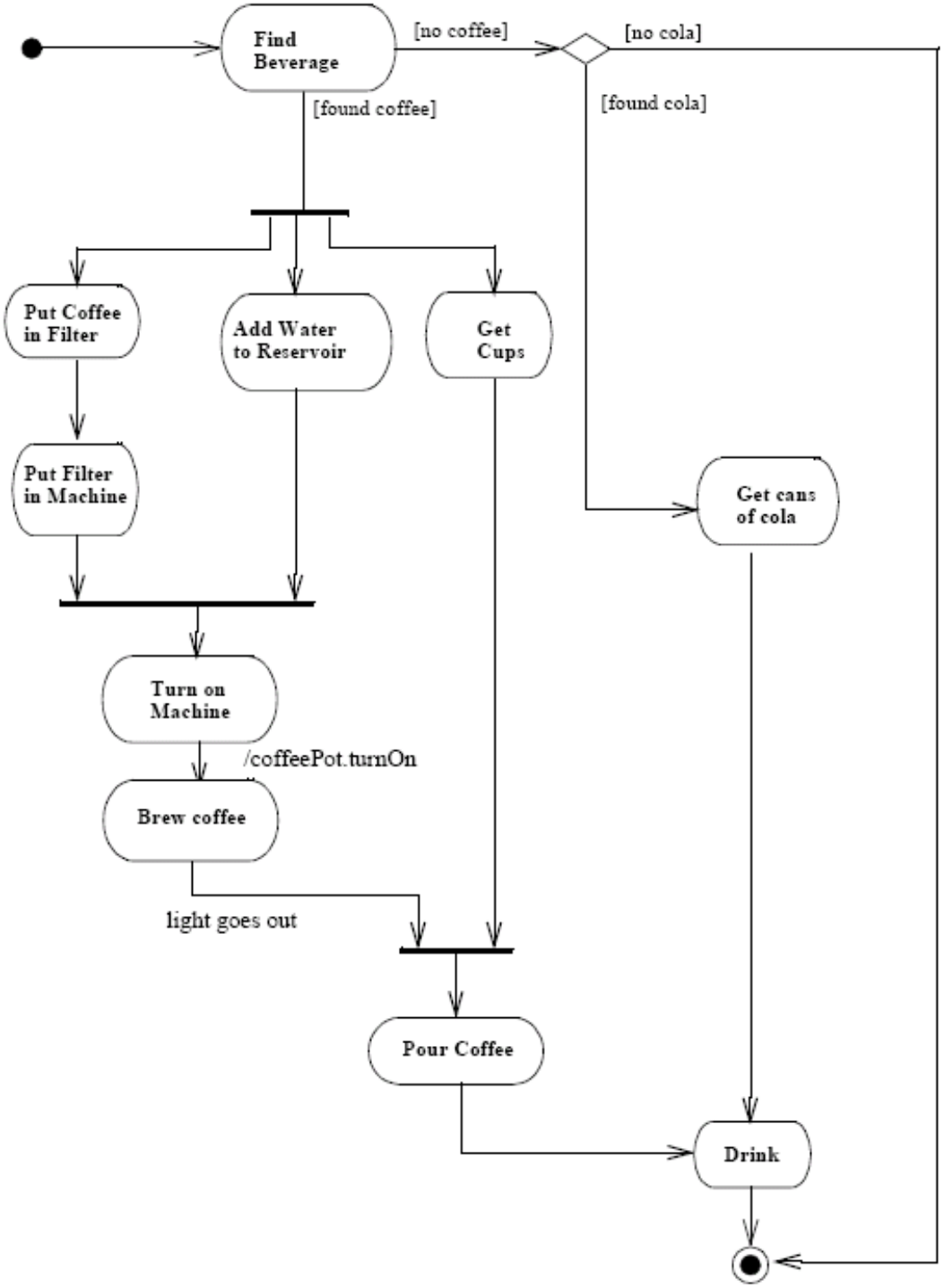
Activity Diagram and exchanged objects



Activity diagram: signals, timeout



https://publib.boulder.ibm.com/bpcsamp/advancedProcessFeatures/eventHandling/activityDiagram.html


Find Beverage
[no coffee]
[no cola]
[found coffee]
[found cola]
Put Coffee in Filter
Add Water to Reservoir
Get Cups
Put Filter in Machine
Get cans of cola
Turn on Machine
/coffeePot.turnOn
Brew coffee
light goes out
Pour Coffee
Drink

UML – other diagrams

type Structure

- *Component diagram*
  - Gives a « physical » view on the components of the system (files, libraries, databases…)
- *Deployment diagram*
  - Shows the hardware elements and the way components are dstributed on the hardware elements and how they interact
- *Composite Structure Diagram*
  - Since UML 2.x, shows the internal structure of a component

type Interaction

- *Communication Diagram*
  - Representation focused on messages exchanges between objects
- *Timing Diagram*
  - Gives the evolution of an object or a set of objects with time
- *Interaction Overview Diagram*
  - Global vue on a set of more detailed diagrams

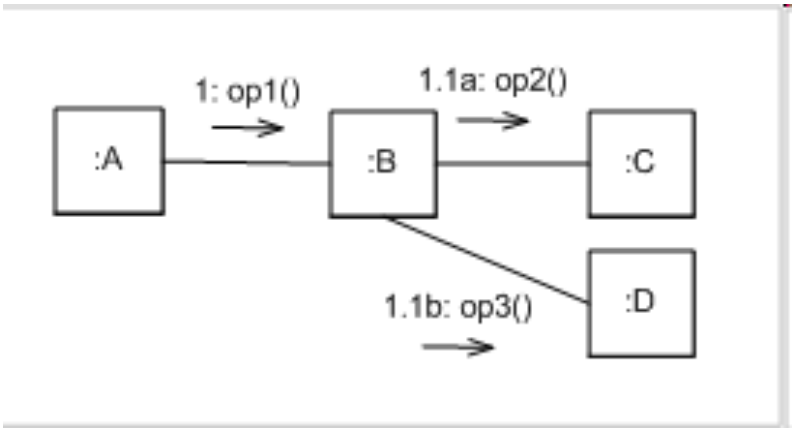
UML – Communication diagram

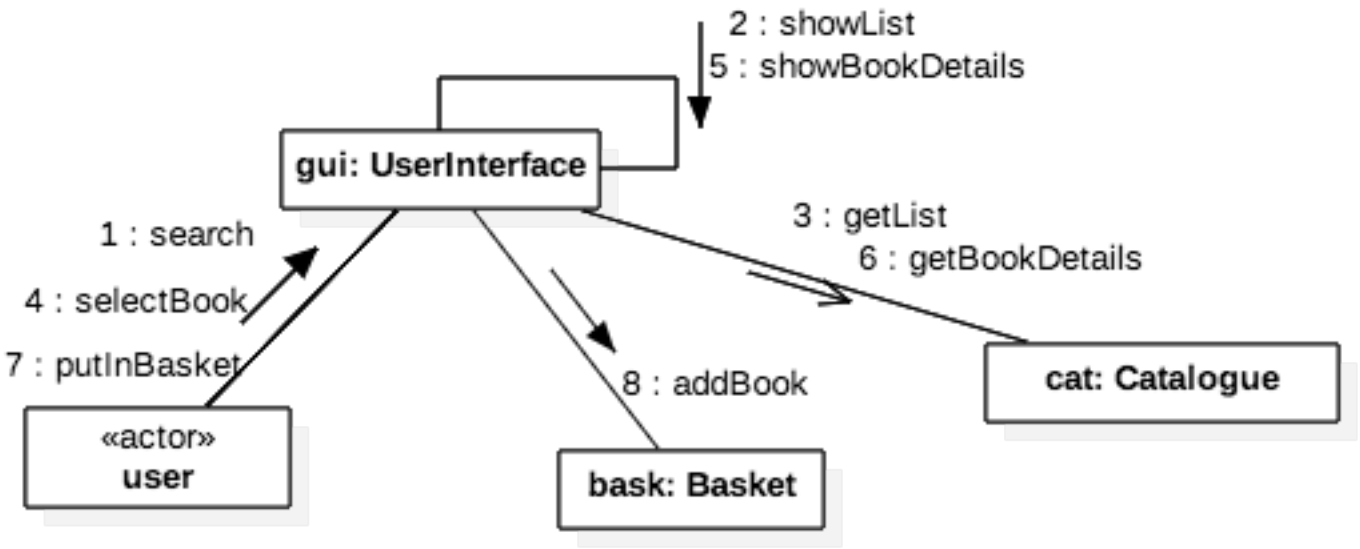
Communication diagram (UML1.x: Collaboration)

type Interaction

- representation focused on messages exchanges between objects
  - Sequence was focused on ordering

Examples


1: op1()
1.1a: op2()
:A
:B
:C
:D
1.1b: op3()


2 : showList
5 : showBookDetails
gui: UserInterface
1 : search
4 : selectBook
7 : putInBasket
3 : getList
6 : getBookDetails
8 : addBook
«actor»
user
bask: Basket
cat: Catalogue

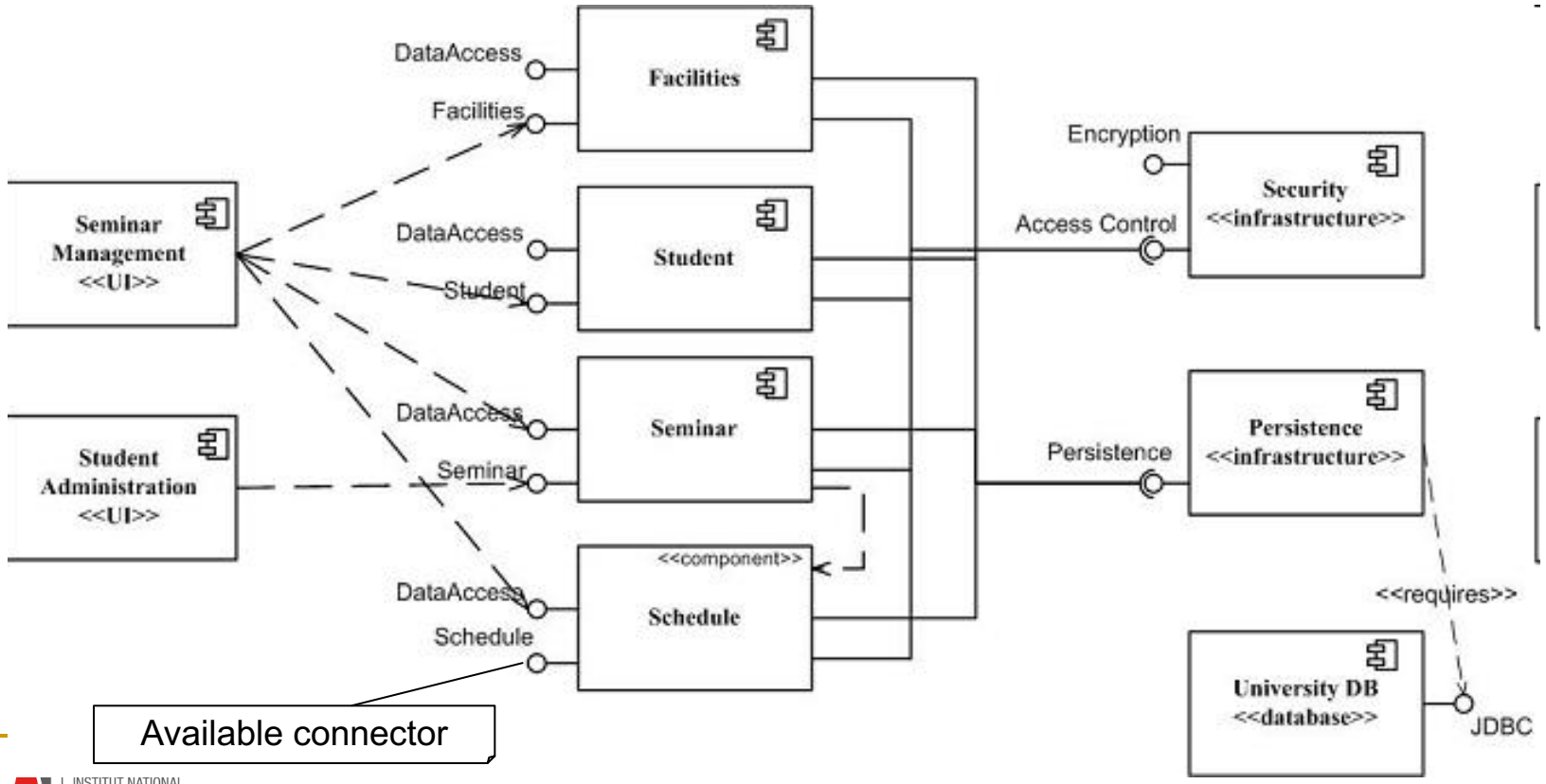
Elements of the diagram

- Participants
  - *Rectangle* with participant name and class
- Messages between participants
  - *Full line* = these participants communicate
  - For each direction of communication:
    - *Triangle arrow gives the direction*
    - A label to name the message
    - Messages can be numbered to express a notion of sequence

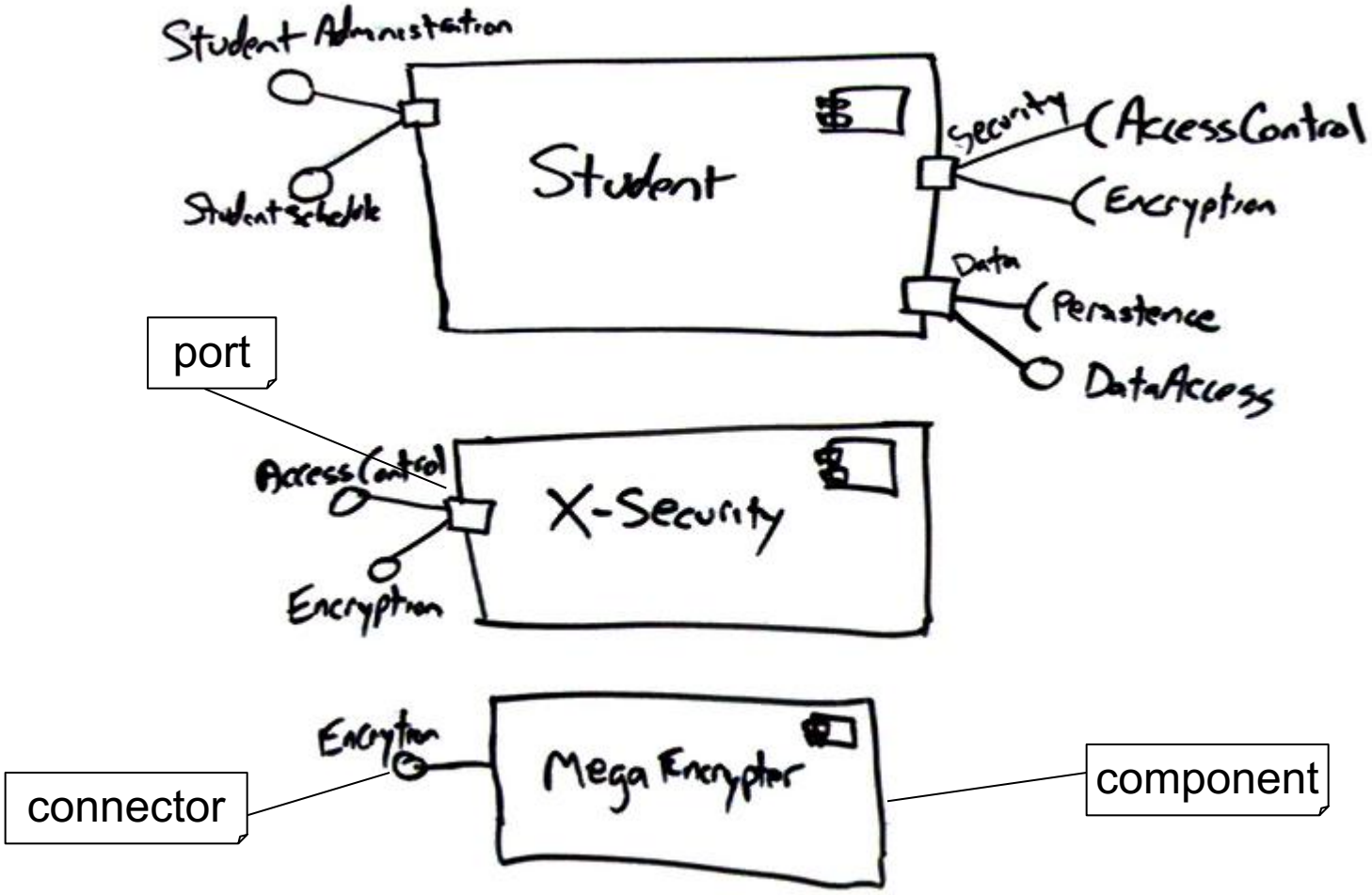
Components diagrams

type Structure

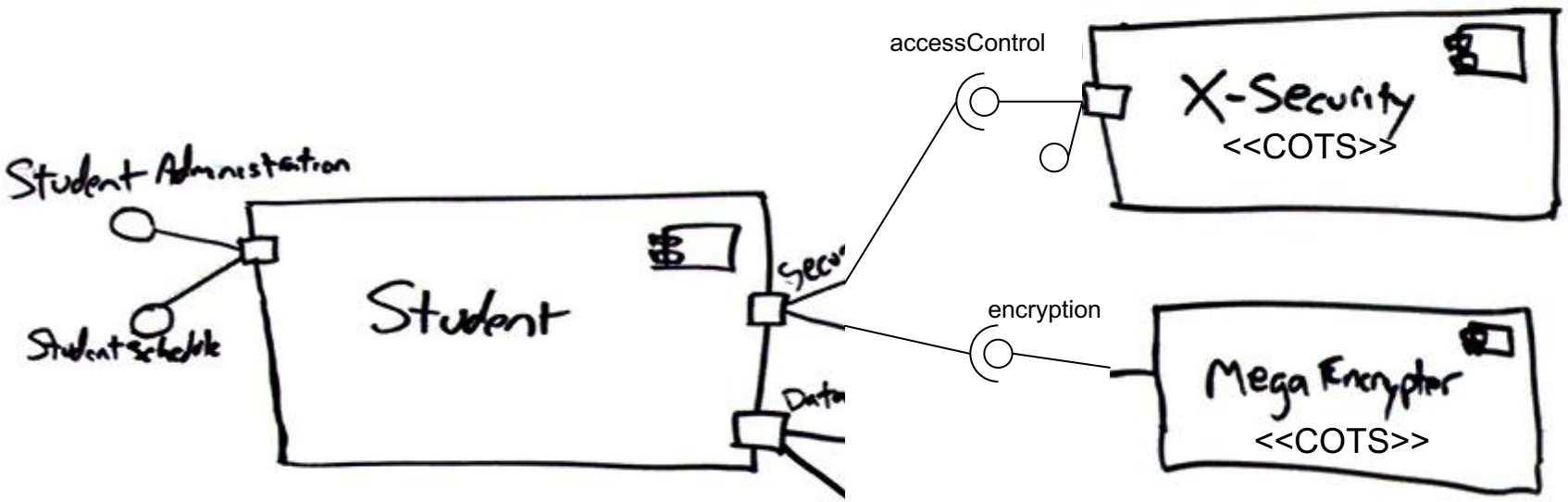
- Component = logical group of software elements, deployment unit. Components have connector to interact with others



Components, connectors and ports

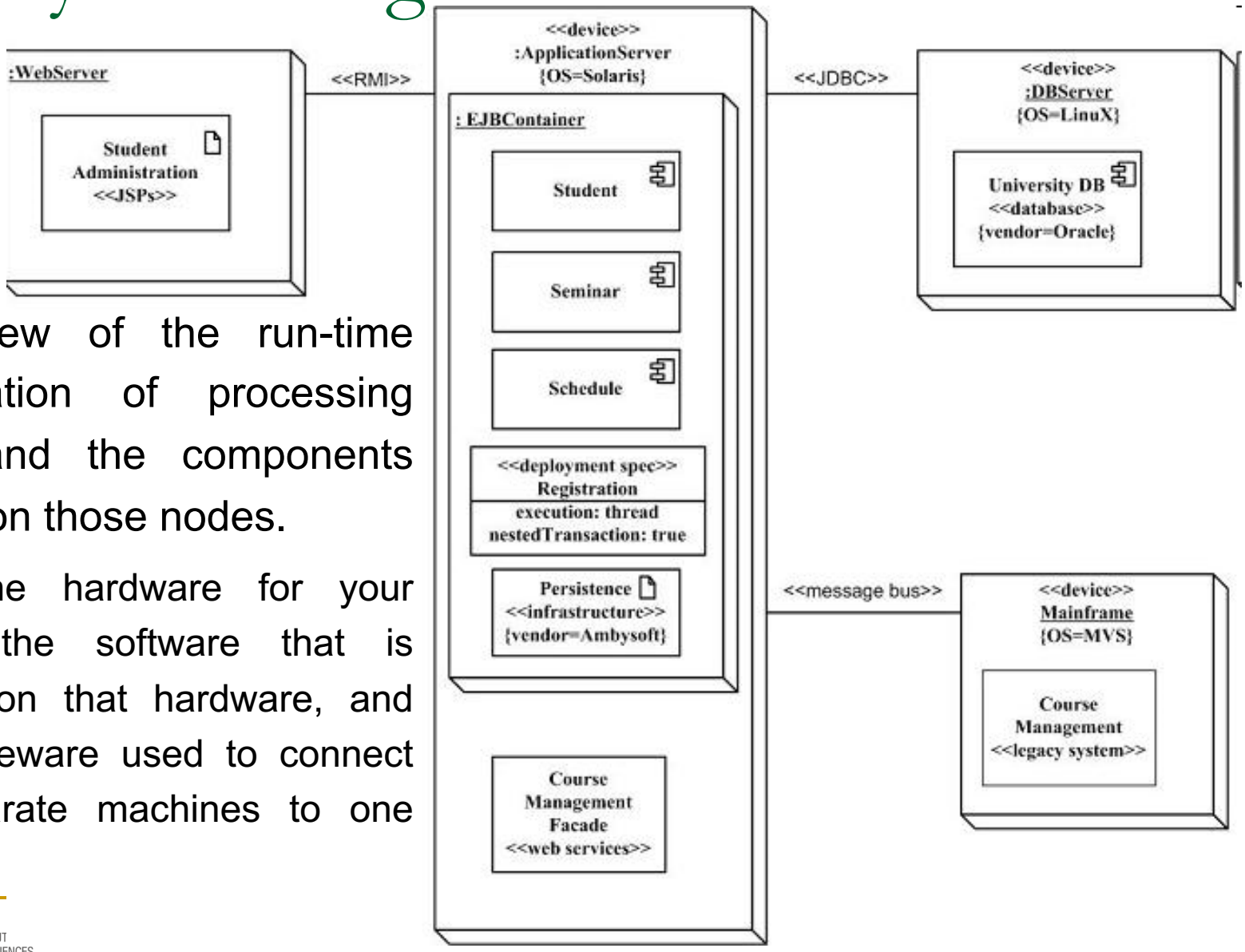


Interconnection



Deployment diagram

http://www.agilemodeling.com/artifacts/deploymentDiagram.htm

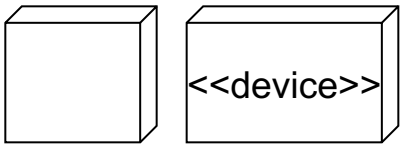


static view of the run-time configuration of processing nodes and the components that run on those nodes.

shows the hardware for your system, the software that is installed on that hardware, and the middleware used to connect the disparate machines to one another.

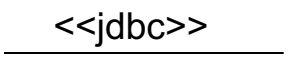
In the deployment diagram

- 3D box : hardware or software node



  - Can be composed
  - Contain software components
  - Can define properties

- lines: links between nodes

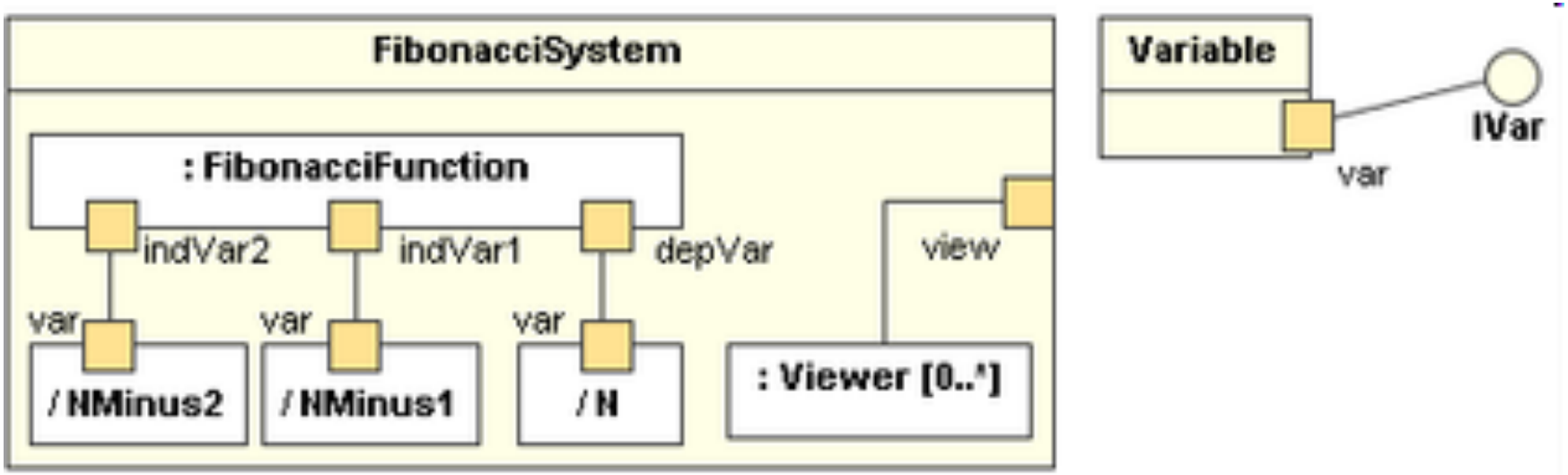


  - They often have a stereotype indicating the communication standard used

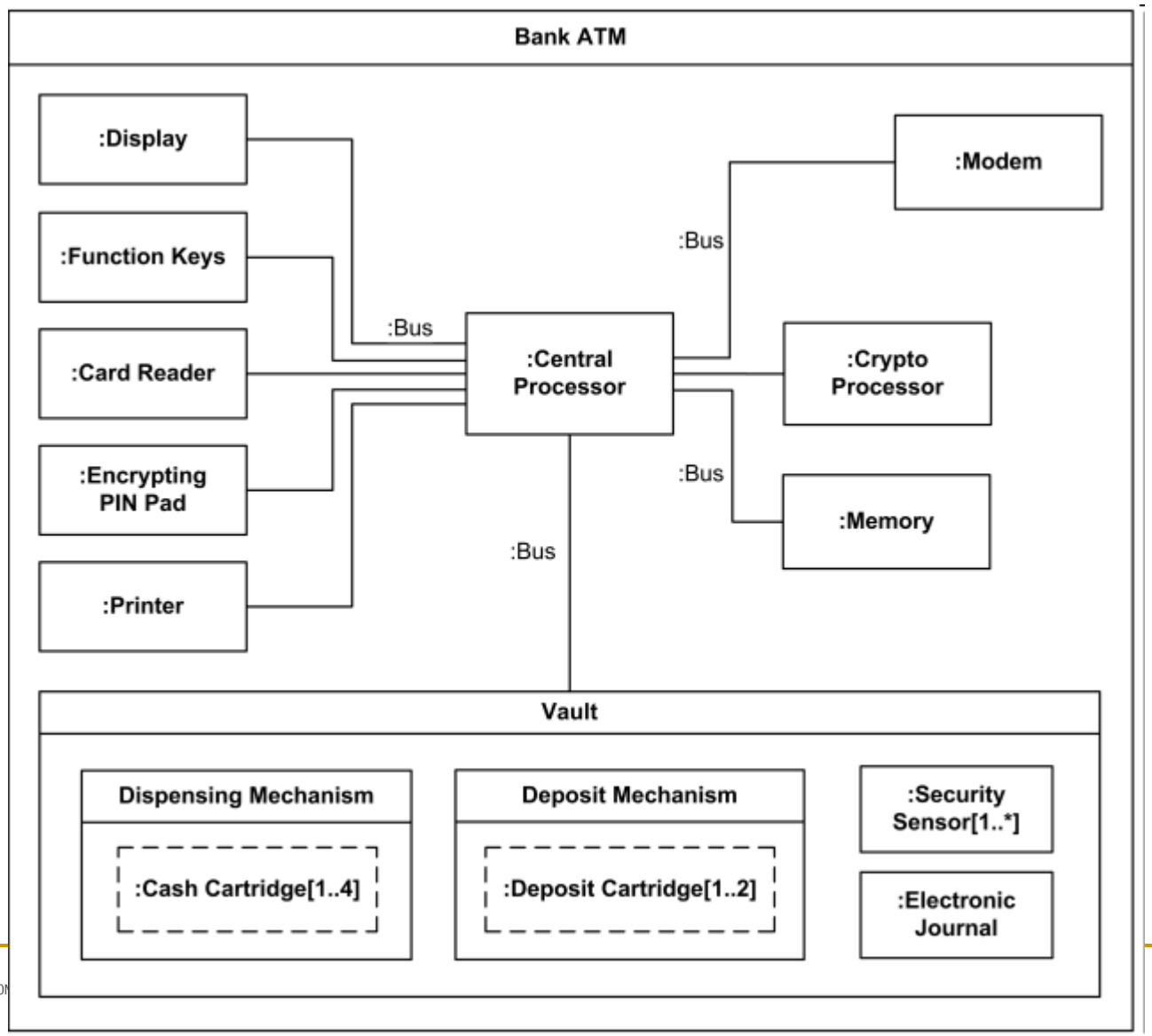
Composite structure diagram

type Structure

- Details the internal structure of a composite UML item (class, node, component…)

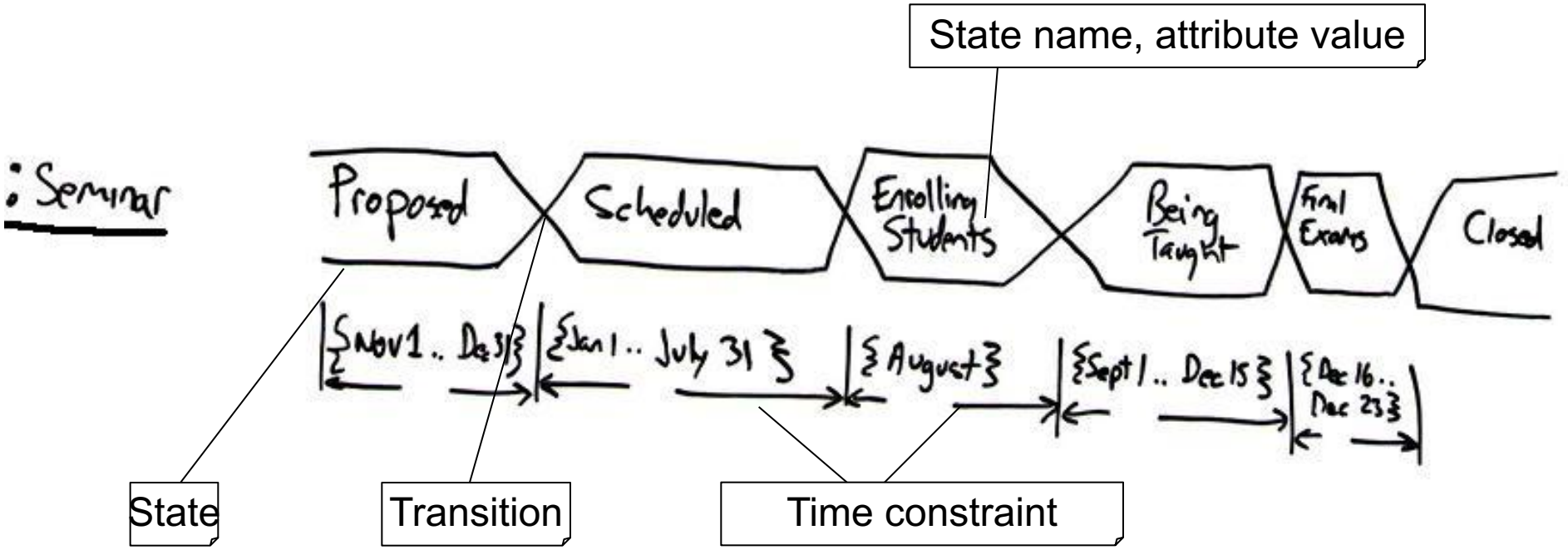


Composite structure diagram example



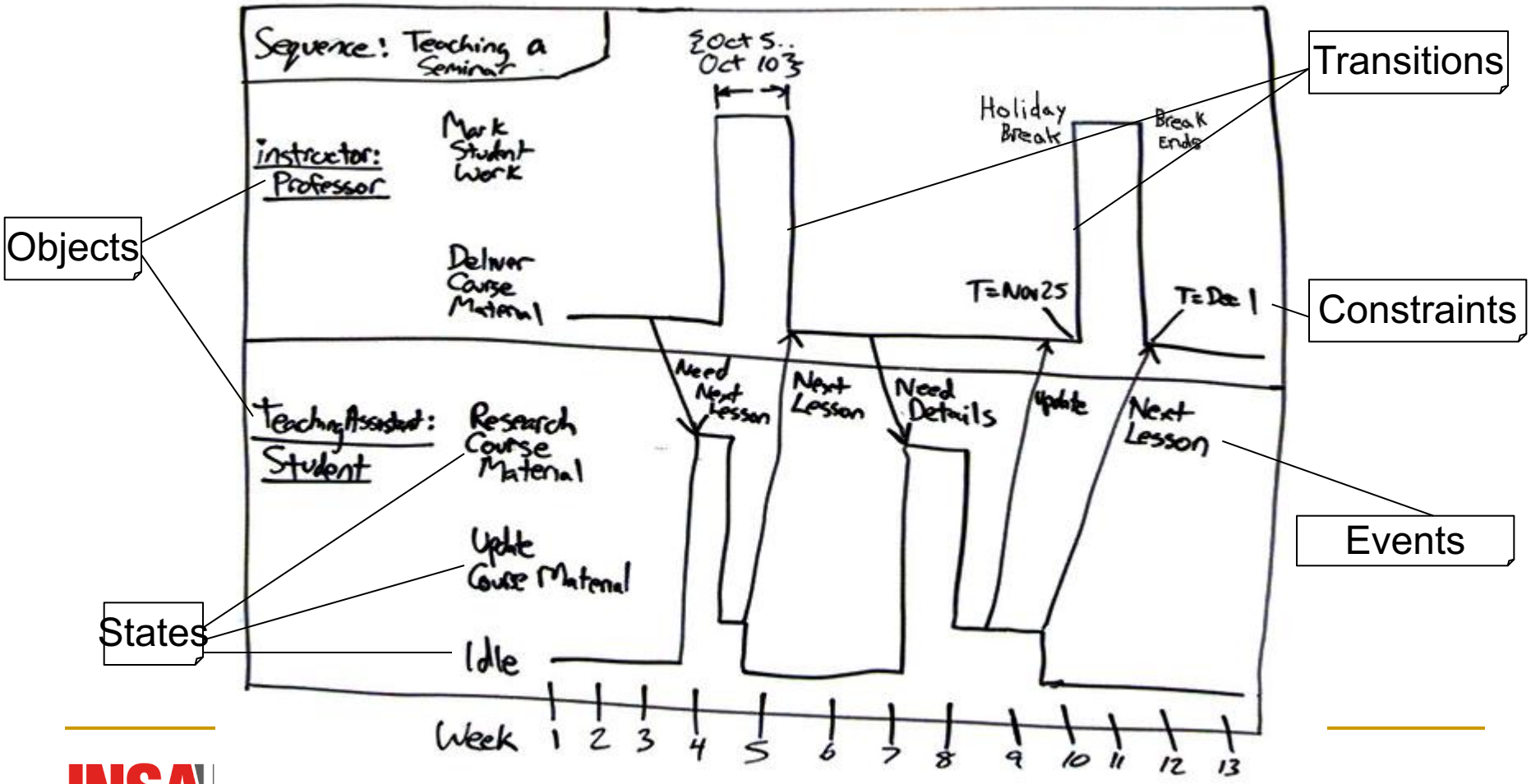
Time diagram

- For one object

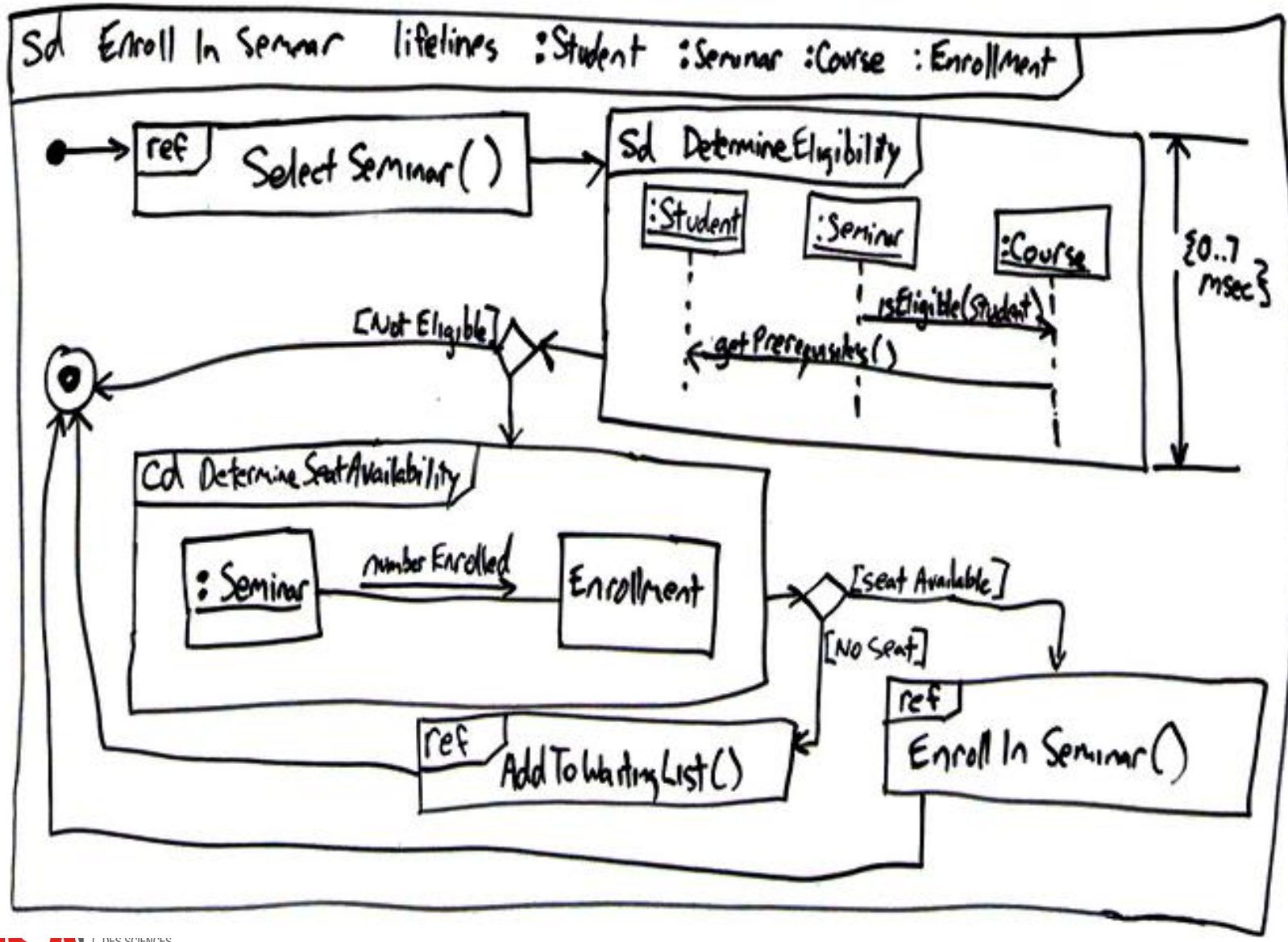


Time diagram

- Many objects



Global Interaction Diagram



Some exercices on the web?

- http://www.scribd.com/doc/8584444/Exercices-UML
- http://www.pearson.fr/resources/titles/2744010095421 0/extras/7466_chap01.pdf