Advanced Object-Oriented Programming
Project presentation

Cecilia ZANNI-MERK
INSA Rouen Normandie

Agenda

- UML
- Simplified SE approach with UML
- UML by the example
- Project
- Deliverables and deadlines

Resources in the Web

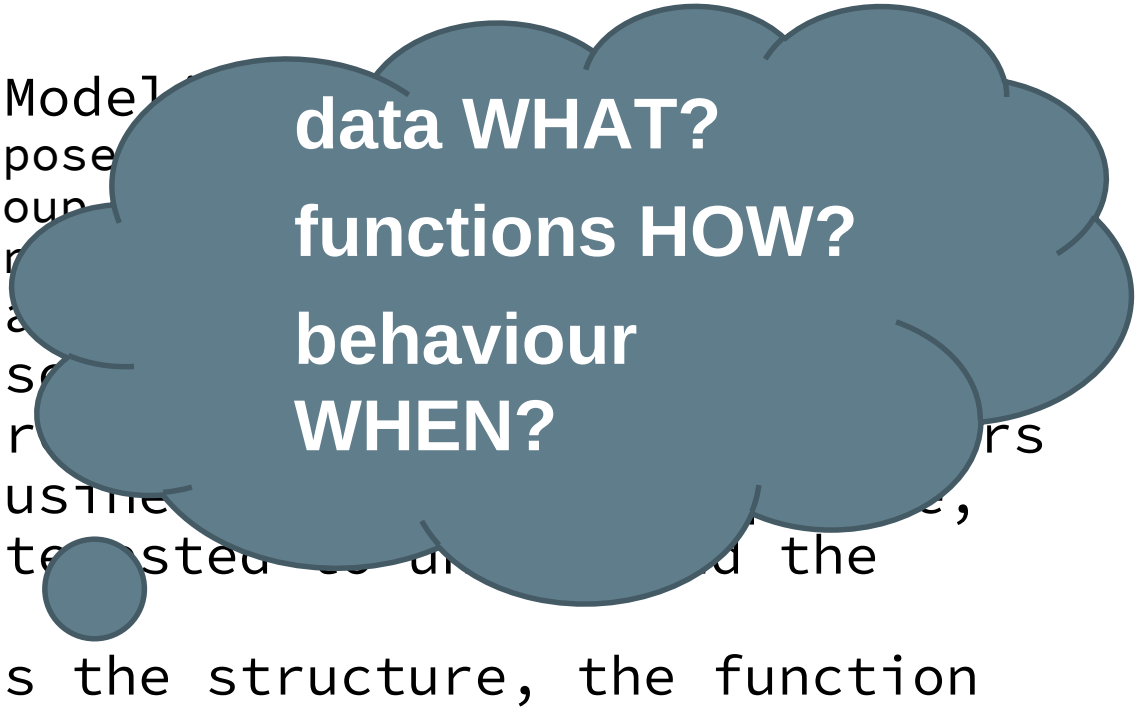
- Tutorials
  - https://www.tutorialspoint.com/uml/index.htm
  - https://sparxsystems.com/resources/tutorials/uml/part1.html
- UML editors
  - https://staruml.io/
  - https://plantuml.com/en/
  - https://www.bouml.fr/
  - http://www.pacestar.com/edge/
  - https://www.eclipse.org/papyrus/

UML

- UML = Unified Modeling Language
  - Initially proposed in 1997 by the Object Management Group (OMG)
  - Latest version is UML 2.0
- It’s a graphical language for modeling objects, in a set of different diagrams
- UML diagrams are not only made for developers but also for business users, common people, and anybody interested to understand the system.
- UML can express the structure, the function and the behaviour of a program

UML

- UML = Unified Model
  - Initially propose
    Management Grou
  - Latest versio
- It's a graphica
  objects, in a s
- UML diagrams ar
  but also for busine
  and anybody inte ested
  system.
- UML can expr s the structure, the function and the behaviour of a program

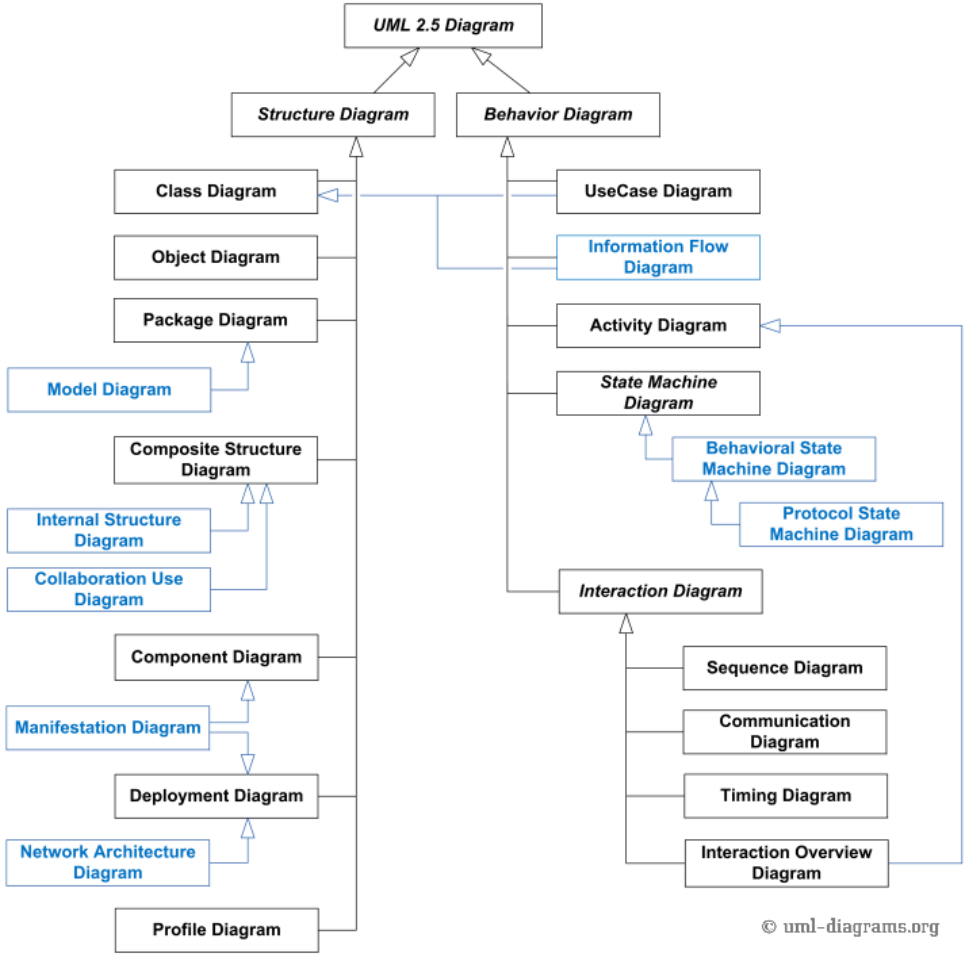


Official definition

- The Object Management Group (OMG) specification states:

"The Unified Modeling Language (UML) is a ***graphical language*** for ***visualizing, specifying, constructing, and documenting*** the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."

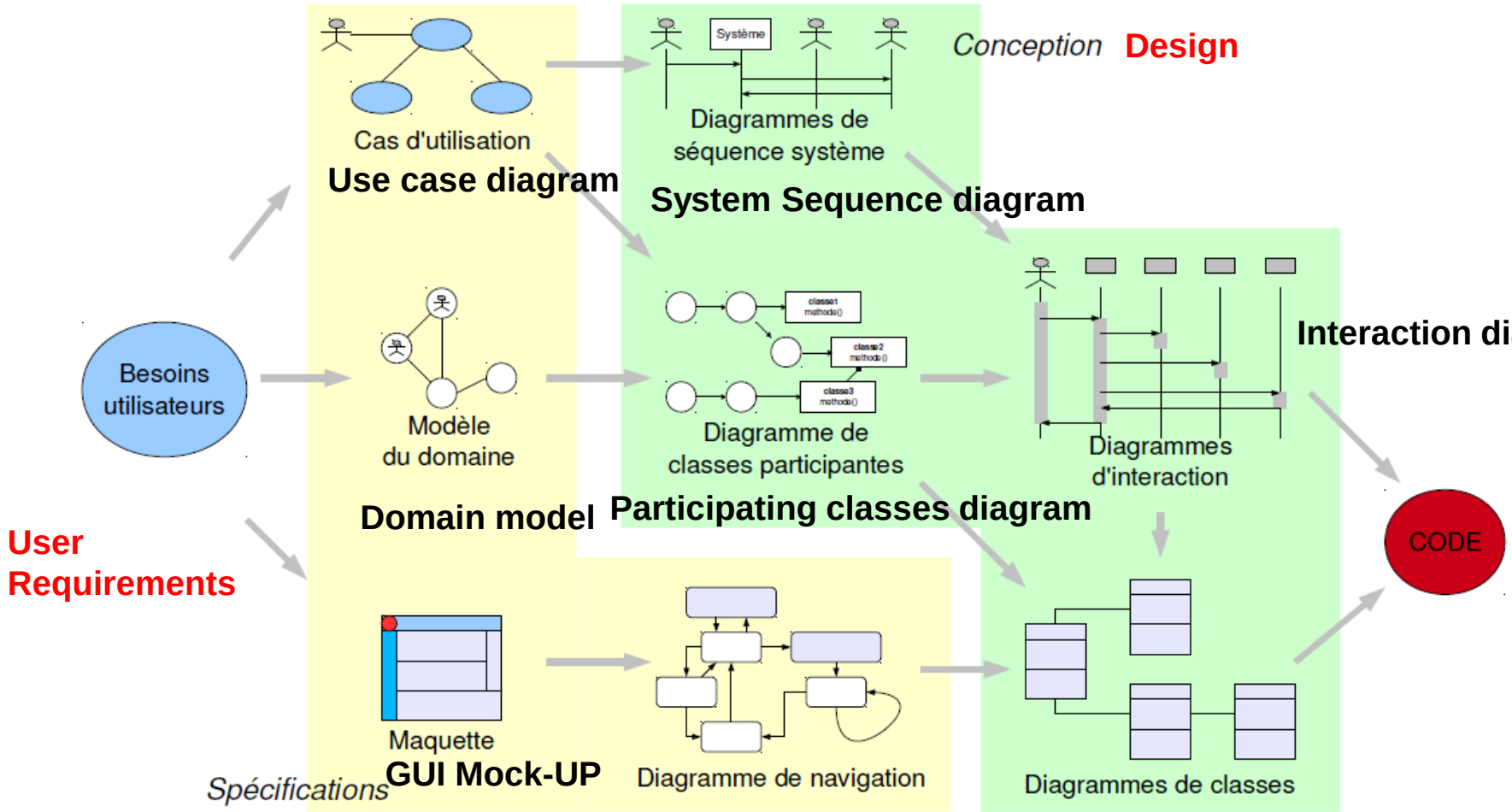
Summary UML 2.0



Summary UML 2.0

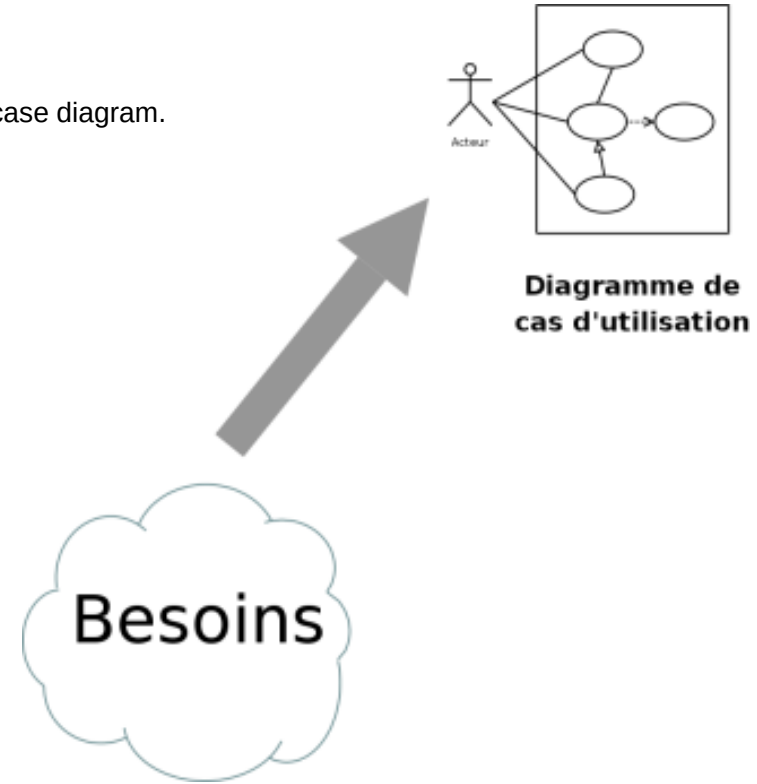
- As already discussed, UML includes thirteen diagrams:
  - package diagram,
  - component diagram, → **implementation**
  - deployment diagram, → **implementation**
  - interaction overview diagram,
  - composite structure diagram,
  - state machine diagram,
  - timing diagram,
  - object diagram,
  - class diagram, → STRUCTURE (design)
  - use case diagram, → FUNCTION (specification)
  - activity diagram, → **BEHAVIOUR**
  - sequence diagram, → **BEHAVIOUR (design)**
  - communication diagram → **BEHAVIOUR**

Simplified SE approach with UML

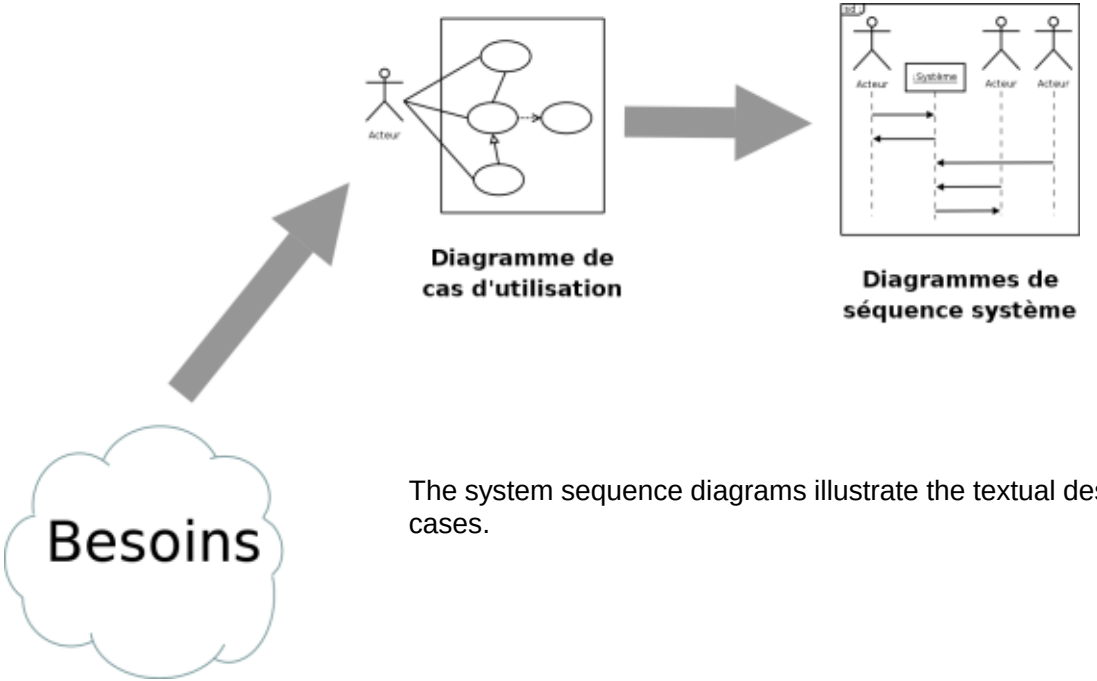

Conception
Design
Cas d'utilisation
Use case diagram
Système
Diagrammes de
séquence système
System Sequence diagram
Besoins
utilisateurs
Modèle
du domaine
Domain model
classe1
methode()
classe2
methode()
classe3
methode()
Diagramme de
classes participantes
Participating classes diagram
Diagrammes
d'interaction
Interaction diagram
User
Requirements
CODE
Maquette
GUI Mock-UP
Spécifications
Specifications
Diagramme de navigation
Navigation
diagram
Diagrammes de classes
Class diagram

Specifications : identification of user requirements

Requirements are modeled by a use case diagram.

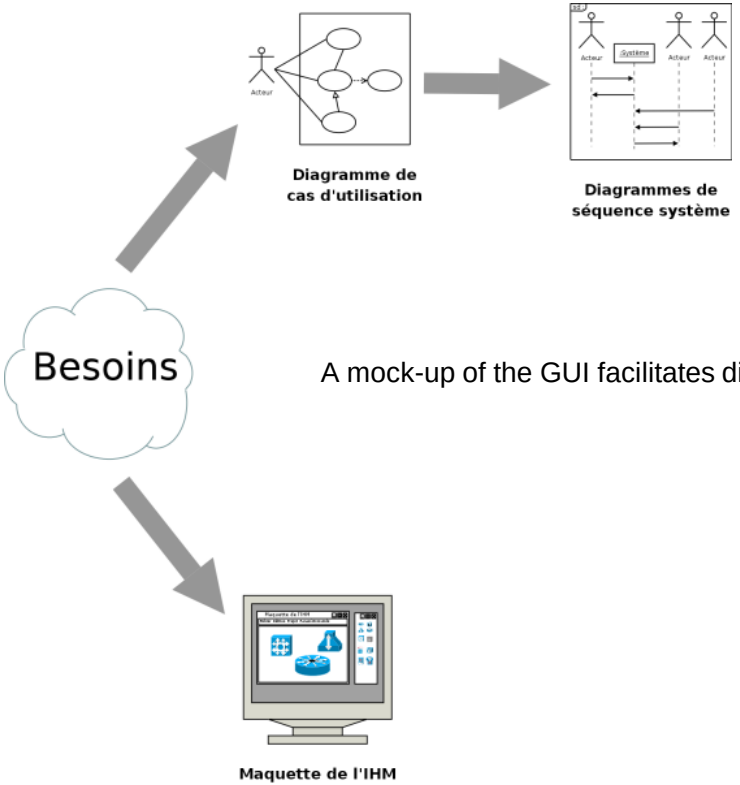


Specifications : identification of user requirements



The system sequence diagrams illustrate the textual description of the use cases.

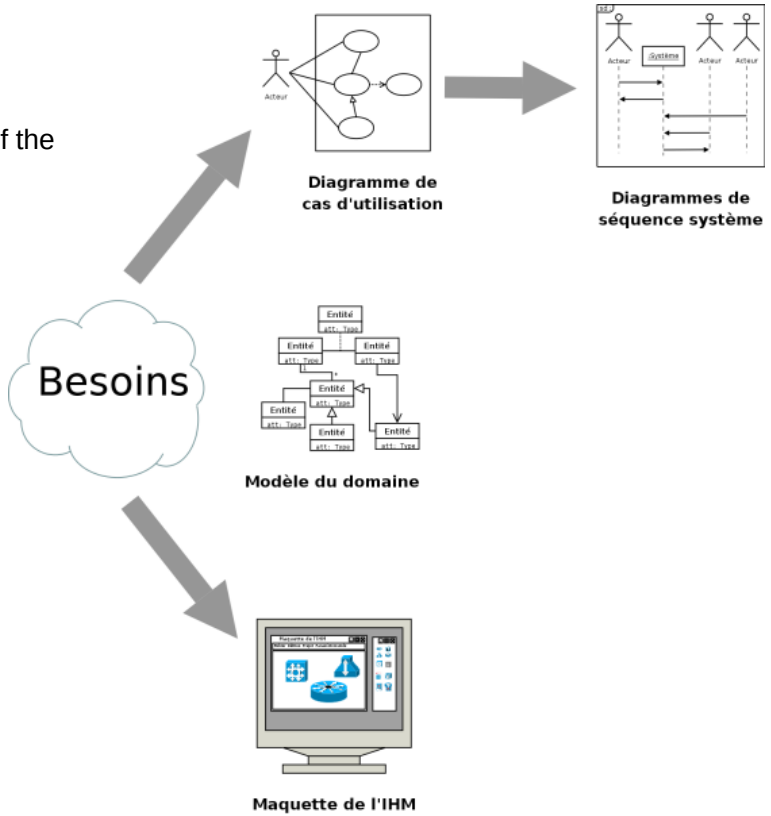
Specifications : identification of user requirements



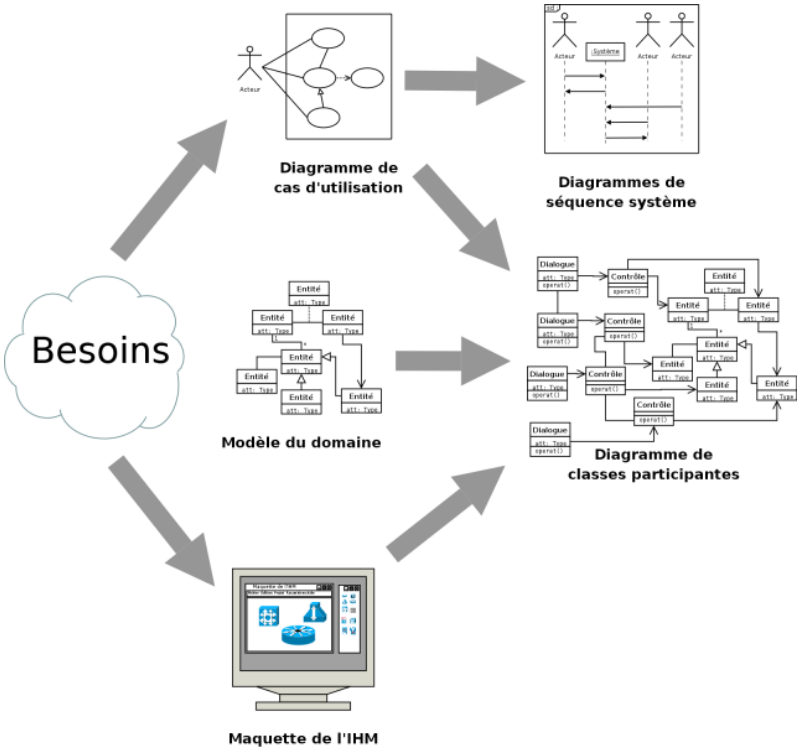
A mock-up of the GUI facilitates discussions with future users.

Specifications

The domain analysis phase is used to develop the first version of the class diagram.

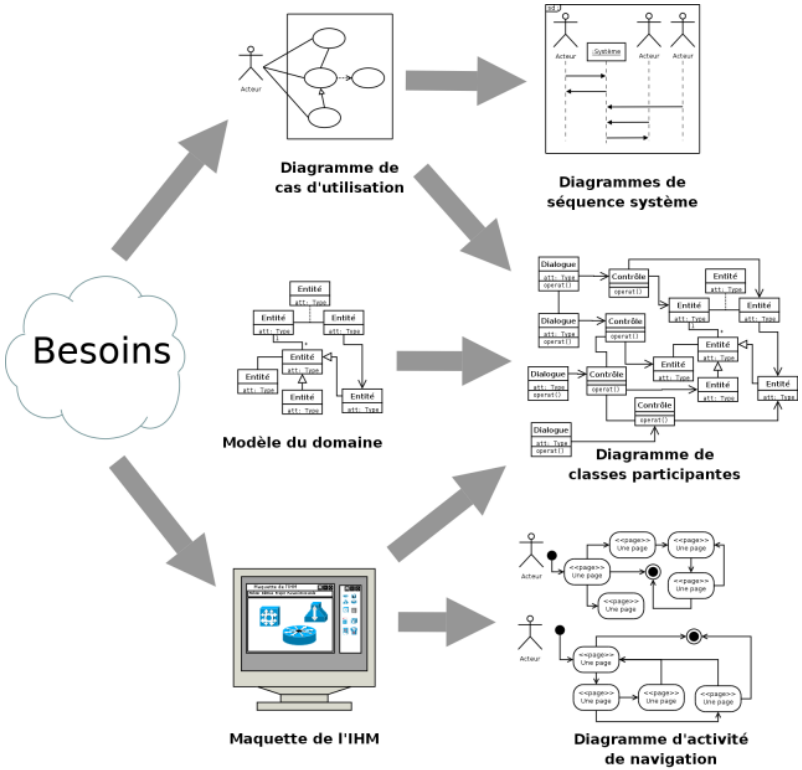


Specifications



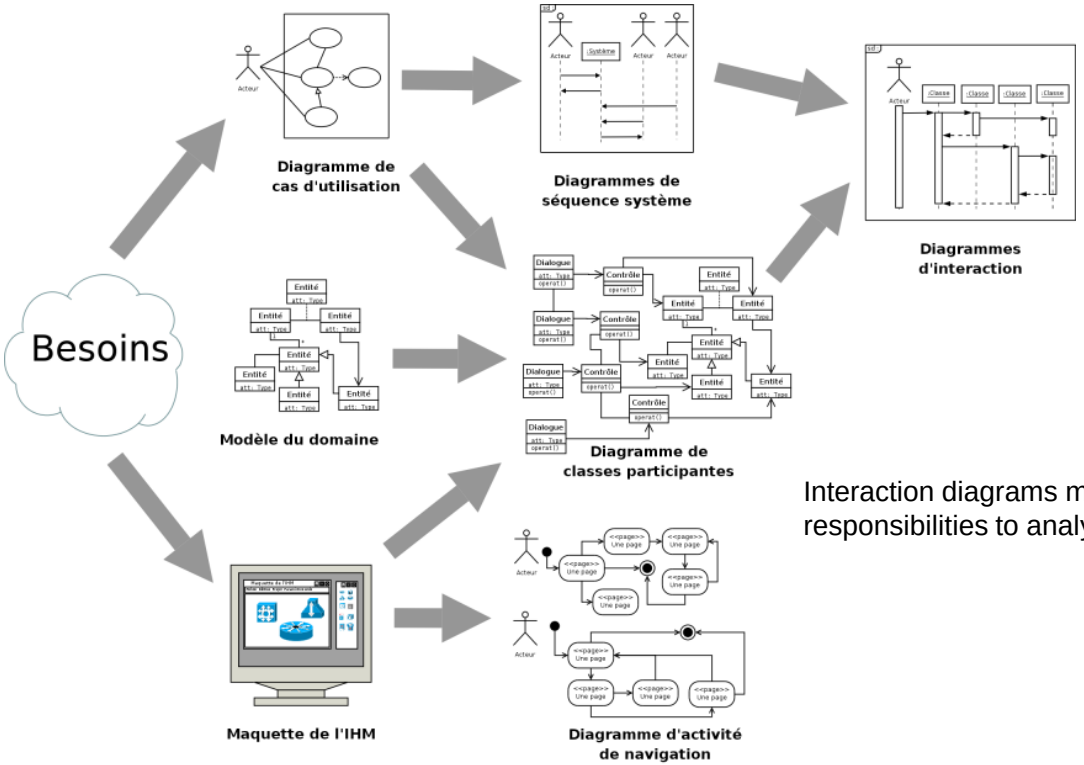
The participating class diagram connects the use cases, the domain model and the mock-ups.

Specifications



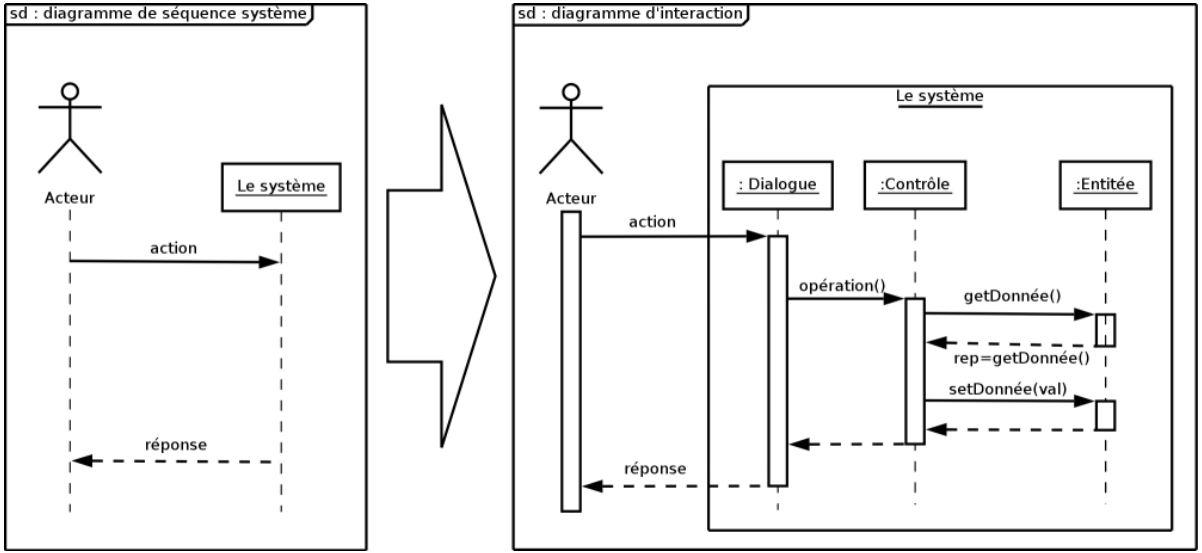
Navigation activity diagrams graphically represent the navigation activity in the GUI

Design phase



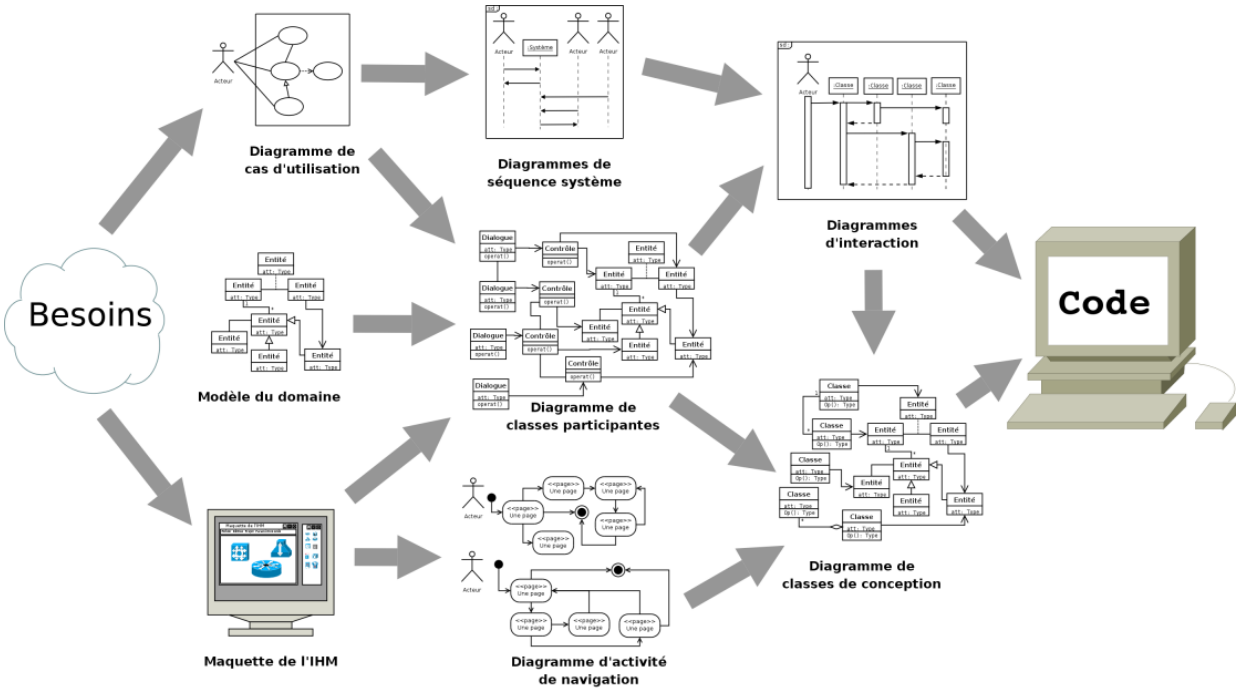
Interaction diagrams make it possible to precisely assign behavioral responsibilities to analysis classes.

Design phase



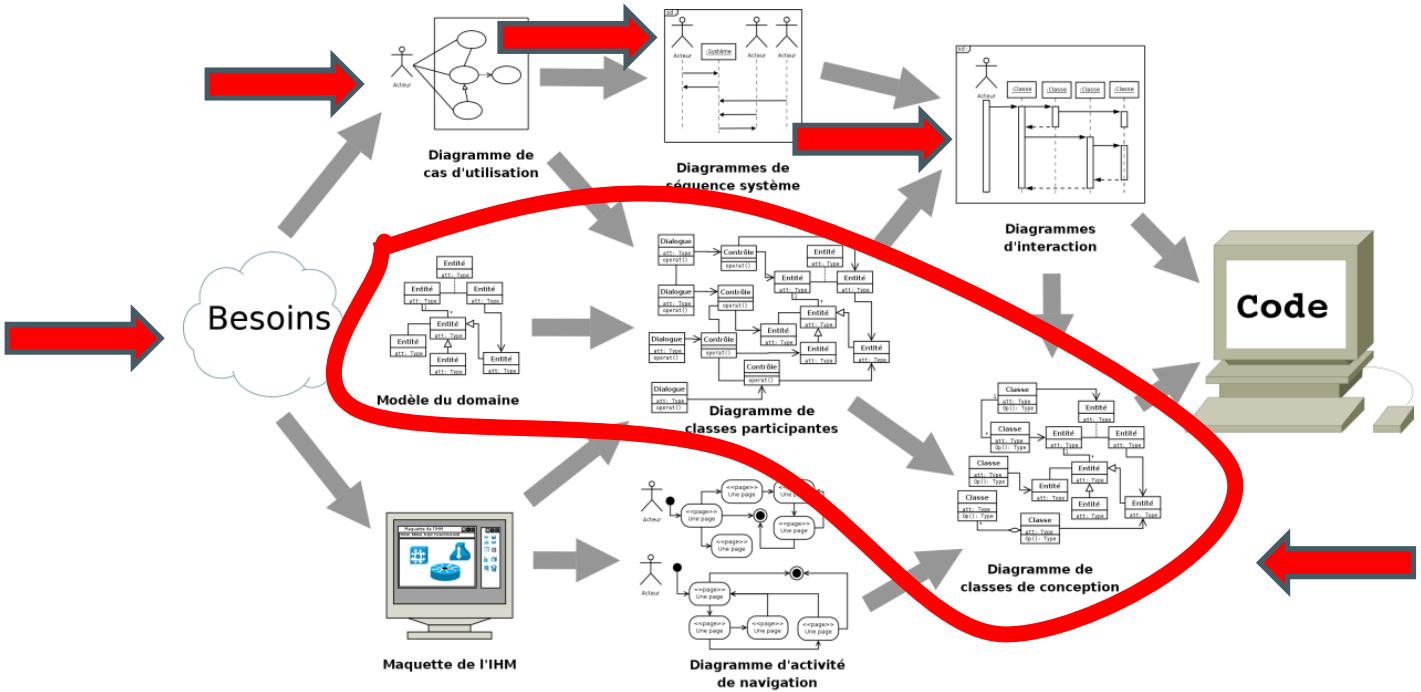
The set of system sequence diagrams, seen as a black box, is replaced by a set of collaborative objects.

Design phase



The design class diagram will be used for the implementation

We will concentrate on …



Sequence diagrams (for the design phase)

- Diagram representing the interactions between classes by chronology of method calls
- ***A sequence diagram illustrates a use case***
- The ***system sequence diagrams*** illustrate the textual description of the use cases.
- Time is represented by a vertical axis

Sequ

- D
  c
- ***A***
- Th
  te
- T

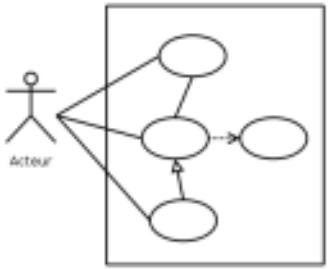
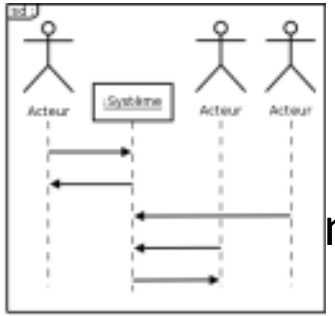


Diagramme de cas d'utilisation



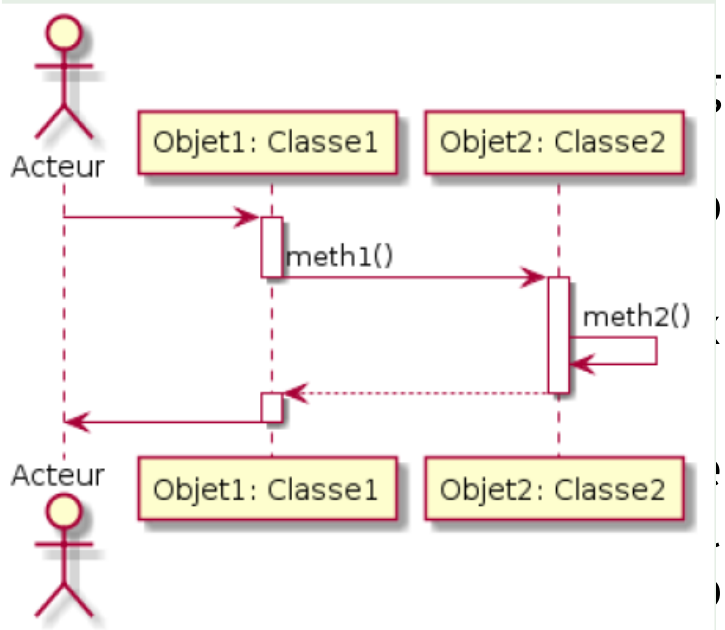
Diagrammes de séquence système

n

Sequence diagrams syntax

- A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.
- Activation boxes are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message

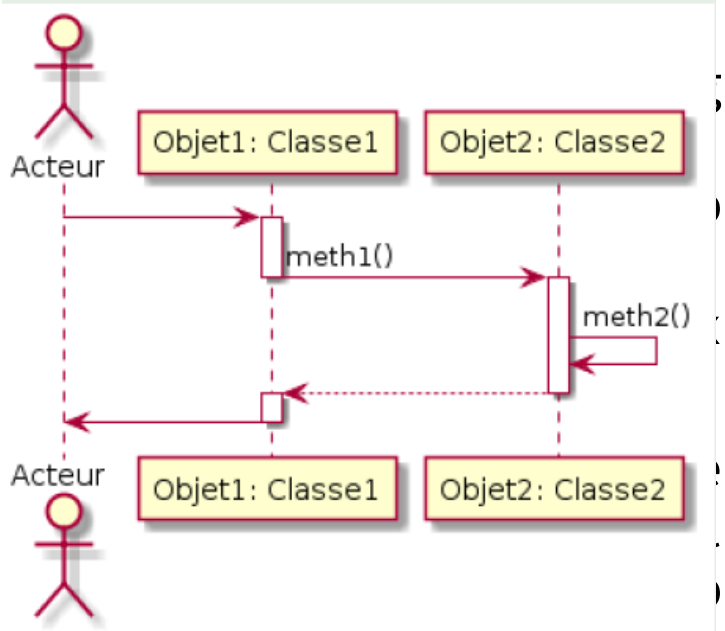
Sequence diagrams syntax

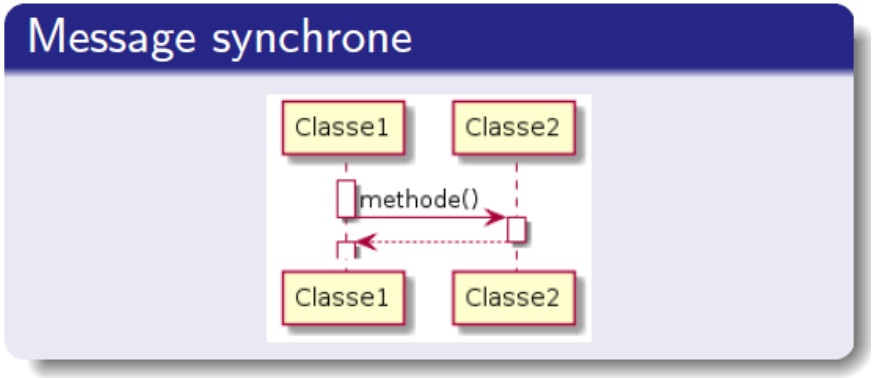


ram shows, as parallel
(*lifelines*), different
jects that live
and, as horizontal arrows,
changed between them, in the
they occur.

s are opaque rectangles drawn
ines to represent that
eing performed in response to
the message

Sequence diagrams syntax

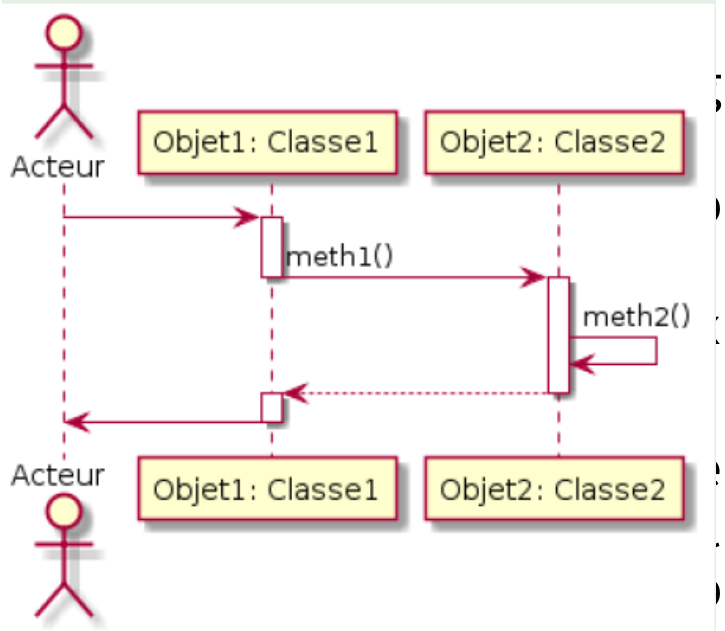




ram show
(*lifelin*
jects that live
and, as horizontal arrows,
changed between them, in the
they occur.

s are opaque rectangles drawn
ines to represent that
eing performed in response to
the message

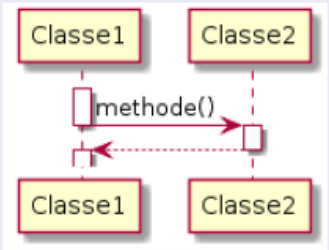
Sequence diagrams syntax



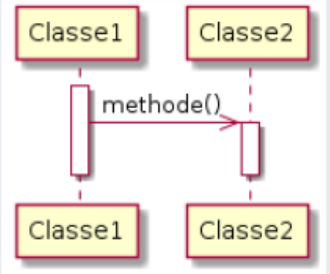
gram show
(*lifelin
jects that live
and, as horizontal arrows.
changed
they occ
s are op
lines to
being per

the message

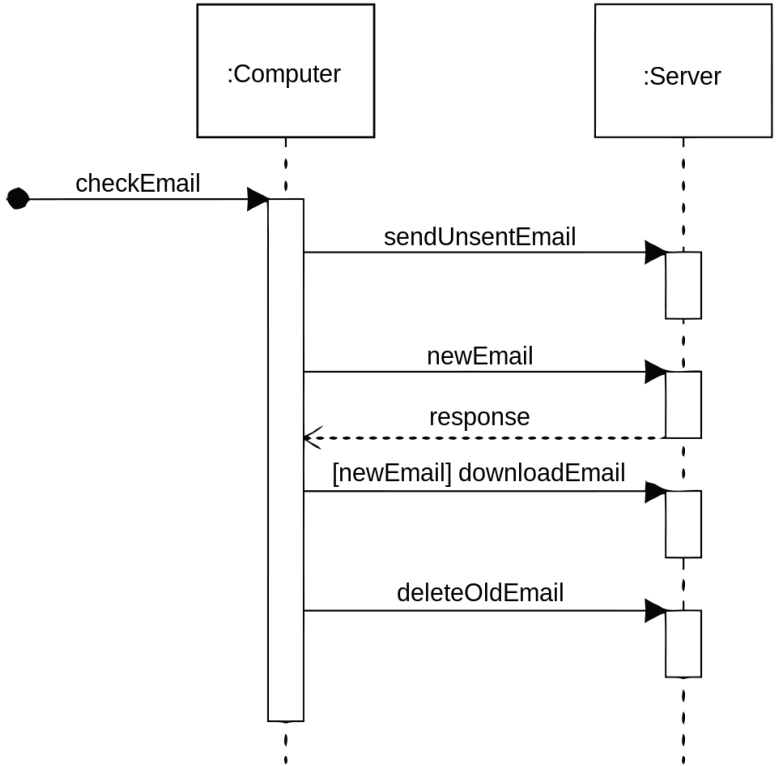
Message synchrone



Message asynchrone



Sequence diagrams: a more complex example

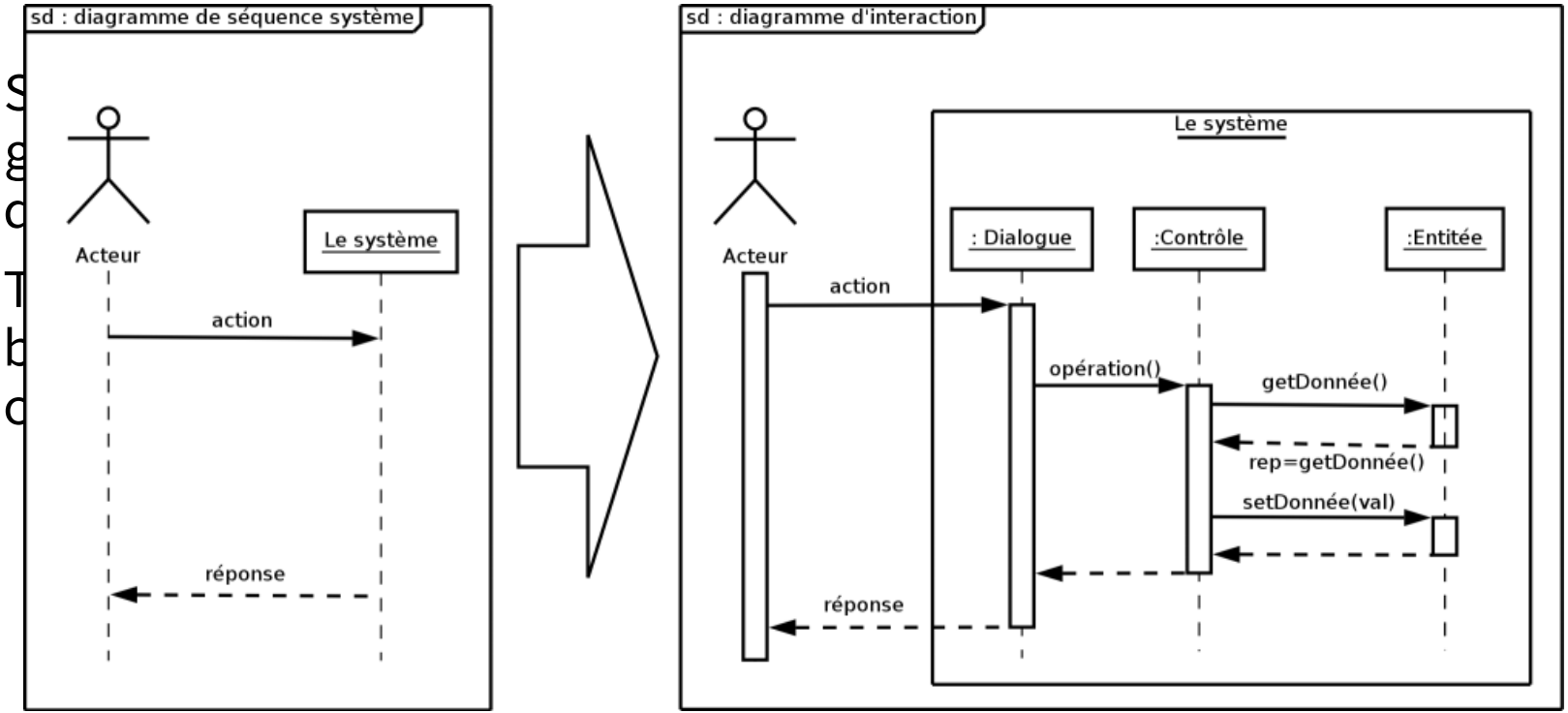


Further in the design phase …

- Sequence system diagrams can be refined (in general after having developed the ***Class*** diagram)
- The system sequence diagram, seen as a black box, is replaced by a set of collaborative objects.

Further in the design phase …

- S
- T



Component diagrams

- A component groups together a set of resources (objects, files, libraries, etc.) from the application to make it a reusable entity and/or to be deployed on a hardware medium.
- It is described by ***provided*** and ***required*** **interfaces**
  - A ***provided interface*** is a functionality implemented by the component that other components can call.
  - A ***required interface*** is a functionality that the component must be able to use and implemented by others.

These notions are essential in distributed programming, because each component is a black box for the other entities of the global system.

Component diagrams

- A component groups together a set of resources (objects, files, libraries, etc.) from the application to make it a reusable entity and/or to be deployed on a hardware medium.
- It is described by ***provided*** and ***required*** **interfaces**
  - A ***provided interface*** is a functionality implemented by the component that other components can call.
  - A ***required interface*** is a functionality that the component must be able to use and implemented by others.

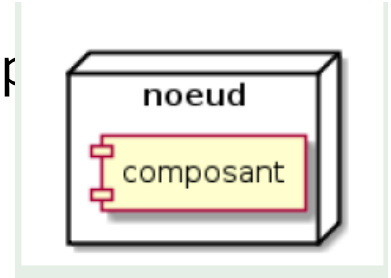
These notions are essential in distributed programming, because each component is a black box for the other entities of the global system.

Deployment diagrams

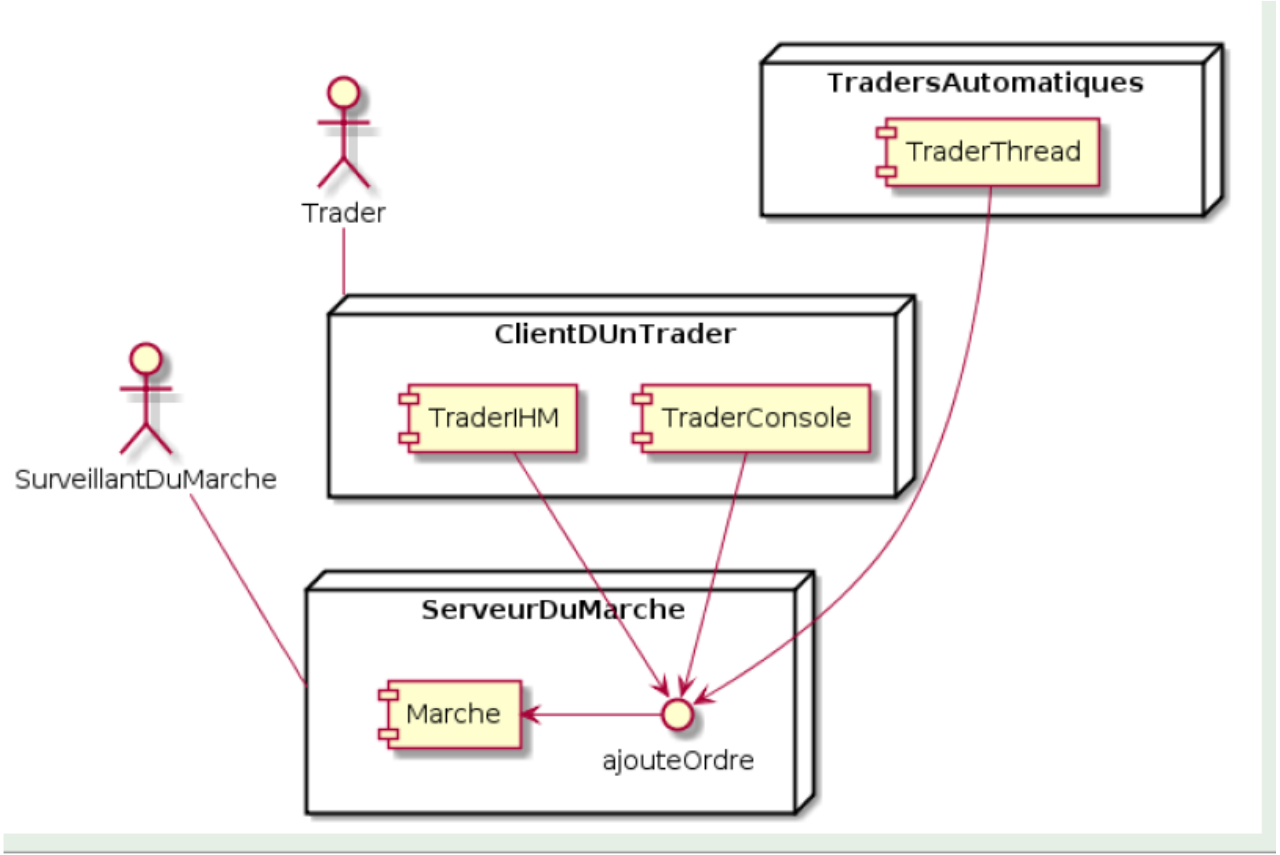
- A deployment diagram is the description of the hardware configuration by the location of software and hardware components on the nodes of the architecture.
- It links the required and provided interfaces of the components to define how they communicate.
- Hardware resources (computer, peripherals, ...) are represented by nodes.

Deployment diagrams

- A deployment diagram is the description of the hardware configuration by the location of software and hardware components on the nodes of the architecture.
- It links the required and provided interfaces of the components to define how they communicate.
- Hardware resources (computer, perip
...) are represented by nodes.



A more complex example



UML by the example

User requirements

Develop software for boat management in a marina

Specification: Goals of the software

- In a marina, there are sailing boats and outboard boats. The boats have a home port and move from port to port.
- ***We want to keep track of the boats' movements and be able to contact their owner***

Specification: Analysis of the needs

- This is the list of features they want to see integrated into the software.
- In general, with the help of the "use cases" of the UML
- Use case diagrams are UML diagrams used to give an overview of the functional behavior of a software system.

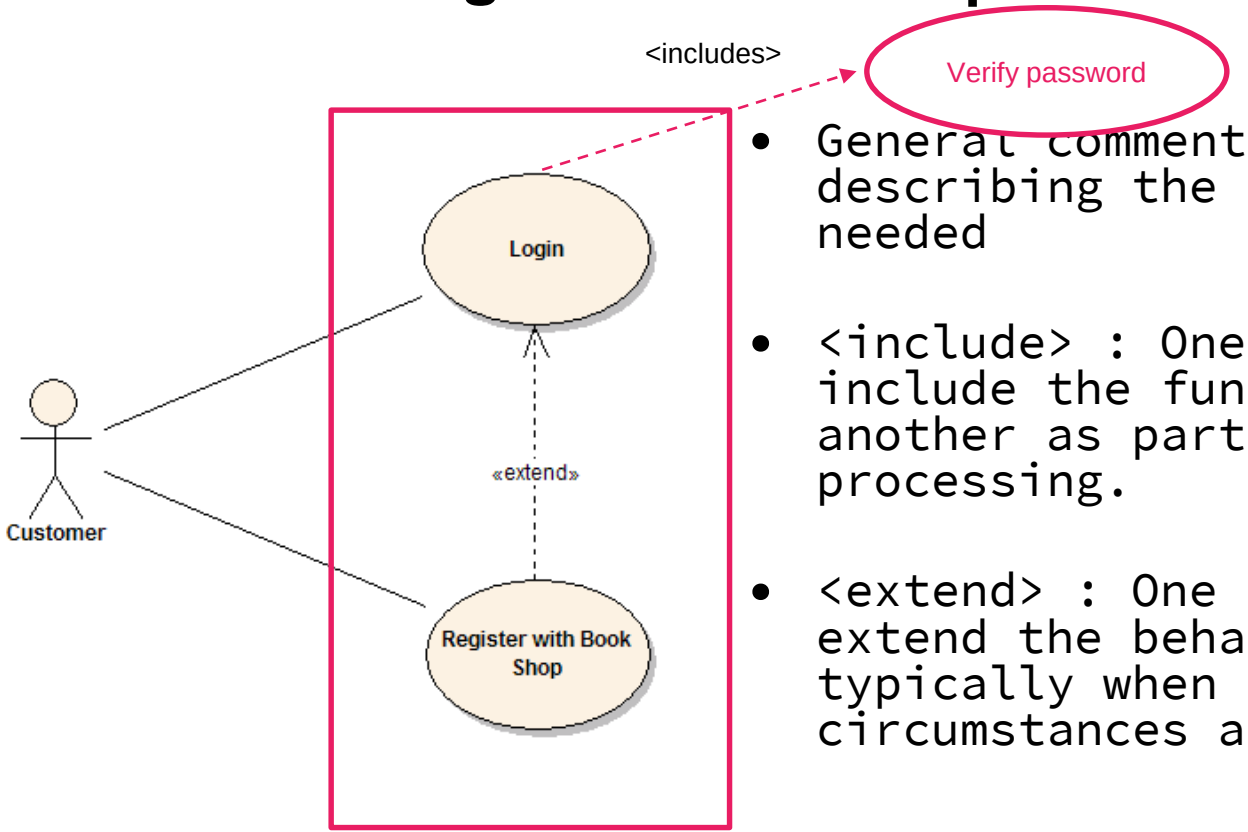
Specification: Analysis of needs

1. Add / delete / modify a boat
2. Find the owner of a boat
3. Calculate the annual tax of a boat in its home port. The tax depends on the length of the boat, and its power if the boat is motorized
4. List the boats for a given home port
5. List the ports visited by a boat
6. Make the history of a boat's movements
7. List the ports sorted by number of attached boats
8. Send a letter to each owner of a boat in a port specifying the amount of the tax to pay

Use Case diagrams

- They are useful for presentations to the management or stakeholders of a project
- A use case represents a discrete unit of interaction between a user (human or machine) and a system. It is a significant unit of work.
- In a use case diagram, users are called actors, they interact with use cases

Use Case diagrams … an example

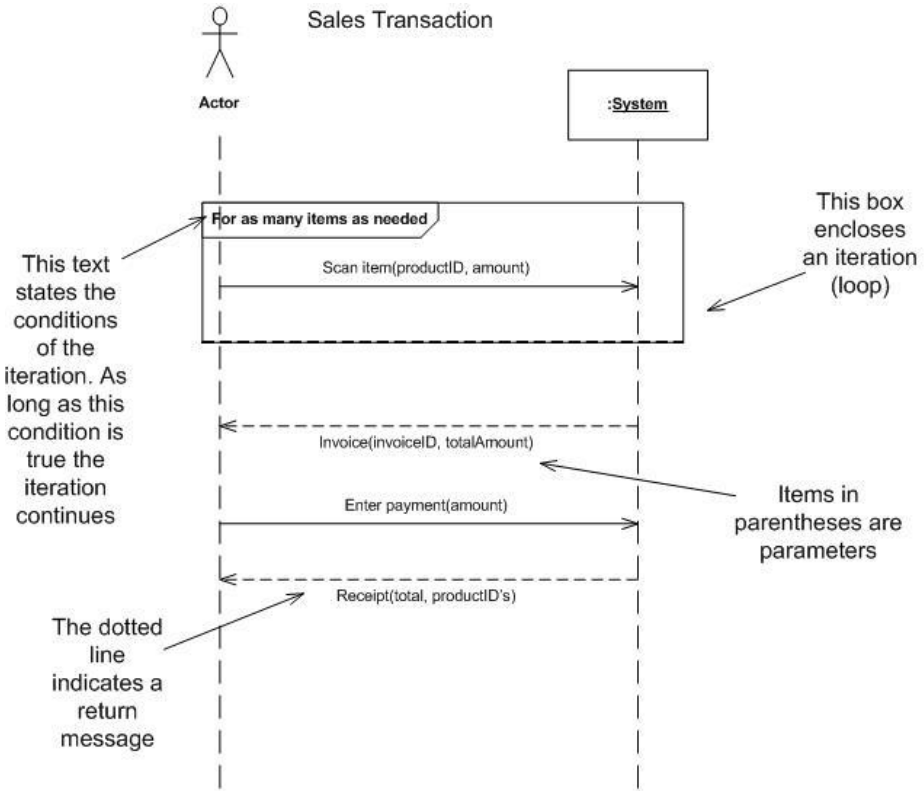


- General comments and notes describing the use case are needed
- <include> : One Use Case could include the functionality of another as part of its normal processing.
- <extend> : One Use Case can extend the behavior of another, typically when exceptional circumstances are encountered.

From Specification to Design

- The scenarios in the textual description of the use cases can be seen as instances of use cases and are illustrated by system sequence diagrams (one by use case)
- At a minimum, the nominal scenario of each use case should be represented by a sequence diagram that accounts for the interaction between the actor(s) and the system.
- The system is considered here as a whole and is represented by a lifeline. Each actor is also associated with a lifeline.

Sequence System Diagrams … an example



Design

Several points of view

- The user's point of view (already addressed during the needs analysis)
- The logical point of view: static view of the internal components of the system : ***UML class diagrams***
- The implementation point of view: decomposition into modules to be coded, tested and modified independently : ***Components or packages UML diagrams***
- The process view: dynamic vision of the system, how components interact over time : ***Interaction UML diagrams***

Design

Using the simplified SE approach introduced earlier, we will only use UML class diagrams and interaction diagrams for design

UML Class diagrams

- Used to describe the static structure of the domain

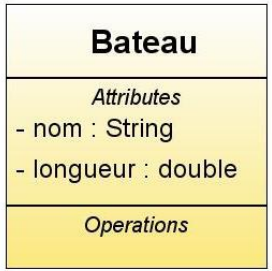
- **Concepts** (or classes or entities) are represented by boxes.
  A concept represents a set of concrete or abstract objects with their own identity. Ex: Person, Car
- **Instance or Class Occurrence**; A particular entity.
  Ex person Jean Dupont, François; Picasso with registration number 222 ABE 67
  - **Attribute** or Characteristic or Property
    Atomic information relating to an entity. E.g.: the name of a person, the registration of a car
    Each entity can only have one value for a given attribute.
  - **Attribute Occurrence**
    The value of an attribute for a particular instance. Ex "222 ABE 67" for the attribute "registration" of the authority "my car"

UML Class diagrams

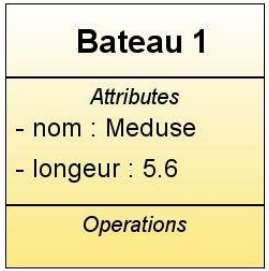
WARNING

- A class defines a set of objects with common characteristics (attributes).
- Attributes are therefore defined at the class level
- An instance has a value for each of the attributes defined at the class level

UML Class diagrams



class



instances

UML Class Diagrams

Association or relationship

- Semantic link between several entities.
- Ex: owner (between Person and Car)

Relationship Occurrence

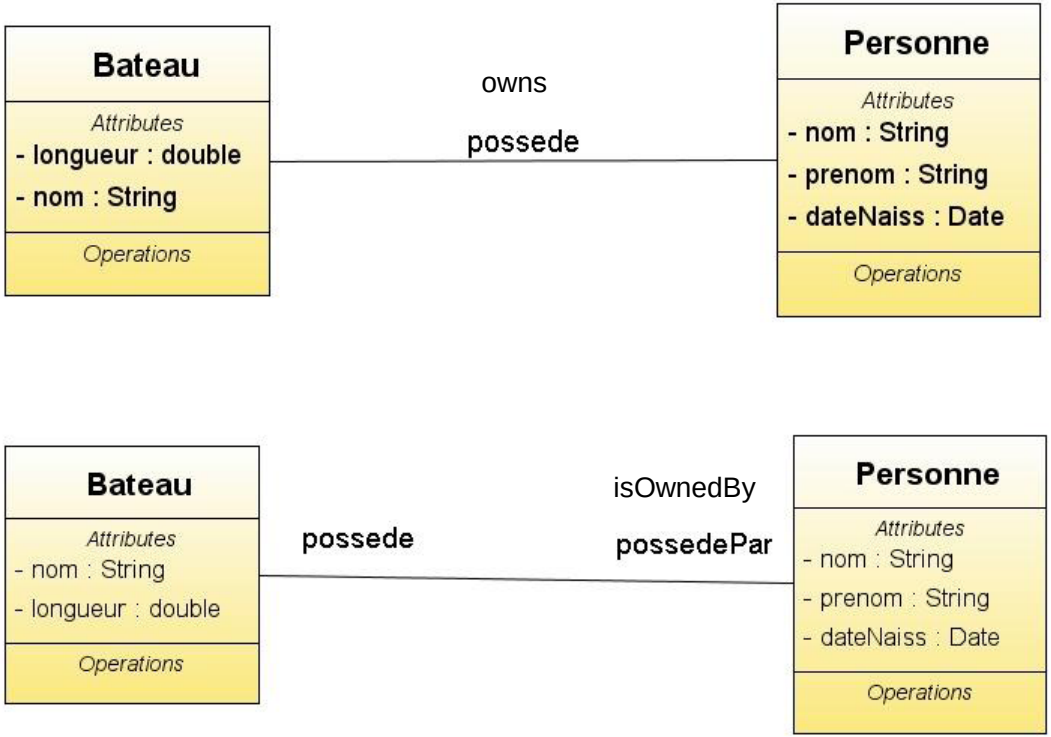
- A tuple of instances.

UML Class diagrams

The associations are represented by simple arrows.

- Can have a name and also a multiplicity.
- Bidirectional associations are represented by a single line, and unidirectional associations by a single arrow

UML Class diagrams



UML Class diagrams

Let R be a relationship between entities E1 and E2

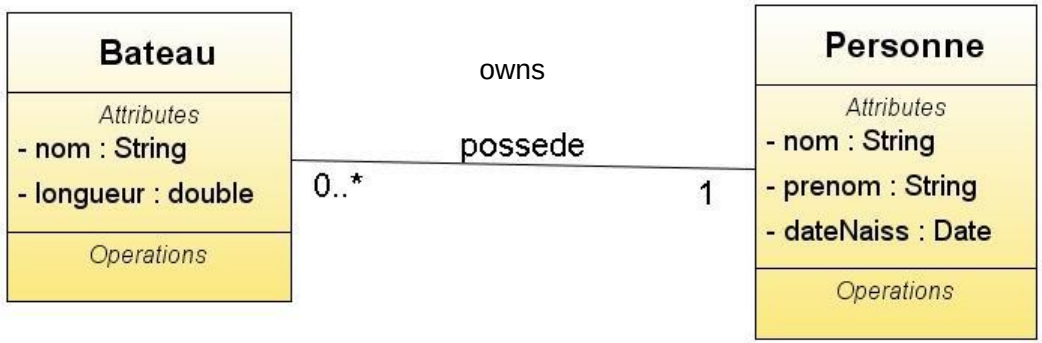
- For e1 an instance of E1, how many (minimum/maximum) entities e2i of E2 can be related to e1
- Similarly, for e2 an instance of E2, how many (minimum/maximum) entities e1i of E1 can be related to e2

owns

UML Class diagrams

Let R be a relationship between entities E1 and E2

- For e1 an instance of E1, how many (minimum/maximum) entities e2i of E2 can be related to e1
- Similarly, fo
  (minimum/maxi
  related to e2



UML Class diagrams

An entity E1 generalizes an entity E2 if any instance of E2 is also an instance of E1. (E2 is included in E1). We can also say:

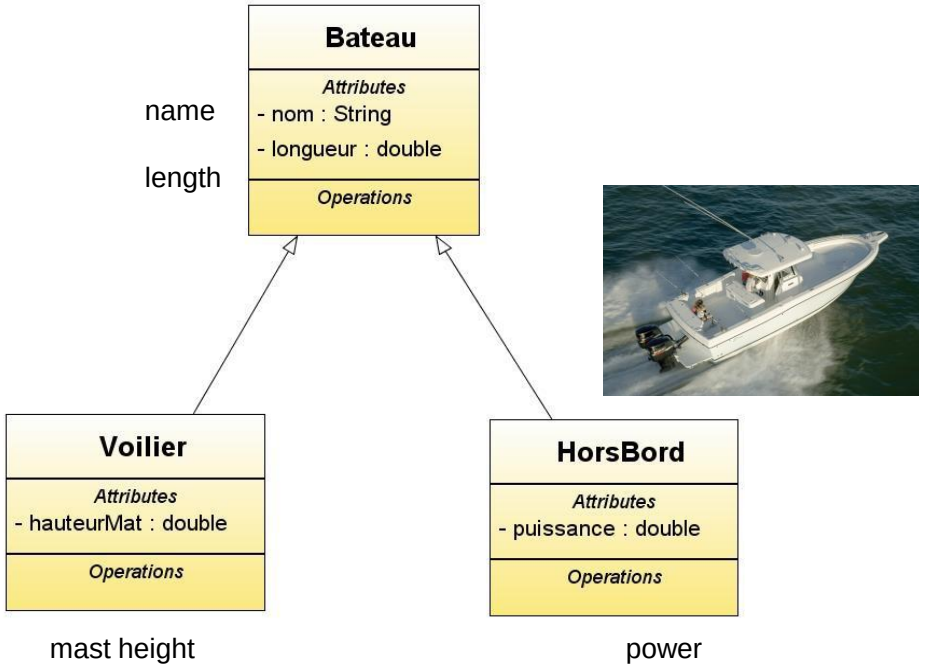
- E1 subsumes E2
- E2 specializes E1
- E2 inherits from E1

Ex: Woman specializes Person

Represented by a small hollow triangle. The arrow points to the superconcept

UML Class diagrams

Inheritance: If E1 specializes E2, any instance e of E1 is a fortiori instance of E2, so has all the attributes of E2, and can intervene in all the relationships defined for E2.



UML Class diagrams

A white "diamond" represents an **aggregation**, which is more specific than associations

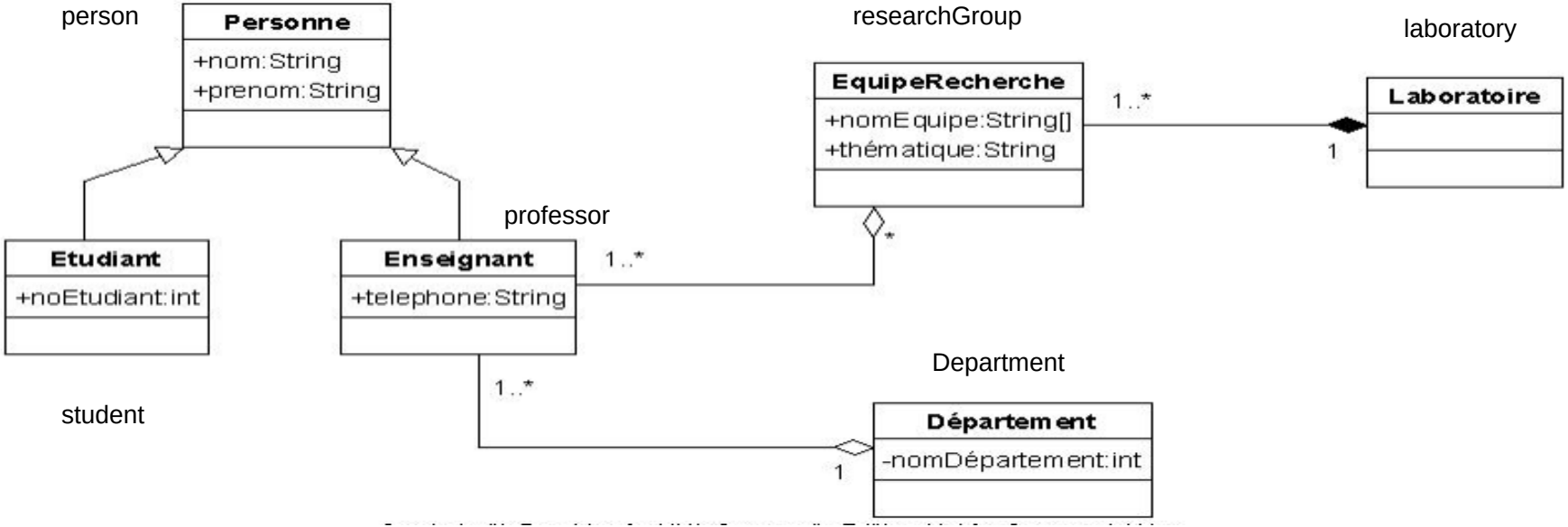
- It is an association that represents a "part-whole" type of relationship
- They can have names and cardinality
- Aggregation occurs when a concept "contains" other concepts. If the container is destroyed, the contents remain.

UML Class diagrams

A solid "diamond" represents a **composition**, a stronger association

- More specific than an aggregation
- In this case, if the container is destroyed, the contents are also destroyed.

Aggregations vs Compositions: an example



Let's return to the example

Goal of the development

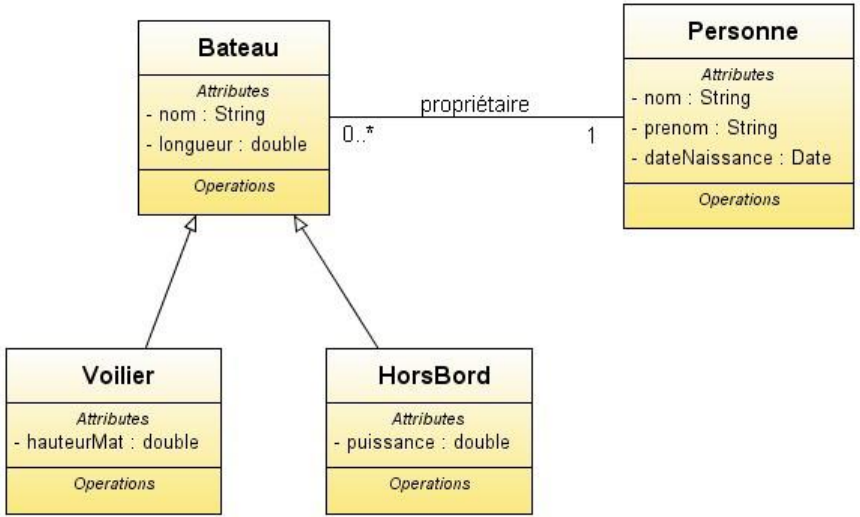
- In a marina, there are sailing boats and outboard boats. The boats have a home port and move from port to port
- We want to keep track of the boats' movements and be able to contact their owner

Let's return to the example

Features to be developed

1. Add / delete / modify a boat
2. Find the owner of a boat
3. Calculate the annual tax of a boat in its home port. The tax depends on the length of the boat, and its power if the boat is motorized
4. List the boats for a given home port
5. List the ports visited by a boat
6. Make the history of a boat's movements
7. List the ports sorted by number of attached boats
8. Send a letter to each owner of a boat in a port specifying the amount of the tax

A first version of the class diagram



Ports and home ports

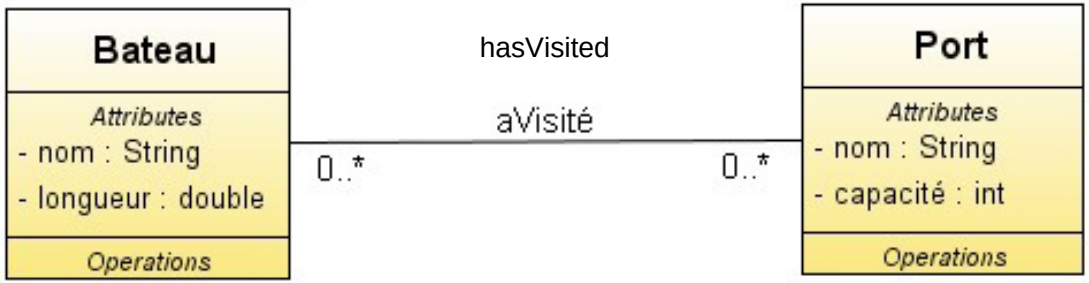
We also need to define the **Port** class, and a **portDAttache (homePort)** association that connects boats and ports.

A boat has at most one home port, but a port is the home port of several boats.

The aVisité (hasVisited) relationship

Point 5 of the needs analysis indicates a new association between Port and Boat.

A ship may visit more than one port, and a port may have been visited by more than one ship.

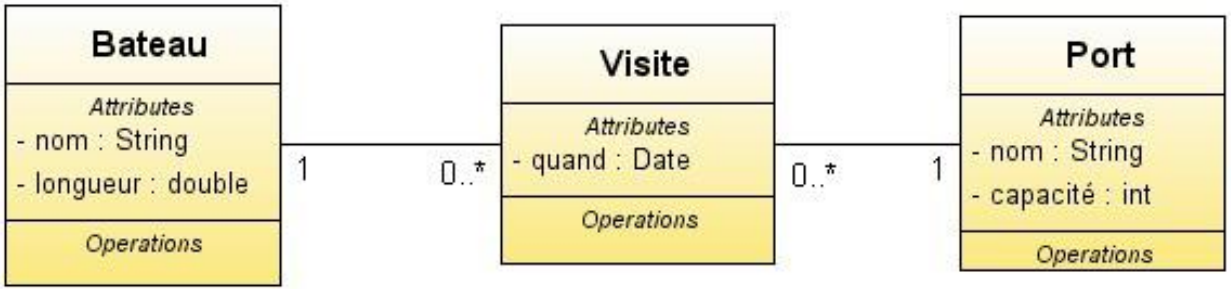


History of visits

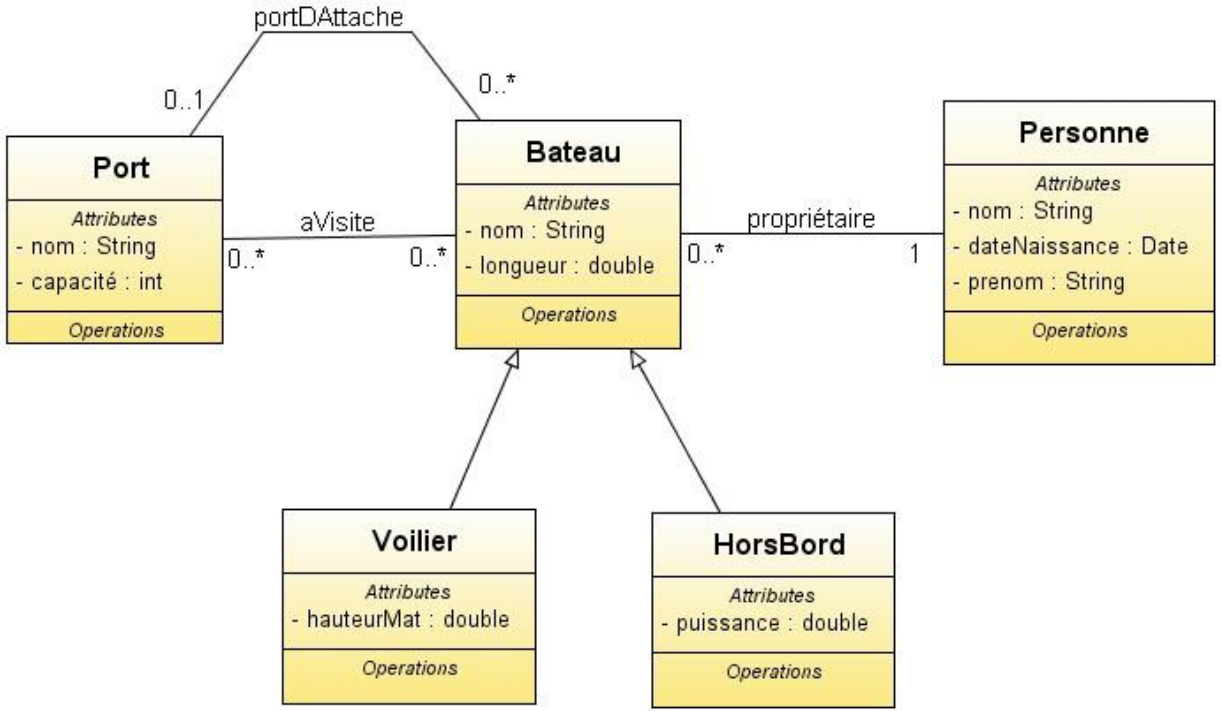
- To be able to make the history of a boat's movements, we need to know when it visited the different ports.
- This information is not a boat attribute (remember that an attribute can only have one value, or a boat can visit several ports, with several visit dates), and is not a port attribute (for the same reason).
- This information should therefore be associated with the relationship itself.

History of visits

- We create a normal class that corresponds to the relationship. This transformation id sone by **reifying** of the relationship.



The whole diagram



What is lacking

The verification of the rest of the needs

- List the ports sorted by number of boats attached to this port
- Sending an invoice

Mini-Project

Mini-Project

The goal of this project is to develop a distributed calculator, meaning with a client/server architecture on positive integers.

Constraints

1. Your calculator will only work on positive integers.
2. It will support 4 types of operation: addition, subtraction, multiplication and integer division.
3. As a request from the client, the server will receive an operation to perform on 2 integers, and as a reply the server will provide the answer or the associated error code.

Mini-Project

4. Several exceptions need to be defined and managed as it is a calculator on positive integers: negative operand, negative result, division by zero or even syntax error in the expression.
5. You will program a Java version for both the client (with a GUI) and the server.

Some tips for the project

- When you only have access to the `.class` files from an external API, you can use the command
  `javap <ClassName>`
  This command will give you a list of the constants and public methods defined in this class.
- Most graphical interfaces are defined by creating a new class as extensions of the `JFrame` class. You will need then to define a `GraphicalCalculator` class with an attribute to send the requests to the distant calculator.

Some tips for the project

- It is generally preferable to first group the different components of the interface in different `JPanels` that are integrated in the `Container` of the main window(`JFrame`). (see the method `Container getContentPane()` of the `JFrame` class to obtain the `Container` of a `JFrame`).
- Define then two `JPanels`. The first one will be used for the keyboard of the calculator. The second one will be used for the display itself and possibly for some other components to configure the display on the screen. In order to be able to view these two `JPanels`, give them different background colors...

Some tips for the project

- The screen can be defined as `a JTextField`, which will be added to the `JPanel` of the screen. However (at least in the first instance) it should not be directly editable.
- To make the keyboard of the calculator, you will use `JButton` components. Since these components are used to control the interface, it is necessary to declare them as attributes of the `GraphicalCalculator` class.

Some tips for the project

- JButtons are generally associated with ActionListener managers to react to mouse clicks. Here each button on the keyboard has a different effect. We can imagine to create a different manager for each button, but with several buttons, it becomes heavy!..
- Another way or to do is to create only one manager, that we associate with each button, but whose ActionPerformed method is capable to adapt its action according to the button that was clicked.
- To identify the button that generated the event, the getSource method of the EventObject class can be used.

Deliverables

Specification document

- Introduction
  - This is an informal description of the project and its context.
  - It should include, among other things, the following information
    - A list of the main functions,
    - The different users and their characteristics,
    - hardware and software constraints.
  - Detailed requirements
    - It is the contractual party strictly speaking since it formalizes the need
    - It consists of 3 distinct parts:
      - the functional specifications
      - interface specifications
      - operational specifications (performance, safety, ...)
  - These different elements can be based in use case diagrams, a domain model diagram, mock-ups and a navigation diagram in UML, depending on the needs.

Design document

- Three sections
  - Introduction
  - Preliminary design
  - Detailed design
- Introduction
  - This is an informal description of the project and its context.
  - It is often relatively similar to the introduction to a specification document.
  - It should therefore include the following information
    - the list of the main functions,
    - the different users and their characteristics,
    - hardware and software constraints.

Design document

- Preliminary design
  - This step consists in carrying out a macroscopic design, i.e. leading to a breakdown into packages with the external signatures of each package.
  - This step can be based in UML on :
    - A domain model diagram (if not specified) ;
    - System sequence diagrams (if not specified);
    - Mock-ups (if not specified);
    - Navigation activity diagrams (if not specified);
    - Interaction diagrams;
    - Preliminary design class diagram;
    - A breakdown into packages and the external signatures of each package.
    - A deployment diagram

Design document

- Detailed design
  - This step consists in detailing by package the elements constituting them.
  - Concretely, it is mainly a matter of specifying the class attributes and methods of all participating classes and grouping them together in a class diagram.
  - The methods of a package that will be considered as non-trivial will have to be commented and their operation detailed by pseudo-code.
  - **The author of each of the methods should also be given in the comments**

Final report

- Must contain
  - Specifications (Specifications document updated if necessary)
  - Design (Design document updated if necessary),
  - Justified technical choices
  - Possible improvements (not foreseen in the specs)
  - Evaluation of the performance of your implementation of the algorithm on the suggested functions

Deadlines

Planning

- **You will prepare the project in groups of 3 students**
  - https://docs.google.com/spreadsheets/d/15njS4UVaC60ucW3_MCwM6mvjTQ4vqbJKcTsOTLsQdtM/edit?usp=sharing
- **Monday March 8th at 20h00 on Moodle**
  - Specification document
- **Monday March 15th at 20h00 on Moodle**
  - Design document

Planning

- You will have 8 practical work sessions with Seymour, who will assist you with the development
- Feedback on your specification and design will be given asap, to allow you to modify / correct for your development
- **Final deadline for the project with all the deliverables to be agreed in the future**