Abdelhafid Boussouf
University Center
Mila

Faculty of Math and Computer Science

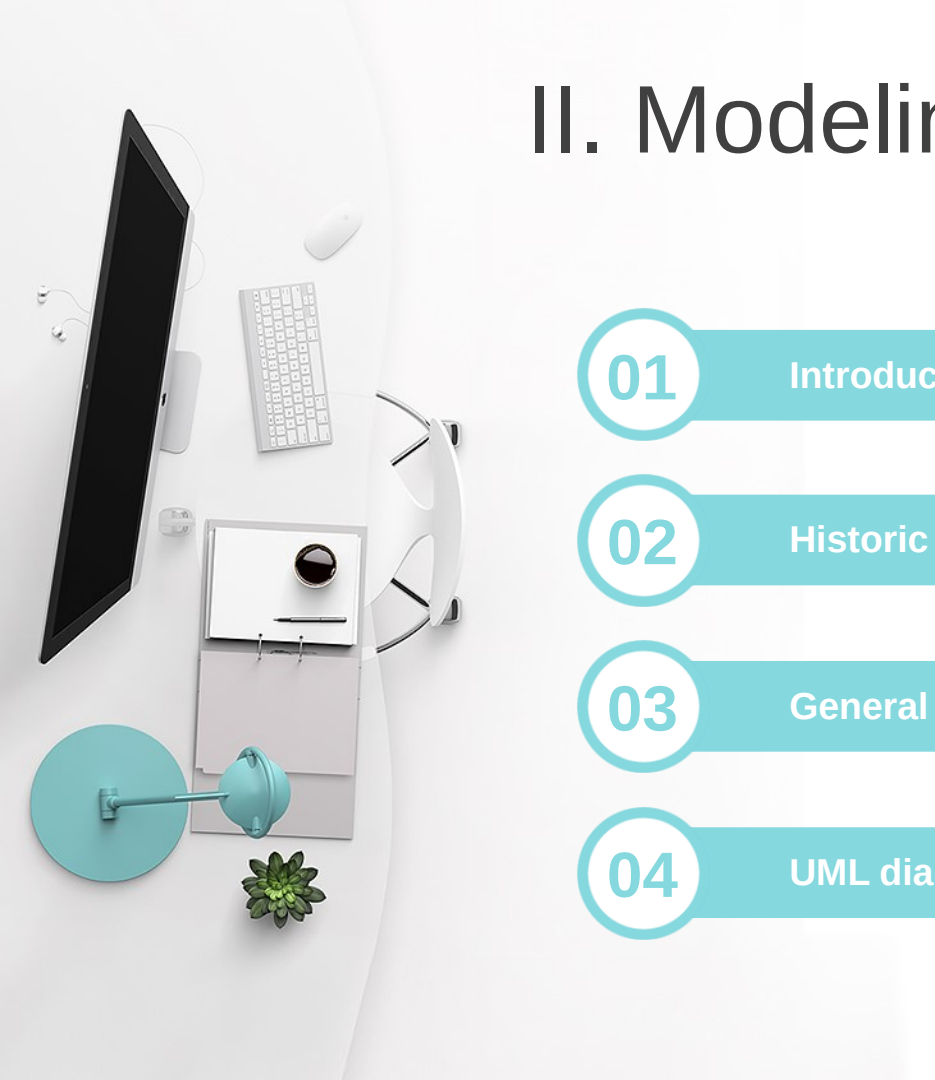
Department of Computer Science

Software engineering

Chapter 2

Modeling with UML

Mrs. S.HEDJAZ



II. Modeling with UML

01 Introduction

02 Historic

03 General elements and mechanisms

04 UML diagrams

01 Introduction

Modeling:

The act of designing a model in a dedicated modeling language. Modeling consists of identifying the interesting or relevant characteristics of a system in order to be able to study it from the point of view of its characteristics.

What is a model?:

- An abstraction to better understand a complex object (representing the system according to different degrees of detail)A simplified representation of a reality (software, building, … etc.)
- The models are often graphic (a small drawing is better than a long speech)

Good model:

- ✓ It must facilitate the understanding of the phenomenon (system) studied
- ✓ It must make it possible to simulate the phenomenon (system) studied

01 Introduction

Why model?

- Making a system easier to understand
- Also allow good communication with the customer
- Allows you to accurately and completely write down requirements without knowing the details of the system
- Allows you to focus on specific aspects of a system (offers different visions)
- Enable testing of a multitude of solutions at a lower cost and in a shorter time frame

01 Introduction

Modeling Method:

A method defines a reproducible approach to obtain reliable results. So, a method of software development describes how to model and build software systems reliably.

Functional modeling methods:

Functional and systemic methods were first imposed (the 70s and 80s), these methods were directly inspired by the architecture of computers. The separation between data and processing as it physically exists in the material to be transposed to the methods.

01 Introduction

Disadvantages:

- A simple software update at a given point can cascade a multitude of other functions
- The major evolution of the software multiplies the maintenance points (the software must be retouched in its entirety)

Solutions:

- Centralizing the data of a type and the associated processing in a single physical entity makes it possible to limit the maintenance points in the code

Solution → Object-oriented methods (OO)

01 Introduction

Important concepts of the object approach:

The object: is a self-contained entity, which brings together a set of consistent properties and associated processing. Such an entity is the founding concept of the approach of the same name.

Encapsulation: consists of hiding the implementation details of an object, by defining an interface. The interface defines the services that are accessible (offered) to the users of the object.

Encapsulation facilitates the evolution of an application because it stabilizes the use of objects: you can modify the implementation of an object's attributes without changing its interface.

01 Introduction

Important concepts of the object approach:

Inheritance: is a mechanism for passing properties from a class (its attributes and methods) to a subclass. Inheritance prevents duplication and encourages reuse.

Polymorphism: represents the ability of the same operation to execute differently depending on the context of the class in which it is located. For example, an operation defined in a superclass can run differently depending on the subclass where it is inherited. Polymorphism increases the genericity of the code.

Aggregation: This is a relationship between two classes, specifying that the objects of on e class are components of the other class. Aggregation allows you to assemble basic obj ects, in order to build more complex objects.

02 Historic

BOOCH method:

Distinguishes 2 levels:

- **Logic**
  - ✓ Class diagrams
  - ✓ Instance diagram
  - ✓ State/transition diagram
- **Physics**
  - ✓ Module diagrams (package principle)
  - ✓ Process Diagram

Grady Booch

02 Historic

OMT (Object Modeling Techninque):

Three axes:

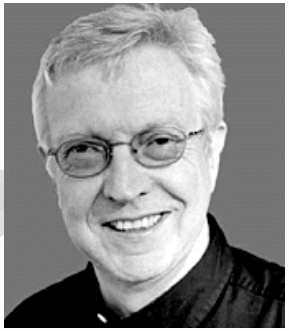
- **Static:** Identifies the properties of objects and their links with o ther objects
- **Dynamic: Defines the life cycle of objects:** behavior of objects, t he different states by and the events triggering these changes of state
- **Functional:** specifies the functions performed by the objects through methods.

James rumbaugh

02 Historic

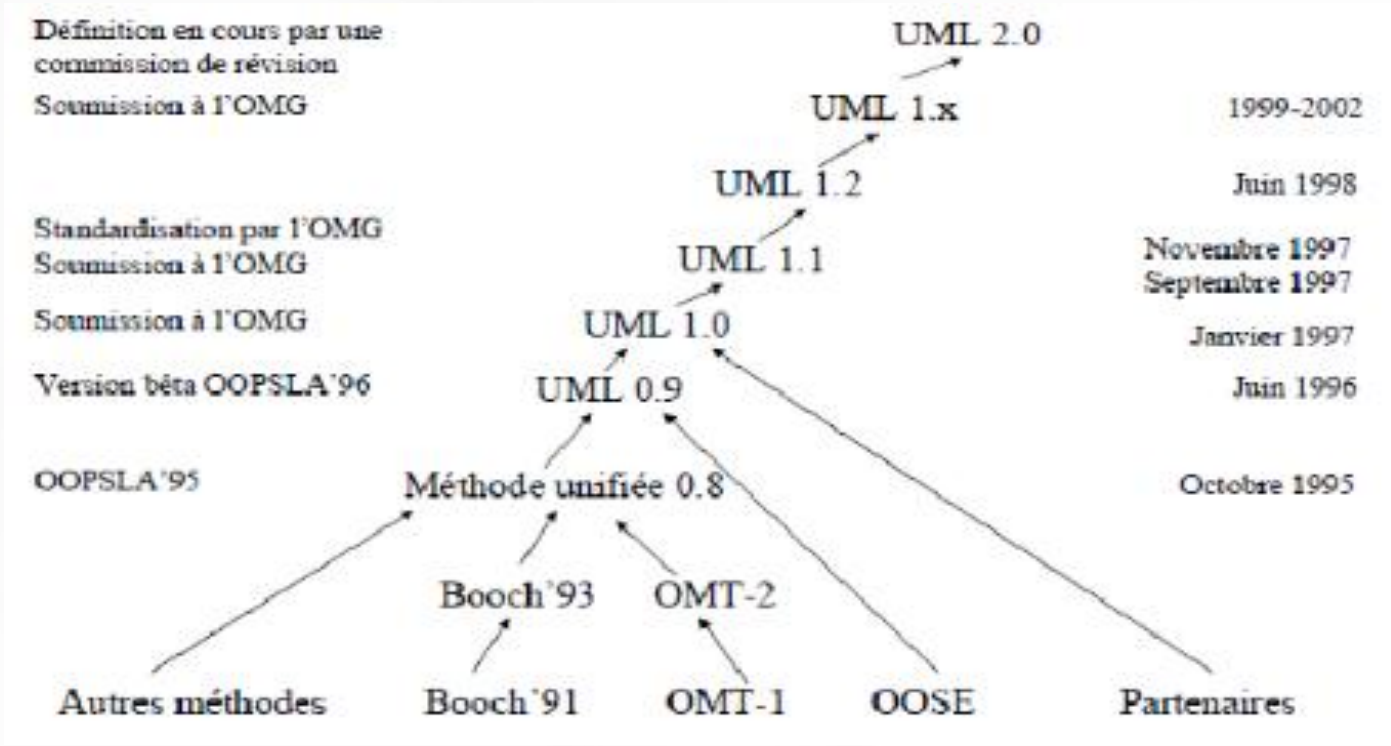
OOSE (Object Oriented Software Engineering):

- ✓ **Five models:**
  - Description of requirements
  - Analysis
  - Conception
  - Implantation
  - Test
- ✓ **3 types of objects**
  - Entities
  - Controls
  - Interfaces
- ✓ **Use Case Concept: Use Cases**



Ivar Jacobson

02 Historic


Définition en cours par une commission de révision
UML 2.0
Soumission à l'OMG
UML 1.x
1999-2002
UML 1.2
Juin 1998
Standardisation par l'OMG
Soumission à l'OMG
UML 1.1
Novembre 1997
Septembre 1997
Soumission à l'OMG
UML 1.0
Janvier 1997
Version béta OOPSLA'96
UML 0.9
Juin 1996
OOPSLA'95
Méthode unifiée 0.8
Octobre 1995
Booch'93
OMT-2
Autres méthodes
Booch'91
OMT-1
OOSE
Partenaires

03 General elements and mechanisms

Why UML?

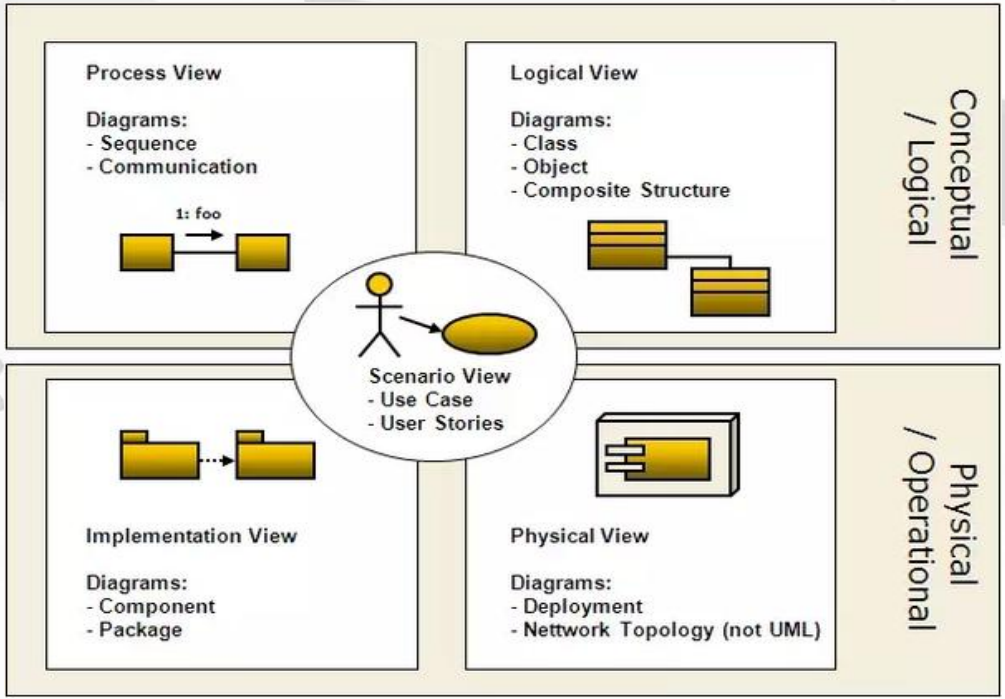
- Graphical and simple
- Uml is a standard
- Represent entire systems
- Common language that can be used by all methods and adapted to all phases of development

UML is not a method, it is an object modeling language

- Iterative and incremental
- Guided by user needs
- Architecture-centric

03 General elements and mechanisms

Architecture 4+1



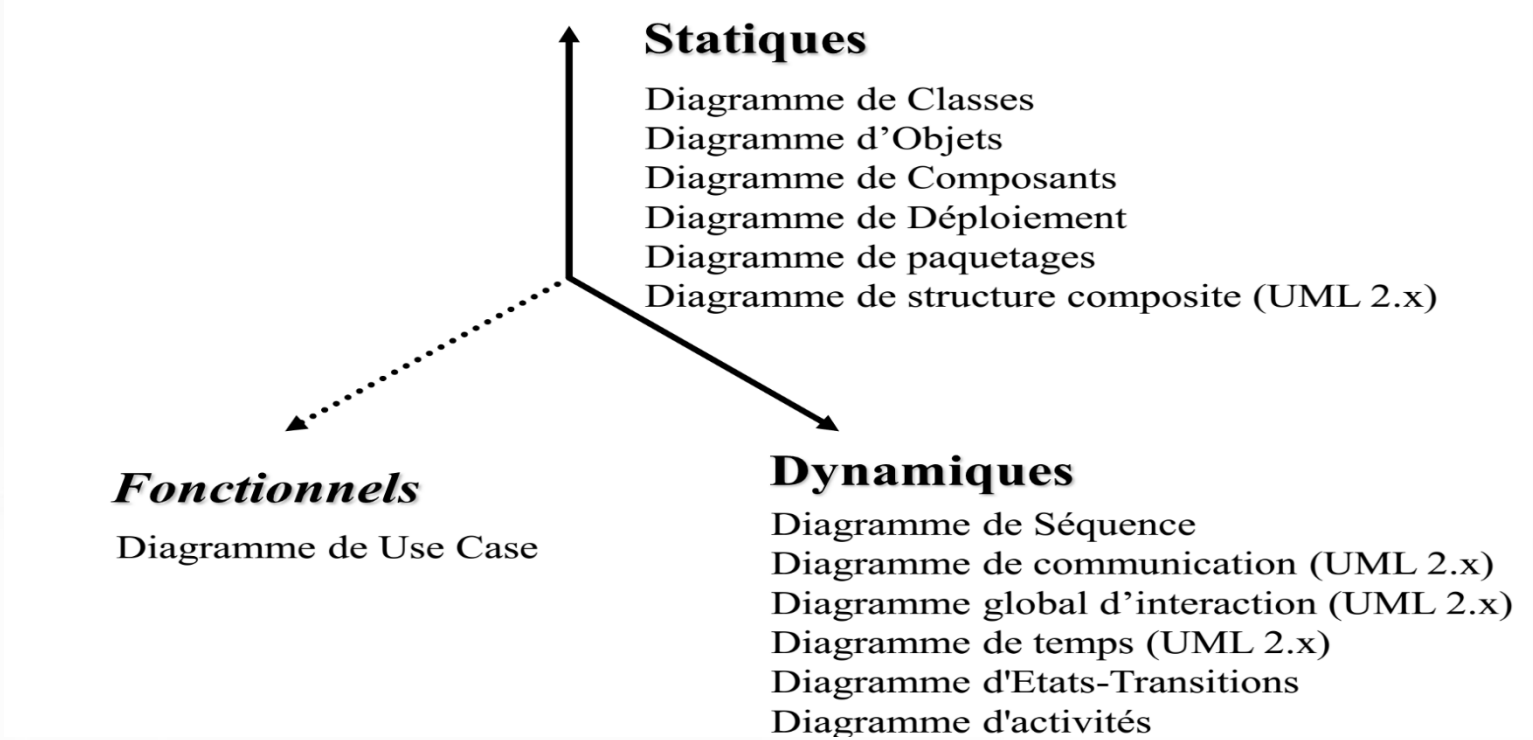
03 General elements and mechanisms

Architecture 4+1

- **The view of the use cases:** this is the description of the model "seen" by the actors of the system. It corresponds to the needs expected by each player
- **The logical view:** this view expresses the abstract perspective of the solution in terms of classes, objects, relations, transition state machine, etc.
- **The implementation or component view:** this view defines the dependencies between the modules
- **The behavior or process view:** this is the temporal and technical view, which implements the notions of competing tasks, stimuli, control, synchronization, etc.
- **The deployment view:** This view describes the geographic position and physical architecture of each element of the system

03 General elements and mechanisms

Modelling axes with UML:



04 UML diagrams

1. Use Case Diagram:

It makes it possible to identify the possibilities of interaction between the system and the actors (external stakeholders), i.e. all the functionalities that the system must provide.

2. Class Diagram:

It represents the classes involved in the system.

3. Object Diagram:

It is used to represent the instances of classes (objects) used in the system.

04 UML diagrams

4. Component Diagram:

It allows you to show the components of the system from a physical point of view, as they are implemented (files, libraries, databases, etc.)

5. Deployment Diagram:

It is used to represent the hardware elements (computers, peripherals, networks, storage systems, etc.) and how the components of the system are distributed on these hardware elements and interact with each other.

6. Package diagram:

A package is a logical container for grouping and organizing elements in the UML model. The package diagram is used to represent the dependencies between packages, that is, the dependencies between sets of definitions.

04 UML diagrams

7. Composite Structure Diagram:

Allows you to describe the relationships between components of a class in a white-box format.

8. Sequence Diagram:

Sequential representation of the processing process and interactions between the elements of the system and/or its actors

9. Communication Diagram:

Simplified representation of a sequence diagram that focuses on the exchange of messages between objects

04 UML diagrams

10. Global Interaction Diagram:

Allows you to describe the possible sequences between previously identified scenarios in the form of sequence diagrams (variant of the activity diagram)

11. Time Diagram:

Allows you to describe the variations of a piece of data over time

12. Transition State Diagram:

Allows you to describe the behavior of the system or its components in finite state machine form

13. Activity Diagram:

Allows you to describe the behavior of the system or its components in the form of a flow or chain of activities

Bibliographies

- **Uml 2 pratique de la modélisation**, Benoît Charroux, Yann Thierry-Mieg, Aomar Osmani https://fr.slideshare.net/nassimamine3994/uml-2-pratique-de-la-modlisation
- **Uml 2 par la pratique, Pascal roques**
- **Les cahiers du programmeur, Pascal roques**
- …