Software Engineering Course

Chapter 2
Modeling with UML

2020-2023

Dr. Sofia Kouah

Plan

1. Introduction: Modeling, Model, Object Oriented Modeling, UML in application.

2. General Elements and Mechanisms

3. UML diagrams

4. Packages

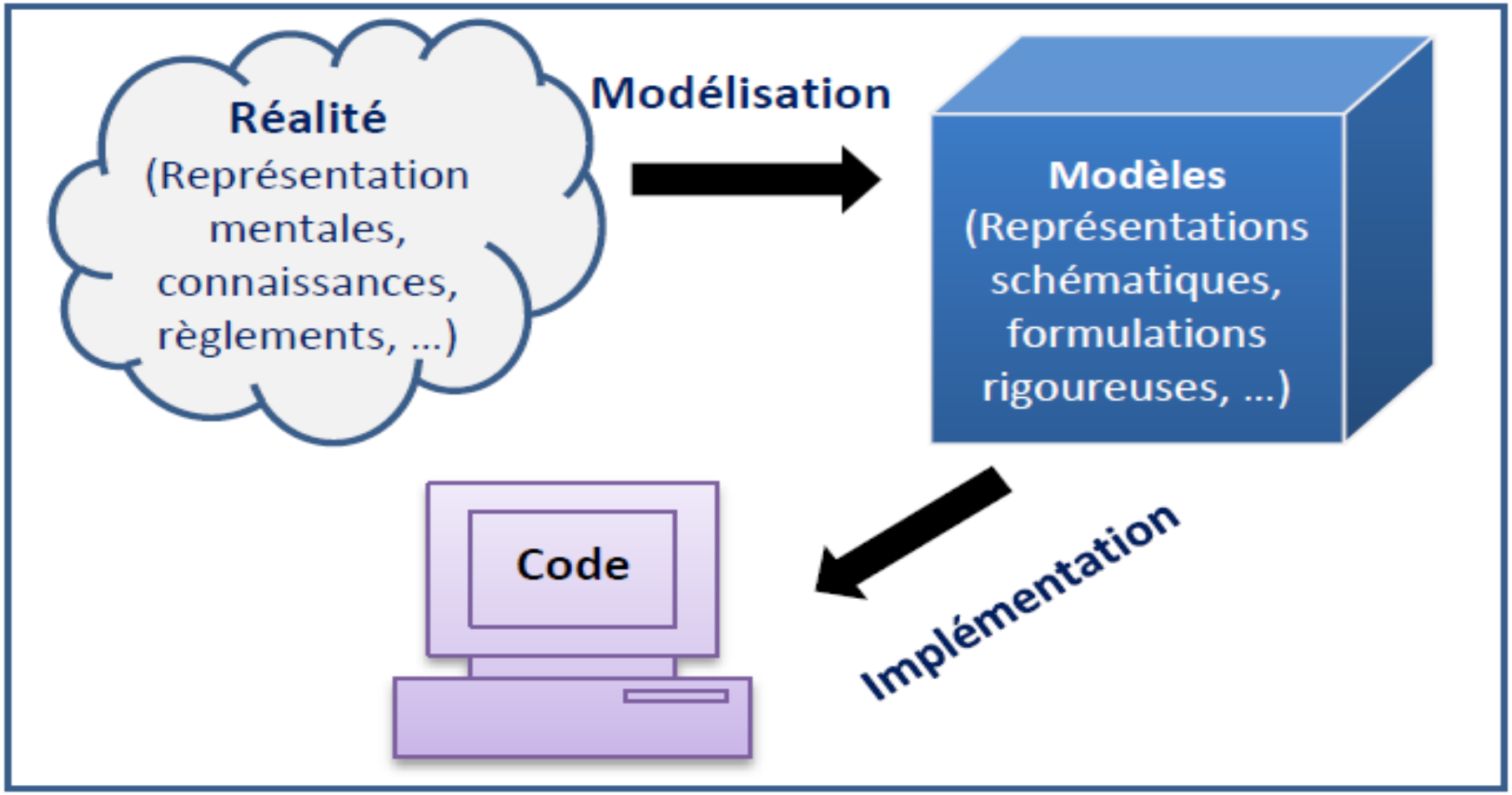
Introduction

Modeling

- ✓ **Model:** Simplification of reality, abstraction, subjective view
- ✓ **Examples:** meteorological, economic, demographic models
- ✓ **Modeling a concept or an object, why?**
  - ✓ to better understand it (modeling in physics)
  - ✓ to build it better (Engineering Modeling)
- ✓ **In software engineering:**
  - ✓ Modeling = specification + design
  - ✓ Help the realization of a software from the requirements of the customer.

Introduction

Modeling



Introduction

Model, Method, Tool ?

- ✓A **Model** is an abstraction of reality. it is an abstract representation of all or part of reality.
- ✓A **model** makes it possible to capture the relevant aspects responding to an objective defined in advance.
- ✓A **Tool** is an automated or semi-automated **medium** to support methods (Formalism or language for expressing a model).
- ✓A **Method** is the way we will conduct an activity, it uses models and tools, and follows an implementation process.

Introduction

Software design methodology provides a logical and systematic means of proceeding with the design process as well as a set of guidelines for decision-making.

Introduction

Design methods

1) Functional method

✓ **System** = set of functions

✓ **System state** (data): centralized and shared by functions.

2) Object-oriented methods

✓ **System** = set of objects

✓ **Object** = data + functions

✓ **Examples of object-oriented methods:**

- OOA (Object-Oriented Analysis), Coad.

- OOD (Object-Oriented Design), Grady Booch.

OMT (Object Modeling Technique), James Rambaugh.

OOSE (Object-Oriented Software Engineering), var Jacobson.

3) Data-Driven Methods

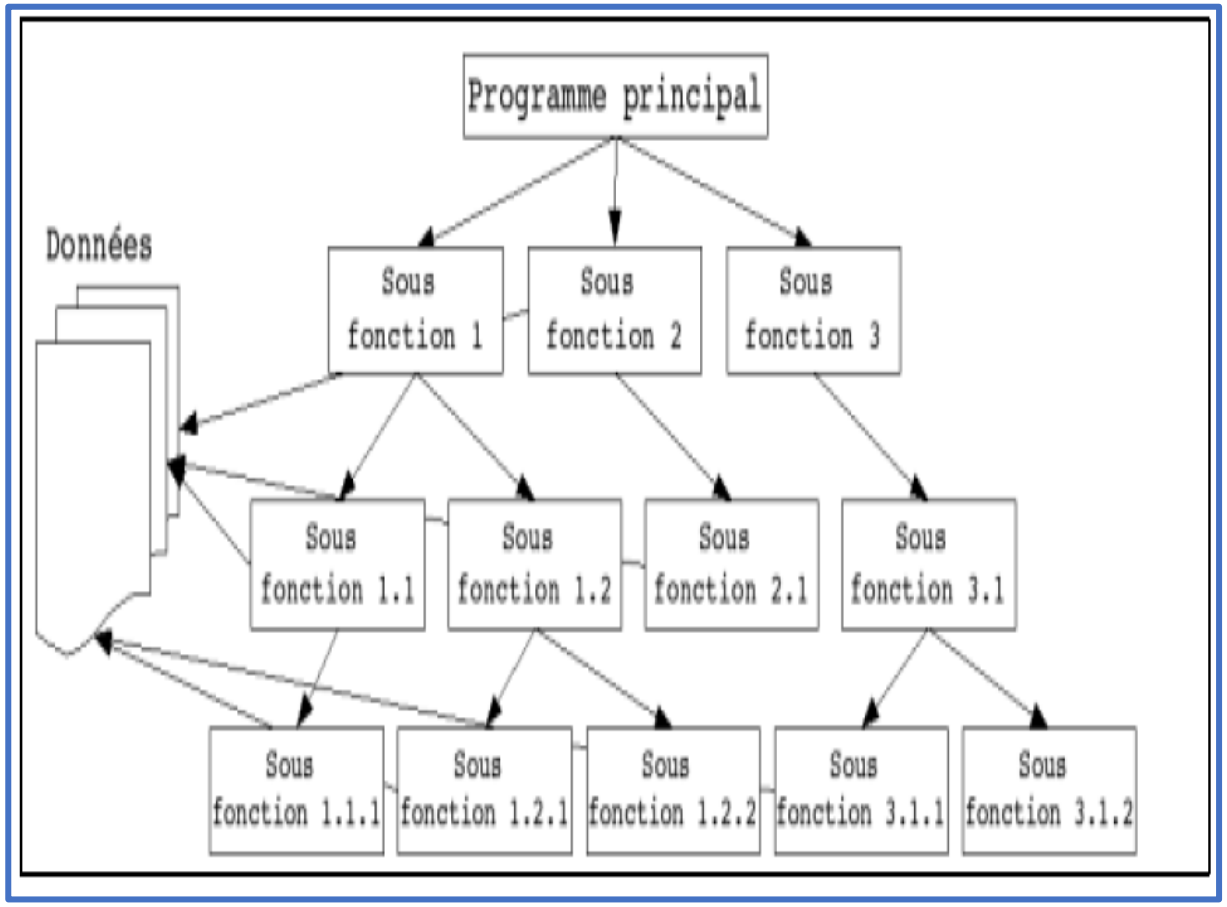
✓ The structure of a system can be deduced from the structure of the data appearing at the input and at the output of this system.

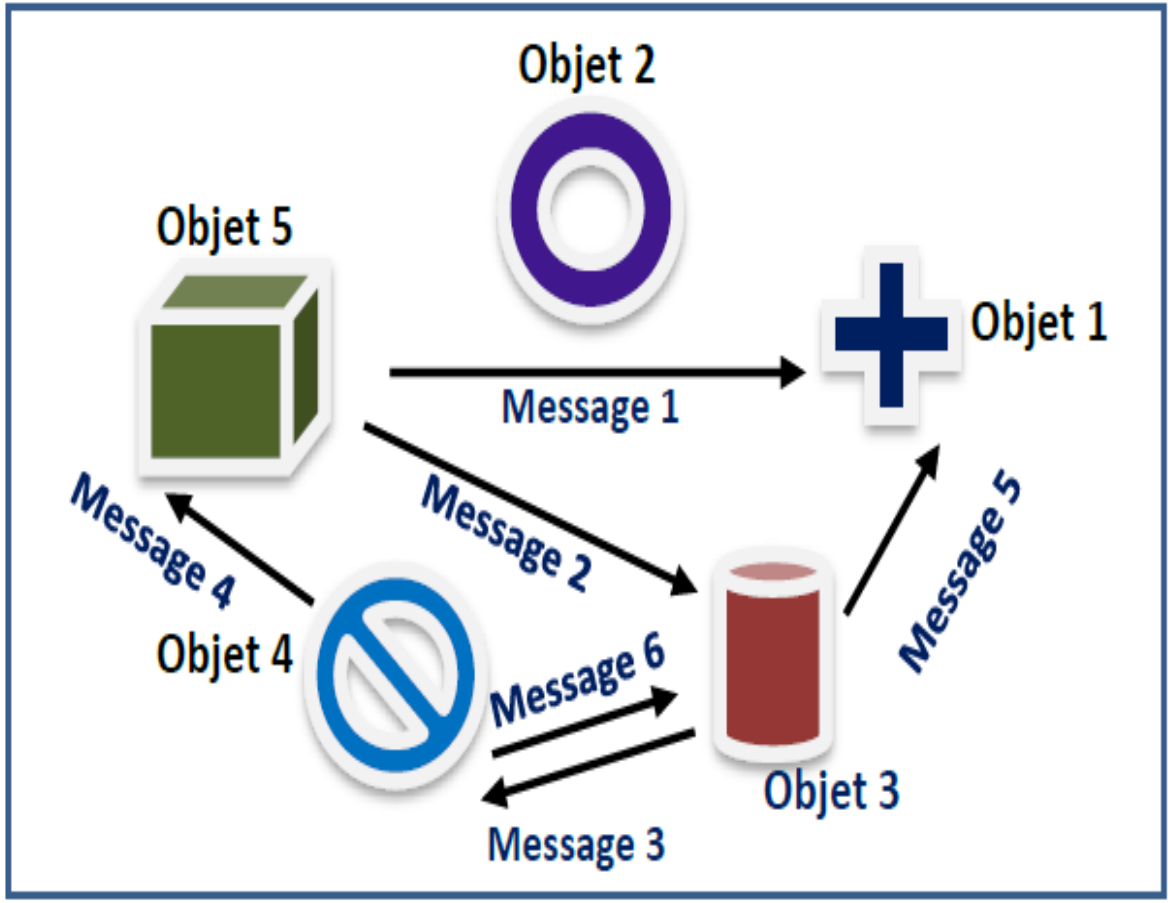
4) **Systematic methods**

✓ Merise

Introduction

Functional method Vs Object-oriented methods





Introduction

Object Oriented Modeling

✓Principles:

- Application domain concept = **object**
- Described by: **state** (attributes) + **behavior** (operations)
- Links between concepts: inheritance, aggregation, composition, etc.

✓Characteristics of objects

- **Identity**: Object = Single entity (same attributes ⇏ same object)
- **Classification**: grouping of objects of the same nature (attributes + operations)
- **Polymorphism**: different behavior of the same operation in different classes
- **Inheritance**: hierarchical sharing of attributes and operations

Introduction

UML in application

UML: **Unified Modeling Language**

✓**Language :**

- Syntax and writing rules
- Standardized graphical notations

✓**Modeling language:**

- Abstraction of the functioning and structure of the system
- Specification and design

✓**Unified language:**

- Merger of several previous notations: Booch, OMT, OOSE
- Standard defined by the **Object Management Group** (OMG)

Introduction

UML in application: **Unified Modeling Language?**

- ✓**Graphical language** for **visualization, specification, documentation, and building** Software.
- ✓**Graphical language**: **a set of diagrams** to software model according to **different views** and **at different abstraction levels.**
- ✓**Object-oriented modeling**: modeling the system as a **set of objects that interact together.**
- ✓UML is not a design method

Introduction

UML in application: Why UML ?

✓**Need to model in order to build software:**

- Modeling of static and dynamic aspects
- Modeling in multiple views and at different abstraction levels.

✓ **Independent of the development process**

✓Need for **standardized languages** for modeling

- Semi-formal language
- Standard widely used

✓**Object-oriented design**

General Elements and Mechanisms

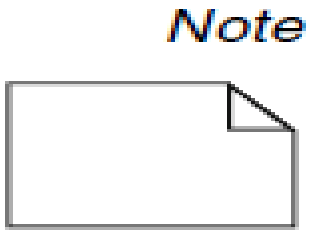
✓**NOTE**: is a concept that could occur in **all UML diagrams**.

✓It can comprise any **form of expression** that **specify the diagram and its elements more precisely**.

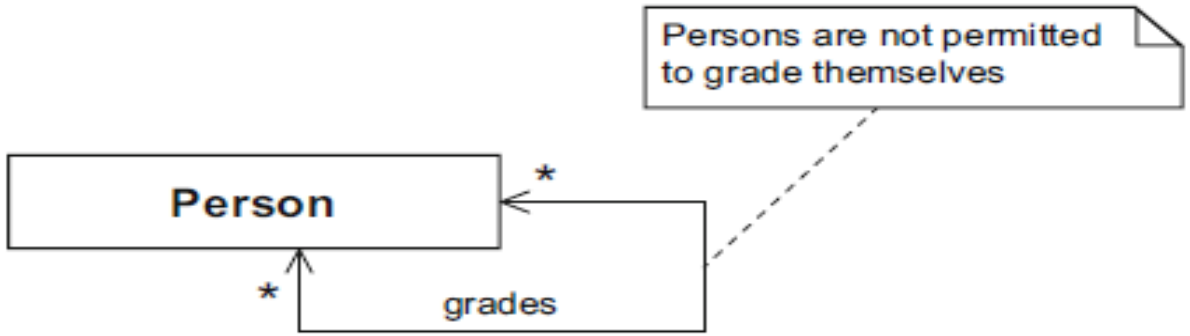
✓**Example**: natural language or Object Constraint Language (OCL).

✓It may be attached to all other model elements.

✓**Notation**:



Example:



General Elements and Mechanisms

- **Encapsulation**:
  - is the mechanism that **protect objects against unauthorized access** to their internal state, through a **uniquely defined interface**.
  - These **various access authorizations** can be ensured by **different visibility levels** of the **corresponding interfaces**.
- For instance, In Java, there is **explicit visibility markers:**
  - public: access **for all**.
  - private: access **only within the object**.
  - and protected: access **for members of the same class, its subclasses, and within the same package.**

General Elements and Mechanisms

- **Messages**: mechanism that ensures communication between Objects.
- The intuition behind sending a message to a given object is to **request** the **execution of an operation**.
- Thus, the object, when receiving a message, decides itself whether and how to execute the required operation, depending on the associated access authorization defined through visibility and a suitable implementation. .

General Elements and Mechanisms

- ✓**Inheritance**: is a mechanism that allows **deriving new classes** from existing ones.
- ✓A class derived from another class is called **subclass**, it **inherits all visible properties** (attributes and operations) of the superclass.
- ✓**A subclass:**
  - ✓Can add new attributes and/or operations
  - ✓Can overwrite (redefine) inherited operations implementation
  - ✓Also, can add its specific behavior (i.e. code) to inherited operations

General Elements and Mechanisms

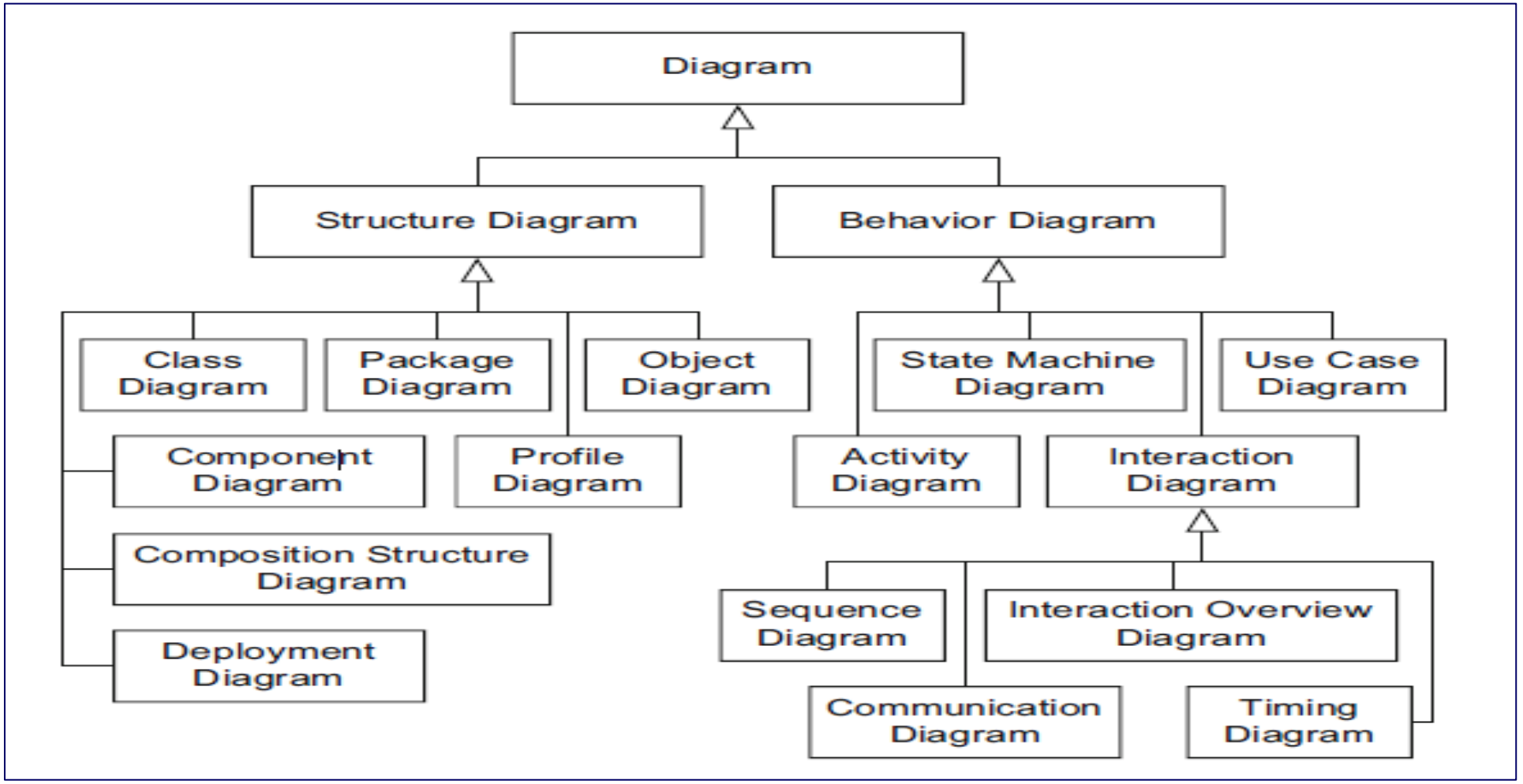
- ✓ **Inheritance** allows classes extensibility, therefore, the **creation of hierarchies of classes, which serves** as the basis of object oriented system development.
- ✓ A **class hierarchy** comprises **classes having similar properties.**
- ✓ Example: Person ← Employee ← Professor ← ...
- ✓ The notation A← B designates that **B is a subclass of A.**
- ✓ The **advantages of the inheritance** can be summarized as follows:
  - ✓ **Reusing program** or parts' model.
  - ✓ Defining interfaces in consistent manner.
  - ✓ Supporting **incremental and gradual development** (a step-by-step refinement of concepts).
  - ✓ Considered as a **modeling assistance** mechanism due to its natural categorization of elements occurrence.

General Elements and Mechanisms

- ✓ **Polymorphism** is the mechanism that enables adopting different forms.
- ✓During the program execution, a polymorphic attribute can refers to objects from diverse classes. When declaring such attribute:
  - ✓a type (e.g., class Professor) is assigned statically during compiling.
  - ✓At runtime, this attribute can also be dynamically bound to a subtype.

UML Diagrams

- ✓ **UML is defined over 14 hierarchically** dependent diagrams
- ✓ Modeling at all levels along the development process



UML Diagrams

- ✓ UML diagrams are divided mainly into:
  - ✓ **Structural diagrams**
  - ✓ **Behavioral diagrams**
- ✓ **Structure diagrams include :** Classes diagram, Object diagrams, Component diagram, Composite Structure diagram, deployment diagram, package diagram, and profile diagram.
- ✓ **Behavior diagrams include:** use case diagram, interaction diagram, activities diagram, and states machine diagram.
- ✓ **At turn, the Interaction diagrams** are divided into: sequence diagram, Timing diagram, communication diagram, and interaction overview diagrams.

UML Diagrams

- **Class diagram** is a static diagram which describes the system structure by displaying its classes and their properties and operations. Also, it shows the relationships between objects.
- **Use Case Diagram** comprises use cases, roles, and the relationships between them. It represents the interaction of users with the system. Also, it defines the corresponding specifications of the use cases.
- **Sequence diagram** models communication between objects in a sequential manner. It displays the chronological order of objects, classes and roles and information engaged in a scenario.
- **Activity diagram** is a behavior diagram which represents a scenario by means of flow of actions. It specifies a sequence of actions, decisions, concurrency, and other features.

UML Diagrams

- **Communication diagram** represents the interaction between objects and parts in term of messages exchange. It is similar to sequential diagram, however, objects are represented horizontally.
- **State Machine Diagram** is a behavior diagram that specifies the state of an object (device, program, module, software, , process, etc.) and the transitions between states.
- **Object diagram** is a structured UML diagram which represents a system or its parts at a specific time. It can models instances with the corresponding values and relationships.

UML Diagrams

- **Package diagram** represents the potential dependencies between packages in a given model. It shows the structure and organization of large-scale projects.
- **Component diagram** shows the provided and/or required interfaces for the various parts of the system and the relationships between them. These parts are represented by means of components and artifacts.
- **Deployment Diagram:** represents the location of components and artifacts (systems, software, modules, etc.) on physical nodes (servers, hardware, databases, etc.) and the relationships between specific parts of the system.

UML Diagrams

- **Composite Structure Diagram** represents the internal structure of a classifier, its parts and ports. It aims to model collaboration between objects having specific roles.
- **Interaction Overview Diagram** shows a view of the interactions in a given system or subsystem at a high level. It represents the processes in a similar way to activity diagrams, however, it utilizes other interaction diagrams and references instead of action nodes.
- **Timing Diagrams** is dedicated to specify temporal systems by focusing primarily on time and describing the changes in the classifier on a timeline.
- **Profile Diagram** allows describing and defining UML language extensions . Such extension mechanism enables adaptation of the language to a specific domain or platform.

Packages

- ✓It shows the **model logical organization** and the **relationships** between packages.
- ✓The package is a **mechanism that groups numerous UML elements** (classes, use cases, components, interfaces, etc.).
- ✓The package diagram is utilized for the following purposes:
  - ✓ Have an **overall view the system under study** (different subsystems).
  - ✓ **Represent the overall system architecture** and help organizing code (java or C#).
  - ✓ Modularize UML diagrams, especially complex ones.

Packages

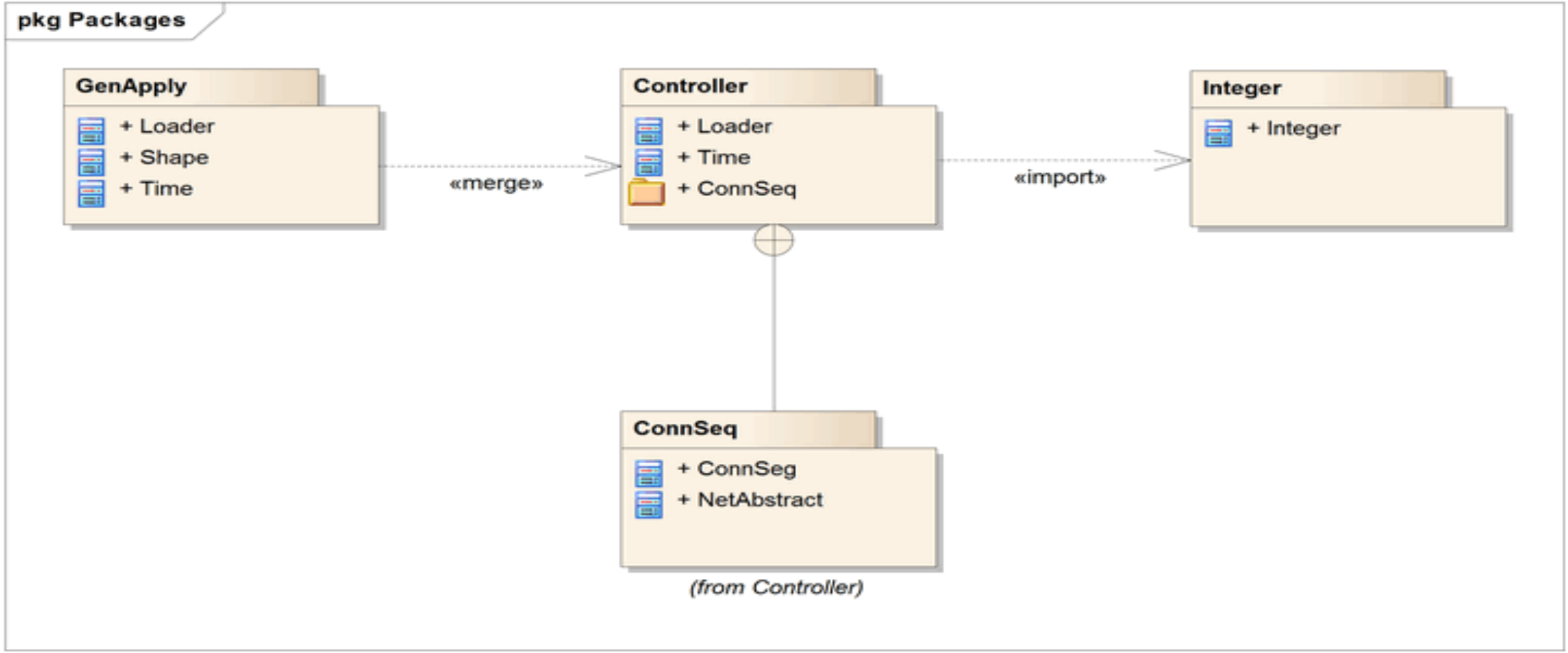
- ✓The **decomposition of a model** (of classes or use cases) is **a delicate activity**. It will be necessary to group the classes from a semantic point of view, that is to say:
- ✓The **classes of the same package** should provide **services of the same nature** to the users.
- ✓**Minimize dependencies** between packages.
- ✓**Dependencies between packages** should reflect the **internal relationships** to the system.
- ✓There should be **no cyclic dependency** between packages.

Packages

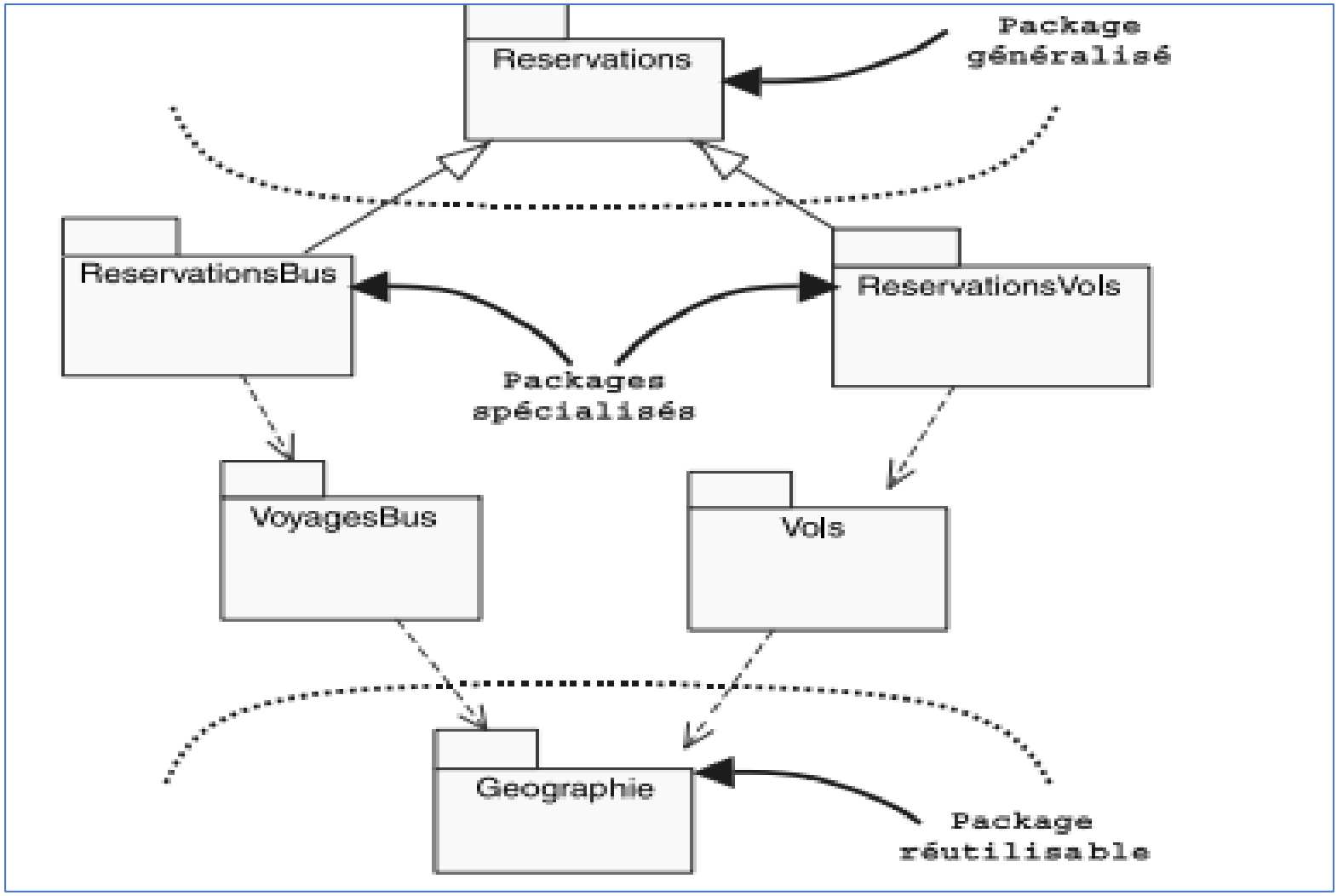
- ✓Package diagram is utilized to reflect the natural organization of packages and their elements.
- ✓Using package diagram to present class elements, it provides a visualization of the **namespaces**.
- ✓**Package diagram is commonly used for organizing** use case diagrams and class diagrams.
- ✓It is not limited to these elements, it can also used to built Packages to describe logical or physical relationships.

Packages

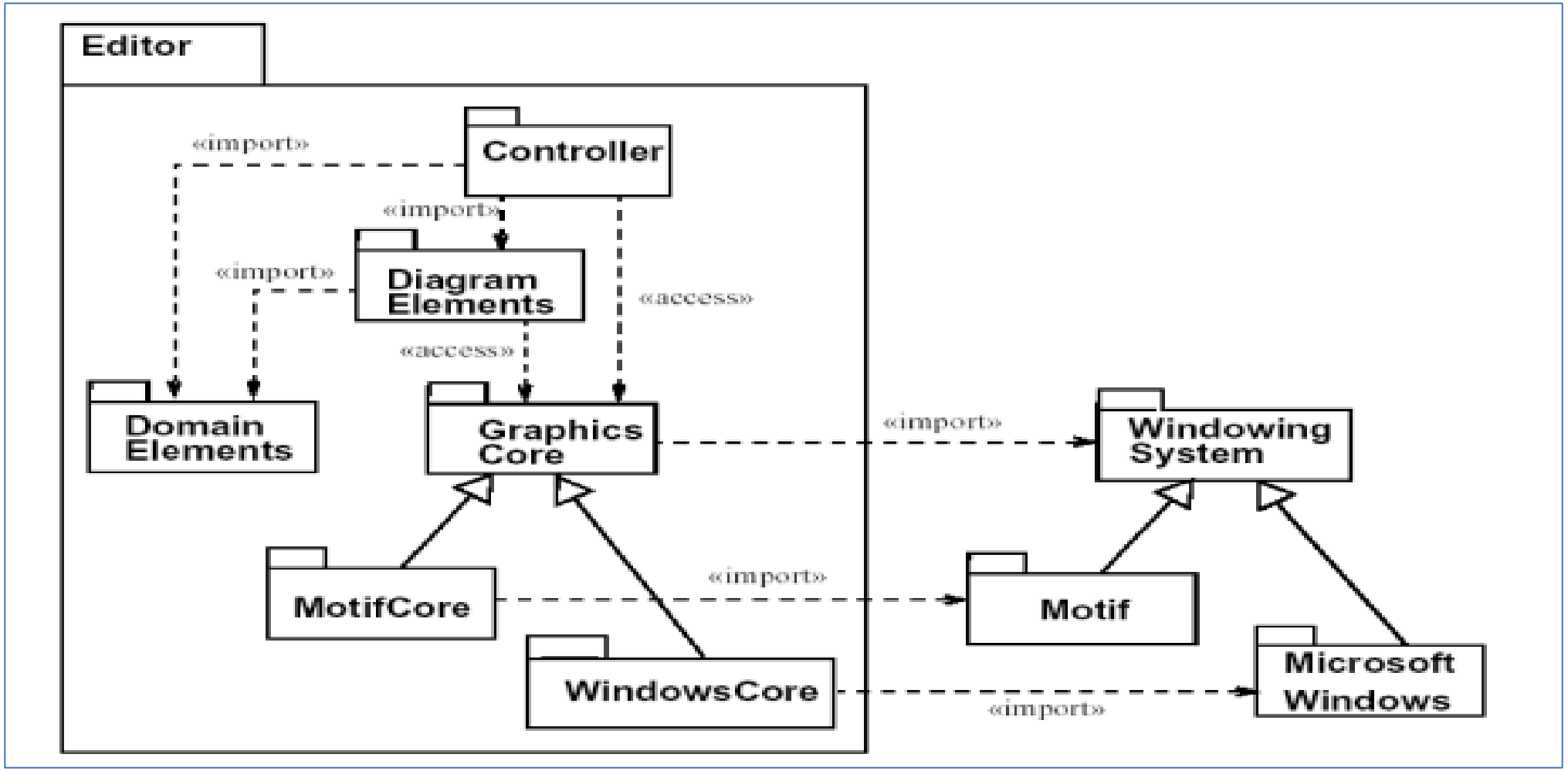
- Example of package diagram.



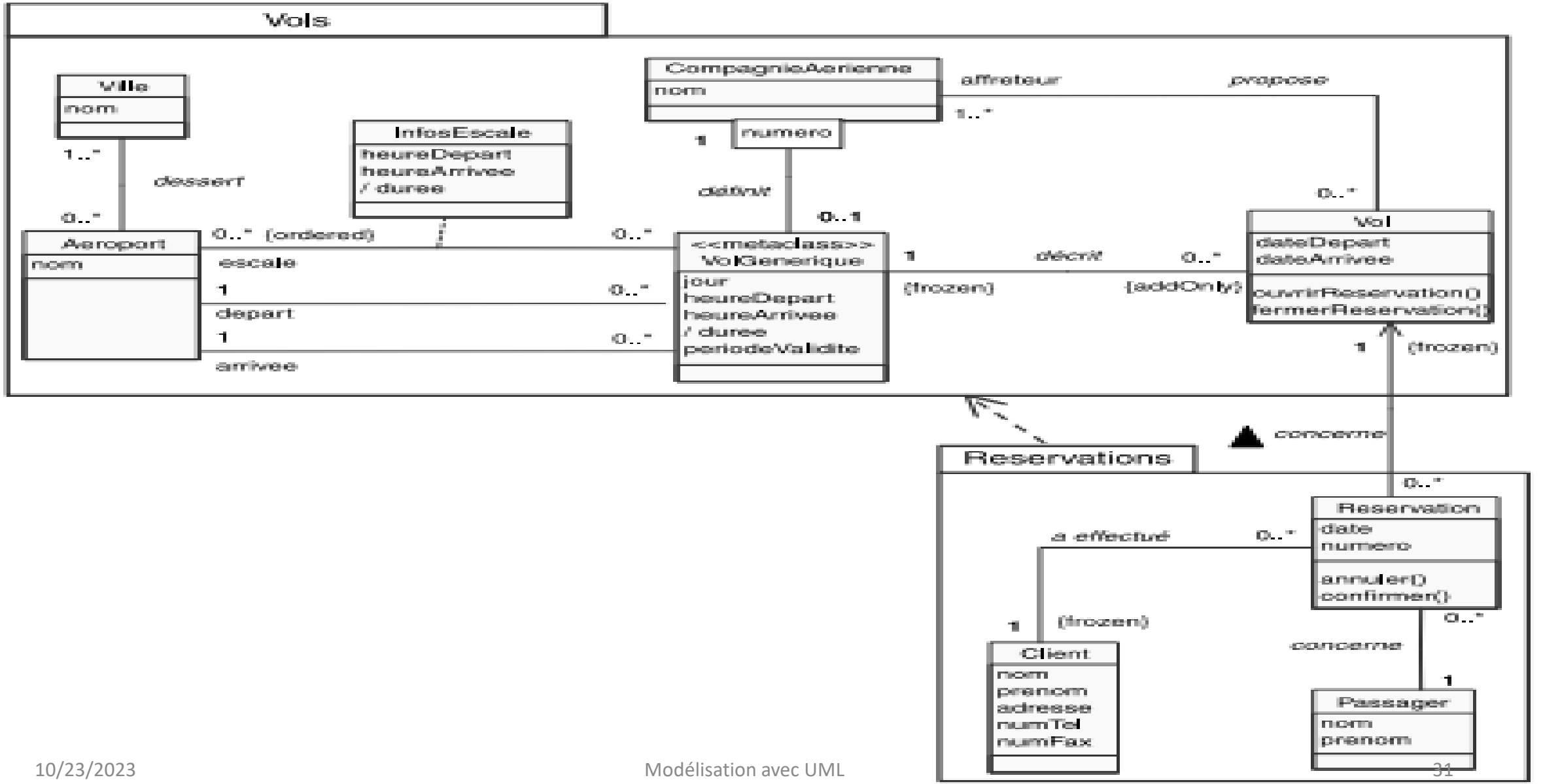
Packages


Reservations
Package généralisé
ReservationsBus
ReservationsVols
Packages apécialiséa
VoyagesBus
Vols
Geographie
Package réutilisable

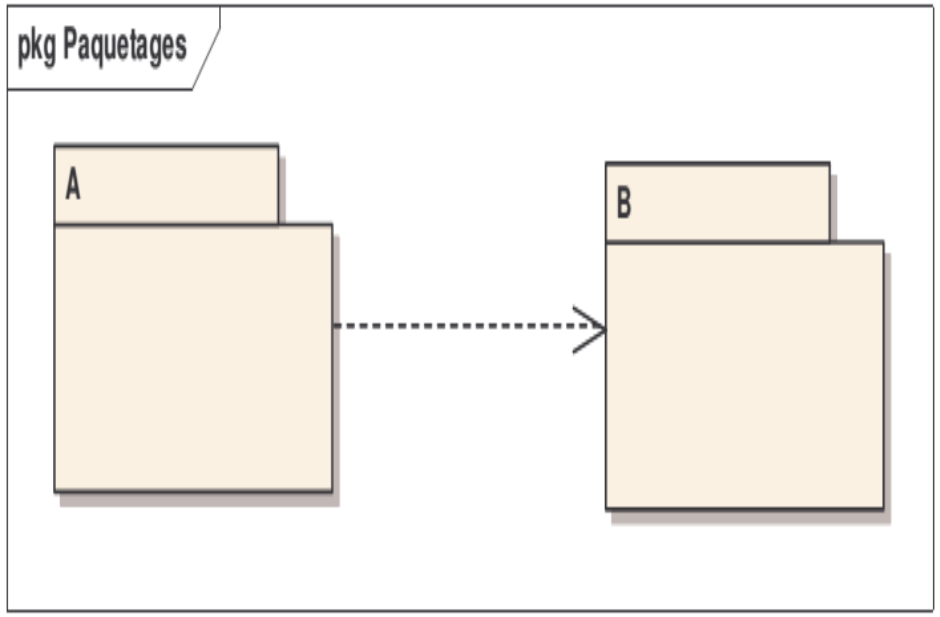
Packages


Editor
Controller
«import»
«import»
«import»
Diagram Elements
«access»
«access»
Domain Elements
Graphics Core
«import»
Windowing System
MotifCore
«import»
Motif
WindowsCore
Microsoft Windows
«import»

Packages


Vols
Ville
nom
1..*
dessert
0..*
Aeroport
nom
InfosEscale
heureDepart
heureArrivee
/ duree
0..* {ordered}
escale
1
depart
1
arrivee
0..*
0..*
0..*
CompagnieAerienne
nom
affreteur
1..*
1
numero
definit
0..1
<<metaclass>>
VolGenerique
jour
heureDepart
heureArrivee
/ duree
periodeValidite
1
{frozen}
décrit
0..*
{addOnly}
propose
0..*
Vol
dateDepart
dateArrivee
ouvrirReservation()
fermerReservation()
1
{frozen}
concerne
Reservations
0..*
Reservation
date
numero
annuler()
confirmer()
a effectué
0..*
1
{frozen}
Client
nom
prenom
adresse
numTel
numFax
0..*
concerne
1
Passager
nom
prenom

Packages

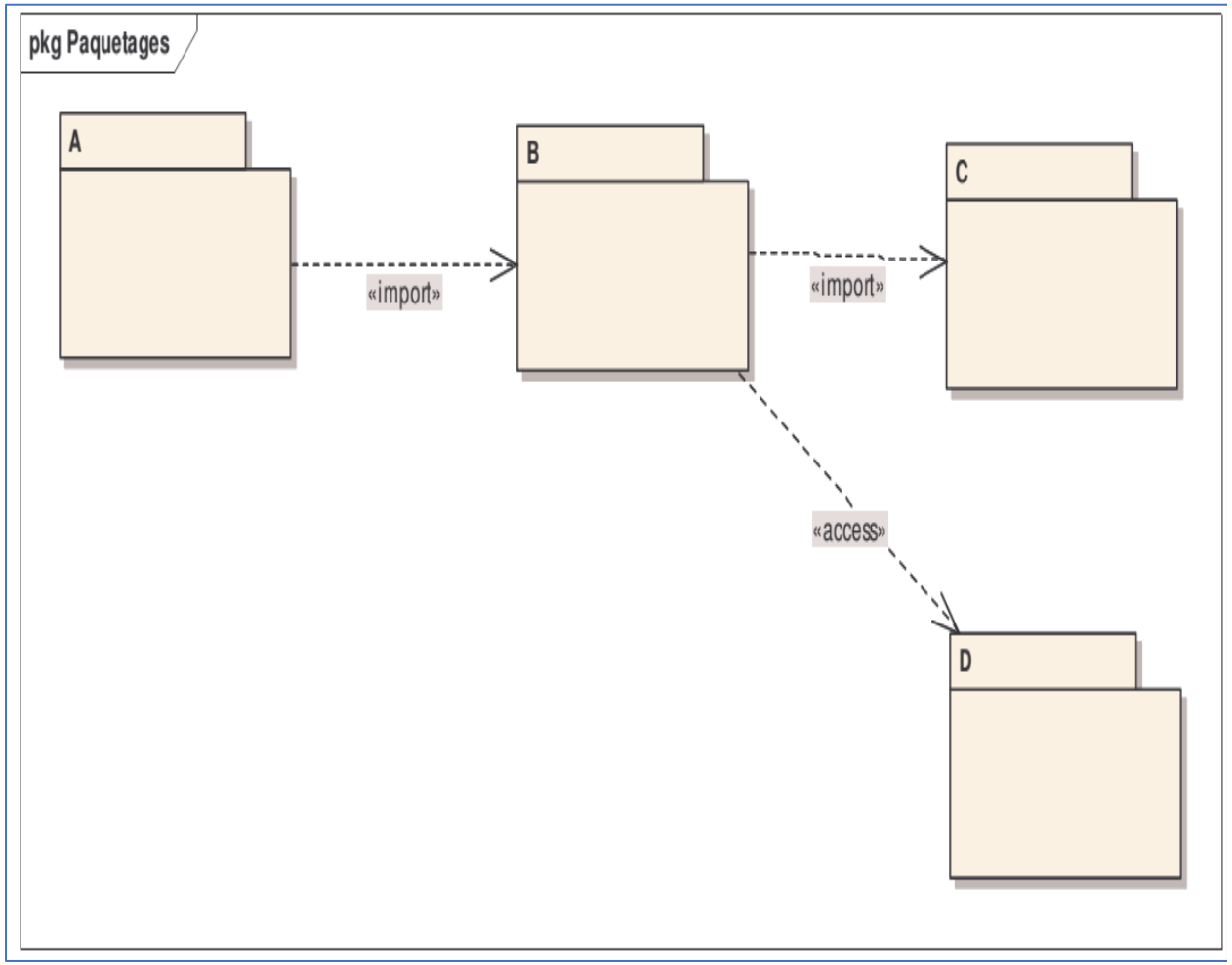


Dependency: the possibility for a given class from one package to use a class from another one. Such necessity creates a dependency between packages.

Example: if an element of **package A** utilizes an element of **package B** , we can say that package A depends on package B .

Packages

Import and Access



✓ **Import** and **access relationships** can be used to model importing classes into a namespace , so that importing namespace elements can reference those in the target namespace , without using its name.

✓ Package B **imports** C and accesses **D:**

- B therefore sees the public elements of C and D.

- Since A imports B, A sees the public elements of B.

- A also sees the public elements of C because C is imported publicly into B ,

- but A does not see the contents of D because D is privately accessed in B.

Conclusion

Using UM Diagrams presents serval advantages:

- Make the entire software design easier to understand prior to software development,
- Reduce development risks.
- Facilitates communication between developers.
- It is an expressive modeling language which allows people acquiring the necessary information they need during software development process.