Cours de Génie Logiciel

Introduction au Génie logiciel et à la modélisation

(Chapitre 2)

2022-2023

Dr. Kouah S.

Plan

1. Introduction: Modélisation, Modèle, Modélisation Orientée Objet, UML en application.
2. Éléments et mécanismes généraux
3. Les diagrammes UML
4. Paquetages

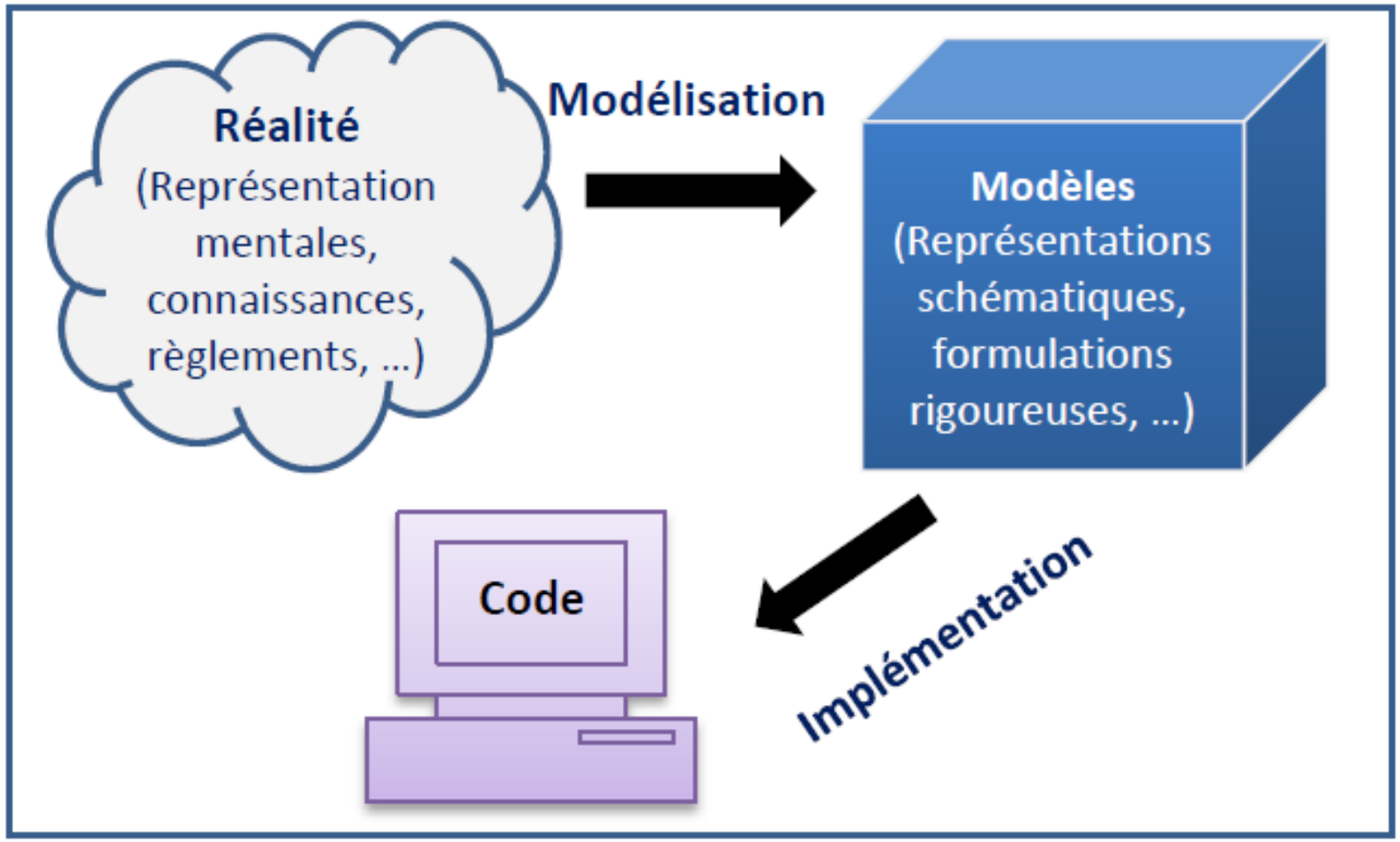
Modélisation

- **Modèle** : Simplification de la réalité, abstraction, vue subjective

Exemples: modèle météorologique, économique, démographique...

- **Modéliser un concept ou un objet pourquoi?**
  - Mieux le comprendre (modélisation en physique)
  - Mieux le construire (modélisation en ingénierie)
- **En génie logiciel :**
  - **Modélisation** = **spécification** + **conception**
  - Aider la réalisation d'un logiciel à partir des besoins du client.

Modélisation



Modèle, Méthode, Outil ?

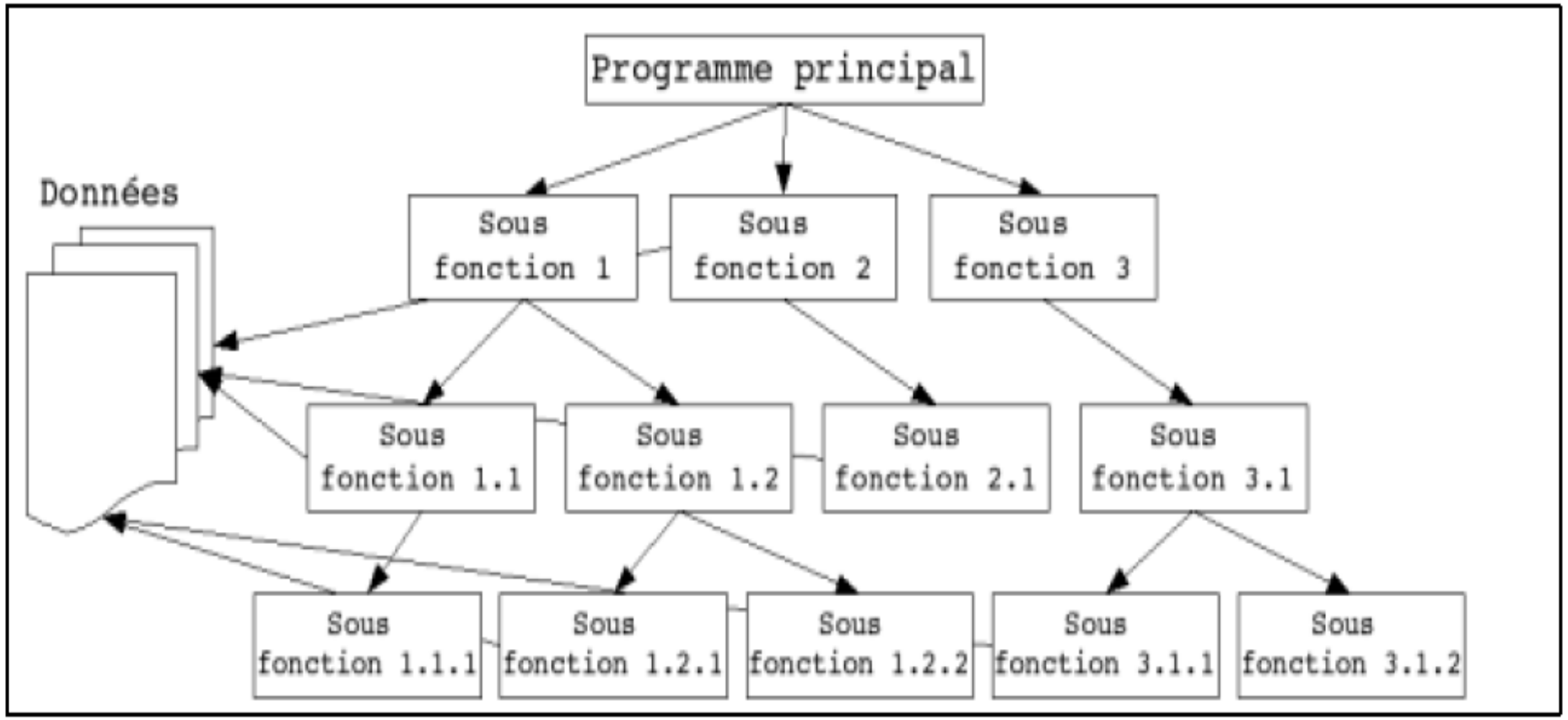
- Un **Modèle** est une abstraction de la réalité. c'est une représentation abstraite de tout ou partie du réel.
- Un **modèle** permet de **capturer les aspects pertinents** répondant à un objectif défini a priori.
- Un **Outil** est un support automatisé ou semi automatisé pour supporter des méthodes (Formalisme ou langage permettant d'exprimer un modèle).
- Une **Méthode** est la manière dont on va conduire une activité, elle utilise des modèles et des outils, et suit une démarche de mise en œuvre.

Méthodes de conception

- **Méthode fonctionnelle**

✓Système = ensemble de fonctions

✓État du système (données) centralisé et partagé par **les fonctions.**



Méthodes de conception

- **Méthodes guidée par les données**

✓la structure d'un système peut se déduire de la structure **des données** apparaissant en entrée et en sortie de ce système.

Méthodes de conception

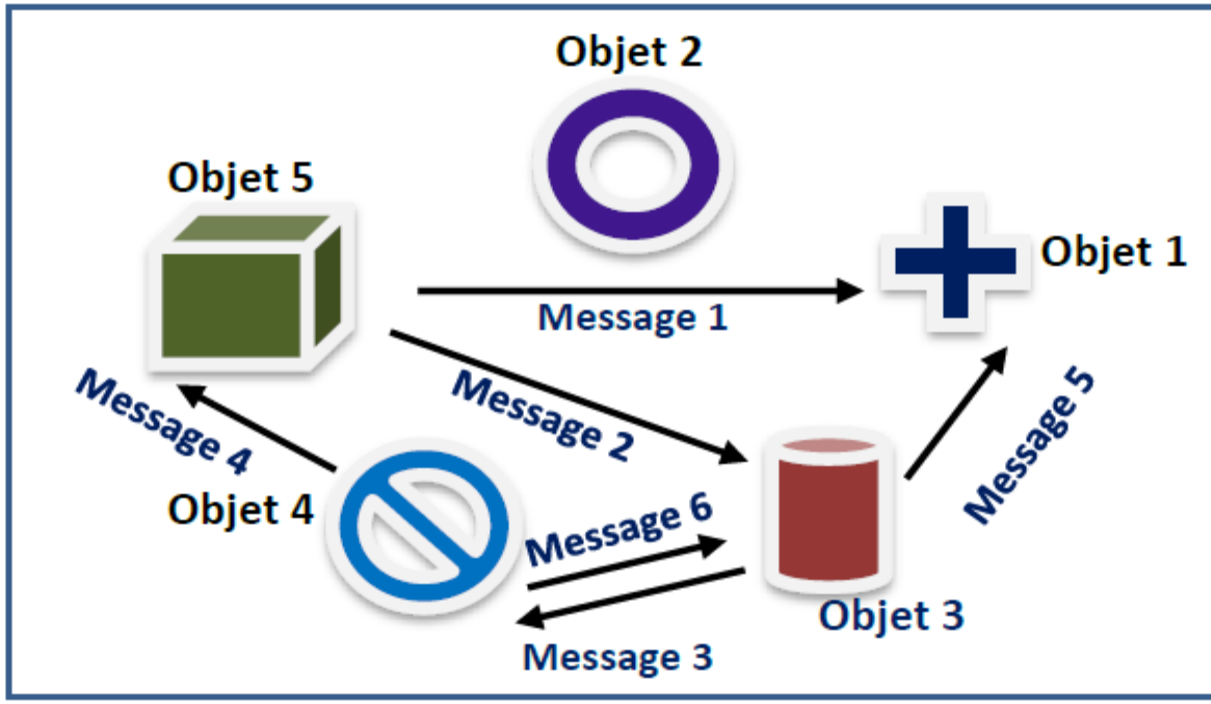
- **Méthodes orientée objets**

✓Système = ensemble d'objets

✓Objet = données + fonctions

✓Exemples de méthodes orientées objets :

- **OOA** (**Object-Oriented Analysis**), Coad.
- **OOD** (**Object-Oriented Design**), Grady Booch.
- **OMT** (**Object Modeling Technique**), James Rambaugh.
- **OOSE** (**Object-Oriented Software Engineering**) Ivar Jacobson.



Méthode Orientée Objet

Conception orientée objet

- Principes

✓Concept du domaine d'application = **objet**

✓Décrit par **état** (**attributs**) + **comportement** (**opérations**)

✓Liens entre concepts : **héritage**, **agrégation**, **composition**...

- **Caractéristiques des objets**

✓**Identité** : objet = entité unique (mêmes attributs ⇏ même objet)

✓**Classification** : **regroupement** des objets de même nature (**attributs + opérations**)

✓**Polymorphisme** : comportement différent d'une **même opération** dans différentes classes

✓**Héritage** : **partage hiérarchique** des attributs et opérations

UML: **Unified Modeling Language?**

- **Langage** :

✓Syntaxe et règles d'écriture

✓Notations graphiques normalisées

- **Langage de modélisation :**

✓Abstraction du fonctionnement et de la structure du système

✓Spécification et conception

- **Langage unifié :**

✓Fusion de plusieurs notations antérieures : Booch, OMT, OOSE

✓Standard défini par l'OMG (Object Management Group)

UML: **Unified Modeling Language?**

- **Langage graphique** pour **visualiser**, **spécifier**, **construire** et **documenter** un logiciel
- Langage **graphique** : Ensemble **de diagrammes** permettant de **modéliser le logiciel** selon **différentes vues** et à différents niveaux d'abstraction
- Modélisation orientée objet : modélisation du système comme **un ensemble d'objets interagissant**
- **UML n'est pas une méthode de conception**

Pourquoi UML ?

- **Besoin de modéliser pour construire un logiciel:**

✓ Modélisation des **aspects statiques** et **dynamiques**

✓Modélisation à différents niveaux d'abstraction et selon plusieurs vues

- **Indépendant du processus de développement**

- Besoin de **langages normalisés** pour la modélisation

✓**Langage semi-formel**

✓**Standard très utilisé**

- **Conception orientée objet**

3. Les diagrammes UML

14 diagrammes hiérarchiquement dépendants

Modélisation à **tous les niveaux** le long du processus de développement

- **Diagrammes structurels :**

✓Diagramme de classes

✓ Diagramme d'objets

✓ Diagramme de composants

✓ Diagramme de déploiement

✓ Diagramme de paquetages

✓ Diagramme de structure composite

✓ **Diagramme de profils**

- **Diagrammes comportementaux** :

✓Diagramme de cas d'utilisation

✓ Diagramme états-transitions

✓Diagramme d'activités

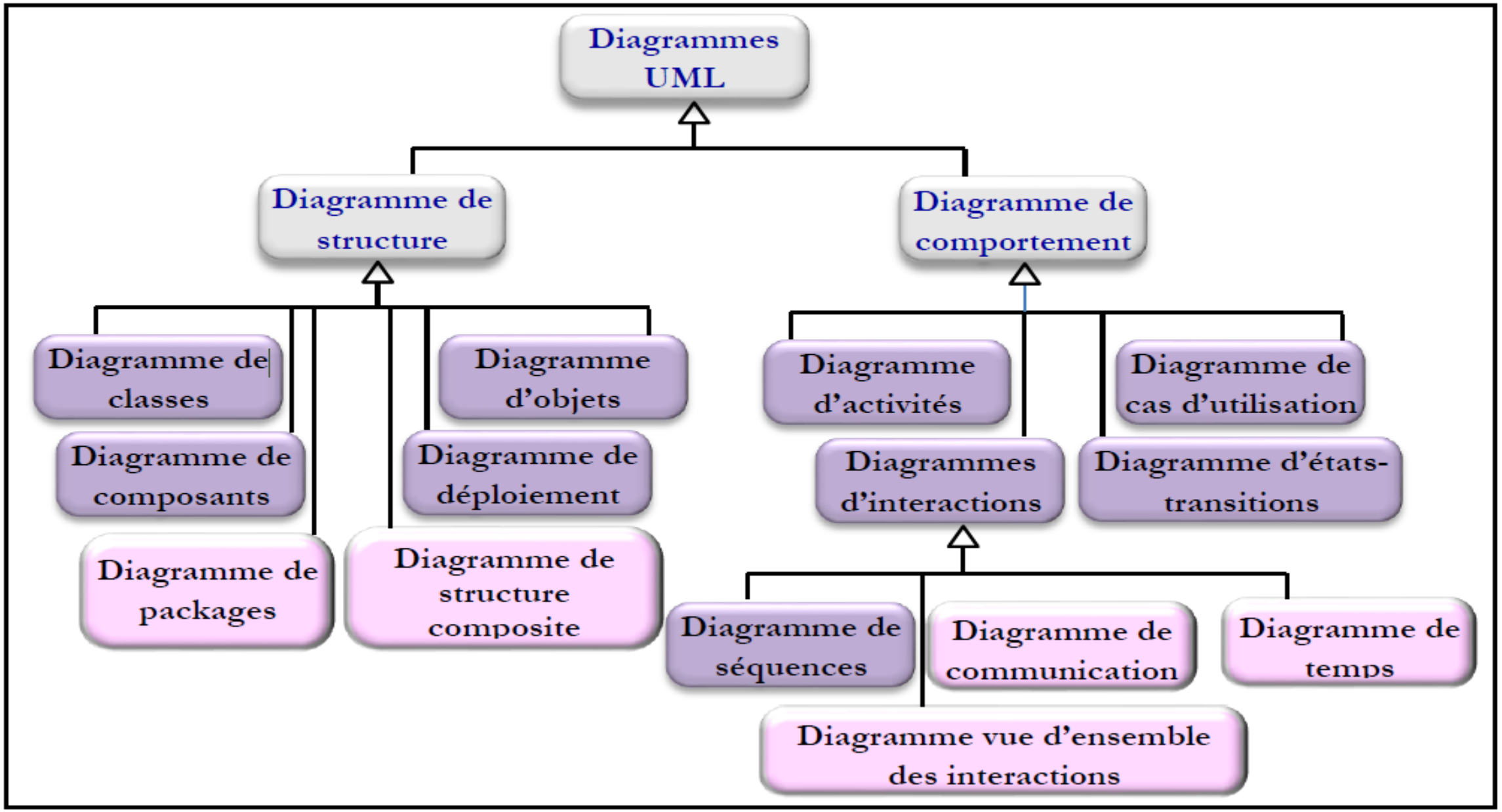
Diagrammes d'interaction :

✓Diagramme de séquence

✓Diagramme de communication

✓Diagramme global d'interaction

✓Diagramme de temps

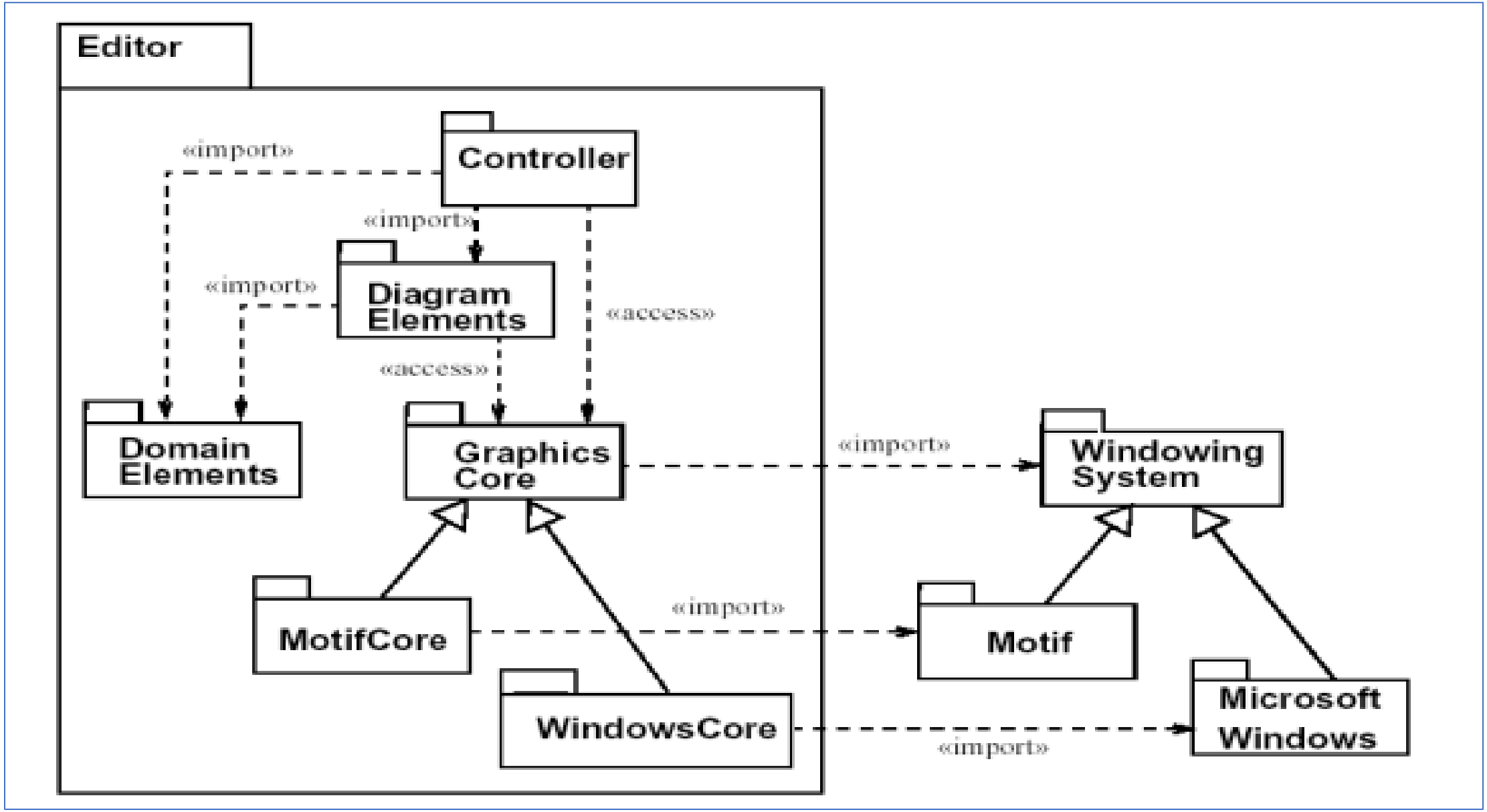

Diagrammes UML
Diagramme de structure
Diagramme de comportement
Diagramme de classes
Diagramme de composants
Diagramme de packages
Diagramme d'objets
Diagramme de déploiement
Diagramme de structure composite
Diagramme d'activités
Diagramme de cas d'utilisation
Diagrammes d'interactions
Diagramme d'états-transitions
Diagramme de séquences
Diagramme de communication
Diagramme de temps
Diagramme vue d'ensemble des interactions

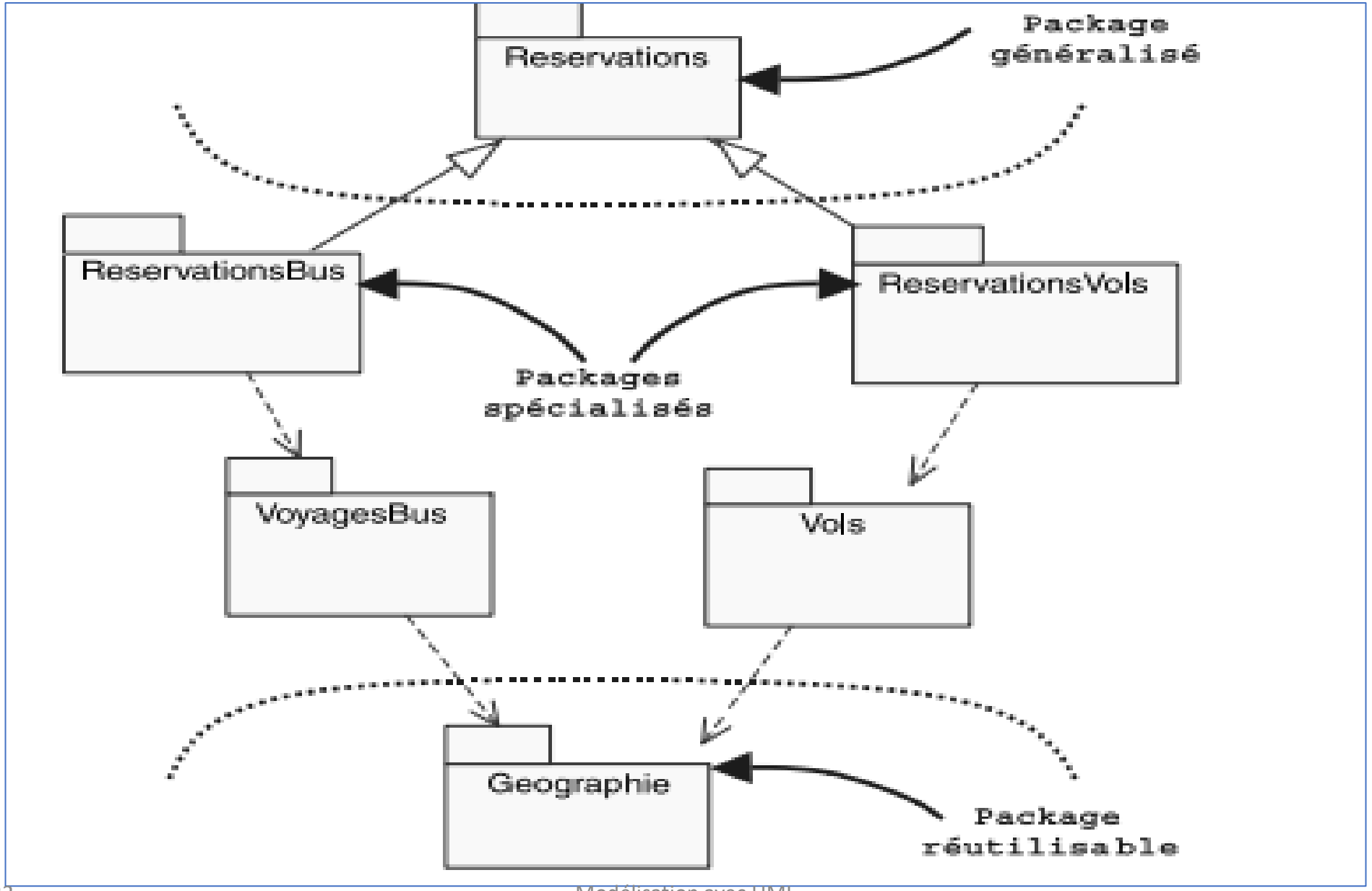
4. Paquetages

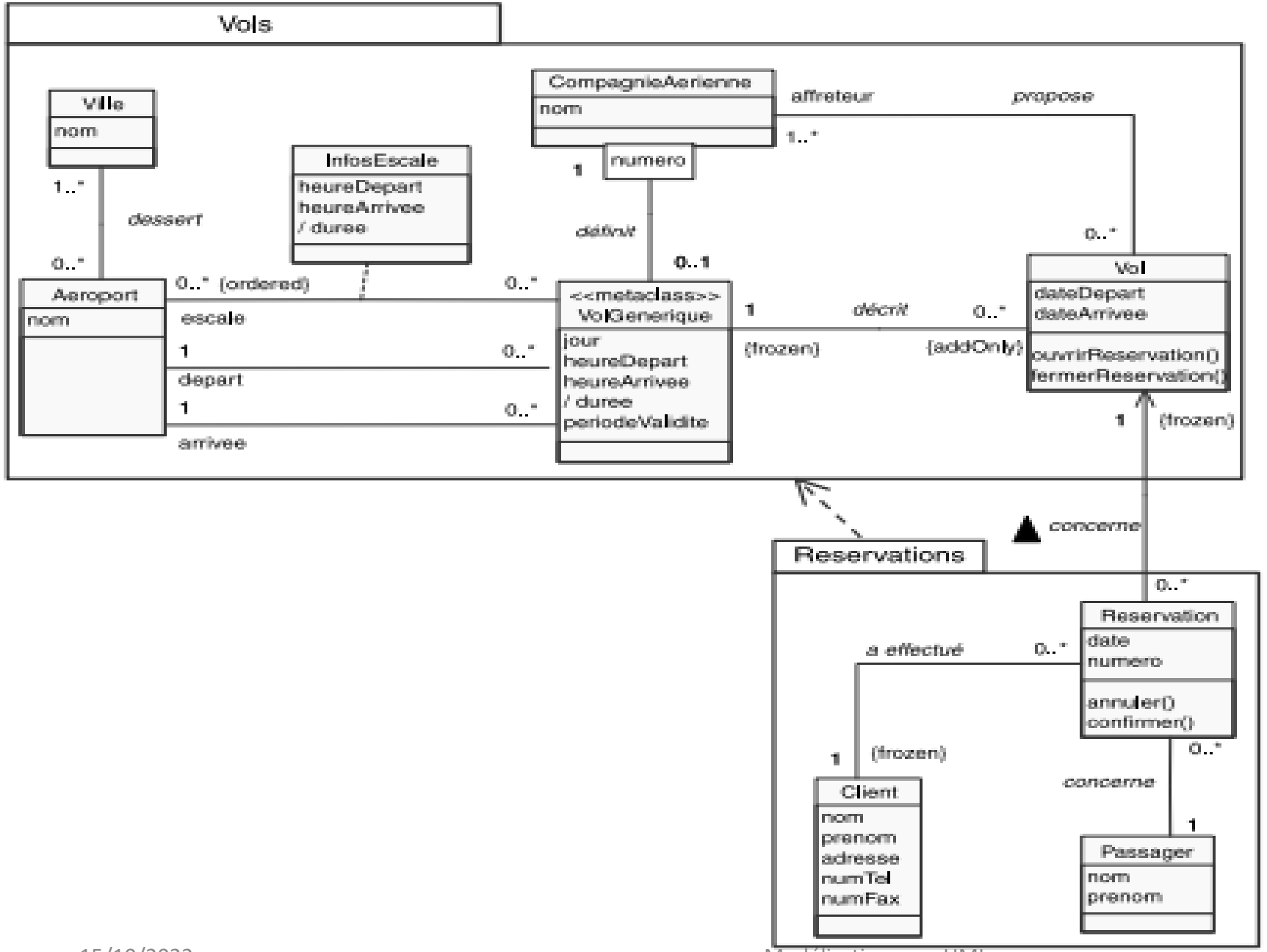
- Il **montre l'organisation logique** du modèle et les **relations** entre packages.
- Le package est un **mécanisme regroupant plusieurs éléments d'UML** (peut regrouper des classes, des cas d'utilisation, des interfaces, des composants….).
- Le diagramme de package sert à :
- Avoir une **vision globale** des différents sous-systèmes du système étudié.
- **Représenter l'architecture globale du système** et aider à organiser du code (java ou C#) .
- Modulariser les diagrammes (UML) les plus complexes.

4. Paquetages

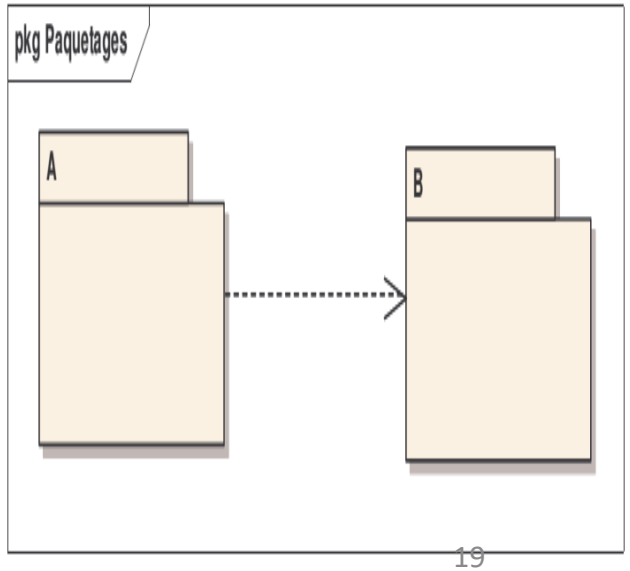
- Le **découpage d'un modèle** (de classes ou de cas d'utilisation) est une **activité délicate.** Il faudra regrouper les classes d'un point de vue sémantique c'est-à-dire :
  - Les classes d'un même package doivent rendre des **services de même nature** aux utilisateurs.
  - **Minimiser les dépendances** entre les packages.
  - Les **dépendances** entre packages doivent **refléter des relations internes au système.**
  - Il ne doit **pas y avoir de dépendance cyclique entre** des packages.

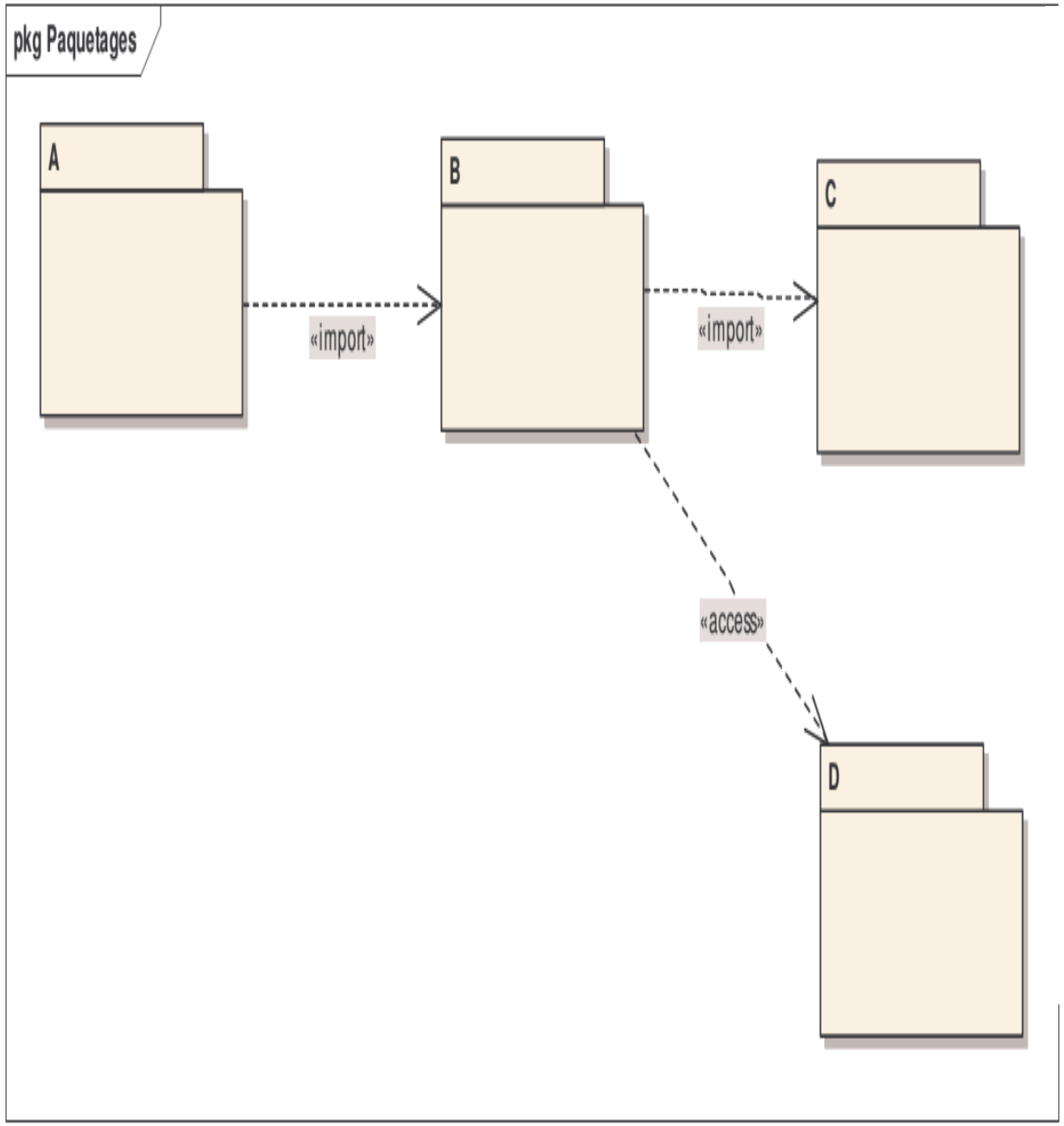

Editor
Controller
«import»
«import»
«import»
Diagram Elements
«access»
«access»
Domain Elements
Graphics Core
«import»
Windowing System
MotifCore
«import»
Motif
WindowsCore
Microsoft Windows
«import»


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

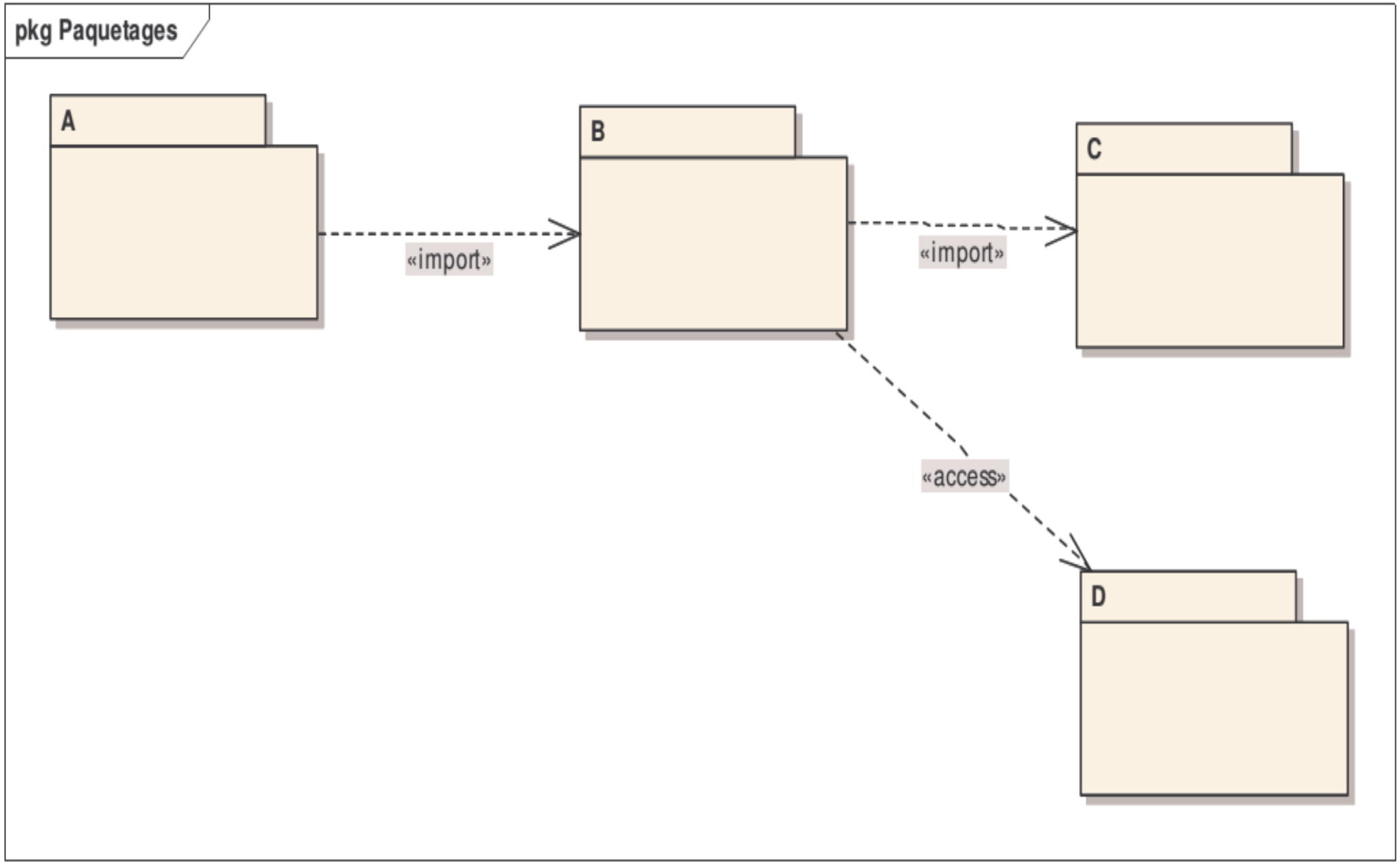


Dépendance : Une classe d'un paquetage a parfois besoin d'utiliser une classe d'un autre paquetage. Cela crée une dépendance entre les paquetages : si un élément du paquetage A utilise un élément du paquetage B alors le paquetage A dépend du paquetage B .





- Les relations **d'importation** et **d'accès** peuvent être utilisées pour modéliser l'importation de classes dans un espace de noms afin que les éléments de l'espace de noms effectuant l'importation puissent faire référence à ceux de l'espace de noms cible sans employer son nom.
- Le paquetage B importe C et accède à D. B voit donc les éléments publics de C et de D. Puisque A importe B, A voit les éléments publics de B. A voit également les éléments publics de C car C est importé **publiquement** dans B, mais A ne voit pas le contenu de D car D est accédé **privativement** dans B.


pkg Paquetages
A
B
C
D
«import»
«import»
«access»

Références

1. Delphine Longuet. UML: Introduction au génie logiciel et à la modélisation. Notes de cours de l'université Polytech Paris-Sud. Formation initiale 3e année. Spécialité Informatique. Année 2017-2018.
2. Pierre Gerard. Génie Logiciel - Principes et Techniques (univ-paris13.fr). Notes de cours de IUT de Villetaneuse - Université de Paris 13 Licence Pro. FC 2007/2008. Lien: https://lipn.univ-paris13.fr/~gerard/docs/cours/gl-cours-slides.pdf
3. Ilham Boussaïd. Cours Génie Logiciel. USTHB - Année Universitaire 2009-201.
4. Joseph Gabay, David Gabay. UML 2 Analyse et conception: Mise en œuvre guidée avec études de cas.

- Stefano Zacchiroli. Génie Logiciel Avancé- Cours 1 — Introduction. Notes de cours. Laboratoire PPS, Université Paris Diderot - Paris 7. 3 Février 2011.