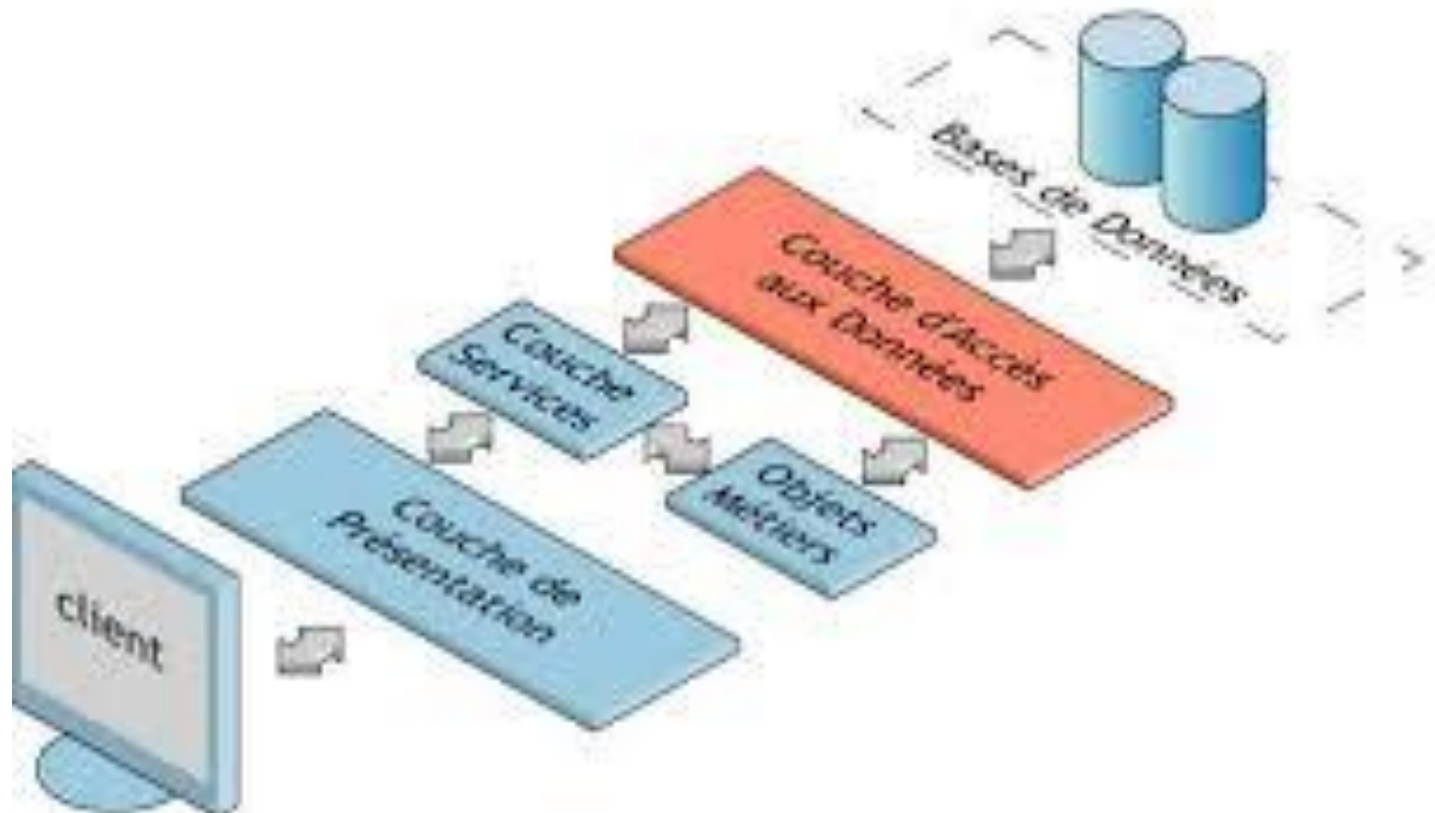


Cours Architectures Logicielles

Mohamed ZAYANI, Sameh Hbaieb Turki

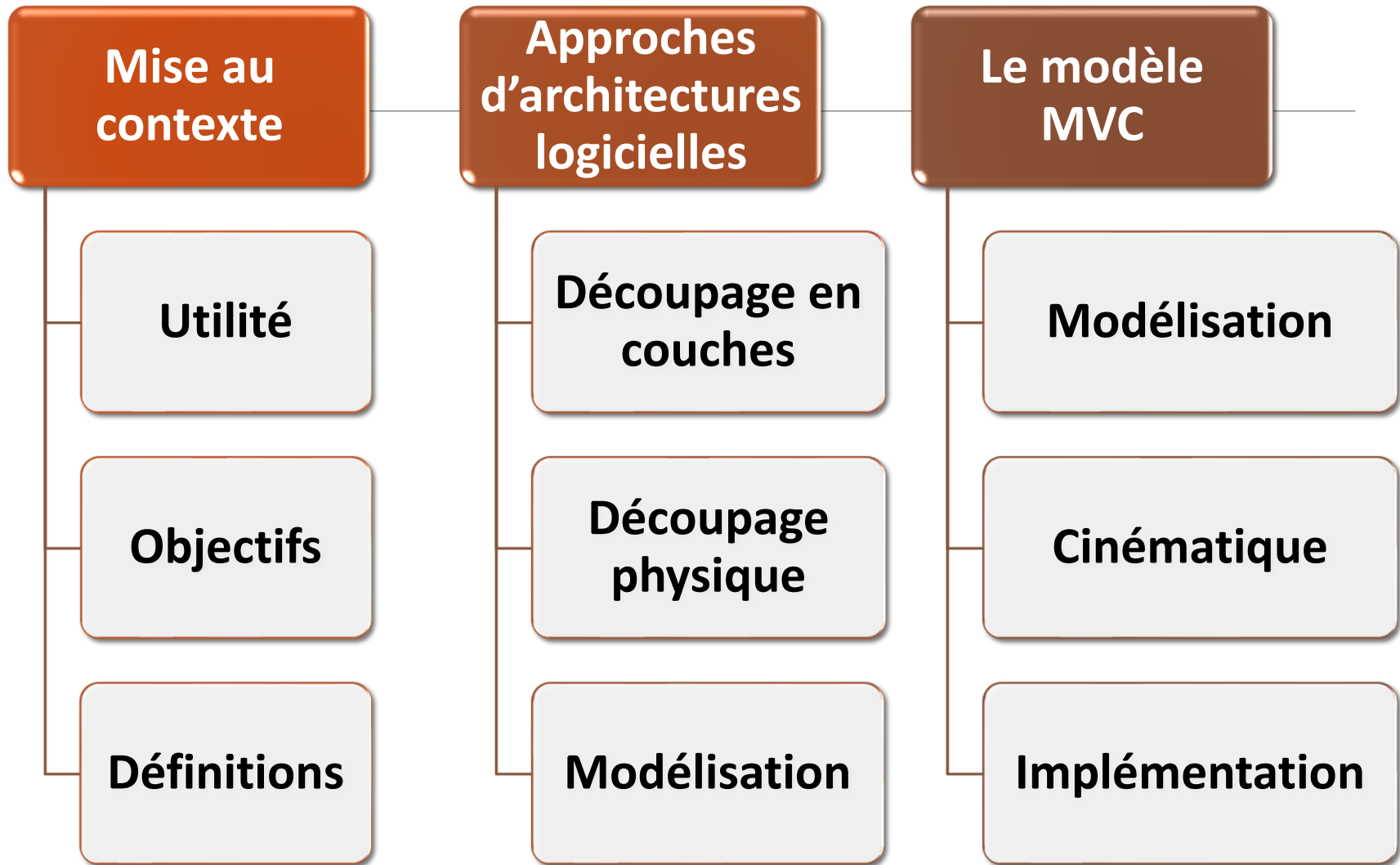
ISET-SFAX-2020/2021



Chapitre 1

Introduction à l'architecture logicielle

Plan



Utilité d'une architecture logicielle (Garlan 2000)

➤ Compréhension

- faciliter la compréhension des grands systèmes **complexes** en donnant une vue de haut-niveau de leurs structures et de leurs contraintes.
- Les motivations des choix de conception sont ainsi mis en évidence

➤ Réutilisation

- favoriser **l'identification** des éléments réutilisables, parties de conception, composants, caractéristiques, fonctions ou données communes

➤ Construction

- fournir **un plan de haut-niveau** du développement et de l'intégration des modules en mettant en évidence les composants, les interactions et les dépendances

Utilité d'une architecture logicielle (Garlan 2000)

➤ Évolution

- Déterminer les points où un système peut être modifié et étendu.
- La séparation composant/connecteur facilite une implémentation du type « **plug-and-play** »

➤ Analyse

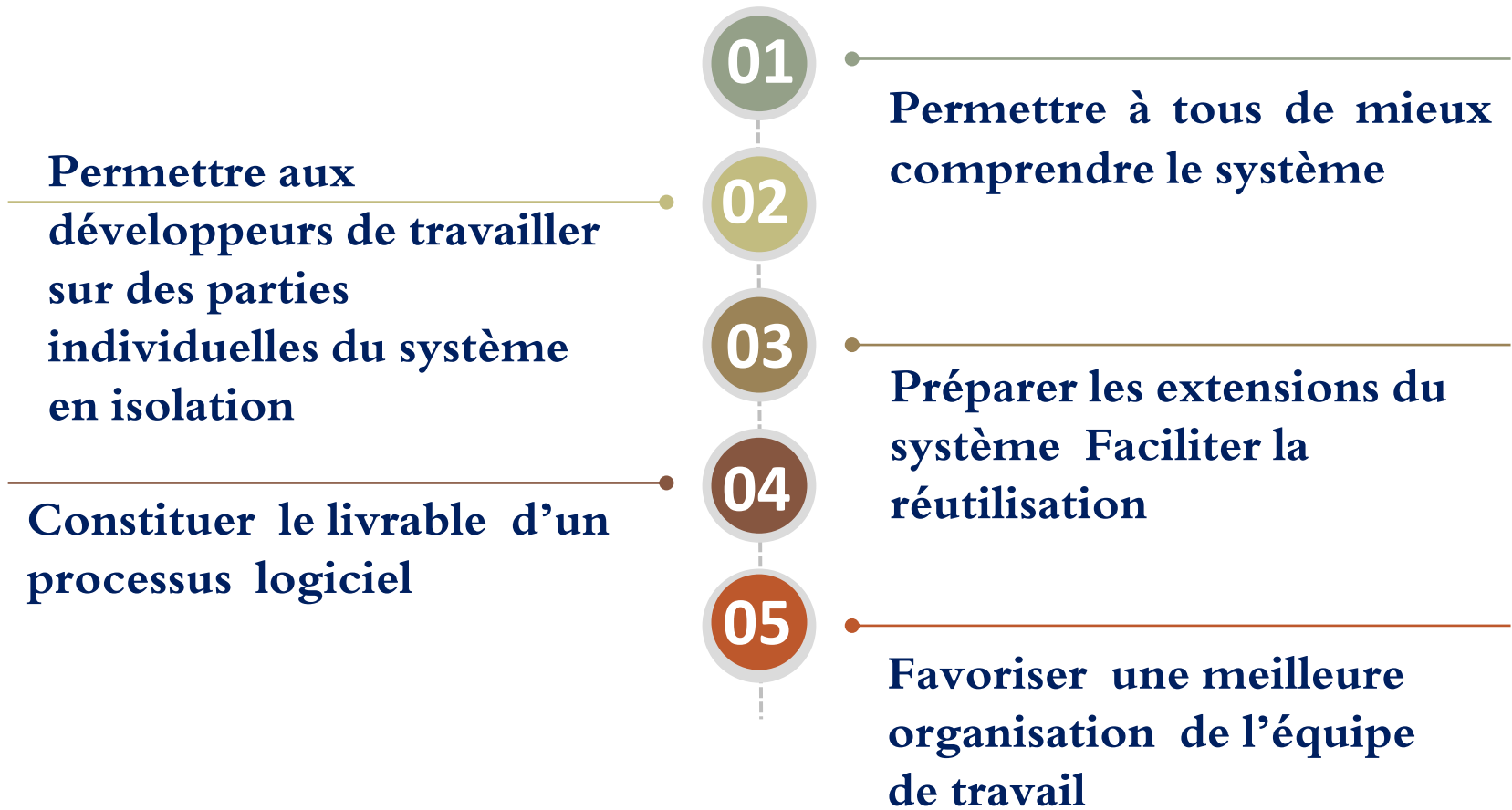
- offrir une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la cohérence, test de conformité, analyse des dépendances

➤ Gestion

- contribuer à la gestion générale du projet en permettant aux différents intervenants de voir comment les différents morceaux du casse-tête seront agencés.
- L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Objectifs globaux

A quoi sert de développer une architecture logicielle?

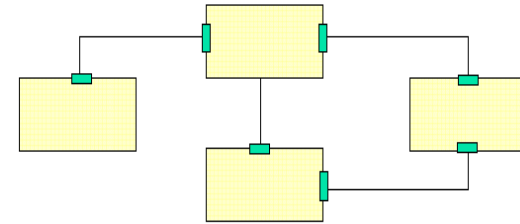


Qu'est ce qu'une architecture logicielle ?

• Définition 1

« L'architecture d'un logiciel est **la structure des structures** (modules) d'un système. Elle inclut:

- Les composants logiciels
- Les propriétés externes visibles de ces composants
- Les relations entre ces composants »



Cf. [Bass, Clements, and Kazman (1998)]

- ❖ L'architecture décrit l'organisation générale d'un système et sa décomposition en sous-systèmes ou composants
- ❖ Elle ne se préoccupe pas des détails d'implémentation

Qu'est ce qu'une architecture logicielle ?

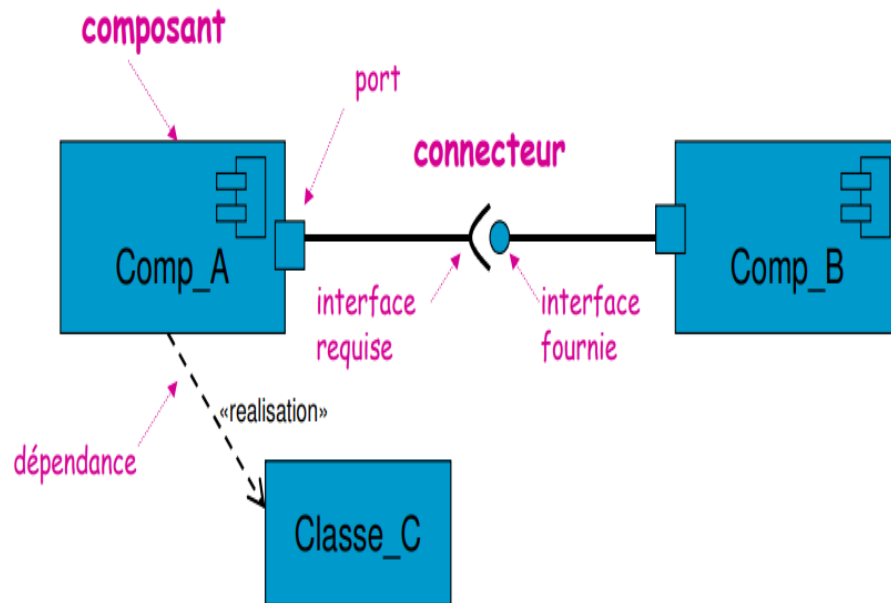
- Définition 2

« Une architecture logicielle est une **représentation abstraite** d'un système exprimée essentiellement à l'aide de **composants logiciels** en interaction via des **connecteurs** » [1]

- Définition 3

« L'architecture logicielle **décrit** d'une manière symbolique et schématique les différents **éléments** d'un ou de plusieurs systèmes informatiques, **leurs interrelations** et leurs interactions. » [2]

Les éléments architecturaux



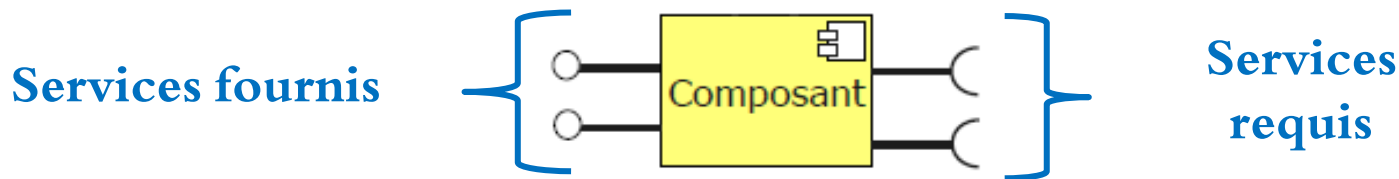
- Deux ou plusieurs composants interagissent via un connecteur
- Chaque élément architectural possède une structure et/ou comportement pouvant être décrit par un modèle UML approprié

Les éléments architecturaux

1. Le composant

- Définition:

« Les composants logiciels sont des **spécifications** d'**unités fonctionnelles** clairement définies, sémantiquement cohérentes et compréhensibles » [1]



- ❖ Un composant encapsule un traitement et/ou des données
- ❖ Les composants peuvent être développés ou acquis (réutilisation)
- ❖ Possède des dépendances explicitement définies pour exprimer les contraintes requises par son contexte d'exécution ou sa réalisation

Les éléments architecturaux

1. Le composant

Dépendances entre composants= relations entre deux composants

Les Types de dépendances

- Un composant peut dépendre d'un autre composant qui lui fournit un service ou une information
- Un composant peut dépendre d'une classe qui implémente une partie de son comportement-> Dépendance de réalisation
- Un composant peut dépendre d'un artefact (code source, fichier .jar, etc.) qui l'implante concrètement -> Dépendance de manifestation

Les éléments architecturaux

2. Le connecteur

- Définition:

« *Les connecteurs sont des éléments qui assurent les **interactions** entre composants* » [1]



Les éléments architecturaux

2. Le connecteur

Un connecteur peut avoir un ou plusieurs rôles

- **Communication** : rôle le plus fréquent
- **Coordination** : contrôle du calcul, de la transmission des données, etc.
- **Conversion** : permet l'interaction entre composants développés indépendamment dans des langages différents par exemple
- **Facilitation** : permet l'interaction entre composants conçus pour interagir ensemble. Synchronisation, accès contrôlés aux données partagées, etc.

Types d'architectures

Une architecture peut être conçue principalement sur deux axes:

Logique

RÉPONDRE À LA QUESTION
COMMENT ?

Structurer l'application de façon **logique** et la structurer en couches, composants, et modules de conceptions

Physique

RÉPONDRE À LA QUESTION AVEC
QUOI ?

Décrire et organiser les différents moyens **matériels** (clients, serveur,...), les logiciels de bases (OS, SGBD, AGL,...) et les moyens de communication (middleware, réseaux,...)»

A. Architectures en couches

- ❖ Une architecture en couches = architecture répartie sur plusieurs **briques**
- ❖ Le principe est de découper les applications (surtout les plus complexes) en des modules ayant leurs propres responsabilités.
- ❖ Le découpage est indépendant des contraintes **matérielles** mais prend un aspect logique qui cherche à regrouper les fonctionnalités similaires et cohérentes dans un composant isolé (couche)
- ❖ Le nombre de couches et le degrés de partition reste un choix conceptuel et surtout relatif à la complexité de l'application

Les principales couches



Objectifs d'un découpage logique

- ❖ Maîtriser la complexité des applications
- ❖ Faciliter la maintenance, le paramétrage d'un logiciel
- ❖ Réaliser les test unitaires (chaque couche peut être isolée et tester indépendamment)
- ❖ Favoriser la réutilisation des composants (classes, bibliothèques,..)
- ❖ Faciliter les échanges entres les applications

Modèle MVC

- Définition:
 - « MVC (Model–View–Controller) *est un motif d'architecture logicielle*
 - *destiné aux interfaces graphiques* ,
 - *lancé en 1978 (dédié initialement au Smaltalk)*
 - *et très populaire pour les applications web* » [3]

- ❖ Ce modèle de conception est composé de **trois** types de modules ayant trois responsabilités différentes:
 1. Un **modèle** contient les données à afficher.
 2. Une **vue** contient la présentation de l'interface graphique
 3. Un **contrôleur** contient la logique concernant les actions effectuées par l'utilisateur.

Composants MVC

1. Modèle:

- représente le fond (sujet d'étude)
- s'intéresse à la **représentation** des données de la couche métier (Business layer)

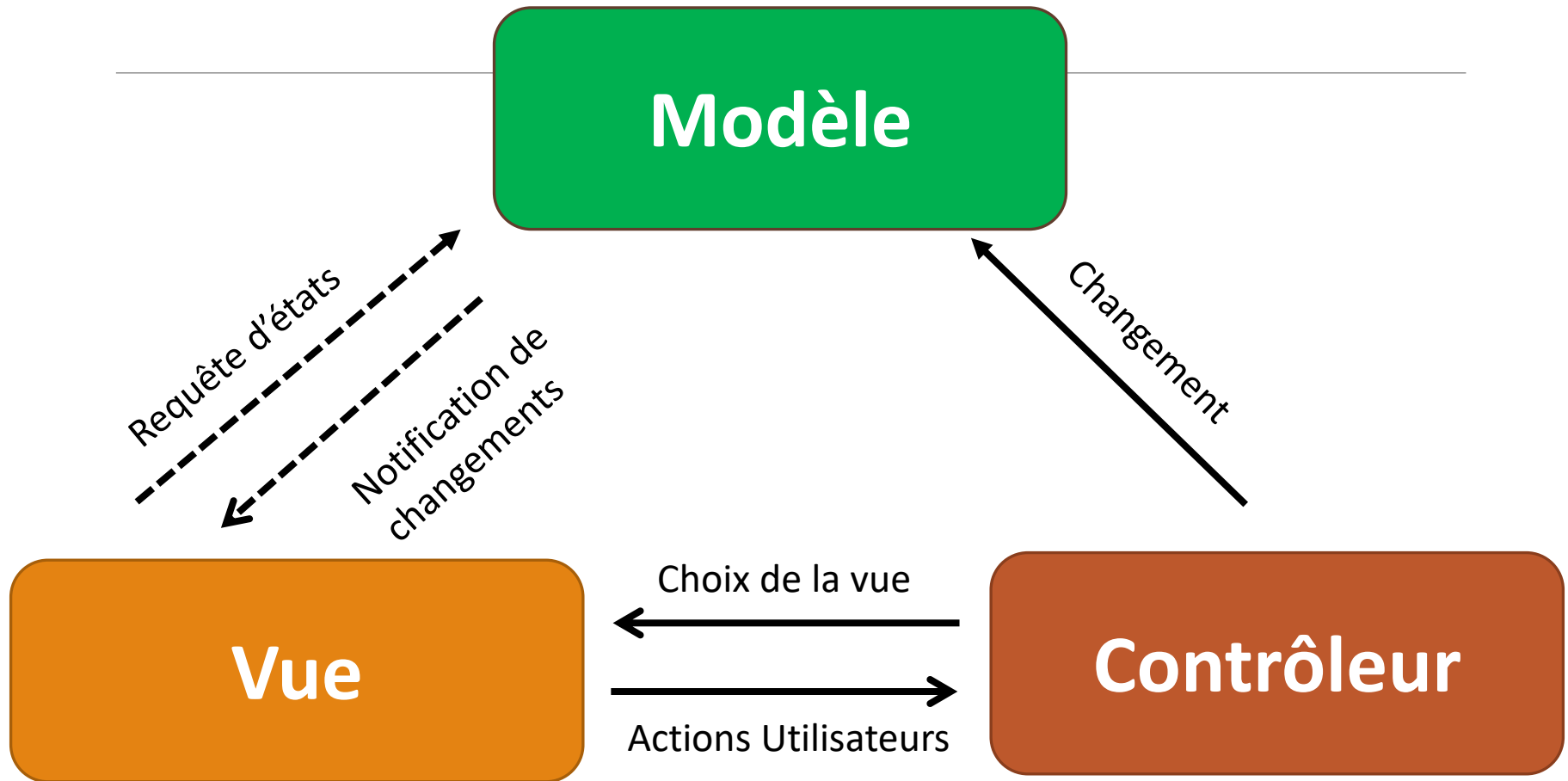
2. Vue:

- Expose les données du modèle à l'écran
- Interagit avec le modèle
- Il peut exister plusieurs vues

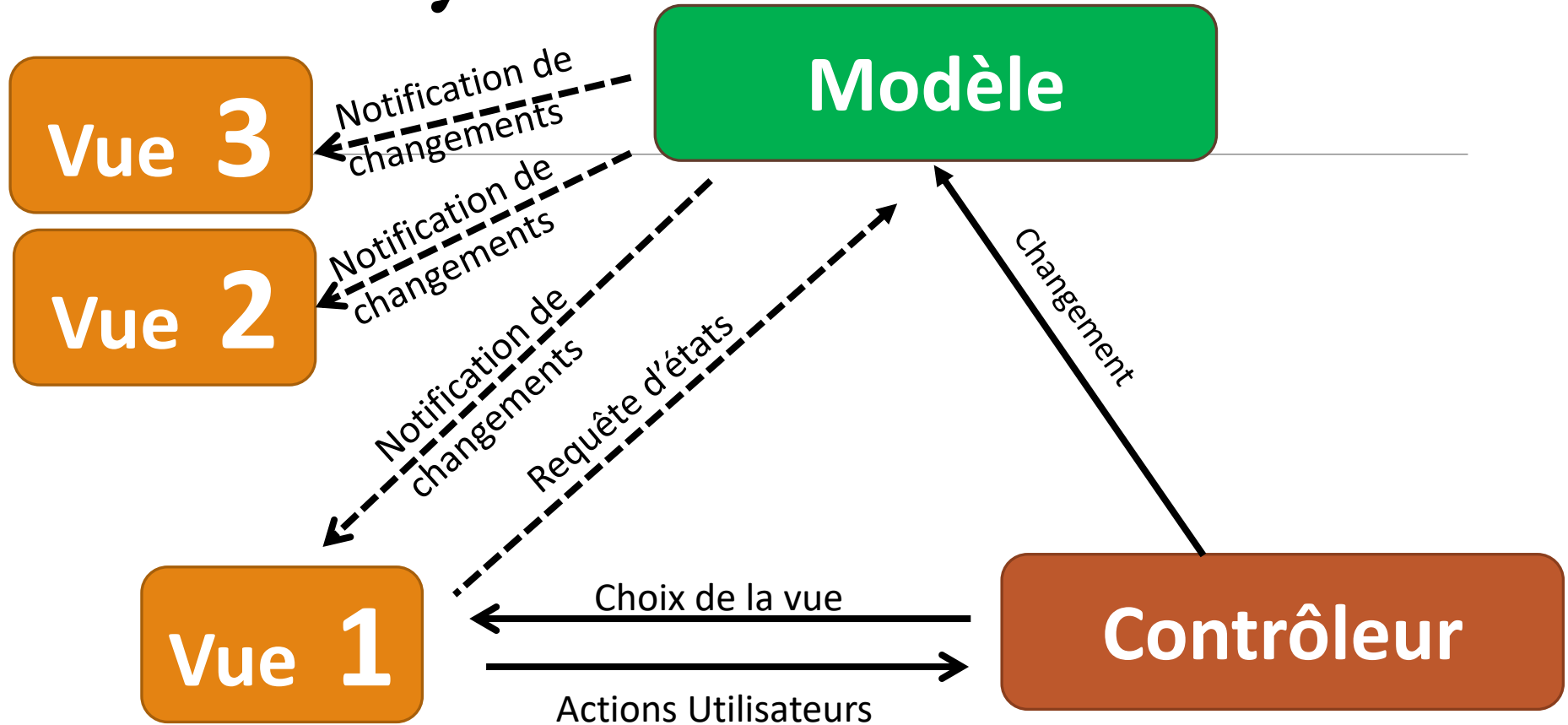
3. Contrôleur:

- Gère les interactions avec l'utilisateur
- Appelle le modèle en lecture et écriture
- S'occupe à la réécriture des URL

Interactions entre les couches MVC



MVC et Synchronisation



1. Il est possible d'avoir **plusieurs vues** qui présentent les données d'un même modèle.
2. Toutes les vues qui montrent la même chose sont **synchronisées**

Modèle en 5 couches

Présentation

- *Gère et assure l'affichage et l'impression graphiques (IHM)*
- *Intercepte les événements utilisateurs*
- *Assure l'appel à la couche « Coordination » (contrôleur)*

Coordination

- *Contrôle la cinématique des écrans*
- *Gère les erreurs et les exceptions levées*
- *Gère les sessions et les droits d'accès*

Services

- *Implémente les fonctionnalités du système*
- *Gère les appels aux objets métiers*

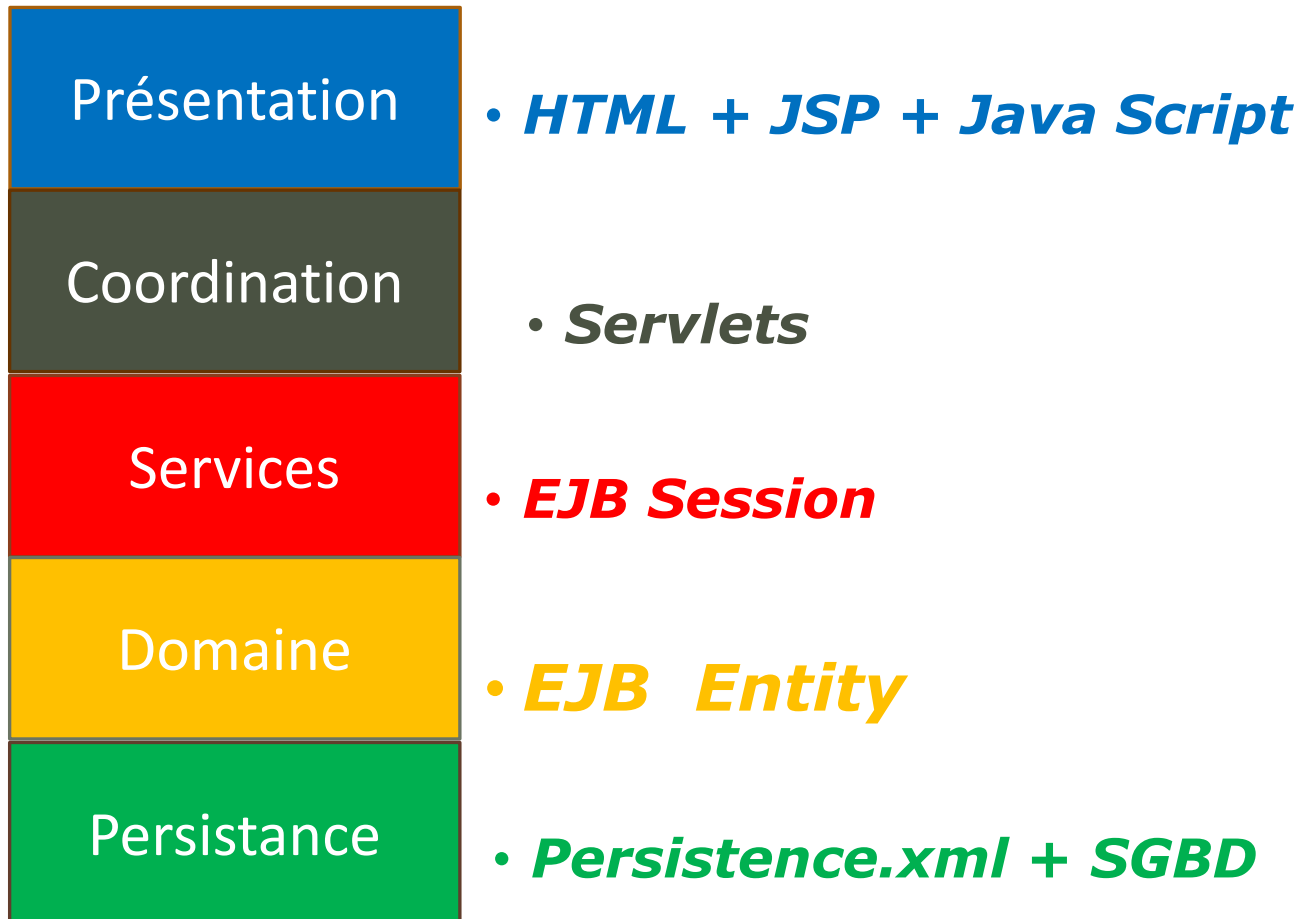
Domaine

- *Gère l'intégrité du modèle métier*
- *Fournit les moyens d'accès aux sources de données*

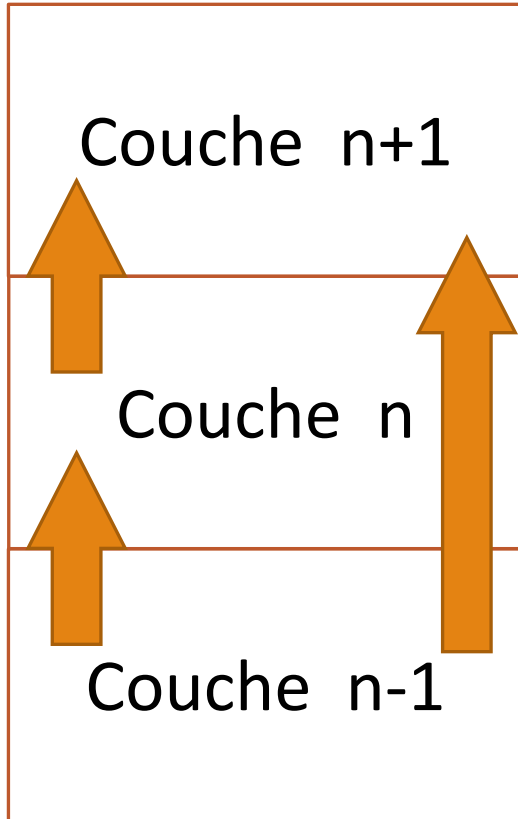
Persistance

- *Fournit les services de stockage des données*
- *Transforme les données en représentation objet*
- *Réalise des opérations CRUD (create, Read, update, Delete)*

Exemple de Modèle en couches: l'architecture JEE




Fonctionnement des couches



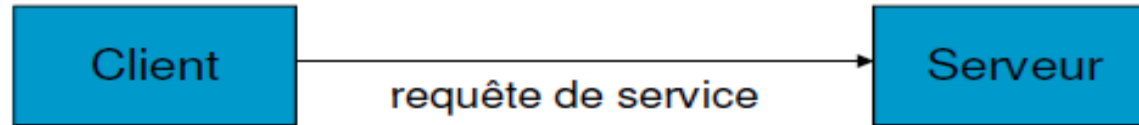
- Les couches inférieures publient leurs services à travers des interfaces aux couches supérieures
- Ces interfaces présentent un contrat entre les couches
- les couches externes sont plus abstraites (haut niveau) que les couches internes
- Les connecteurs dépendent du protocole d'interaction souhaité entre couches (systèmes fermés, systèmes ouverts)

B. Architectures physiques (N-niveaux)

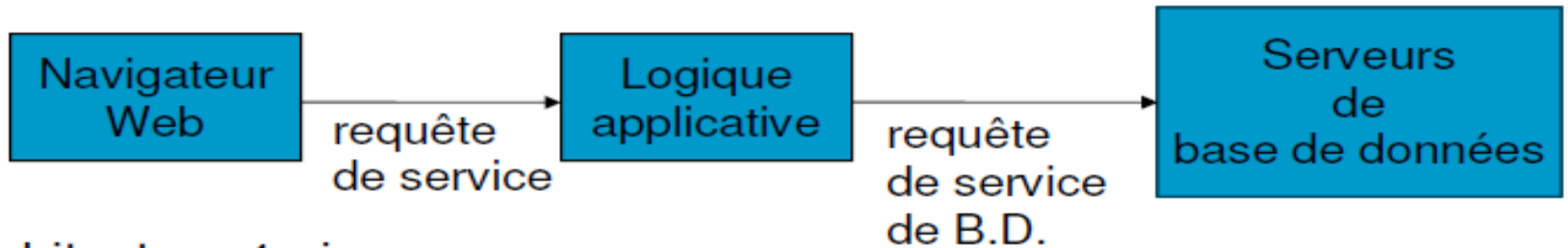
- ❖ *Le découpage physique sert à identifier les différents niveaux qui peuvent être répartis sur des composants matériels divers*
 **Décrire les systèmes distribués**
- ❖ *Un changement de niveau (ou tiers) est identifié dès qu'un module logiciel doit passer par un intermédiaire de communication (*middleware*)*
- ❖ Les tiers peuvent être ou non exécutés sur des machines différentes
- ❖ Un composant peut être un client ou un serveur et un connecteur peut être un protocole de communication (http, ftp, RMI, jdbc,..)

Architecture n-tiers

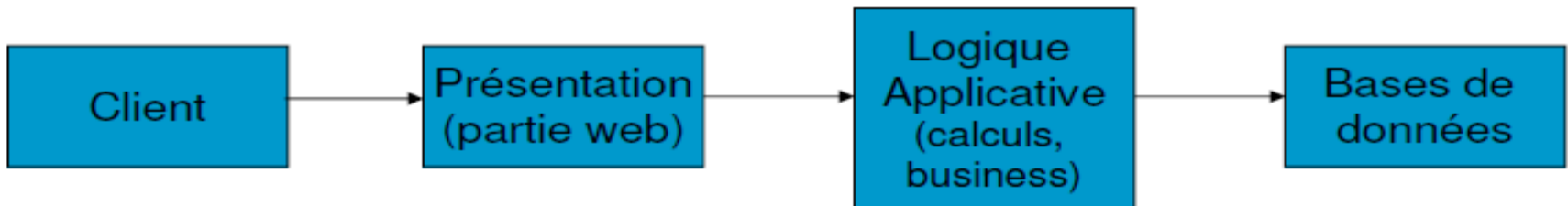
- Architecture 2-niveaux (client-serveur ou client lourd)



- Architecture 3-niveaux (client léger)

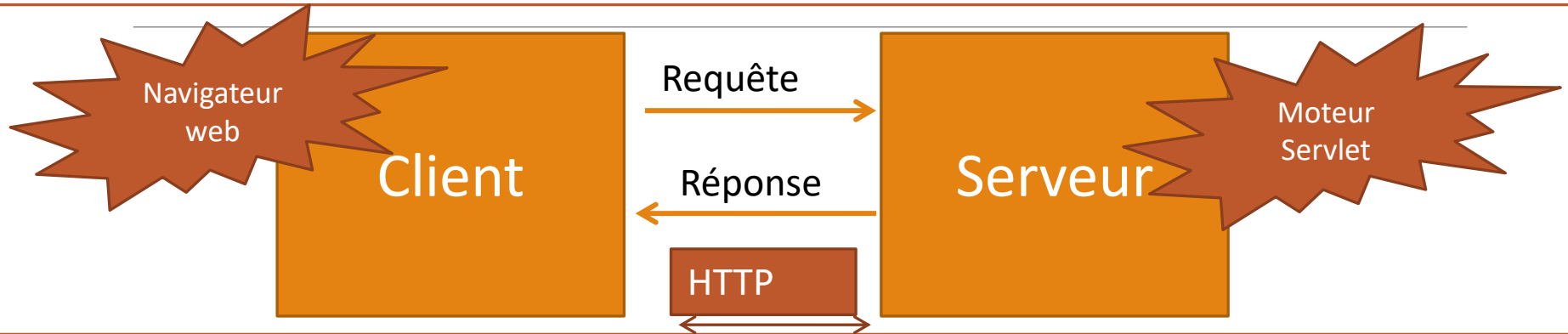


- Architecture 4-niveaux



Architecture 2-tiers

❖ Connue par « Client / Serveur »



- ❖ Le client demande une ressource et le serveur lui fournit directement en utilisant ses propres ressources
- ❖ Le client s'occupe de la présentation « IHM »
- ❖ Le serveur gère la logique métier et la gestion des données
- ❖ les ressources sont centralisées sur le serveur et le client gère les données indirectement (architecture sécurisée)

Architecture 3-tiers

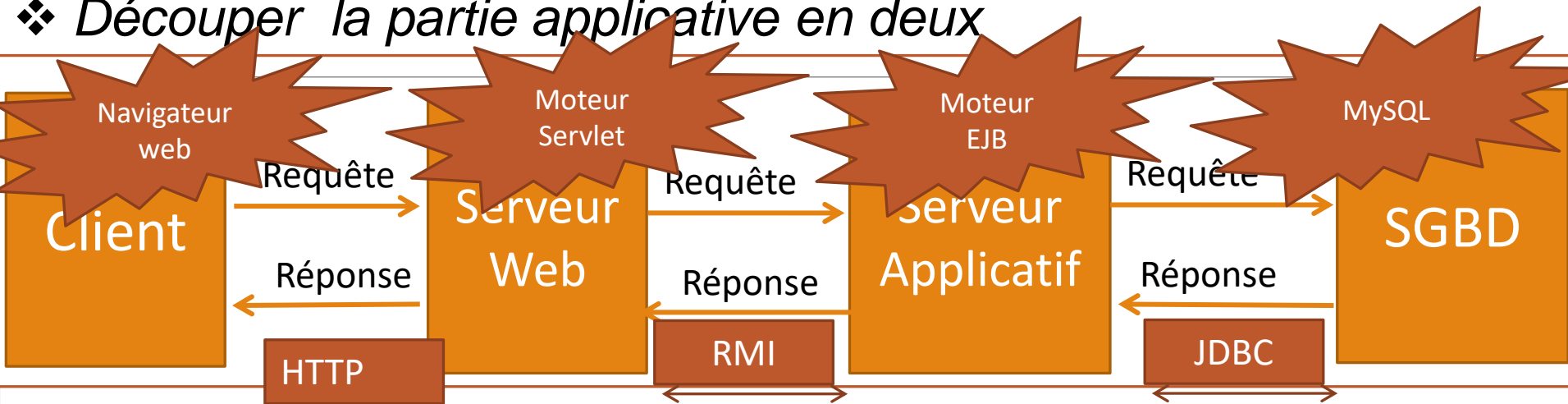
- ❖ Insertion d'un niveau *intermédiaire* entre le client et la BD



- ❖ Il s'agit de l'architecture la plus développée de nos jours
- ❖ Elle présente plus de *flexibilité* et dote le système d'un aspect *dynamique* et *évolutif*
- ❖ Cette architecture , plus divisée, favorise un développement plus ciblé et spécifique et permet un *travail de groupe* mieux réparti

Architecture 4-tiers

❖ *Découper la partie applicative en deux*



❖ *Le niveau applicatif constitue le cœur des traitements du système*

❖ *Le serveur web est un intermédiaire pour centraliser les requêtes web des clients*

❖ *Bien adaptée pour les applications Web devant supporter plusieurs types de clients (léger, lourds, mobiles, web,...)*

Comment développer un modèle architectural ?

❑ Commencer par faire une esquisse de l'architecture

- Décomposition en sous-systèmes
- Déterminer les principaux composants requis
- Sélectionner un style architectural: en couches, n-niveaux, etc

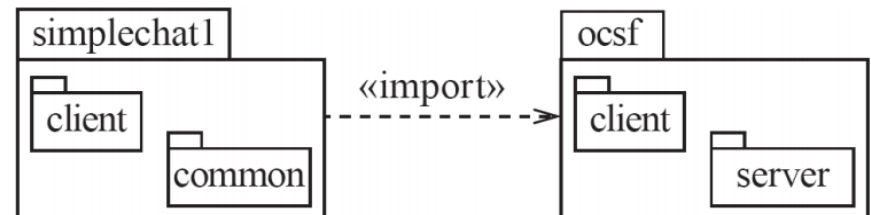
❑ Raffiner l'architecture

- Identifier les principales interactions entre les composants et les interfaces requises
- Décider comment chaque donnée et chaque fonctionnalité sera distribuée parmi les différents composants
- Déterminer si on peut réutiliser un composant existant ou si on peut construire un composant réutilisable

❑ Détailler l'architecture et la faire évoluer

Comment développer un modèle architectural ?

- ❑ Décrire l'architecture avec UML
- ❑ Tous les diagrammes UML peuvent être utiles pour décrire les différents aspects du modèle architectural
- ❑ Trois des diagrammes UML sont particulièrement utiles pour décrire une architecture logicielle
 - **Diagramme de package**

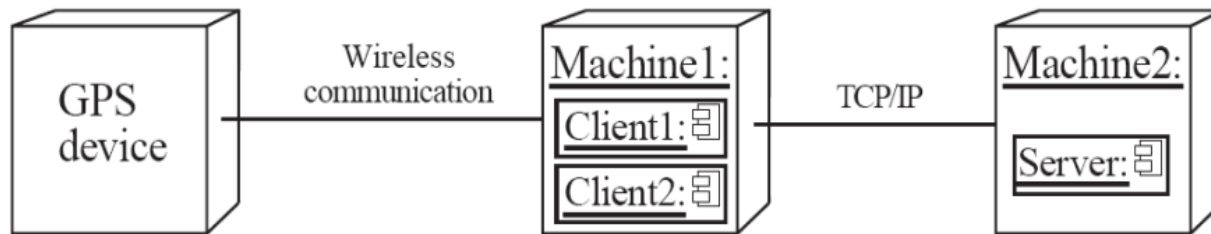


Comment développer un modèle architectural ?

- **Diagramme de composants**



- **Diagramme de déploiement**



Références

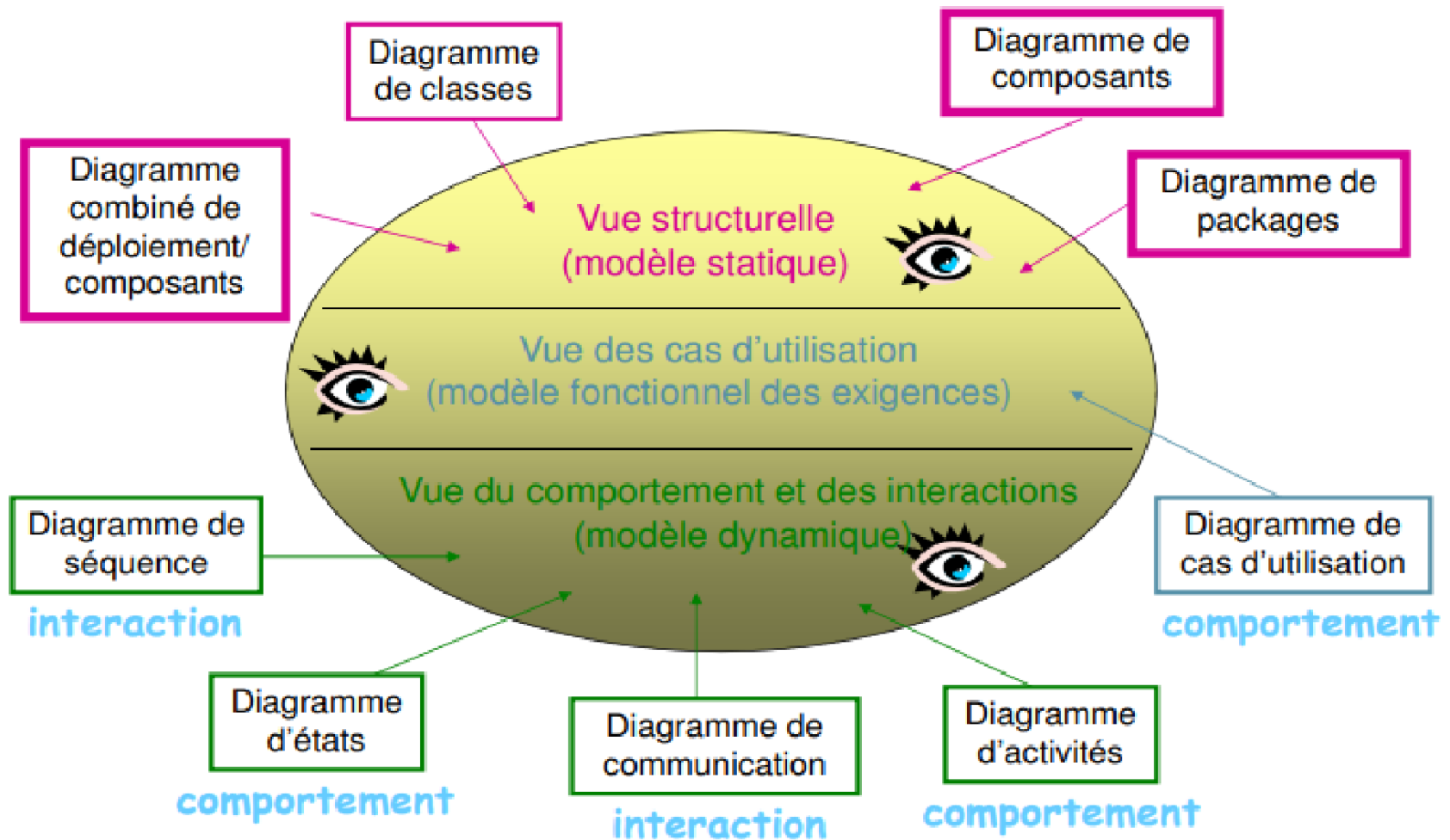
1. Cours «Architecture logicielle » ,*Lydie du Bousquet*
2. Wikipédia , *https://fr.wikipedia.org/wiki/Architecture_logicielle*
3. Wikipédia , *https://fr.wikipedia.org/wiki/Architecture_logicielle*



Chapitre 2

Modélisation architecturale en UML

Les différentes vues d'un programme



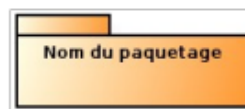
Modéliser avec UML

Les vues (structurelles) d'une architecture logicielle:

- **Vue logique.** Description logique du système décomposé en sous-systèmes (modules + interface) -> UML : diagramme de paquetages
- **Vue d'implémentation.** Description de l'implémentation (physique) du système logiciel en termes de composants et de connecteurs-> UML : diagramme de composants
- **Vue de déploiement.** Description de l'intégration et de la distribution de la partie logicielle sur la partie matérielle -> UML: diagramme combiné de composants et de déploiement

1 Le diagramme de paquetages

- Lorsque nous sommes en présence d'un système de grande taille, il peut être intéressant de le décomposer en plusieurs parties (appelées paquetage).
- Un paquetage est donc un regroupement de différents éléments d'un système (regroupement de classes, diagrammes, fonctions, interfaces...).
- Le paquetage permet une meilleure organisation de l'application.
- Il est représenté par un dossier avec son nom à l'intérieur:



2 Le diagramme de composants

- Offre une vue de haut niveau de l'architecture du système
- Utilisé pour décrire le système d'un point de vue implémentation
- Permet de décrire les composants d'un système et les interactions entre ceux-ci
- Illustre comment grouper concrètement et physiquement les éléments (objets, interfaces, etc.) du système au sein de modules qu'on appelle composants

2 Le diagramme de composants

Le composant:

- Unité autonome fournissant un service bien précis
- Représenté en UML par un classeur portant le stéréotype <<component>>
- Comportant une ou plusieurs interfaces requises ou offertes
- Pouvant contenir d'autres composants
- Composé de deux parties :
 - une partie masquée : ensemble de classes
 - une partie visible : ensemble d'interfaces

2 Le diagramme de composants

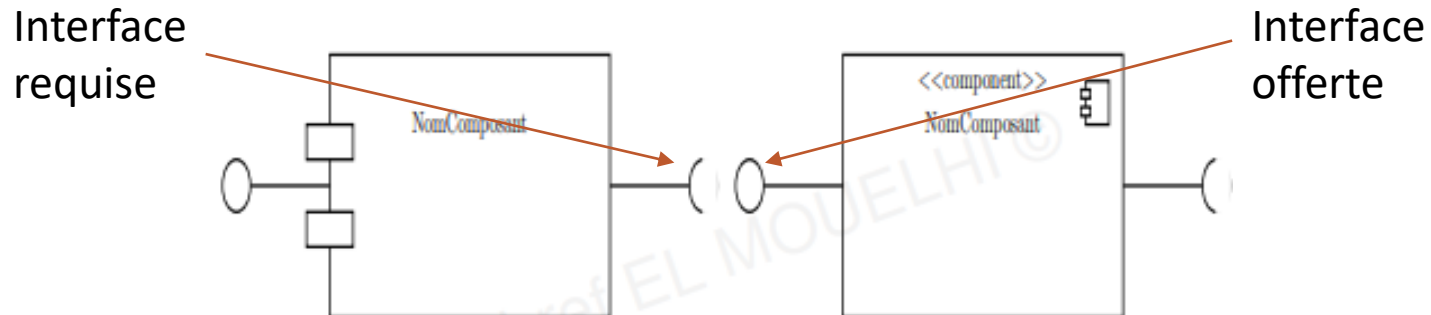
- Unité autonome fournissant un service bien précis
- Représenté en UML par un classeur portant le stéréotype <<component>>
- Comportant une ou plusieurs interfaces requises ou offertes
- Pouvant contenir d'autres composants
- Composé de deux parties :
 - une partie masquée : ensemble de classes
 - une partie visible : ensemble d'interfaces

2 Le diagramme de composants

- Deux présentations possibles d'un composant en UML:

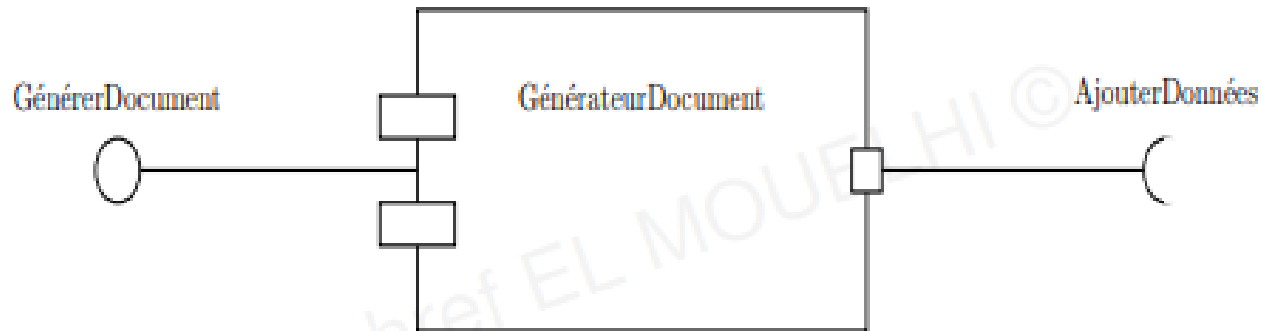


- Pour visualiser les interfaces requise et offerte

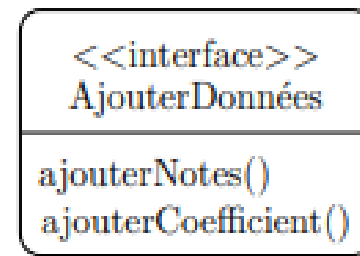
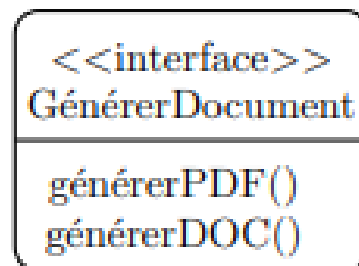


2 Le diagramme de composants

- Exemple d'un composant permettant de générer des documents (diplôme, attestation de réussite...) pour les étudiants

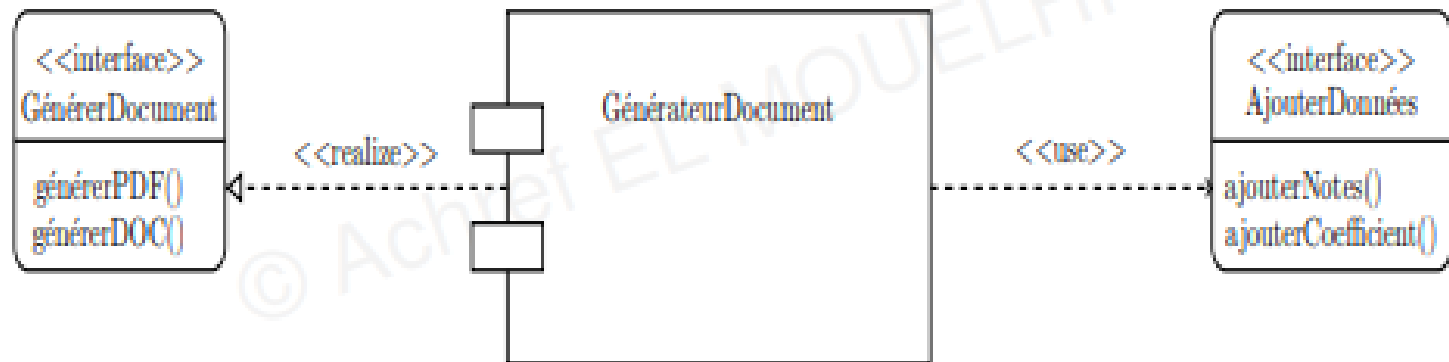


- Les interfaces:



2 Le diagramme de composants

- On peut représenter explicitement la relation avec les interfaces:



- Un diagramme de structure d'UML (statique)
- Décrivant l'architecture physique ainsi que les relations entre les composants logiciels et matériels d'une application
- Expliquant le déploiement de l'application en terme de réseau et communication

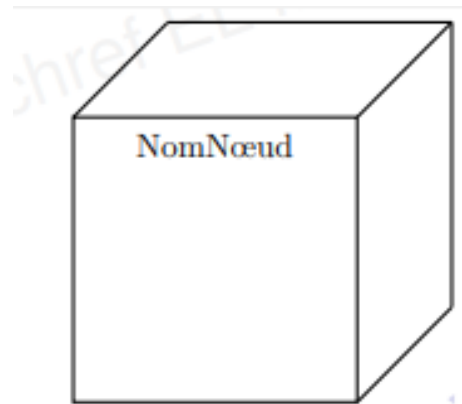
Le diagramme de déploiement

Les éléments constituant un diagramme de déploiement:

- Nœud
- Lien
- Stéréotype
- Artefact
- Composant

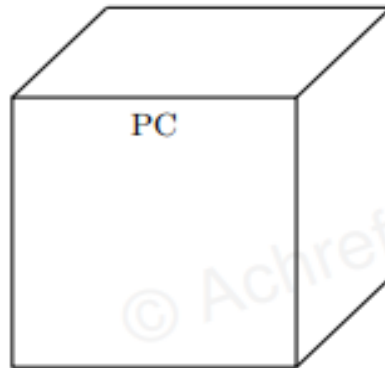
Nœud:

- Composant mécanique : ordinateur, serveur, imprimante, environnement d'exécution...
- Pouvant contenir d'autres nœuds ou artefacts
- Représenté par des cubes

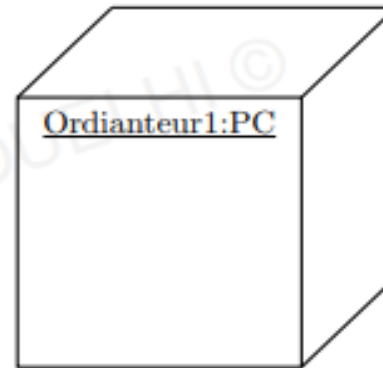


Le diagramme de déploiement

- Exemple d'un nœud et une instance de nœud



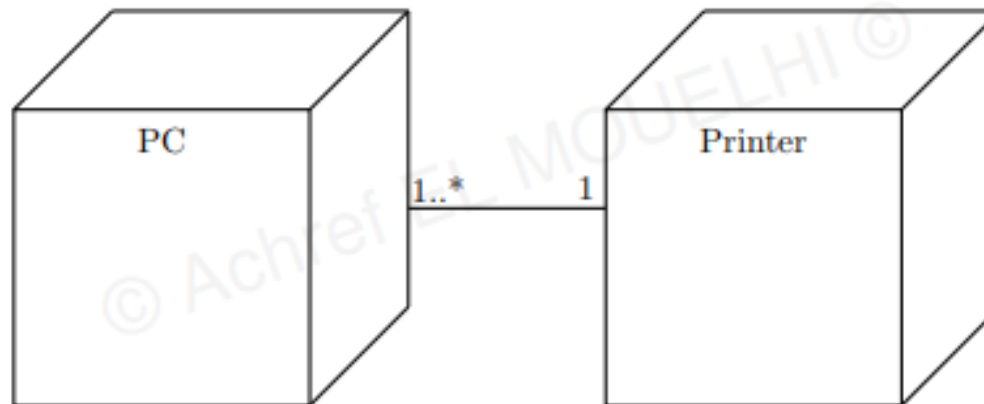
Un nœud



Une instance de nœud

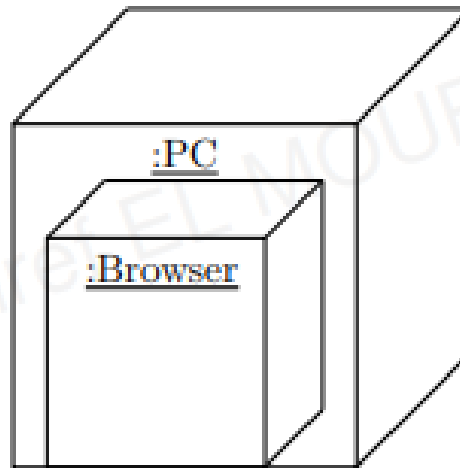
Lien:

- élément permettant de connecter les nœuds
- On peut définir aussi la multiplicité



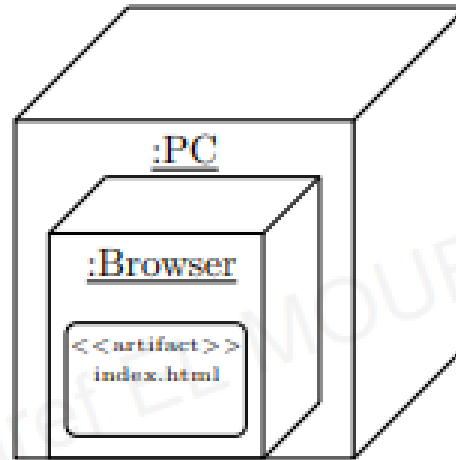
Le diagramme de déploiement

- Un nœud peut contenir d'autres nœuds



Le diagramme de déploiement

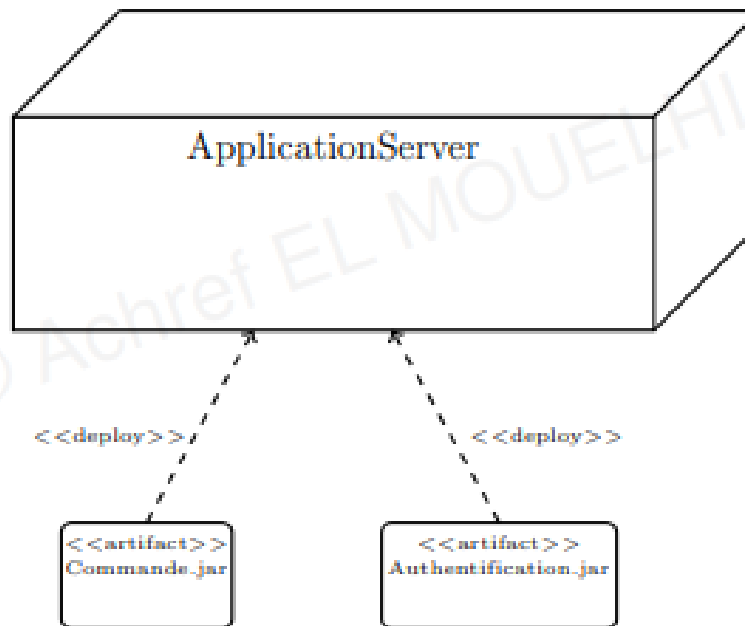
- Un nœud peut contenir aussi un artefact



- Artefact: Élément concret de l'application (fichier contenant du code source, table d'une base de données, script...)

Le diagramme de déploiement

- Un artefact déployé dans un nœud peut être représenté en utilisant le stéréotype `<<deploy>>`



3

Le diagramme de déploiement

- Exemple de composants répartis sur un déploiement

