CHAPITRE 8

Modèle architectural
Architecture 3 niveaux et paquetages

Sommaire:

Vue architecturale

- Architectures à 3 niveaux et multiniveaux
- Paquetages (« *Packages* »)
- Introduction aux patrons (« *Patterns* ») et cadres de référence (« *Frameworks* »)
- Diagrammes de composants
- Diagrammes de déploiement

Définition

L'architecture logicielle d'un système est sa décomposition en un certain nombre de sous-systèmes, parmi lesquels on retrouve, entre autres, un ou plusieurs sous-systèmes regroupant les entités liées au domaine du problème.

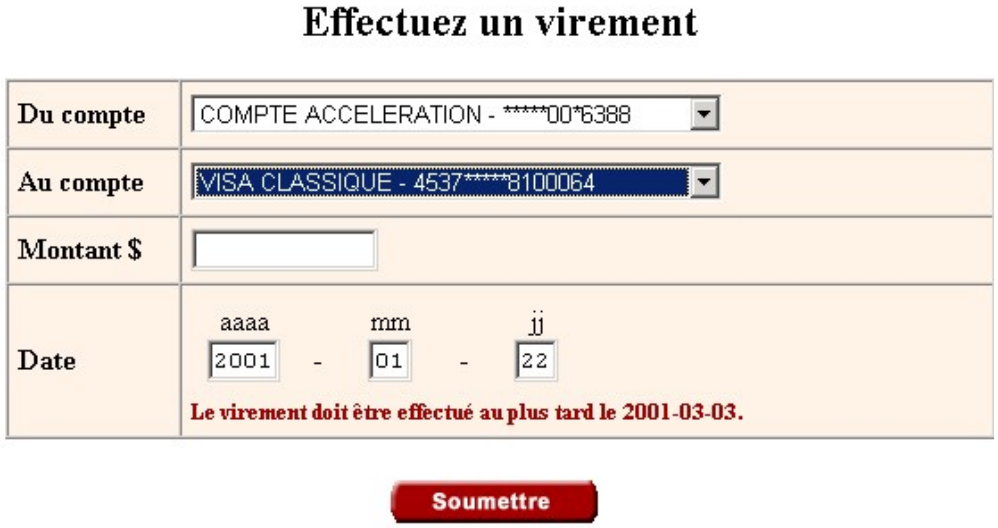
Architecture classique à trois niveaux

Une architecture classique pour la conception d'applications comprenant une interface usager et un système de sauvegarde persistante des données est l'architecture à trois niveaux:

1. Présentation,
2. Logique d'application,
3. Système de sauvegarde.

Architecture classique à trois niveaux

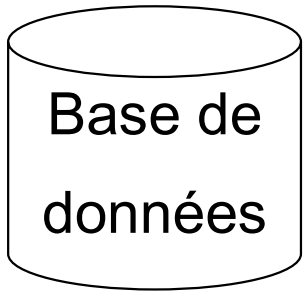
Présentation



Logique d'application

GestionDesComptes EnregistrementTransactions

Système de sauvegarde

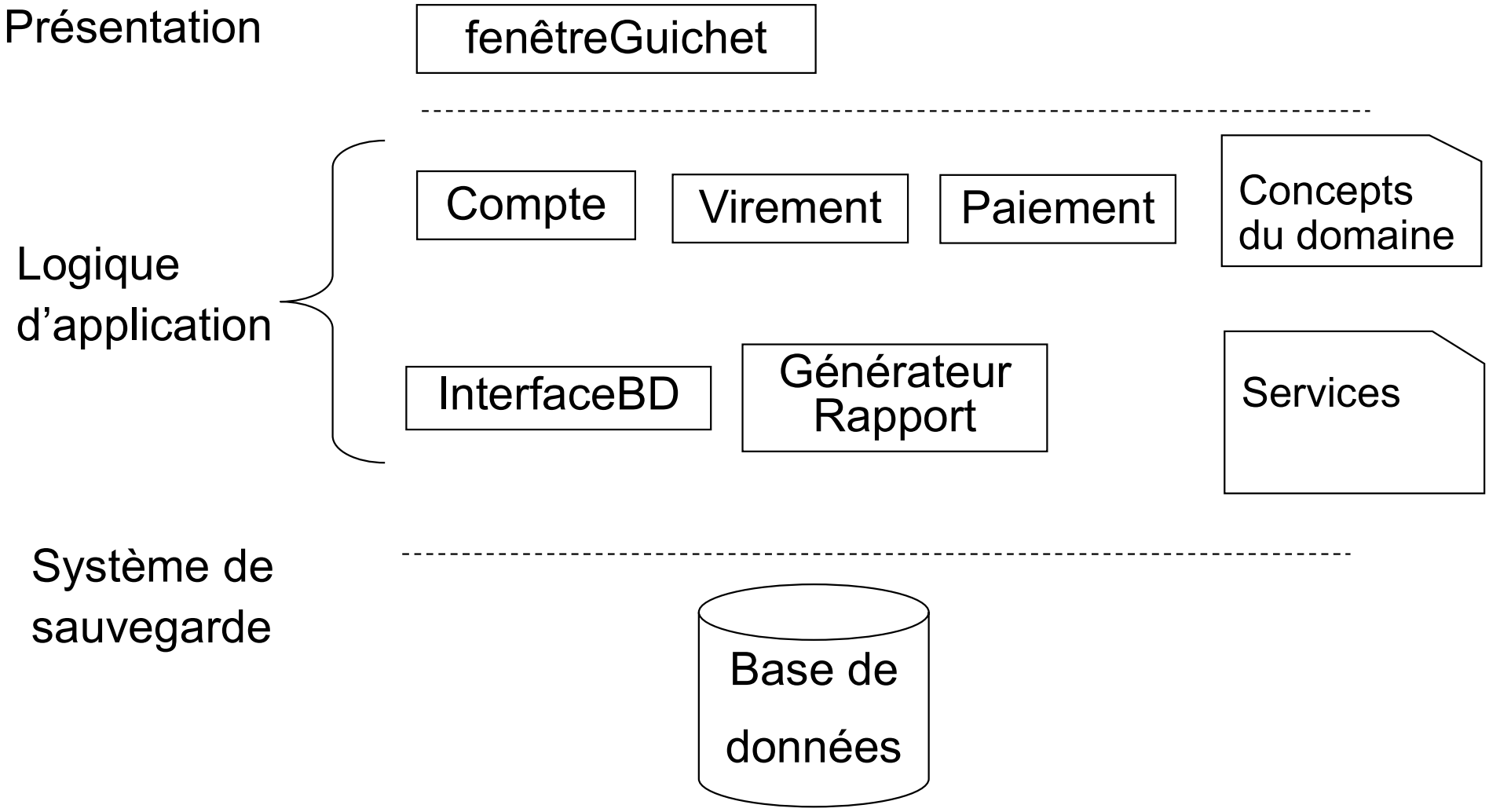


Architecture multiniveau

Les architectures multiniveaux des systèmes orientés objet incluent la séparation des responsabilités proposée par l'architecture à trois niveaux.

Le niveau de la logique d'application est généralement décomposé en couches plus minces organisées autour de classes.

Architecture multiniveau



Architecture multiniveau

Les niveaux de logique d'application et de services peuvent être divisés en niveaux plus étroits:

Ex. Services:

- Haut niveau: génération de rapports, formatage, algorithmes de calcul (vecteurs, matrices, etc.)
- Bas niveau: gestion de fichiers, communication.

Architecture multi niveaux : déploiement

Un système *logiquement* conçu selon une architecture à trois niveaux peut être *physiquement* déployé selon différentes configurations:

1. Présentation et logique d'application sur un ordinateur client, système de sauvegarde sur un ordinateur serveur,
2. Présentation sur un ordinateur client, logique d'application sur un serveur d'application et sauvegarde sur un serveur séparé de sauvegarde.

Architecture multiniveau : avantages

Les avantages d'une architecture multiniveau incluent:

1. La séparation de la logique d'application dans des composantes séparées qui peuvent être réutilisées dans d'autres systèmes,
2. La possibilité de répartir les niveaux sur différents nœuds de calcul, et dans différents processus,
3. L'assignation de développeurs à la construction de chaque niveau: parallélisation des efforts et spécialisation des intervenants.

Modéliser les vues architecturales: Diagrammes de paquetages

Vue logique

Diagrammes de paquetages
Diagrammes de classes
Diagrammes d'objets

Vue implémentation

Diagrammes de composantes

Vue utilisateur

Diagrammes de cas

Vue comportement

Diagrammes d'état
Diagrammes d'activités
Diagrammes de séquence
Diagrammes de collaboration

Vue déploiement

Diagrammes de déploiement

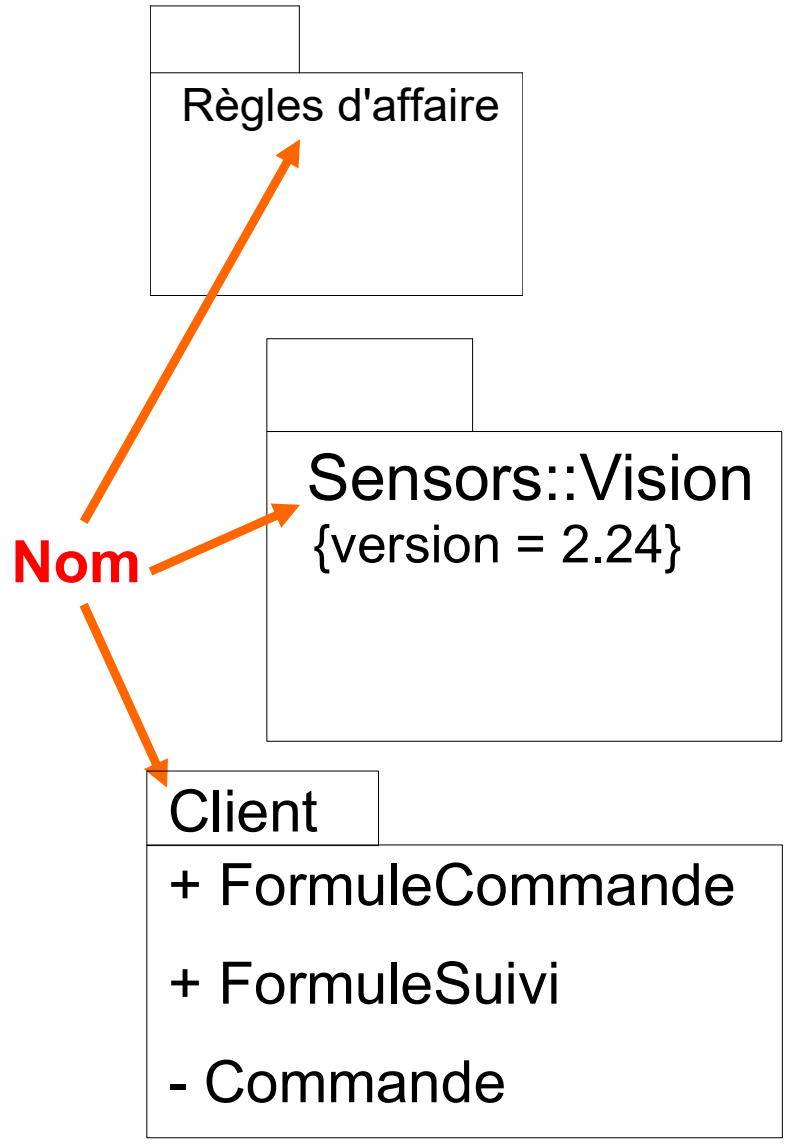
Documentation de l'architecture, les paquetages: pourquoi ?

- Visualiser, spécifier, construire et documenter de grands systèmes implique la manipulation d'un grand nombre de classes, interfaces, composantes, noeuds, diagrammes et autres éléments.

- Il est donc utile de regrouper les éléments communs en paquetages afin de pouvoir les comprendre et les manipuler plus facilement.

Paquetage: nom

- Chaque paquetage doit avoir un nom unique,
- Chaque éléments d'un paquetage doit avoir un nom unique.
  - On ne peut avoir deux classes nommées `liste` dans le même paquetage, mais on peut avoir une classe nommée `liste` dans le paquetage `p1` et une autre nommée `liste` dans le paquetage `p2`. Ces deux classes peuvent être distinguées par le paquetage source (`p1::liste` et `p2::liste`).
  - Deux éléments de types différents peuvent avoir le même nom dans un paquetage, mais il est préférable d'avoir des noms différents.

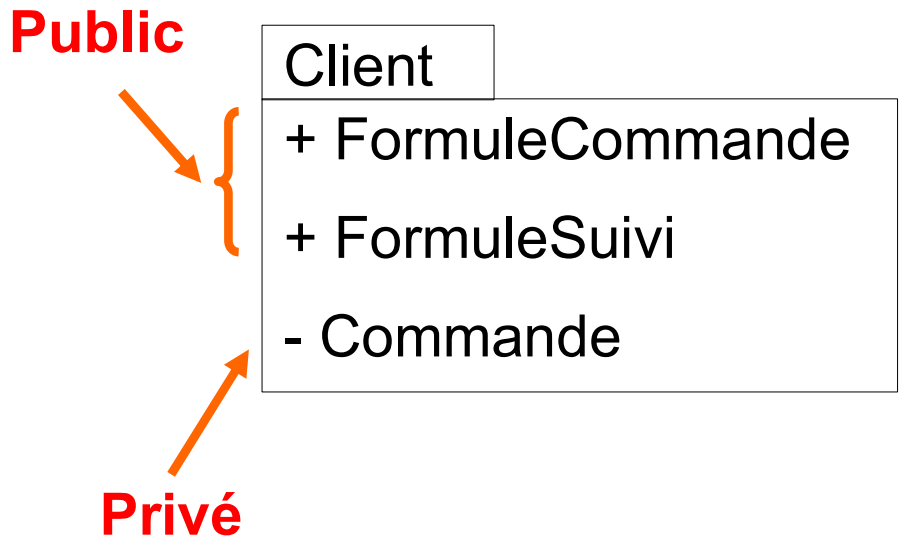


Paquetage: éléments inclus

- Un paquetage peut inclure des éléments tels des classes, des interfaces, des composants, des nœuds, des collaborations, des cas d'utilisation, des diagrammes et même d'autres paquetages,
- Chaque élément appartient à un et un seul paquetage,
- Si le paquetage est détruit, tous les éléments inclus le sont aussi,
- Un paquetage peut contenir d'autres paquetages.
  - `Véhicule::Voiture::Moteur`
- Il est recommandé d'arrêter à trois niveaux d'inclusion.
- Il existe des mécanismes d'importation et d'exportation entre les paquetages pour résoudre le problème.

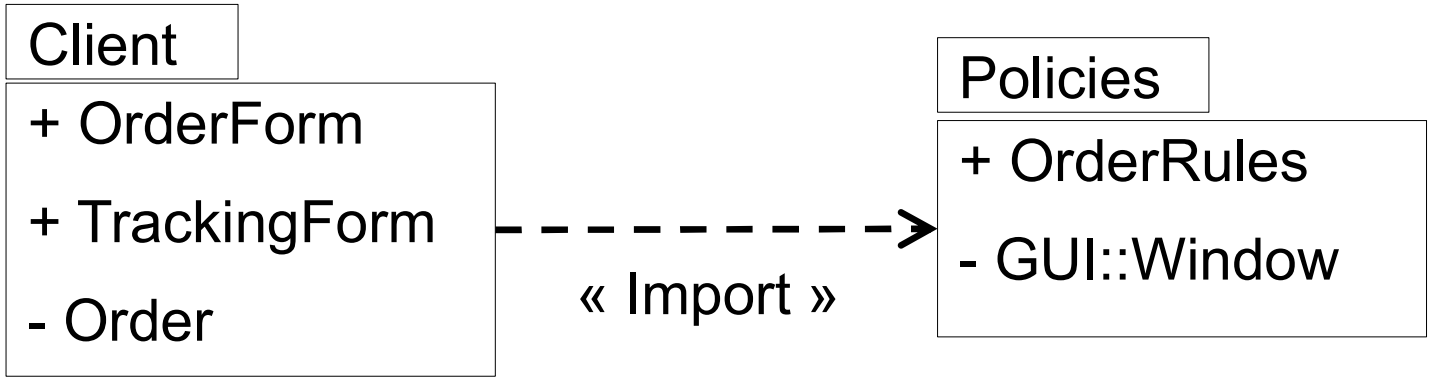
Visibilité d'une classe

- Classe publique (+)
  - La classe est visible pour tous les éléments qui importent le paquetage englobant la classe.
- Classe privée (-)
  - La classe n'est visible que pour les classes qui appartiennent au même paquetage.
- Classe protégée (#)
  - La classe est visible pour les classes qui appartiennent au même paquetage et pour les classes contenues dans les paquetages enfants du paquetage englobant.



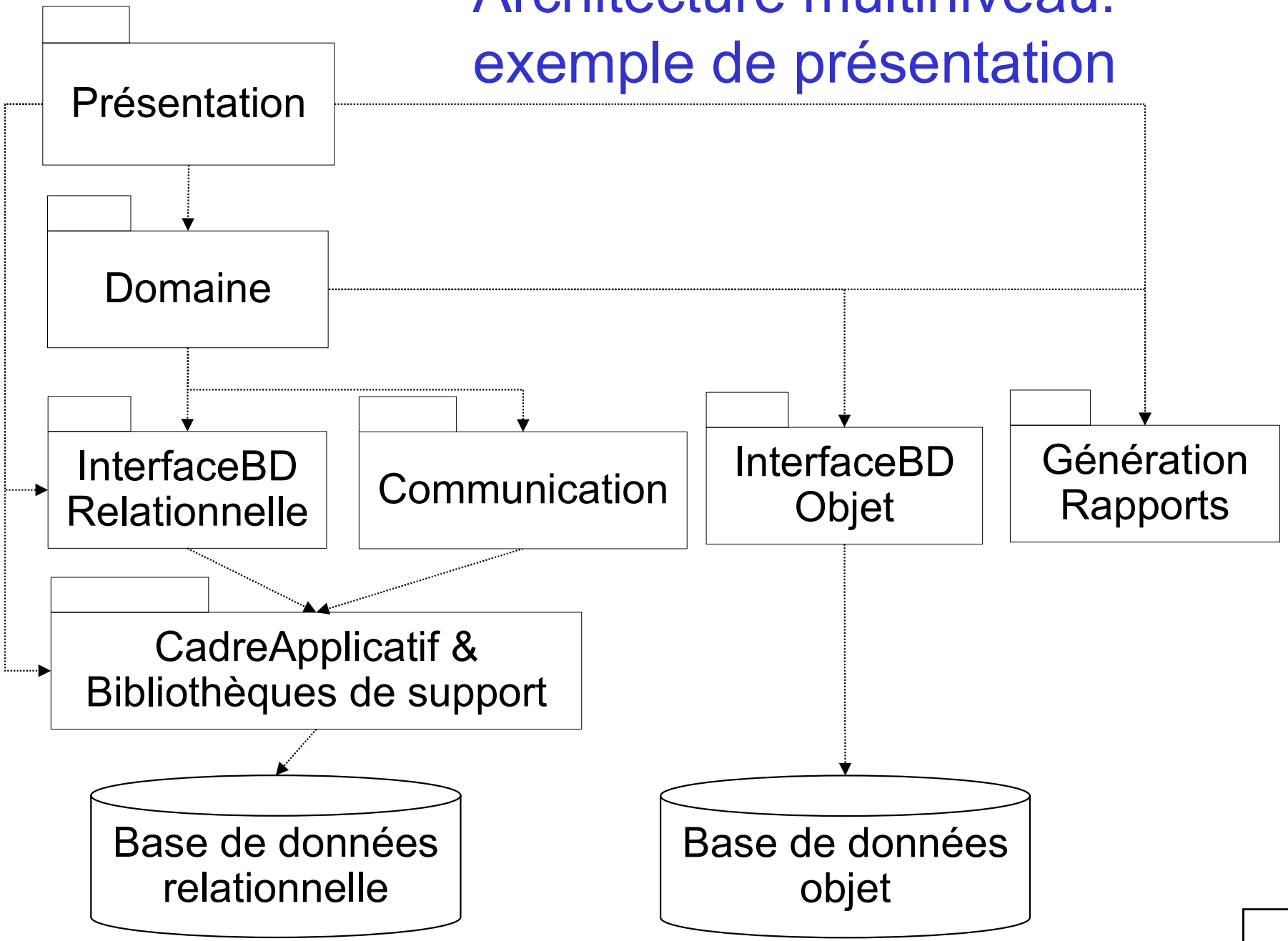
Paquetage: importer et exporter

- La source voit la cible, mais pas l'inverse.
- Représenté par une dépendance avec le stéréotype « import » ou « access ».
- Les éléments exportés doivent être publics.



- Le paquetage *Client* importe la classe *OrderRules* du paquetage *Policies*. Les classes du paquetage *Client* peuvent utiliser la classe *OrderRules.*

Architecture multiniveau: exemple de présentation



Identification des paquetages

On regroupe dans un paquetage:

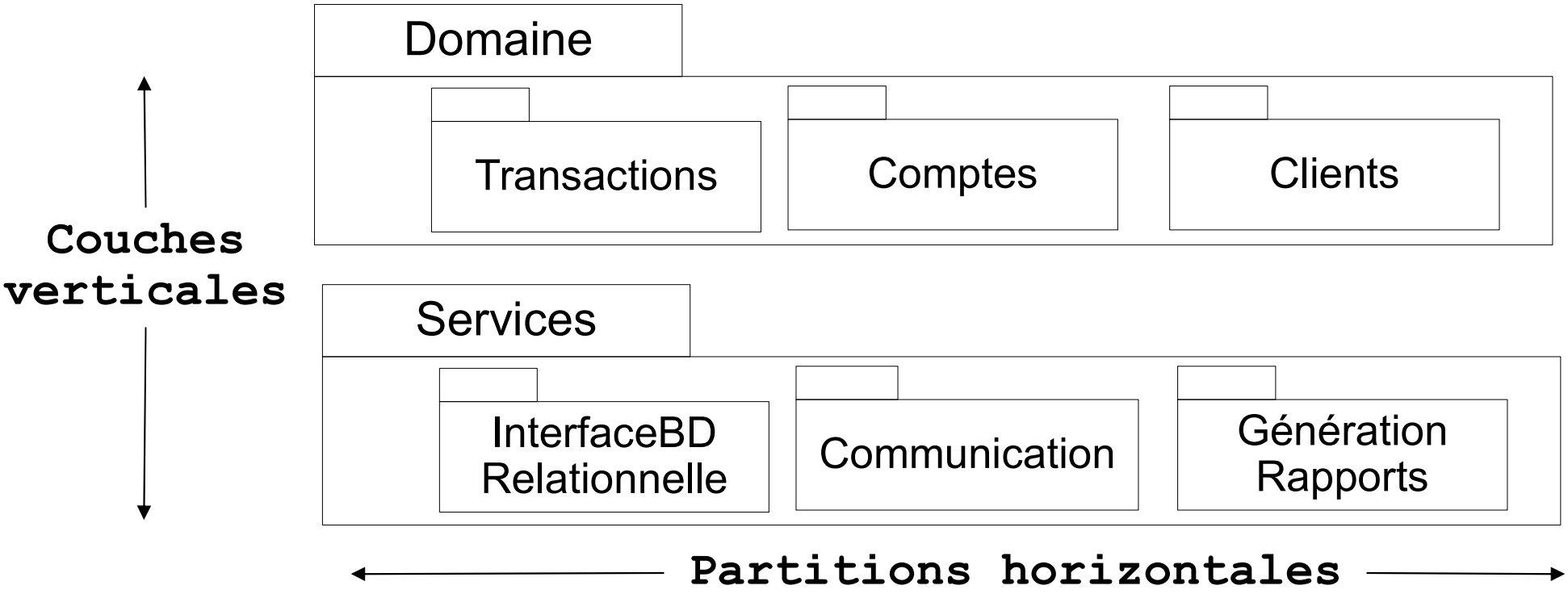
- Des éléments qui fournissent un service commun ou une famille de services reliés les uns aux autres,
- Des éléments qui peuvent être vus, à un certain niveau d'abstraction, comme un seul concept avec un ensemble de responsabilités fortement reliées.

On met dans des paquetages séparés:

- Des éléments ayant peu de couplage ou de relations entre eux,
- Des éléments qui peuvent être réutilisés séparément.

Partition et subdivision des niveaux

Les couches de l'architecture représentent les subdivisions verticales dans l'architecture, alors que les partitions sont les subdivisions horizontales de sous-systèmes parallèles à l'intérieur d'une couche:



Visibilité entre les paquetages

Les relations entre les composants des paquetages satisfont généralement aux principes suivants:

- **Accès aux paquetages de présentation** – aucun autre paquetage n'a un accès direct aux entités de la couche de présentation. Si nécessaires, les communications sont indirectes (*Patron Observer*),
- **Accès aux paquetages du domaine** – les autres paquetages, ex. présentation, accèdent à plusieurs classes représentant les concepts du domaine,
- **Accès aux paquetages de service** – les autres paquetages, ex. domaine et présentation, n'accèdent qu'à une seule ou à un nombre très limité de classes (*Patron Façade*).

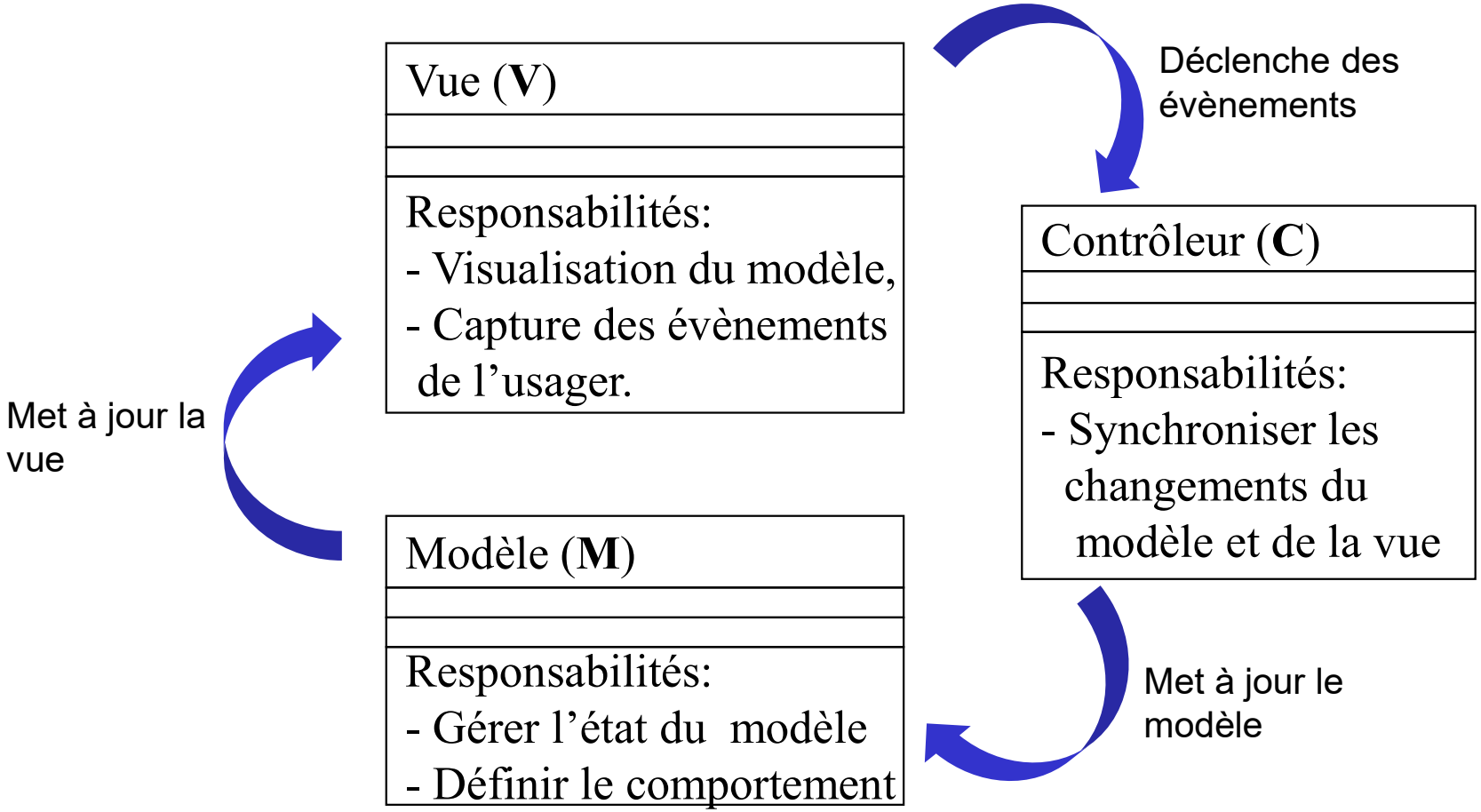
Pas d'accès direct à la présentation: le modèle MVC (Modèle-Vue-Contrôles)

On ne veut généralement pas fournir d'accès direct aux classes de présentation aux autres paquetages du système. Normalement, les classes de présentation sont très liées à l'application alors que les classes du domaine et des services pourraient être réutilisées dans d'autres projets:

➔ Principe de la séparation entre le Modèle, la Vue et les Contrôles.

Modèle MVC

Responsabilités dans le modèle MVC



Modèle MVC

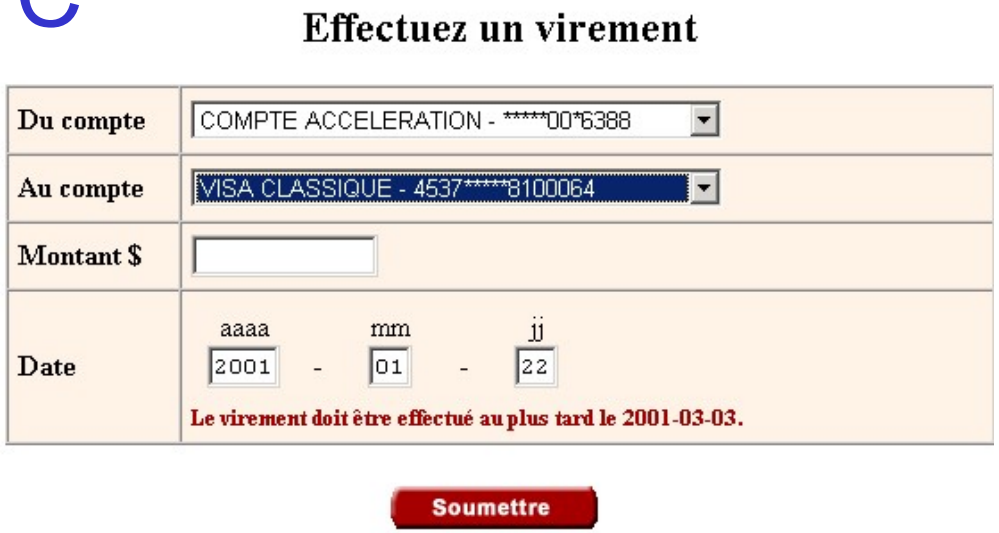
Caractéristiques architecturales du modèle MVC:

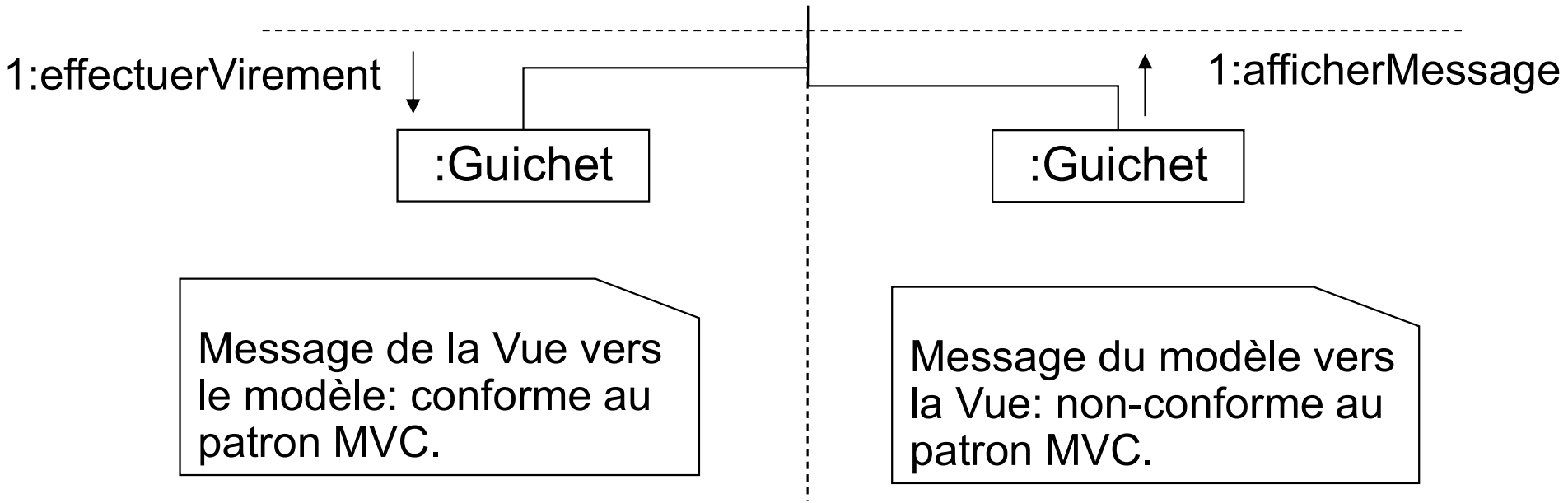
- Il ne devrait pas y avoir d'énoncés dans les méthodes d'une classe du niveau Modèle (logique d'application) qui envoient directement un message à un objet du niveau Vue (présentation),

  ➔ Pas de relation de dépendance du Modèle sur la Vue

- Les classes du niveau Modèle devraient encapsuler complètement l'information et le comportement liés à la logique d'application,
- Le niveau des classes de Vue est relativement « mince ». Elles sont responsables des entrées et des sorties, mais ne contiennent pas de données ou de fonctionnalités liées à l'application.

Modèle MVC





Justification du modèle MVC

- Propose une définition cohérente du modèle, axée sur les processus issus du domaine, et non pas sur les interfaces personne-machine,
- Permet une séparation des tâches de développement des niveaux,
- Minimise l'impact de changements dans les spécifications liées à l'interface sur le niveau de la logique d'application,
- Permet l'ajout simple de nouvelles vues par dessus une couche liée au domaine, sans modifications à la couche domaine.

Justification du modèle MVC (suite)

- Permet la définition de vues multiples sur les mêmes données,
- Permet l'exécution de la couche liée au modèle de façon indépendante de l'interface, par exemple dans un environnement d'exécution en lots,
- Garantit une plus grande portabilité en facilitant la migration de la couche modèle vers un nouveau type d'environnement de gestion d'interface.

Communications indirectes

La communication entre le Modèle et la Vue du modèle MVC est un exemple de nombreuses situations architecturales nécessitant une communication indirecte entre des entités d'un système.

Ce type de communication nécessite l'introduction de Patrons spécialisés qui seront discutés en détail dans la conception:

- Patron Observer,
- Patron Mediator.

Interface aux paquetages de service: le patron Façade

Comment les autres composantes du système devraient-elles interfacer avec les classes dans un paquetage de la couche Service ?

➔Lorsqu'une classe est définie pour fournir une interface standard à un groupe d'autres composants ou à un ensemble disparate d'interfaces, on appelle cette classe une **Façade**.