Master Informatique

Génie Logiciel 2

178UD01

Cours n°2

UML pour la conception + Etude de cas (1/3) + Trois patterns GRASP + Conception architecture par niveau

Plan du cours

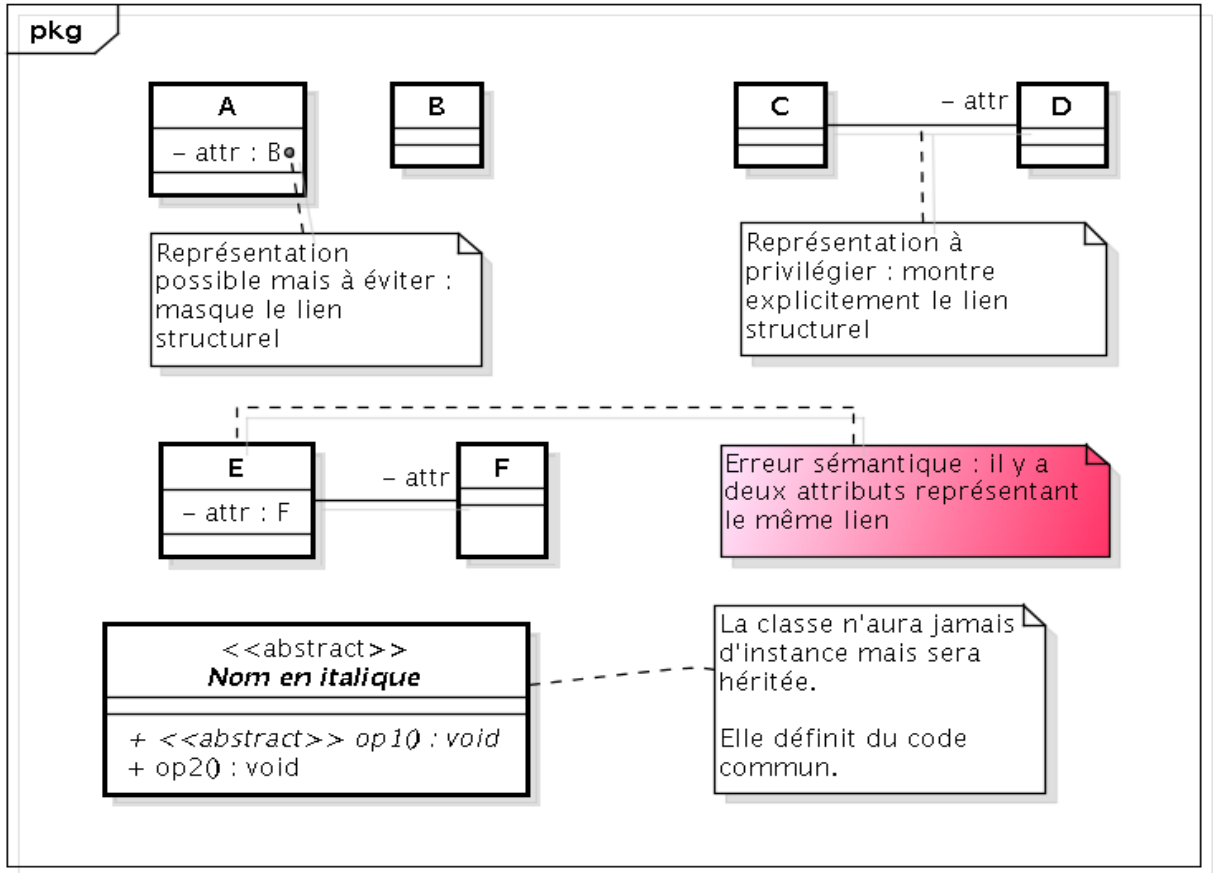
- UML pour la conception logicielle
- Présentation de l'étude de cas
- Trois patterns GRASP
- Conception de l'architecture par niveau

Plan du cours

- UML pour la conception logicielle
- Présentation de l'étude de cas
- Trois patterns GRASP
- Conception de l'architecture par niveau

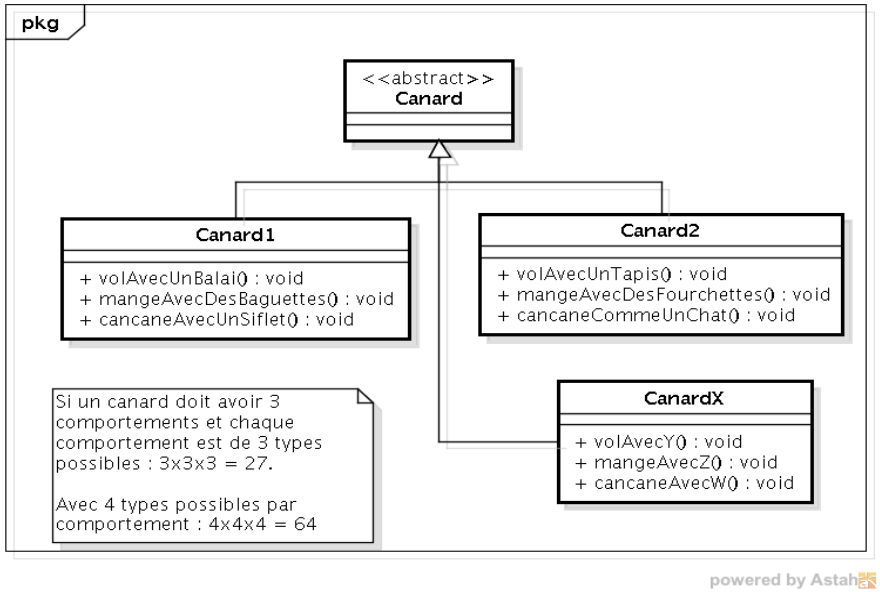
Les classes (1/5)

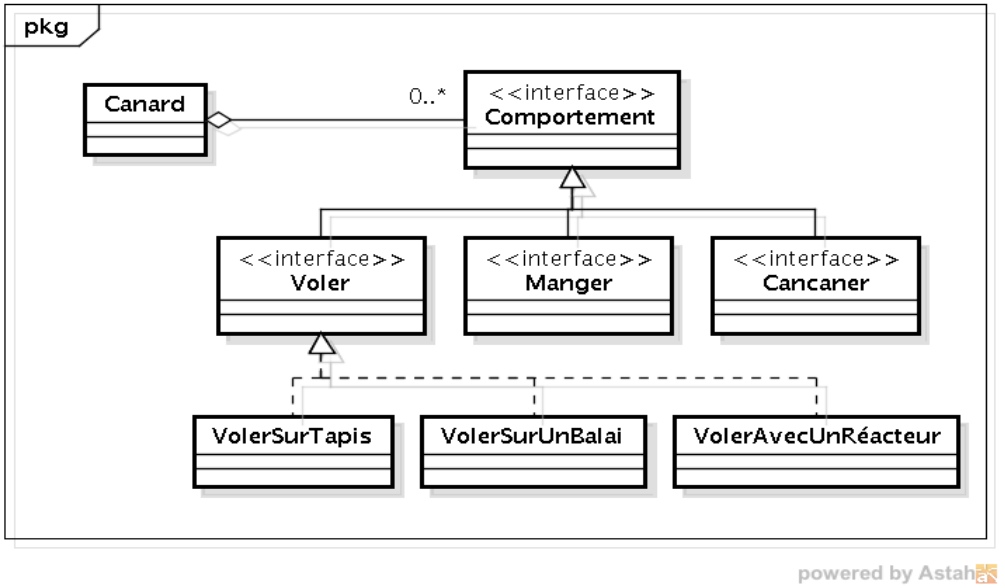
- Les classes
  - attribut de type objet : défini un lien structurel
  - méthode : peut être abstraite mais la classe doit contenir au moins une méthode concrète (ou des attributs statiques)



Héritage, classe abstraite *vs* interface (2/5)

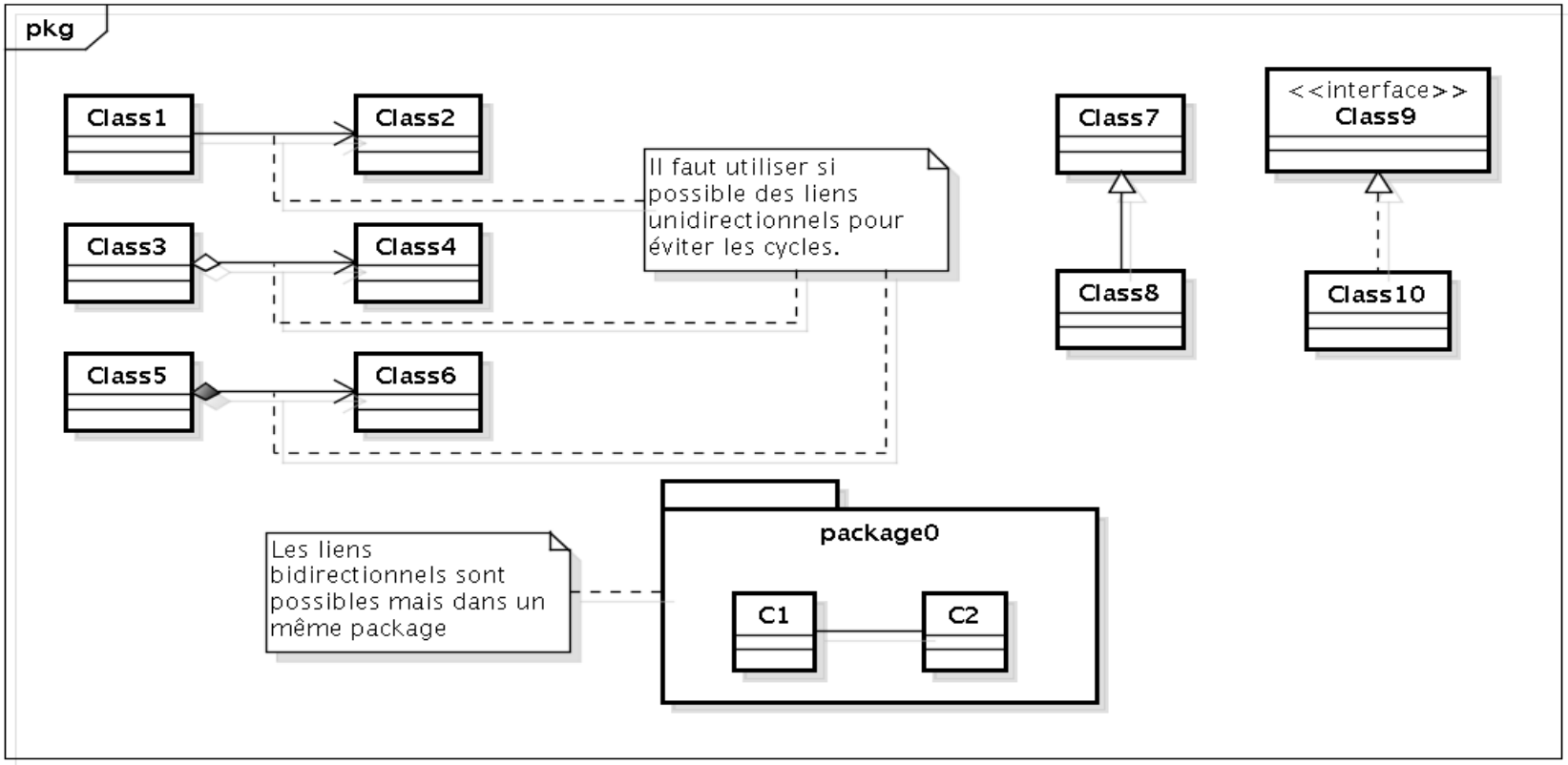
- Héritage :
  - Permet de définir du code commun
  - Permet de définir des (petites) hiérarchies de classification qui ne changent pas
  - Cas particulier : les hiérarchies d'exceptions (environ 70 classes Java héritent de *java.lang.Exception*)
- Classe abstraite :
  - Elle est utile que dans le cadre d'un héritage pour définir des comportements communs
  - Pour définir un représentant prototypique utilisez :
    - Soit une classe POJO à comportement modifiable (composition *versus* héritage)
    - Soit une classe interface : définition de ce que doit faire toute classe se réclamant de ce prototype





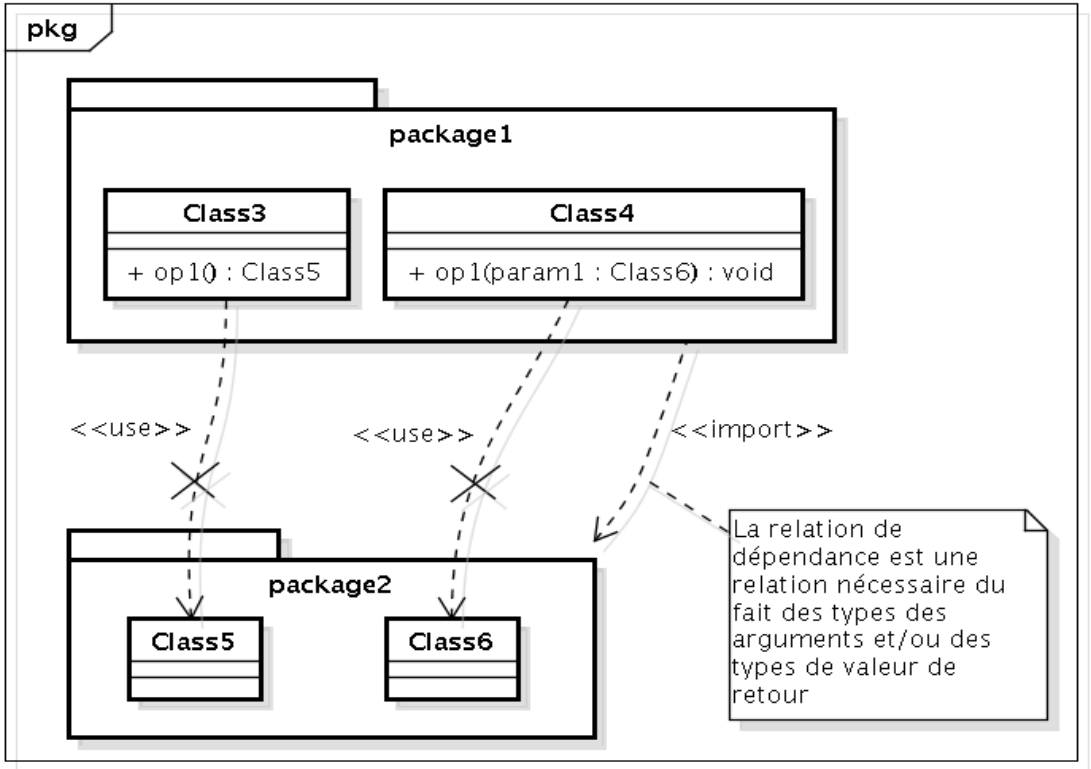
Les relations structurelles (3/5)

- Elle représente une relation sémantique durable existant entre instances durant l'exécution
- Elle est implémentée par des attributs de type objet (y compris des collections d'objets)
- Les relations structurelles peuvent être nombreuses dans un même package mais doivent limitées entre packages (voire inexistante entre couches d'une architecture)



Les relations temporaires (4/5)

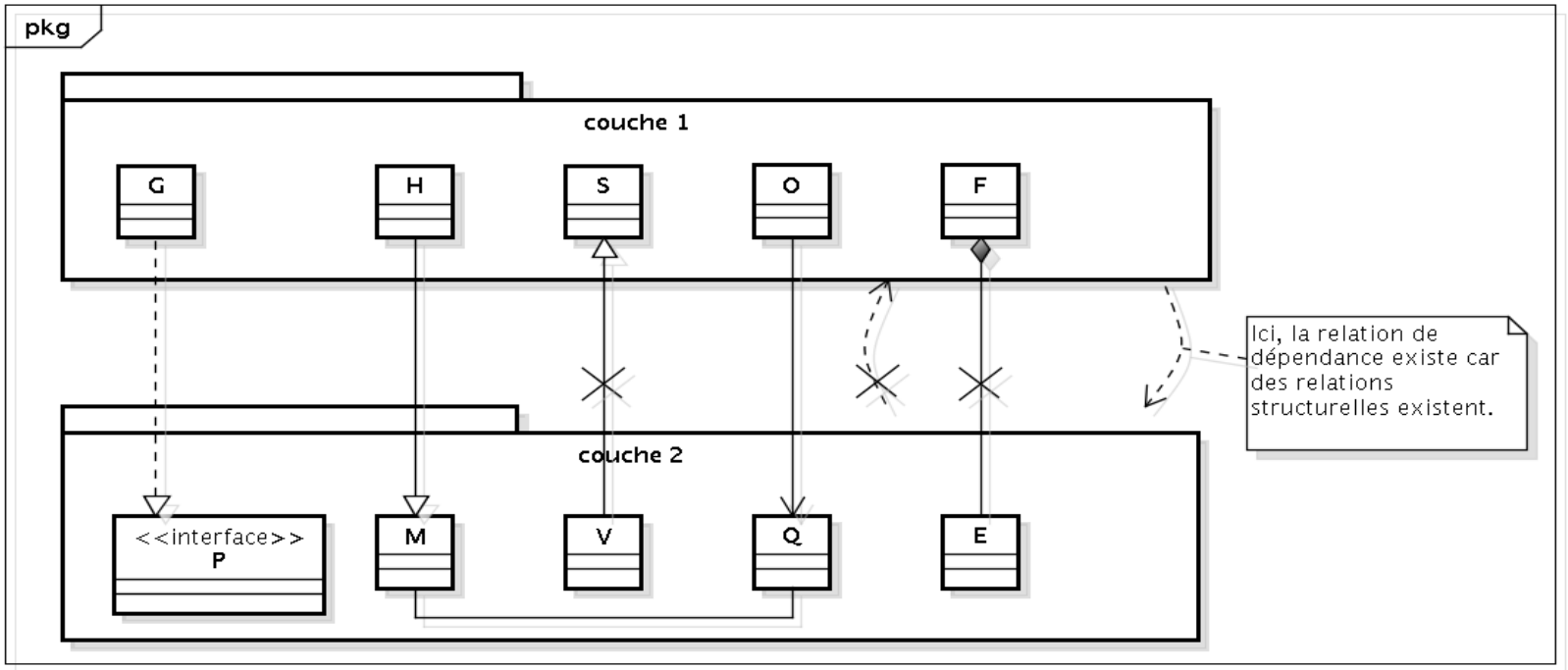
- Elle représente une relation de courte durée existant entre instances durant l'exécution
- Elle représente une relation n'ayant pas besoin d'être mémorisée par un attribut
  - Relation durant l'exécution d'une méthode (type des paramètres ou type de la valeur de retour) : non représentée en UML généralement
  - Les dépendances dues à cette relation : représentée en UML



Les relations entre packages (5/5)

- Les relations entre packages sont dues :
  - Soit du fait de relation structurelles
  - Soit du fait de relation temporaires (cf. diapositive précédente)
- Il faut éviter :
  - Les cycles de dépendances : utilisez des relations undirectionnelles (cf. cours n°2)
  - Les relations montantes dans une architecture par niveau : les dépendances sont descendantes
    - Les éléments de la couche présentation dépendent de l'application et de la technologie utilisée
    - Les éléments de la couche métier/domaine dépendent du domaine
    - Les éléments de la couche technique ne dépendent pas du domaine

    => les éléments les plus indépendants de l'application et génériques sont dans les couches inférieures
  - Le couplage entre couche : utilisez des façades (cf. cours n°2)



Plan du cours

- ✓ UML pour la conception logicielle
- ➢ Présentation de l'étude de cas
- Trois patterns GRASP
- Conception de l'architecture par niveau

Etude de cas (1/6)

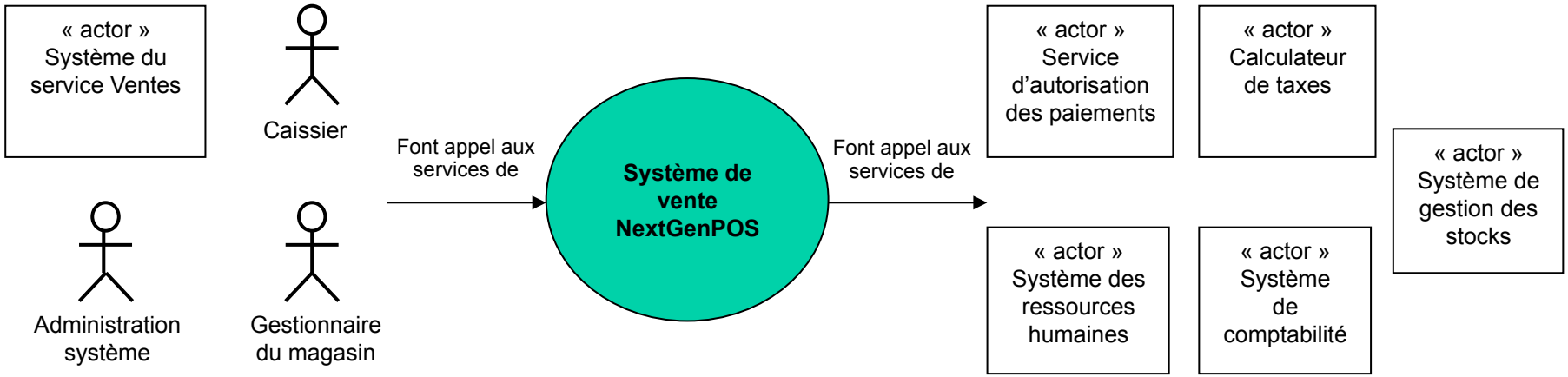
- Le système de vente NextGenPOS
  - Point de vente pour les détaillants
  - Caractéristiques :
    - Enregistrement des ventes
    - Traitement des paiements
    - Constitué d'un ordinateur avec un lecteur de code barres et d'un logiciel
    - Connecté à d'autres applications informatiques distribuées (calculateur de taxe, système de gestion des stocks, etc.)
    - Destiner à différents clients ayant des règles de gestion commerciale différentes
    - En cas de défaillance réseau, le système doit pouvoir continuer à saisir les ventes et gérer au moins les paiements en espèces
    - Gestion de divers terminaux et interfaces de clients (navigateur Web, interface Java Swing, écrans tactiles, tablette)
  - Architecture en couche

Etude de cas (2/6)

- Cas d'utilisation principal en abrégé :

> Un client arrive à la caisse avec les articles qu'il veut acheter. Le caissier utilise le système NextGenPOS (*Next Generation of Point Of Sale*) pour enregistrer chaque article acheté. Le système présente le détail des articles et le total. Le client entre les informations concernant le paiement, que le système valide et enregistre. Le système met à jour l'inventaire. Le client obtient un reçu du système et part avec les articles.

- Principaux acteurs :

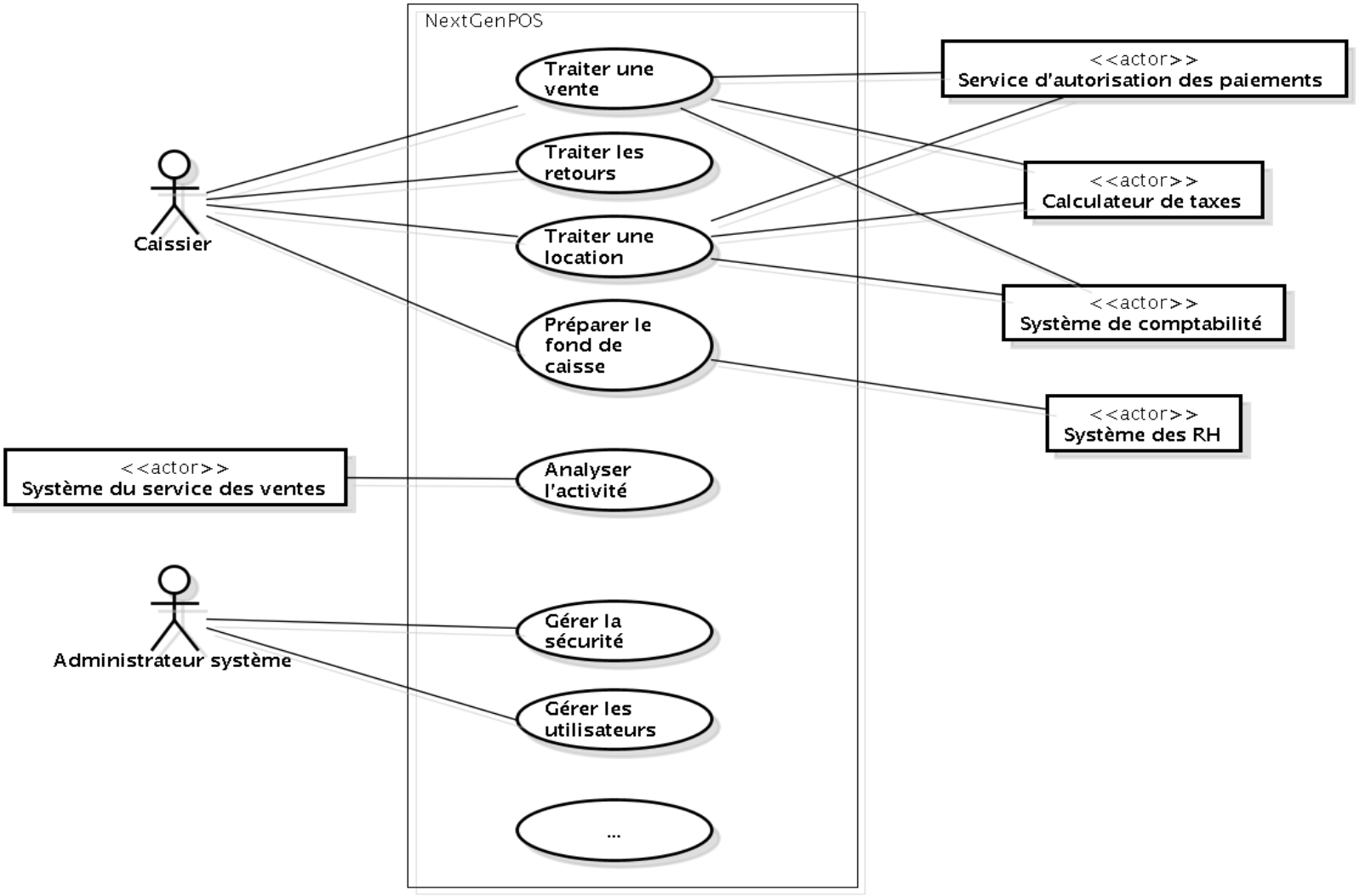


Etude de cas (3/6)

Acteur	Intentions métier
Caissier	Traiter les ventes Traiter les locations Traiter les retours Encaissement Décaissement
Gestionnaire du magasin	Démarrer le système Arrêter le système
Service d'autorisation des paiements	Autoriser les paiements par carte bancaire ou par chèque
Système de comptabilité	Obtenir l'ensemble des paiements effectués
Administrateur système	Gérer les utilisateurs Gérer la sécurité Gérer les données
Système du service des ventes	Analyser les données sur les ventes et les performances
Calculateur de taxes	Calculer les taxes selon le produit et le pays
Système des ressources humaines	Obtenir des informations pour contrôler l'activité des caissiers

Etude de cas (4/6)

- Extrait des cas d'utilisation :



Etude de cas (5/6)

- Extrait du diagramme de séquence du cas d'utilisation « Traiter une vente »

Scénario simple de paiement en espèces pour le cas Traiter une vente

1. Le client arrive à la caisse avec ses achats.
2. Le caissier entame une nouvelle vente.

3. Le caissier saisit le code des articles et la quantité.
4. Le système entregistre les références d'article et présente leur description, leur prix et le total courant.

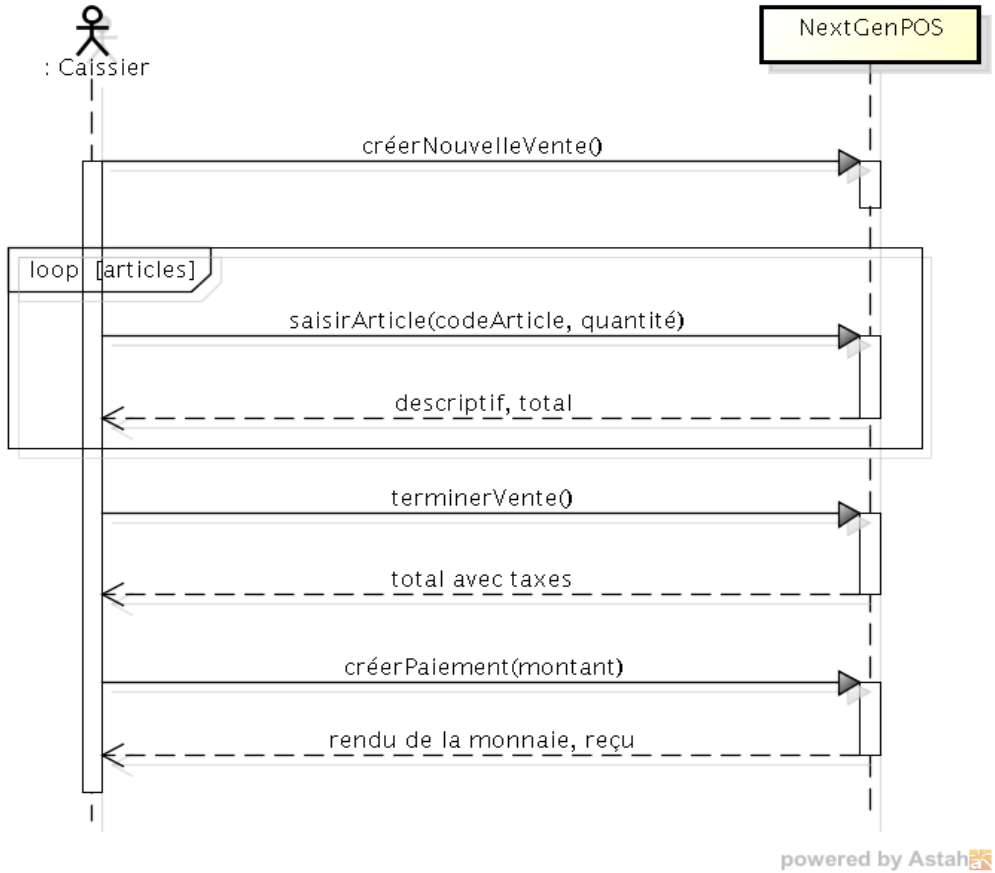
Le caissier répète l'opération pour tous les articles.

5. Le caissier finalise la vente.

6. Le système présente le total avec taxes.

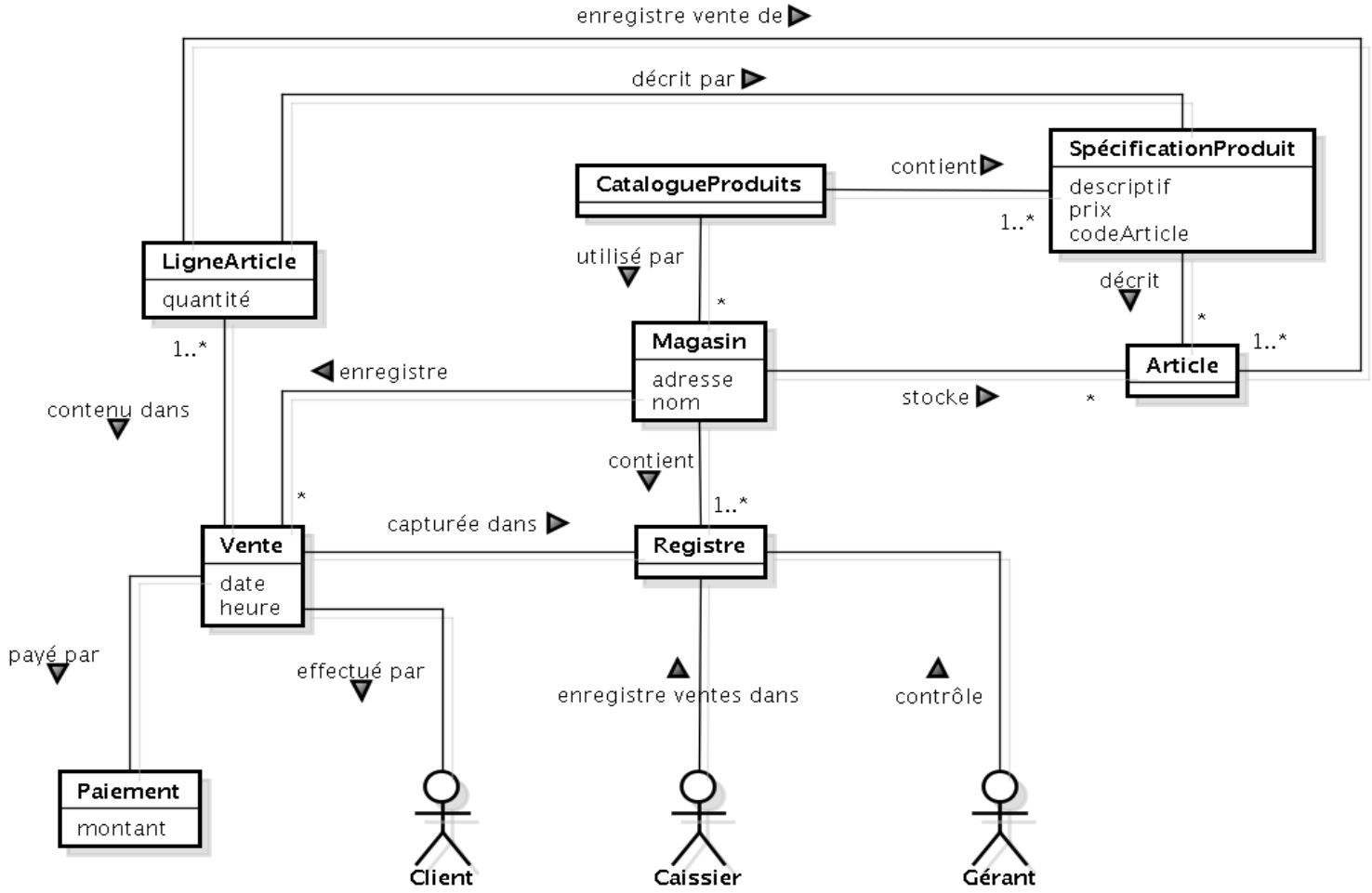
7. Le caissier indique le moyen de paiement et le montant.

8. Le caissier rend la monnaie et un reçu au client.



Etude de cas (6/6)

- Extrait du modèle du domaine :



Plan du cours

- ✓ UML pour la conception logicielle
- ✓ Présentation de l'étude de cas
- ➢ Trois patterns GRASP
- • Conception de l'architecture par niveau

Trois patterns GRASP (1/2)

- Les patterns GRASP (*General Responsibility Assignment Software Patterns*) :
  - Patterns de conception proposés par Craig Larman qui « formalise » des principes et des règles de conception objet pour l'assignement de responsabilité.

- ***Contrôleur*** : affecter la responsabilité de la réception ou du traitement d'un événement système entrant à une classe qui correspond à l'un des deux cas suivant :
  - La classe représente l'ensemble d'un système ou d'un sous-système (*contrôleur de façade*).
  - La classe représente un scénario dans lequel l'événement se produit (*contrôleur de cas d'utilisation* ou *contrôleur de session*).

Contrôleur :

- Un contrôleur délègue aux autres classes les tâches à effectuer. Il sert à coordonner ou à contrôler l'activité des objets du domaine. Il sert d'interface entre les couches représentation et domaine.
- Le choix entre les deux catégories de contrôleur doit respecter les principes de faible couplage et de forte cohésion : un contrôleur de façade ne doit pas avoir trop de responsabilités sinon utiliser des contrôleurs de cas d'utilisation (un par cas d'utilisation).
- La couche représentation ne doit pas avoir la responsabilité de gérer les événements système. Elle doit uniquement mettre en forme les événements systèmes qui peuvent provenir de plusieurs interactions avec l'utilisateur.

Trois patterns GRASP (2/2)

- ***Faible couplage*** : affecter une responsabilité de sorte que le couplage demeure faible.

Couplage :

- Mesure du degré de liberté d'un élément : un élément faiblement couplé ne dépend pas d'un trop grand nombre d'autres éléments.
- Le couplage doit être faible avec les éléments instables ou susceptibles d'évoluer.
- C'est un principe général qui s'applique à tous les niveaux : entre classes, entre sous-systèmes et entre systèmes.
- Le couplage est intrinsèque à l'approche objet qui est par définition l'animation d'objets connectés.

- ***Forte Cohésion*** : affecter une responsabilité de sorte que la cohésion demeure forte.

Cohésion :

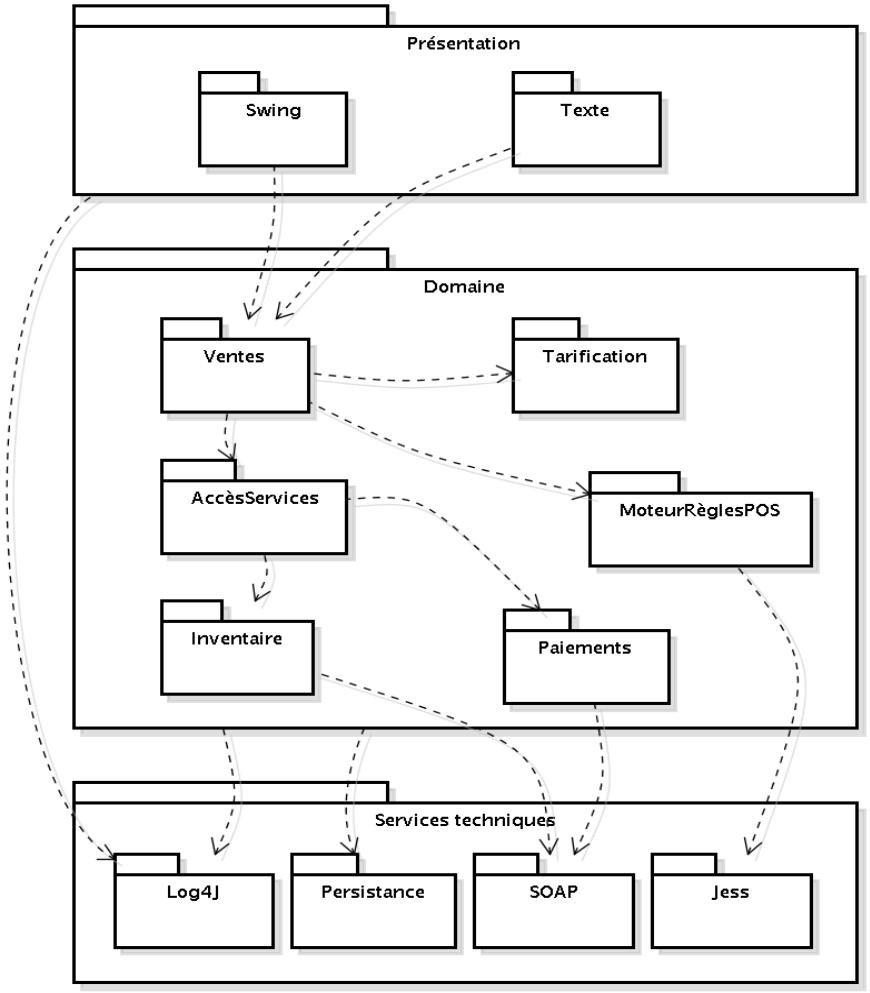
- Mesure du degré de spécialisation d'un élément : un élément fortement cohésif est un élément ayant des responsabilités étroitement liées et n'effectuant pas un travail « gigantesque ».
- C'est un principe général qui s'applique à tous les niveaux : aux classes, aux sous-systèmes et aux systèmes.
- La cohésion est intrinsèque à l'approche objet qui est par définition la délégation de traitements dans des objets connectés.

Plan du cours

- ✓ UML pour la conception logicielle
- ✓ Présentation de l'étude de cas
- ✓ Trois patterns GRASP
- ➢ Conception de l'architecture par niveau

Conception de l'architecture (1/8)

- Architecture logique = organisation du système en couches, packages, classes et interfaces.
- Un diagramme de package permet de représenter les couplages :

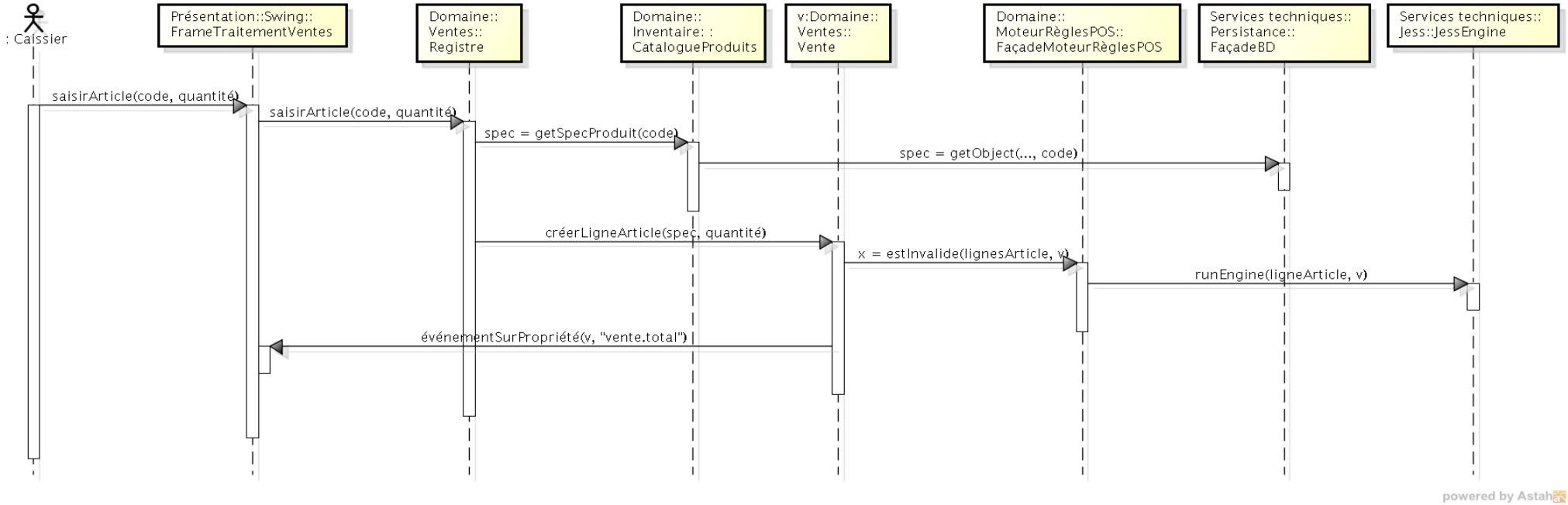


Aucun cycle ne doit exister à ce niveau de description !

Conception de l'architecture (2/8)

- Des diagrammes d'interaction illustrent les scénarios significatifs du point de vue architectural :

Exemple d'une interaction et son traitement à travers différentes couches



Conception de l'architecture (3/8)

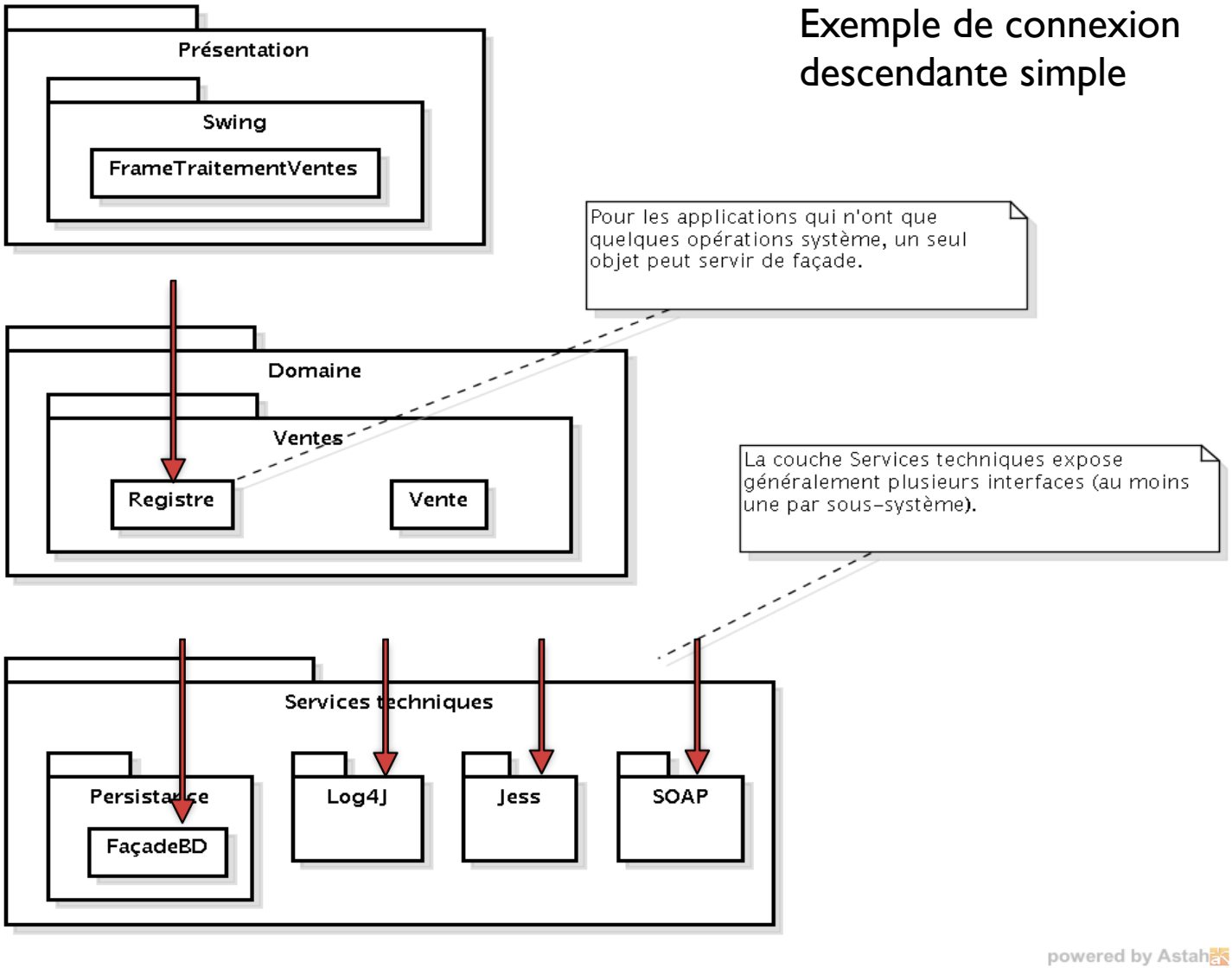
- Conception des liens entre la couche présentation et la couche domaine :
  - La conception doit être telle que le couplage entre ces deux couches soit le plus faible possible pour minimiser leur dépendance.
  - La couche présentation ne doit pas avoir de responsabilité logique. Elle doit faire suivre toute tâche orienté domaine à la couche domaine.
  - Connexion descendante : pattern GoF Façade + pattern GRASP Contrôleur
  - Connexion ascendante : pattern GoF Observateur

Conception de l'architecture (4/8)

- Connexion descendante des couches : pattern GoF Façade + pattern GRASP Contrôleur
  - Une Façade public sert de point d'accès unique à un sous-système.
  - Les clients ne voient que la Façade qui elle délègue le travail réel aux éléments masqués du sous-système.
  - Pour les applications simples, une seule Façade peut suffire dans la couche domaine.
  - Par contre, la couche services techniques possèdent autant de Façade que de services techniques.
  - Pour des applications complexes, plusieurs Façade peuvent exister dans la couche domaine. Dans ce cas, il y a une Façade par cas d'utilisation qui gère les opérations systèmes du cas qu'elle représente.

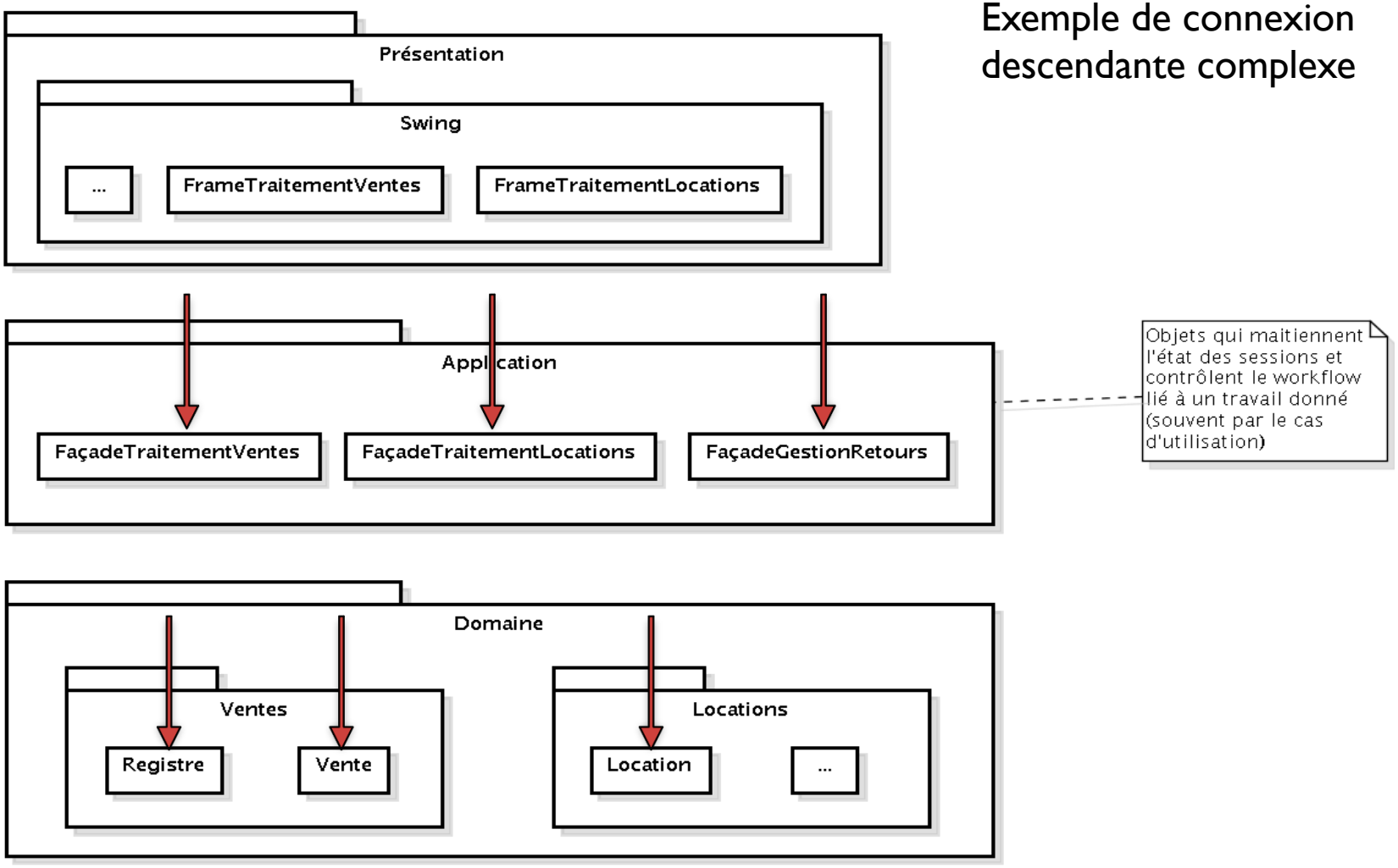
Conception de l'architecture (5/8)

Exemple de connexion descendante simple



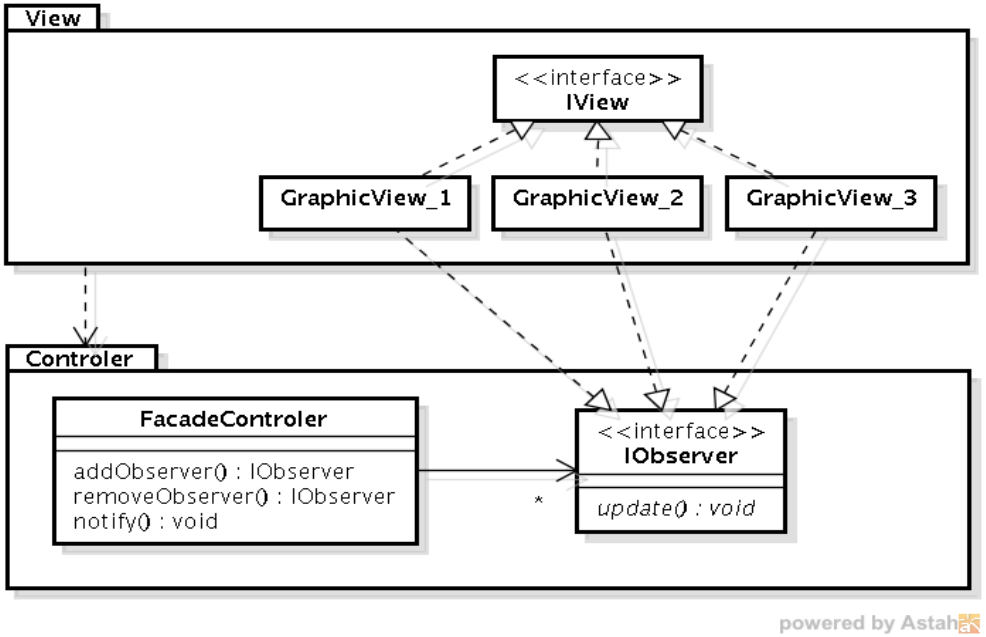
Conception de l'architecture (6/8)

Exemple de connexion descendante complexe



Conception de l'architecture (7/8)

- Connexion ascendante des couches : pattern Observateur
  - Les couches supérieures implémentent des interfaces *IObserver* qui sont les auditeurs.
  - Les couches inférieures sont des diffuseurs. Ils connaissent les auditeurs et les notifient lorsque leurs états internes changent.
  - Il s'agit d'un couplage lâche car les couches inférieures ne sont pas en contact avec les objets concrets des couches supérieures mais ils sont en contacts avec des objets interfaces.



Conception de l'architecture (8/8)

- Couplage et cohésion des packages :
  - Séparer les classes interfaces des classes d'implémentations
  - Utiliser des Fabriques d'objets
  - Séparer les éléments stables des éléments instables
  - Interdire les cycles entre packages