

Partie 6

Diagramme de Packages

D. de Packages

- « gros » programmes
 - Décomposer en sous-systèmes
 - Décomposer framework
 - Framework
 - Langage
 - Applicatif
 - Technique (math)
 - Pour des raisons de :
 - Couplage
 - Cohérence
 - Réutilisabilité

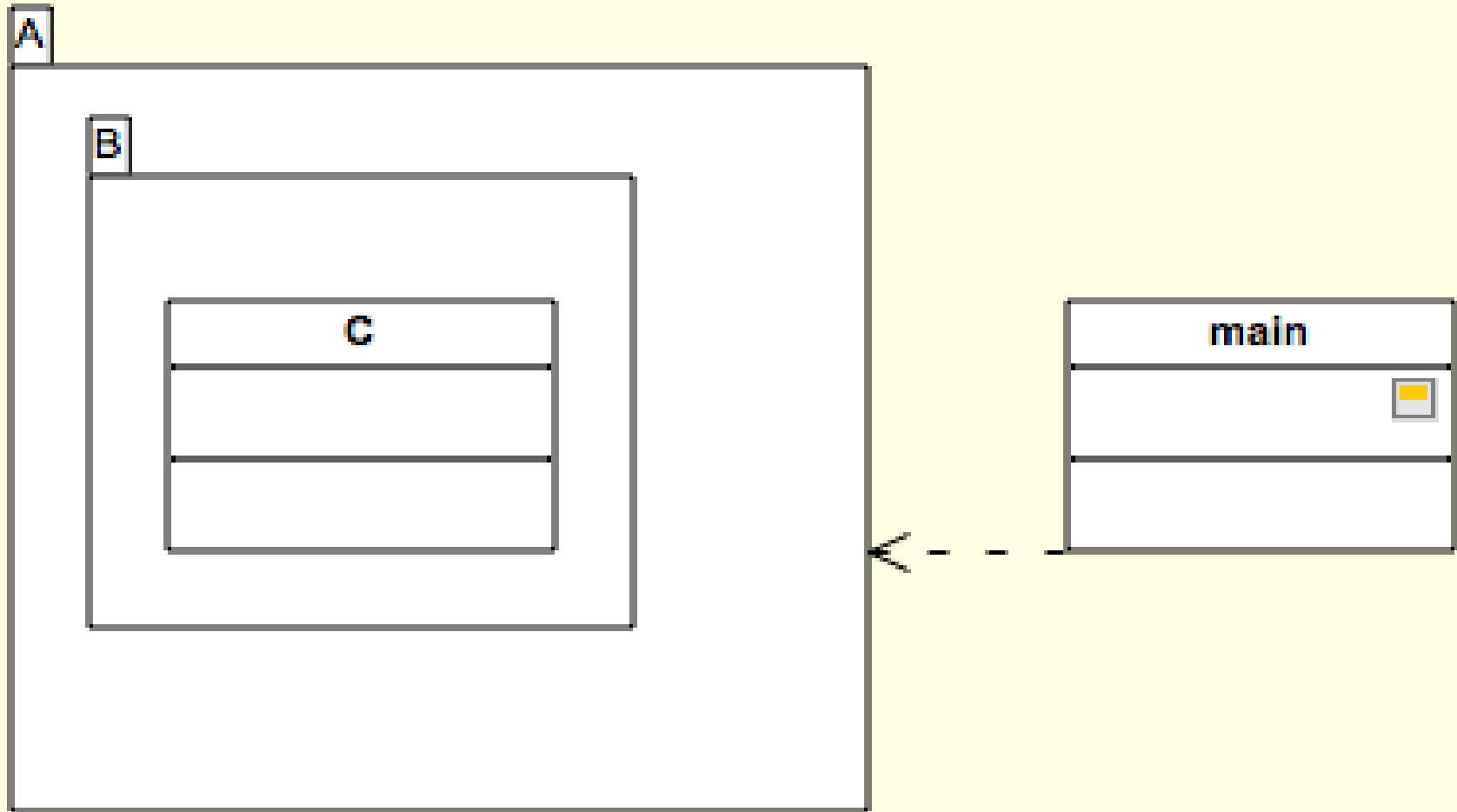
D. de Packages

- Package : regroupement d'objets selon un critère arbitraire
 - Encapsulation de la visibilité
 - Forte cohérence (selon le critère de regroupement) entre les objets
 - (Recherche de) couplage faible entre les packages
 - Un package peut contenir d'autres packages
 - Pas d'héritage des classes contenues

D. de Packages

- Attention !
 - Diag. De Package n'existe pas au sens de la norme UML
 - Diag. Package est un Diag. De classe ne représentant que les packages et leurs dépendances

D. de Packages



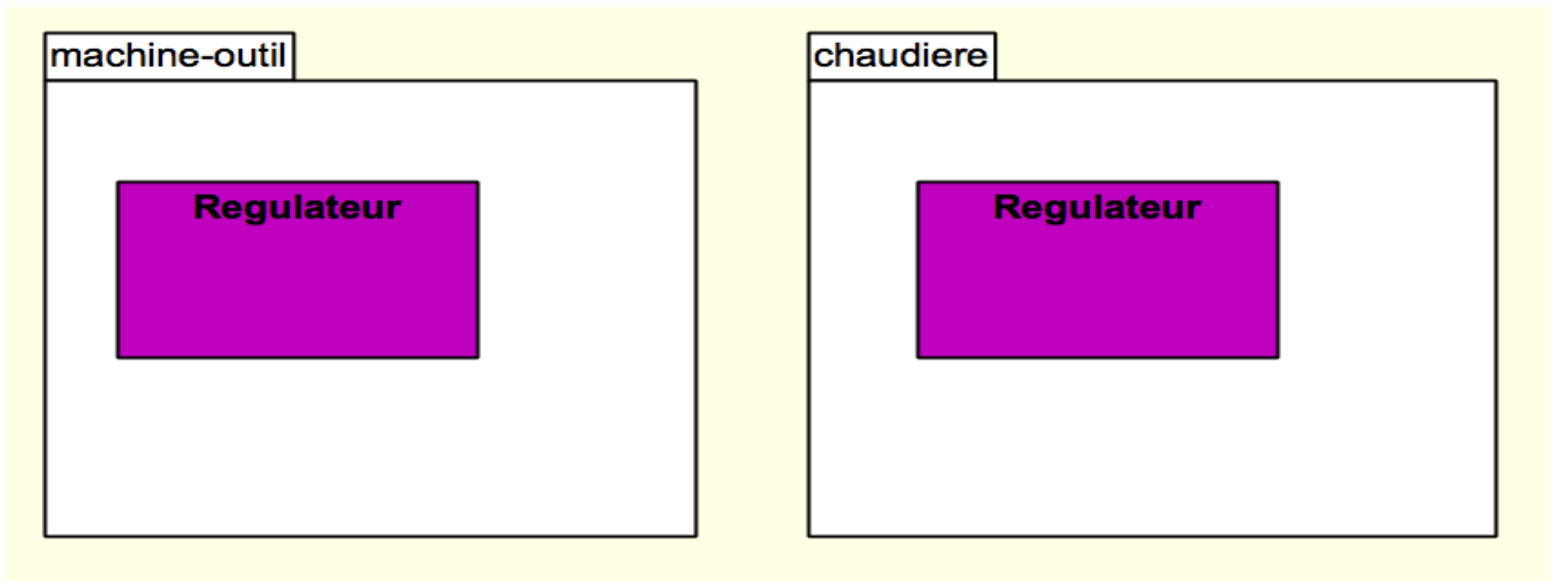
D. de Packages

```
import A.B.*;  
public class main {  
    public static void main(String[] args)  
    {  
        C objC = new C();  
    }  
}
```

D. de Packages

Espaces de nommage

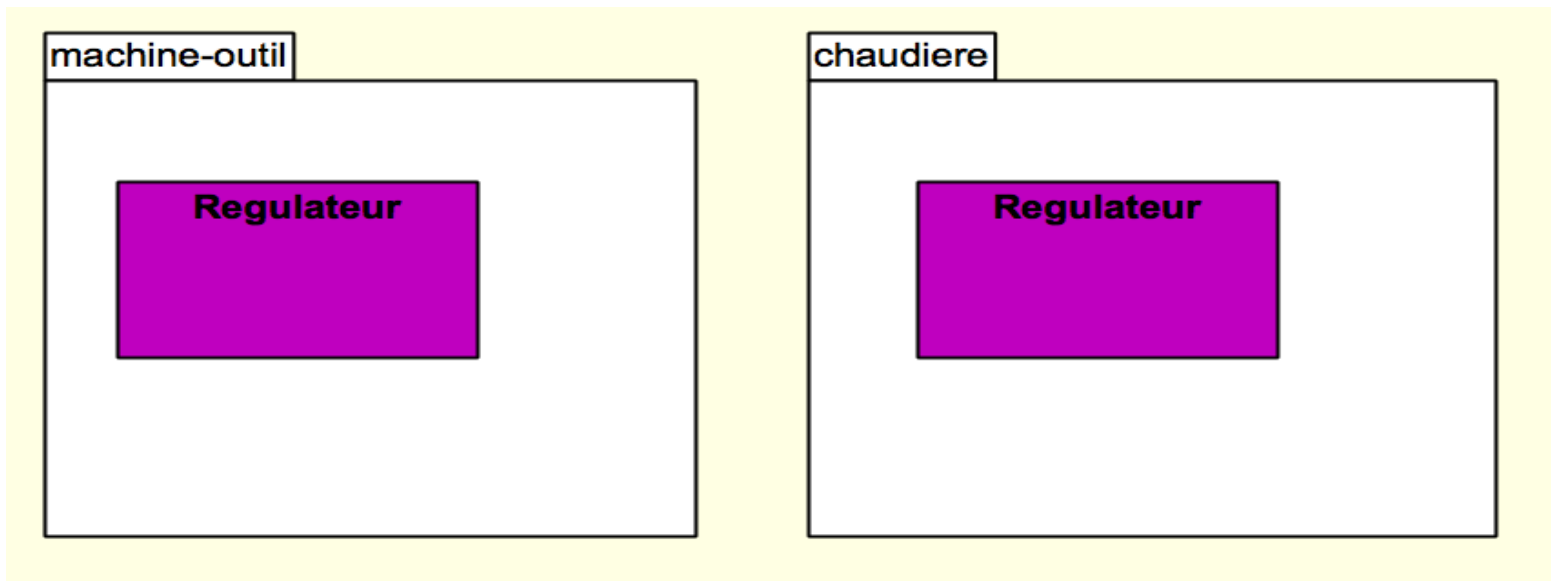
- Appartenir à un package = appartenir à un espace de nommage



D. de Packages

Espaces de nommage

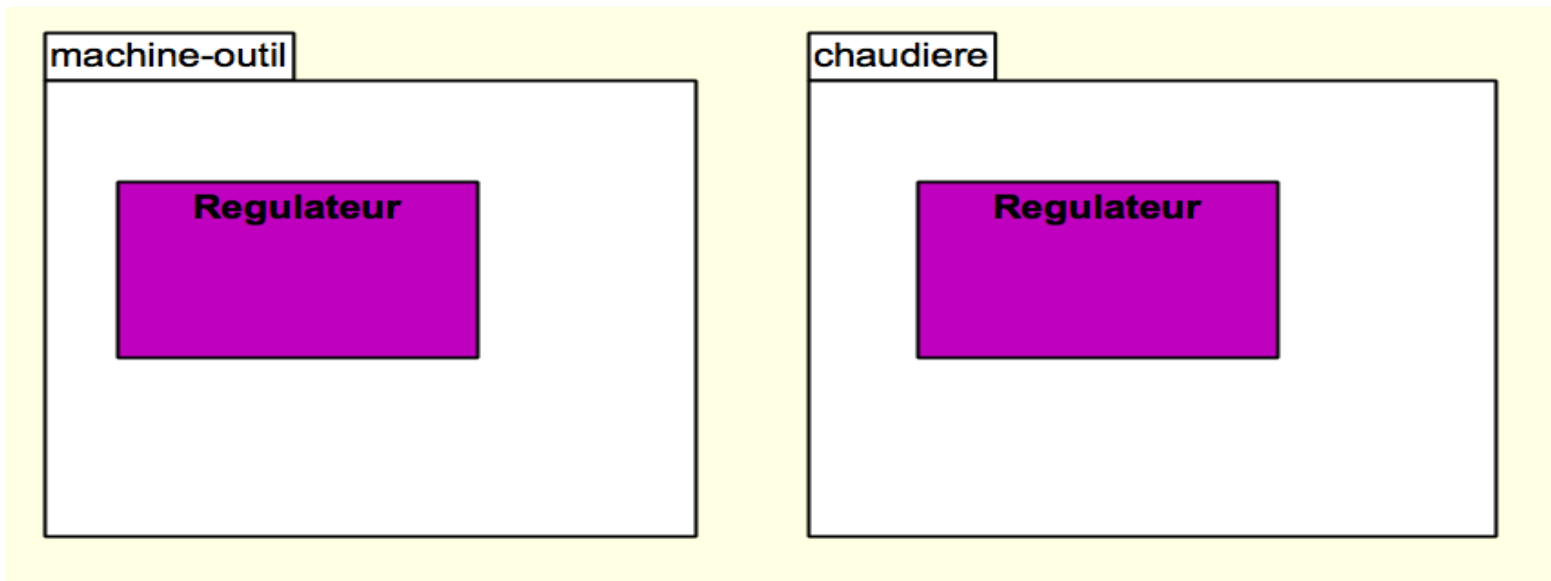
- 2 Objets avec le même nom : permit par les espaces de nommage



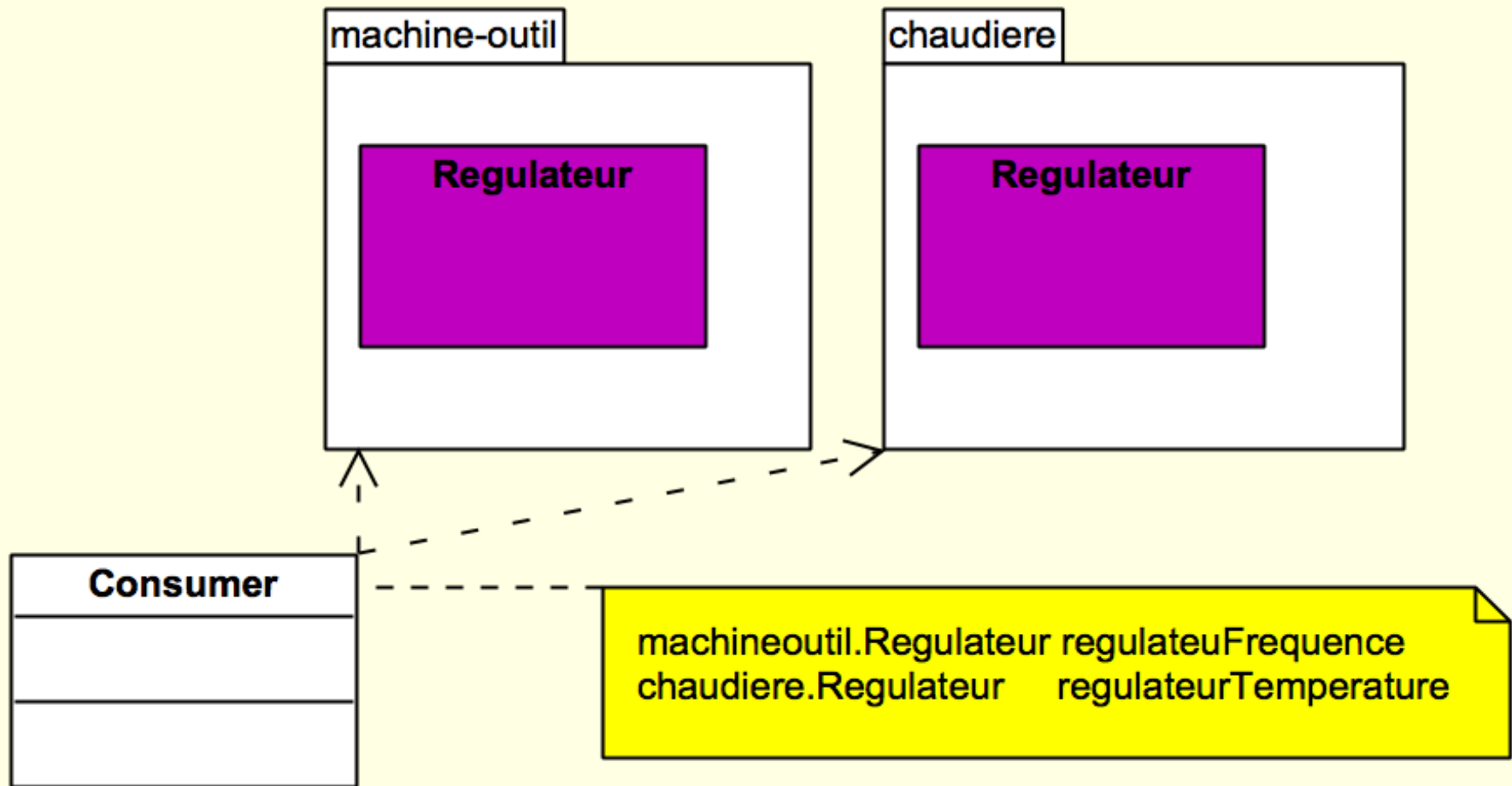
D. de Packages

Espaces de nommage

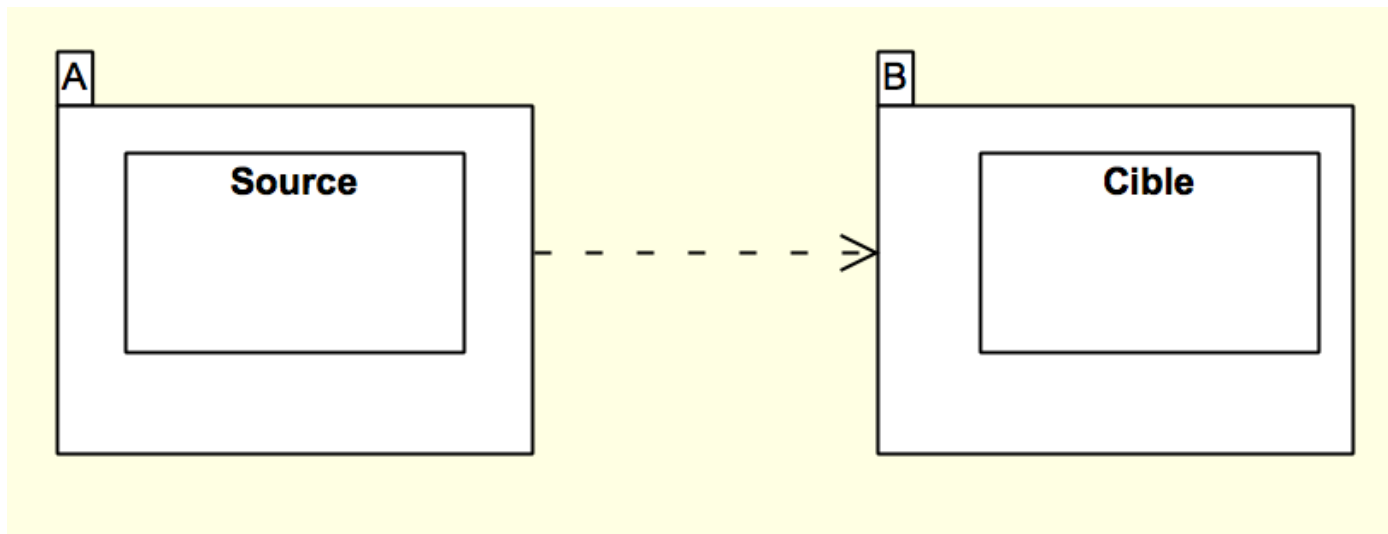
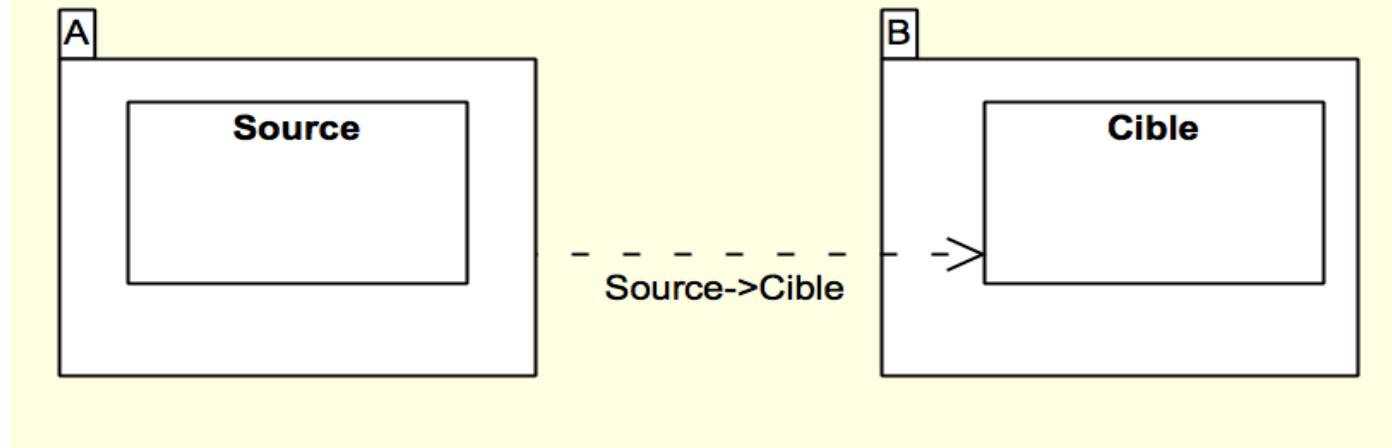
- Exemple en Java



D. de Packages



D. de Packages



D. de Packages

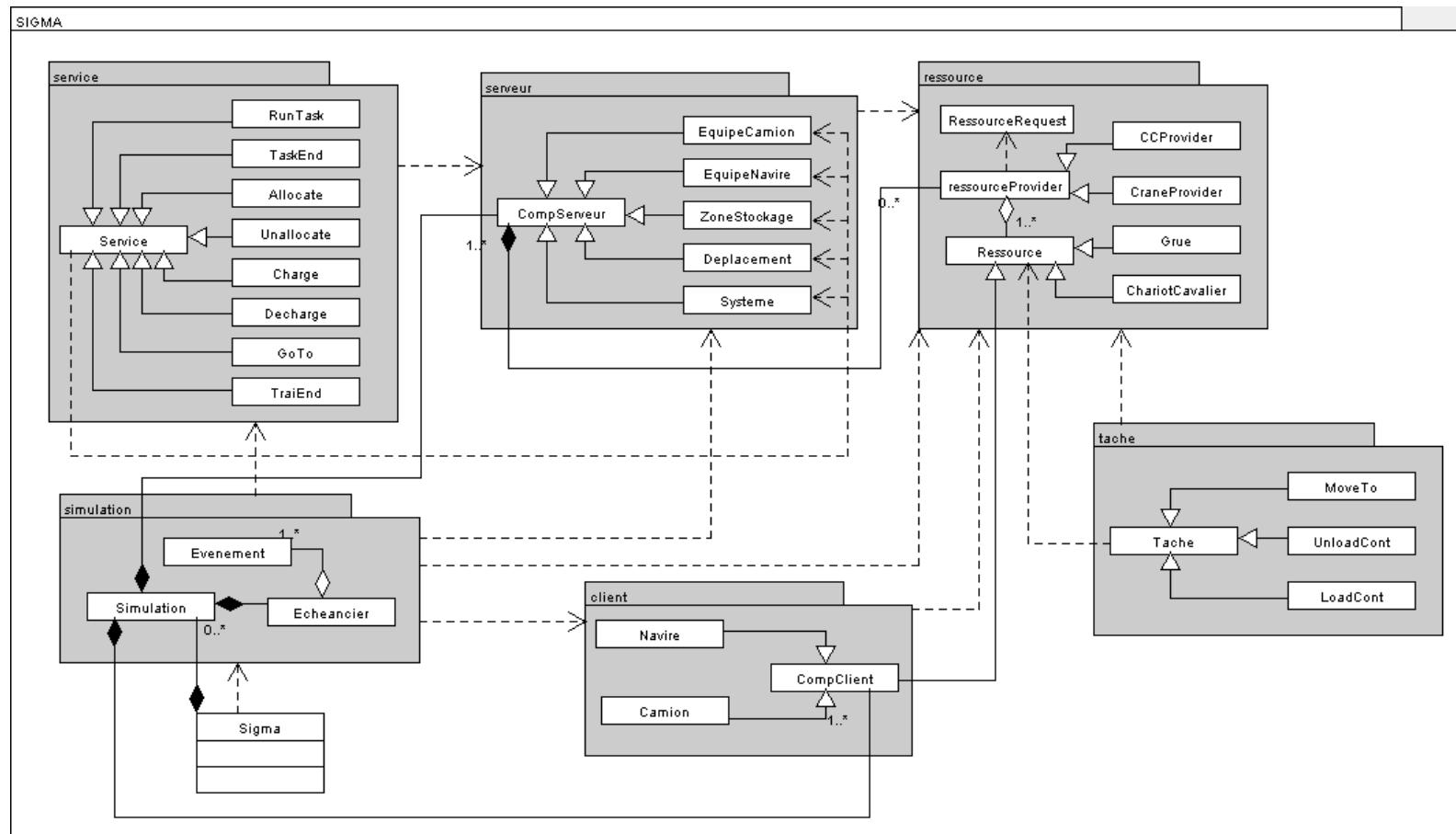
Dépendances entre packages

- Il existe une dépendance en 2 packages dès qu'il existe une dépendance entre, au moins, une classe de chacun des 2 packages
- Il n'existe qu'un type de dépendance

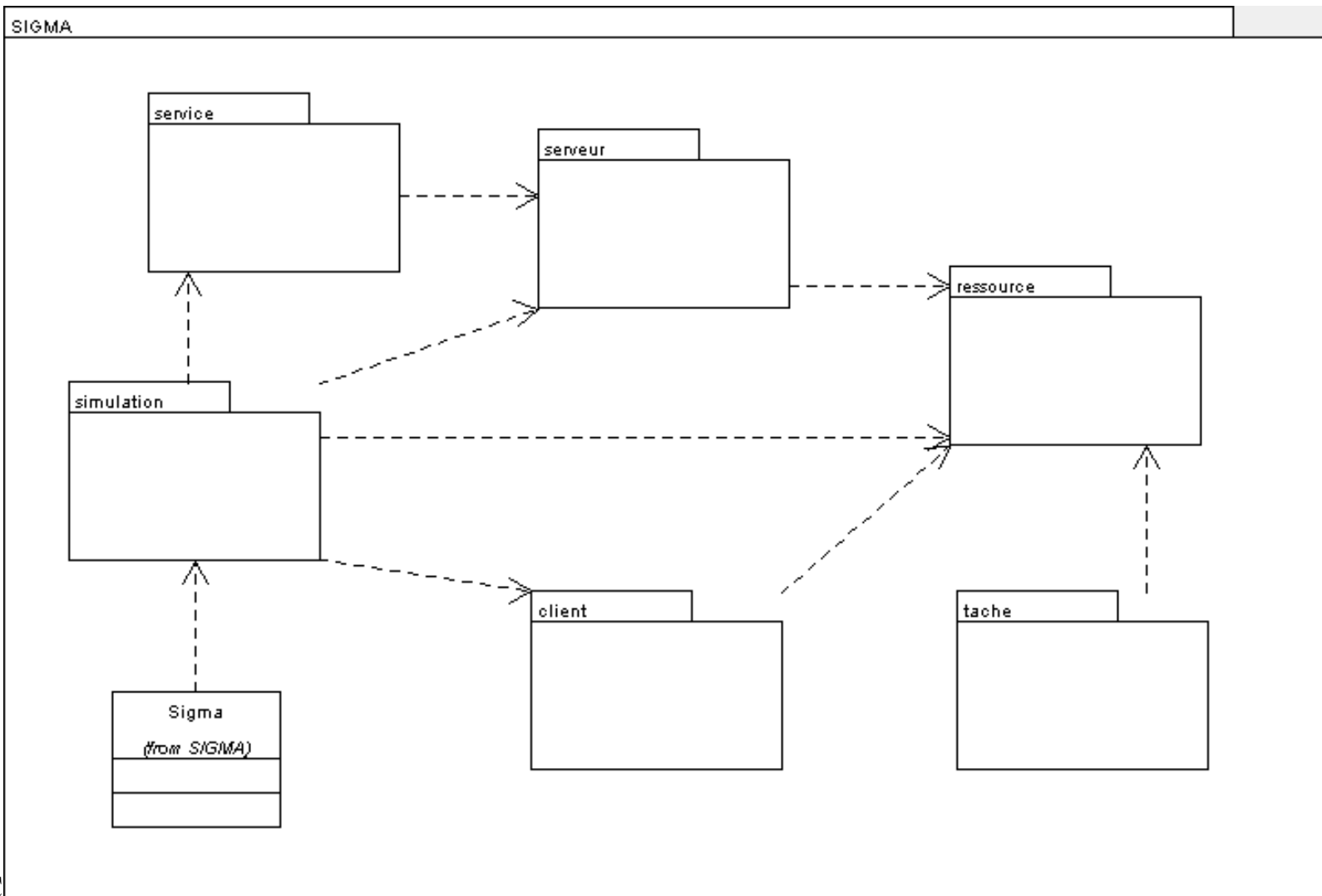
D. de Packages

- Pourquoi cette « simplification fausse » ?

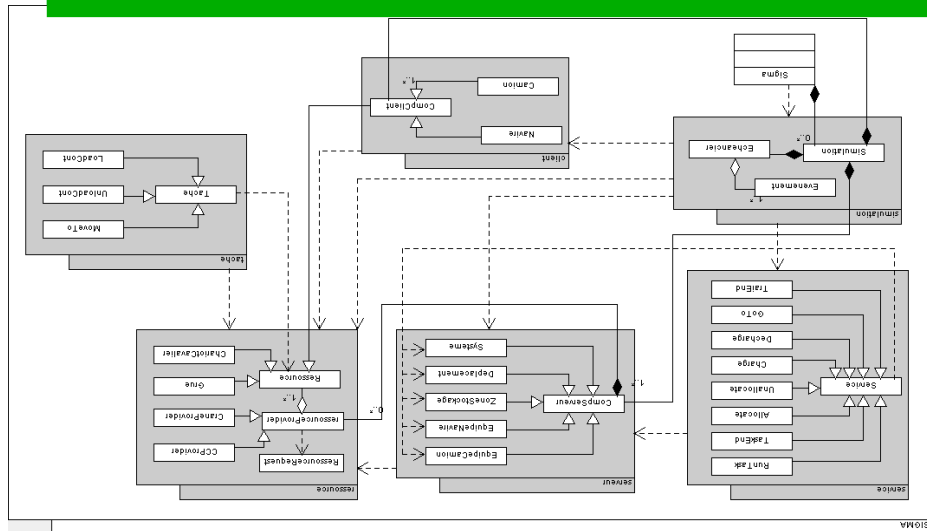
D. de Packages



D. de Packages

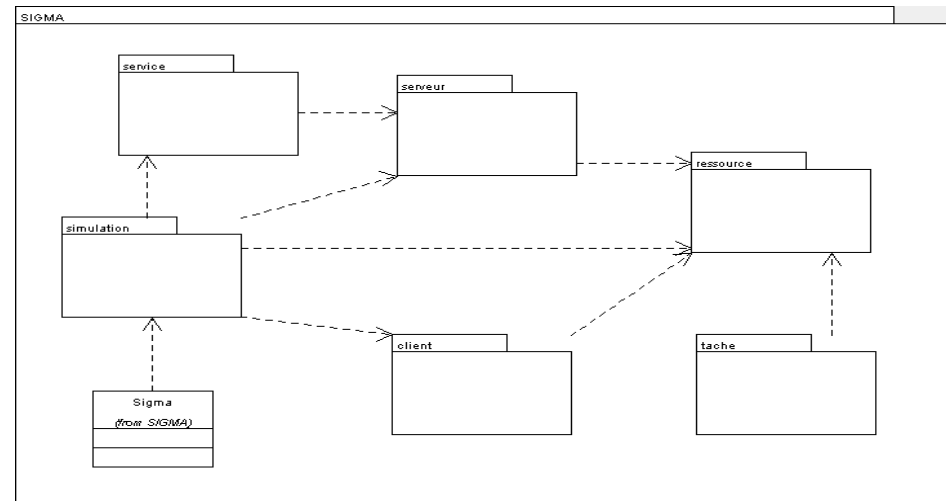


D. de Packages



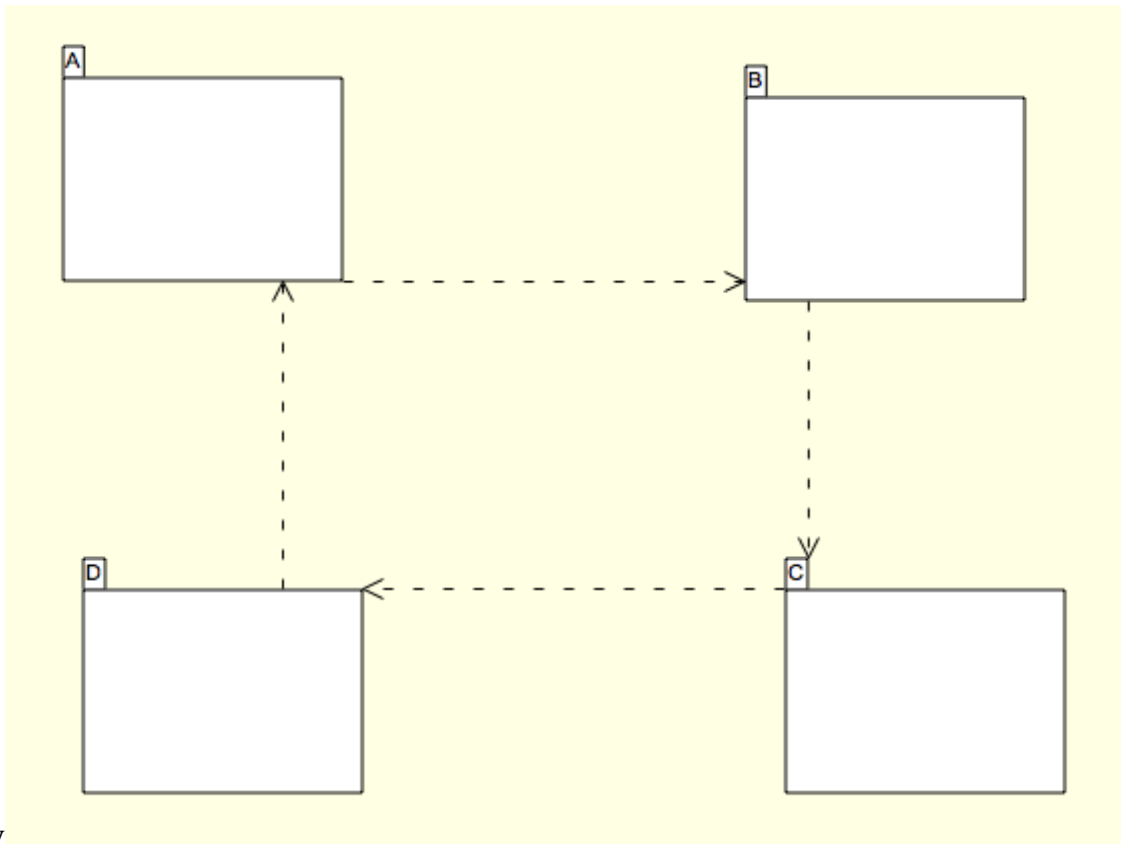
Représentation des dépendances dans diag. de package permet une réflexion sur les dépendances au niveau conceptuel

Echange avec client, chef projet, architecte

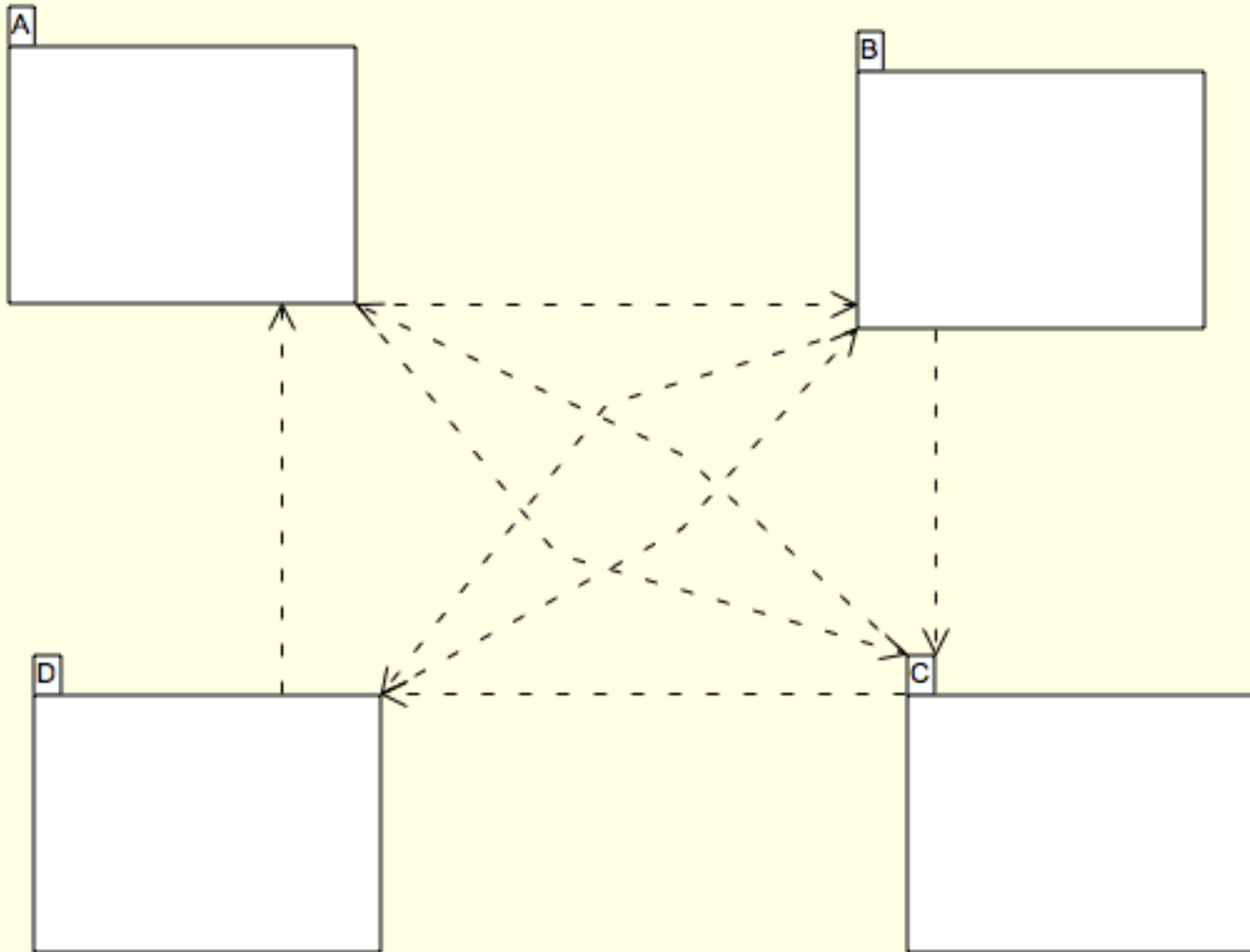


D. de Packages

- Eviter (le plus possible) les couplages multiples et les références circulaires



D. de Packages



D. de Packages

- Eviter (le plus possible) les couplages multiples et les références circulaires
 - Couplages forts
 - Fragilité du logiciel
 - Rigidité du logiciel
 - Réutilisation complexe des composants (un à un)

D. de Packages

- La dépendance entre package n'est pas transitive
 - Limite la volumétrie de modification en cas d'évolution d'un package
 - Limite la propagation des bugs
 - Architecture en couche

D. de Packages

- Comment éviter ce type de problème?
 - Prévenir
 - Etablir, en permanence, des diagrammes de package pour éviter que ce phénomène ne se développe lors de la phase d'étude
 - Pas toujours évident
 - Version 1.1, 1.2, 1.3... du soft
 - Redéfinition des bases et du but
 - Evolution du soft par couche
 - Refactoring
 - Changement du décideur, de l'architecte...
 - Problème humain "complexe"

D. de Packages

- Comment éviter ce type de problème ?
 - Guérir
 - « Eclater » un package en plusieurs packages plus petits
 - « Cacher » les packages derrière un package facade

D. de Packages

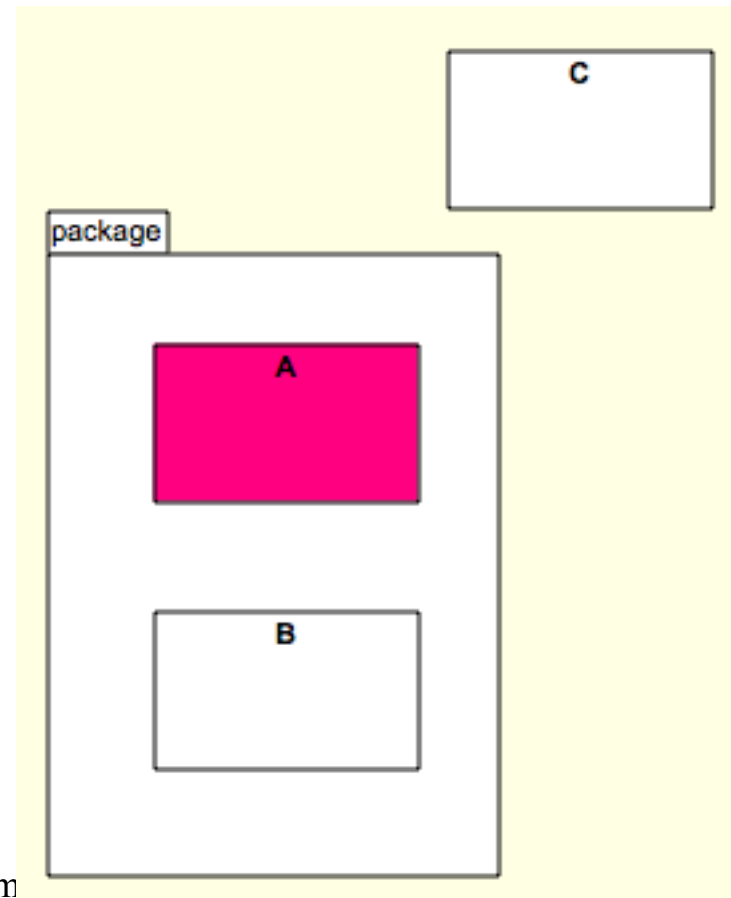
Les stéréotypes des packages

- <<facade>> Vue façade d'un ensemble de packages
- <<framework>> Vue applicative
- <<toplevel>> Package "root" (de plus haut niveau)

D. de Packages

Visibilité des classes dans les packages

- UML
 - Public
 - Private
 - Package
 - Protected
- Équivalent en Java ?
 - Public
 - Default



D. de Packages

Les packages facade et classes adaptatrices

- Cacher des éléments d'un package
- Recombiner les éléments d'un ou plusieurs packages selon une autre vision (ex : multi OS)
- Travailler en QUOI/COMMENT

Cre → Moins il y a de parties publiques, moins il y a de problèmes de couplage

D. de Packages

Les packages facade et classes adaptatrices

- Recombiner les éléments d'un ou plusieurs packages selon une autre vision (ex : multi OS)
 - Ex : intégrer les API de 2 OS pour un dev. multi-OS

D. de Packages

Les packages facade et classes adaptatrices

- Cacher des éléments d'un package
 - Exemple d'un framework réutilisé pour une autre tâche

D. de Packages

Les packages facade et classes adaptatrices

- Travailler en QUOI/COMMENT
 - Travailler en équipe
 - Répartition architecte/dev

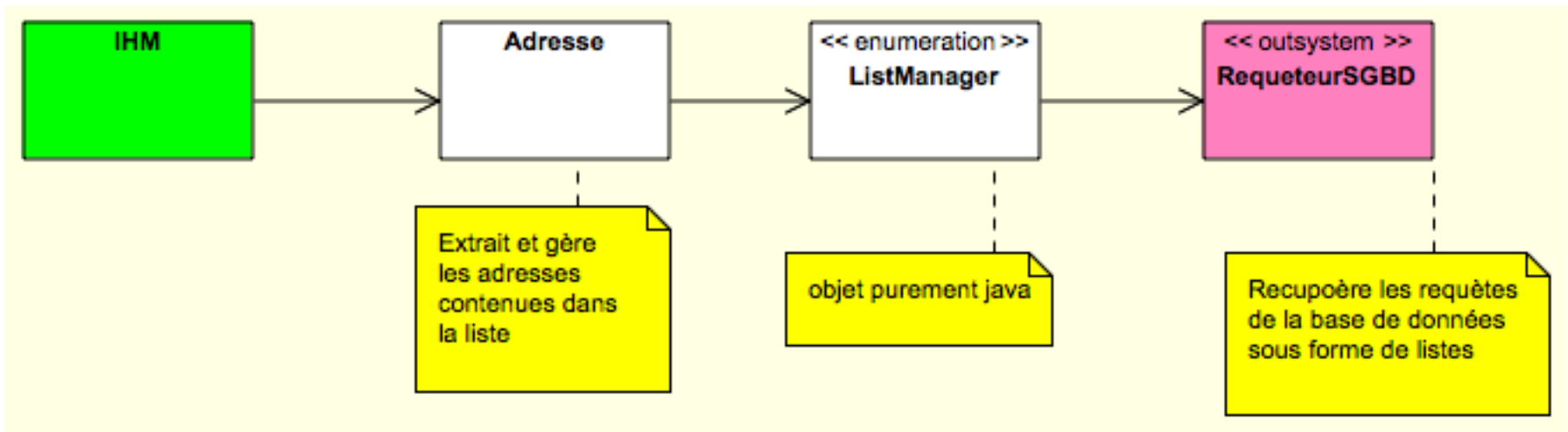
D. de Packages

Les packages

- Définir les briques du système
 - Dev. Modulaire
 - Ex : réalisez un logiciel qui extrait les adresses d'une base de données pour les afficher dans une interface graphique
 - Quelles sont les briques ?
 - Pensez
 - Réutilisabilité des briques
 - Évolution du système

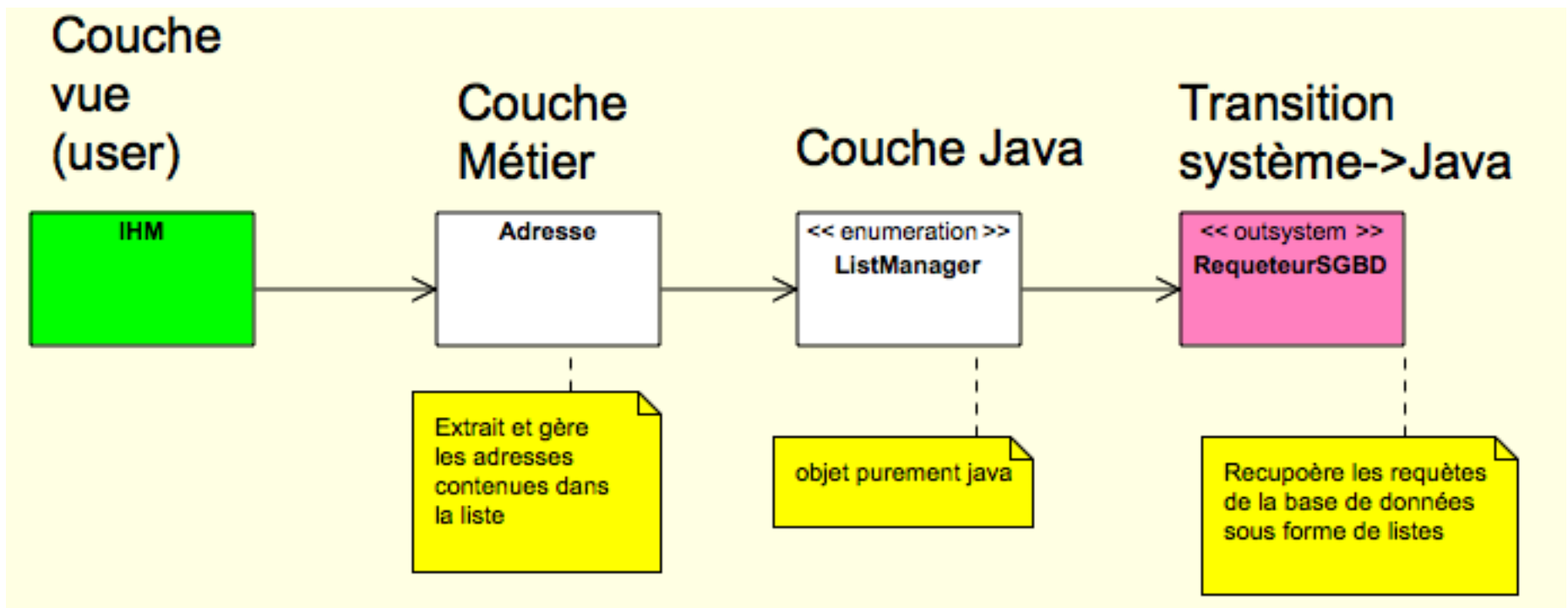
D. de Packages

Les packages



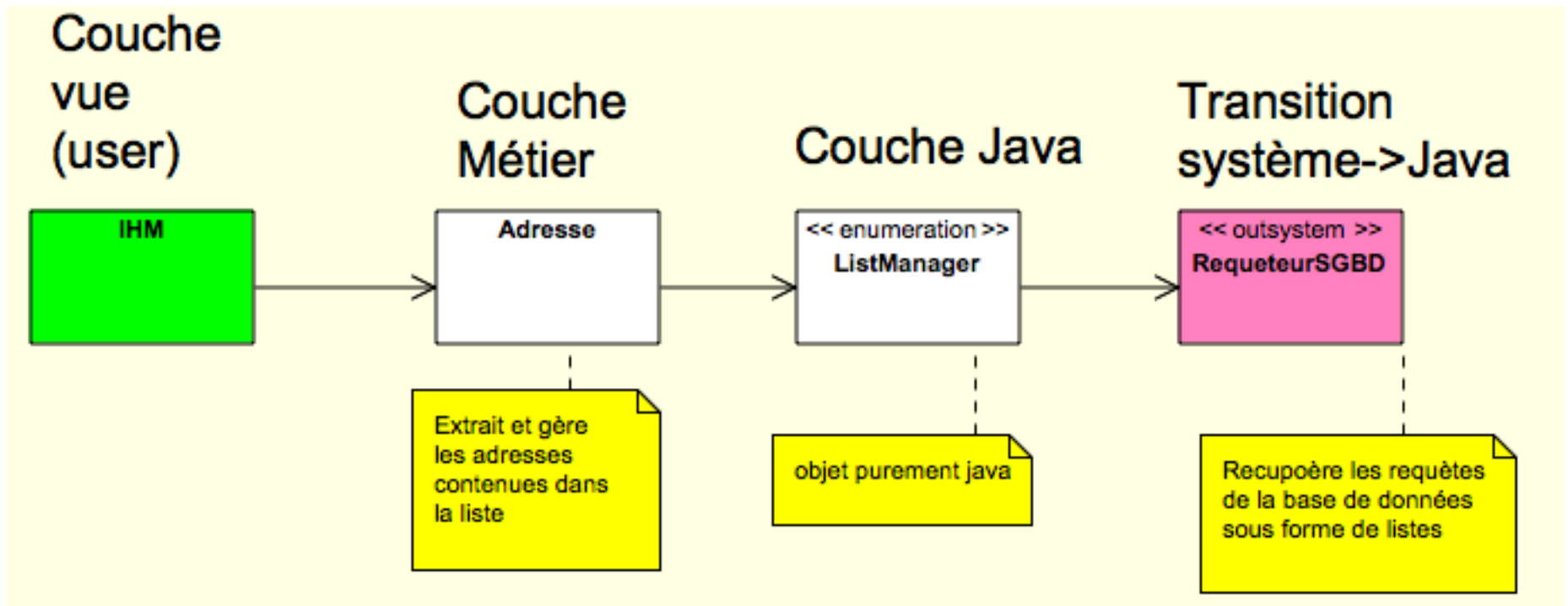
Pourquoi ce découpage (gros projet) ?

D. de Packages



Point de vue projet : répond à des couches différentes (connaissance + tech ou métier)

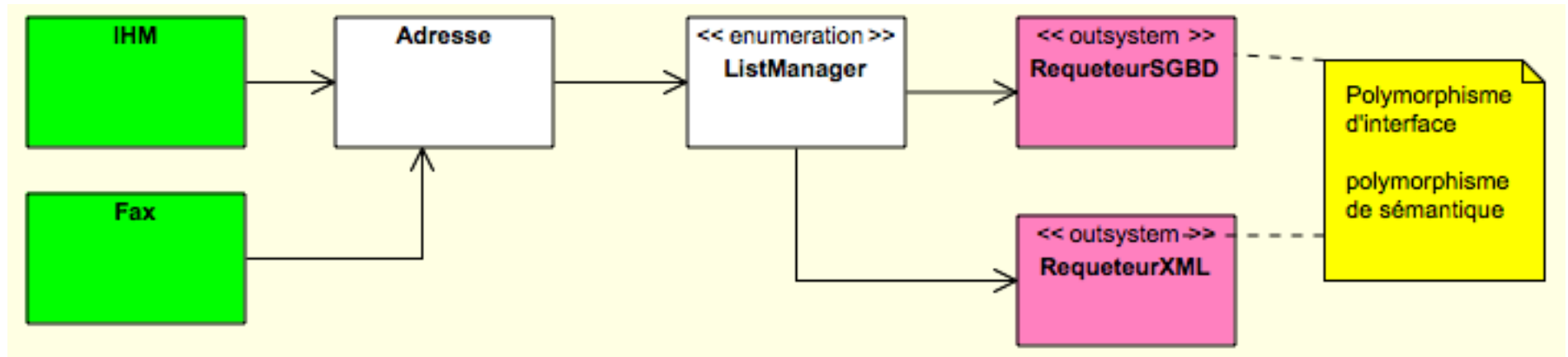
D. de Packages



Faire évoluer vers un système pouvant gérer les entrées en XML et les sorties sur

D. de Packages

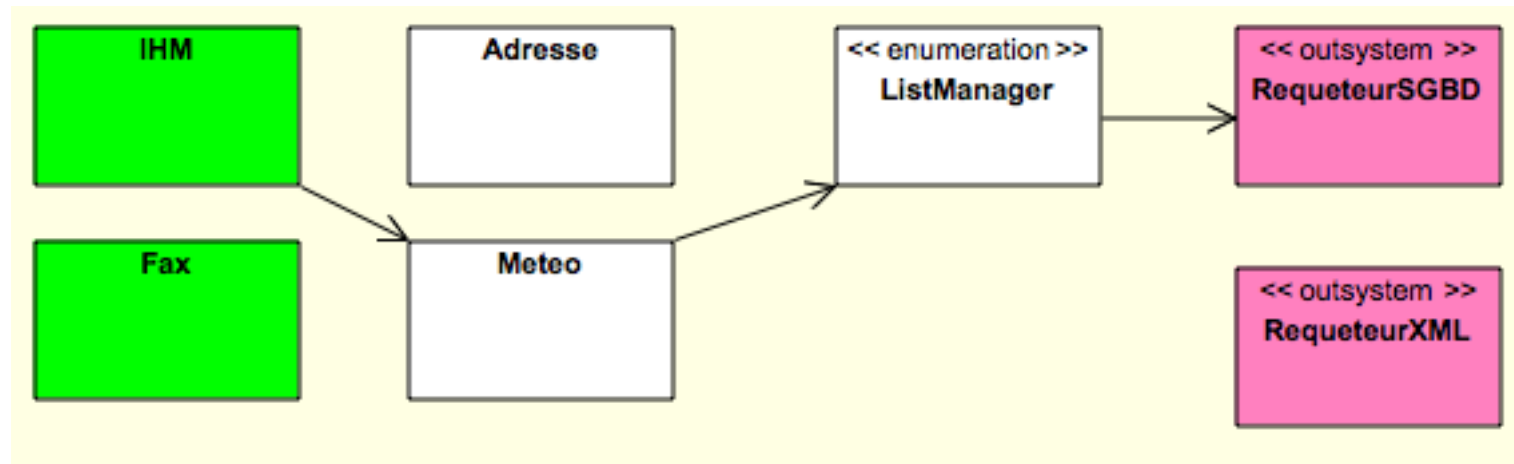
Les packages



Et si on ajoutait une gestion météo au système (température, hygrométrie...) ?

D. de Packages

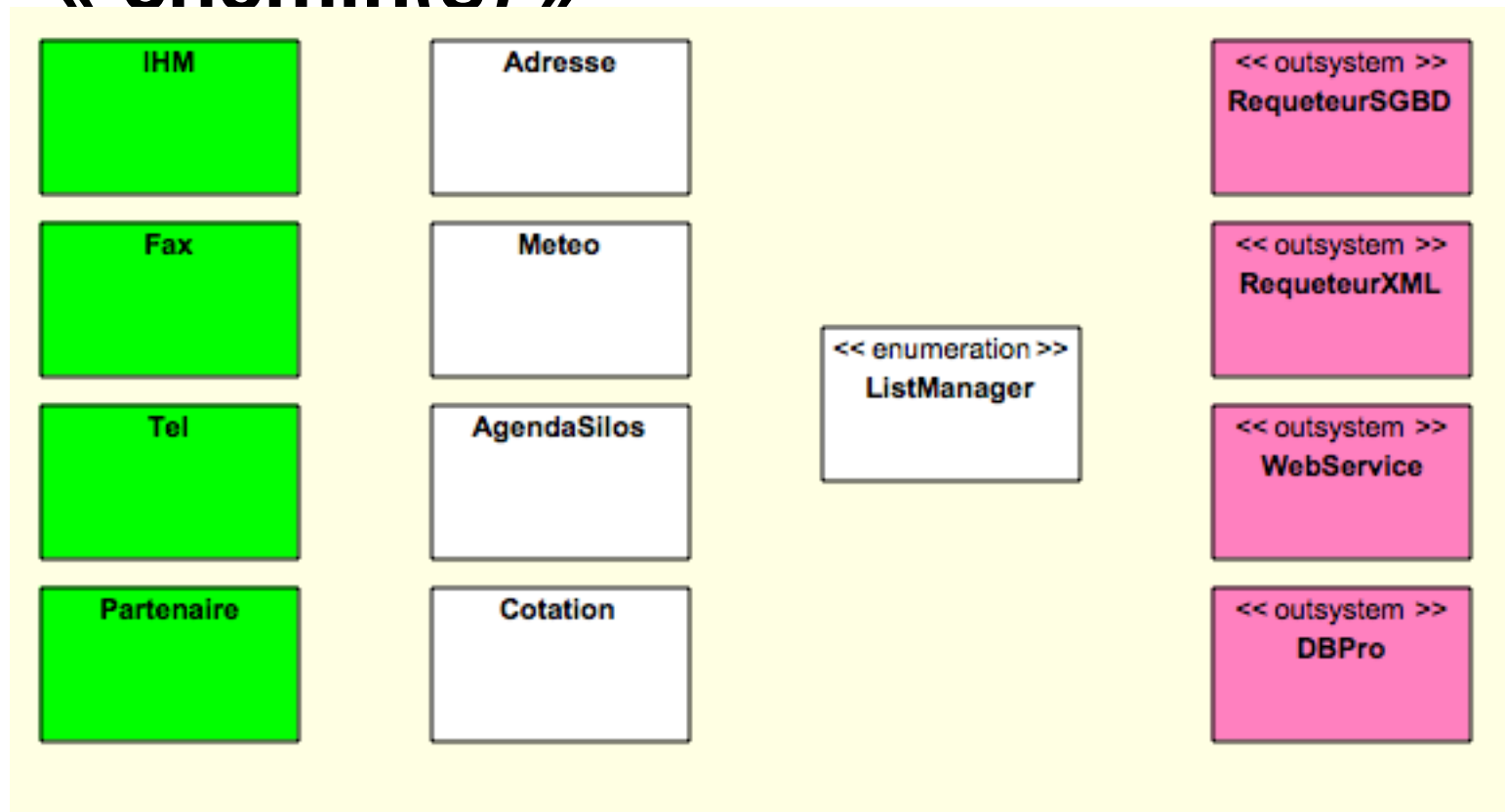
Les packages



Et si on ajoutait une gestion météo au système (température, hygrométrie...) ?

D. de Packages

Le logiciel se définit par son(s)
« chemin(s) »



D. de Packages

Application distributeur bancaire

- Modélisez les Packages du distributeur bancaire