UE1 : Architecture logicielle

Cours 2 :

- Modélisation de l'architecture d'un système avec UML
  - Diagramme de packages
  - Diagramme de composants
  - Diagramme de déploiement
- Gestion des dépendances

Modélisation de l'architecture d'un système avec UML

- Tous les diagrammes UML peuvent être utiles pour décrire les différents aspects du modèle architectural.
- Trois diagrammes UML sont particulièrement utiles :

1. Diagramme de packages
2. Diagramme de composants
3. Diagramme de déploiement

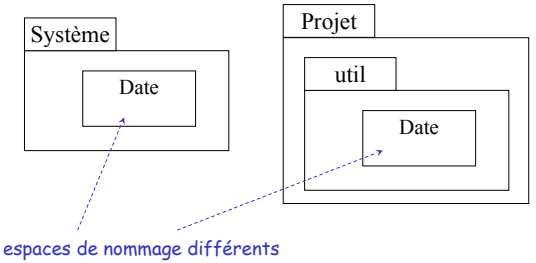
1. Structuration logique : les packages

- Les packages sont une technique importante pour organiser les grands systèmes.
- Ils offrent une structuration des modèles construits sur des classes et permettent de manipuler un ensemble d 'éléments comme un groupe.
- Un package bien construit regroupe des éléments proches qui auront tendance à évoluer ensemble.
- Attention à minimiser les dépendances entre packages.
- On peut contrôler la visibilité des éléments.

Diagramme de package

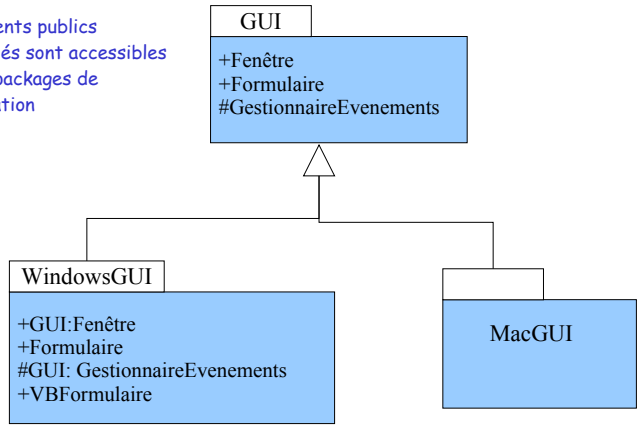
- En UML chaque classe est membre d'un seul package.
- Les packages peuvent être membres d'autres packages d'où une structure hiérarchique des packages.
- Chaque package représente un espace de nommage.
- Dans le diagramme on peut montrer uniquement le nom d'un package ou avec son contenu.

Système::Date
Projet::util::Date



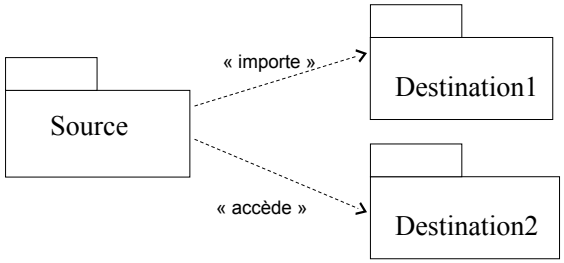
Généralisation/ Héritage entre packages

Les éléments publics et protégés sont accessibles dans les packages de spécialisation

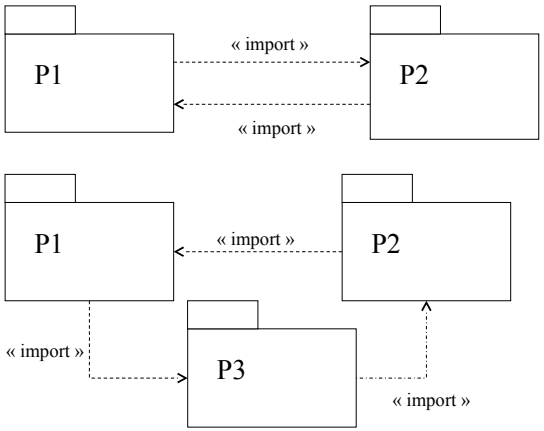


importation et accès

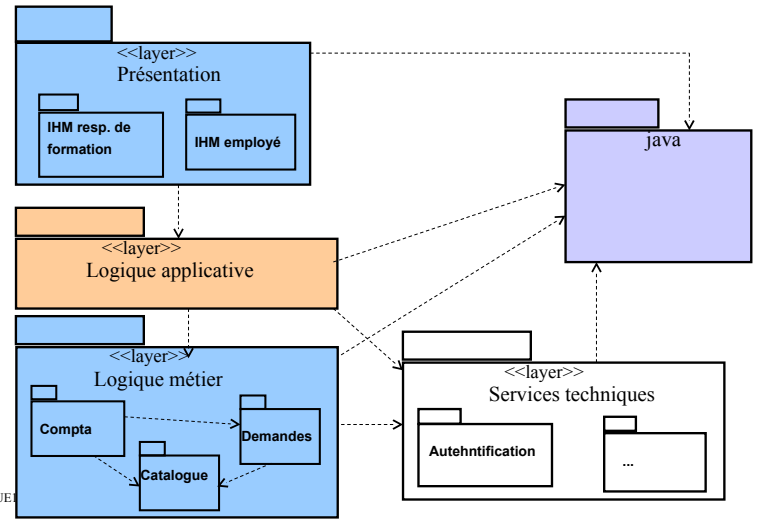
- Relations de dépendance stéréotypées
- La dépendance « *importe* » ajoute les éléments du package destination à l'espace de nommage défini par le package source
- La dépendance « *accède* » permet de référencer des éléments du package destination.



Dépendances circulaires à éviter



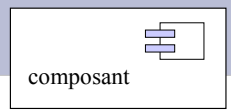
Exemple : composition des packages et architecture en couche



2 . 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 leurs interactions.
- Illustre comment grouper concrètement et physiquement les éléments (objet, interfaces, …) du système au sein de modules appelés composants.

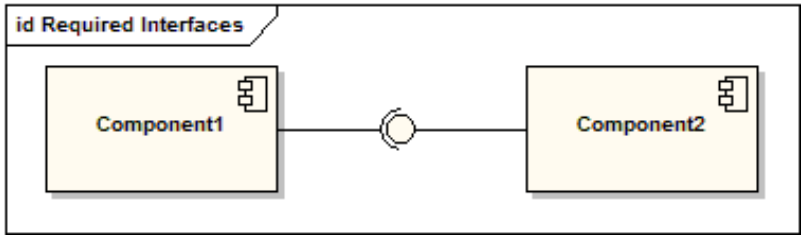
Les composants UML



- En UML tous les éléments physiques (tels que les exécutables, les bibliothèques, les fichiers et les documents) sont modélisés avec des composants.
- Chaque composant contient l'implémentation de certaines classes du système pour lesquelles il spécifie des interfaces et implémente un service.
- Un composant « bien formé » ne dépend pas directement d 'autres composants, mais des interfaces qu'ils supportent. Il est remplaçable par un autre qui implémente les mêmes interfaces.

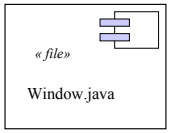
Interface de composants

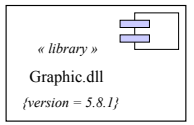
- Types d'interfaces :
  - Interface offerte ou fournie : définit la façon de demander l'accès à un service offert par le composant
  - Interface requise : définit le type de service requis par le composant

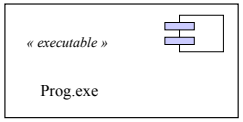


Quelques stéréotypes UML pour les composants

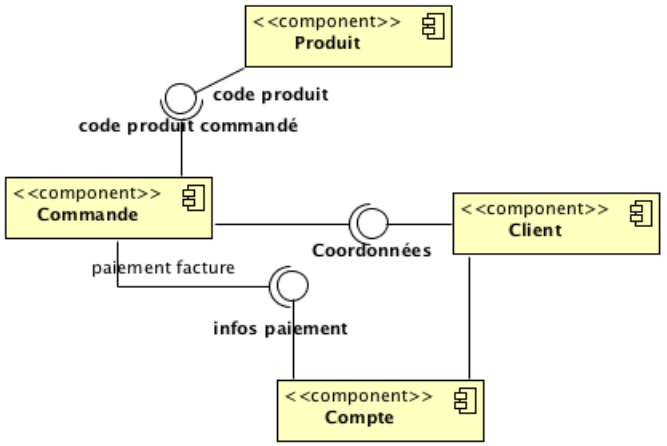
«executable» : composant binaire
«library» :bibliothèque d 'objets statique ou dynamique
«table » : table d 'une base de données
«file» : document contenant du code source ou des données
«document » : un document quelconque





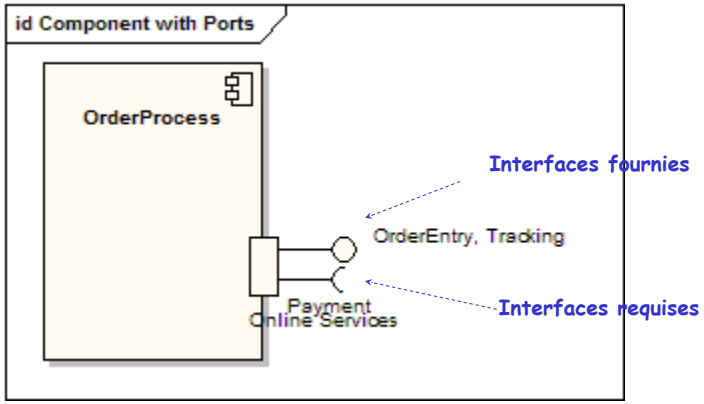


Exemple de diagramme de composants



Ports d'un composant

C'est le point de communication du composant.
Une interface ou plusieurs interfaces peuvent être attachées à un port .



3. Les diagrammes d'implémentation

- Le modèle des composants montre les dépendances entre les différentes parties du code du système.
- Il montre les choix de réalisation des systèmes complexes.
- Le modèle de déploiement montre la structure de l'exécution du système : quelles parties s'exécutent sur tels processeurs et comment le matériel est configuré pour fournir les ressources nécessaires.

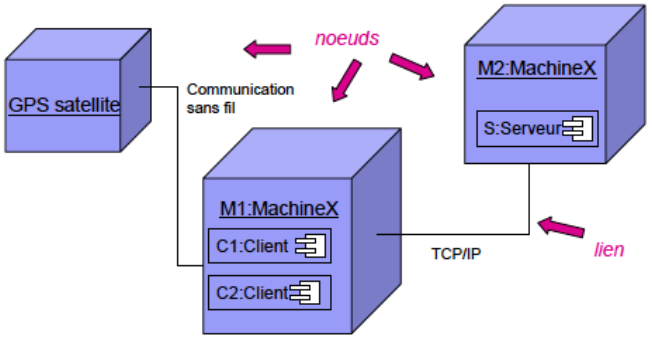
Diagramme de déploiement

- Le diagramme de déploiement montre la configuration physique des différents matériels qui participent à l'exécution du système :
  - processus et objets qui s'exécutent sur ces matériels
  - répartition des instances des composants qu'ils supportent.
- Un diagramme de déploiement est un graphe composé de noeuds interconnectés par des liens de communication.

Les diagrammes de déploiement existent sous deux formes : spécification et instance.

Les noeuds

Chaque ressource matérielle est représentée par un noeud (calculateur, ressources humaines, périphériques…)



Classe de nœud et instance de noeud

- Les diagrammes de déploiement présentent des classes de nœuds ou des nœuds d'instance.
- Un nœud peut donc comme une classe avoir des attributs (vitesse du processeur, quantité mémoire) et des opérations, et participer à des relations (association, généralisation, dépendance)

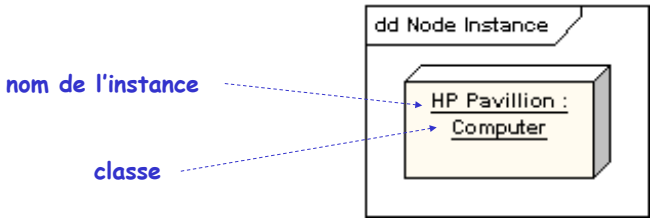
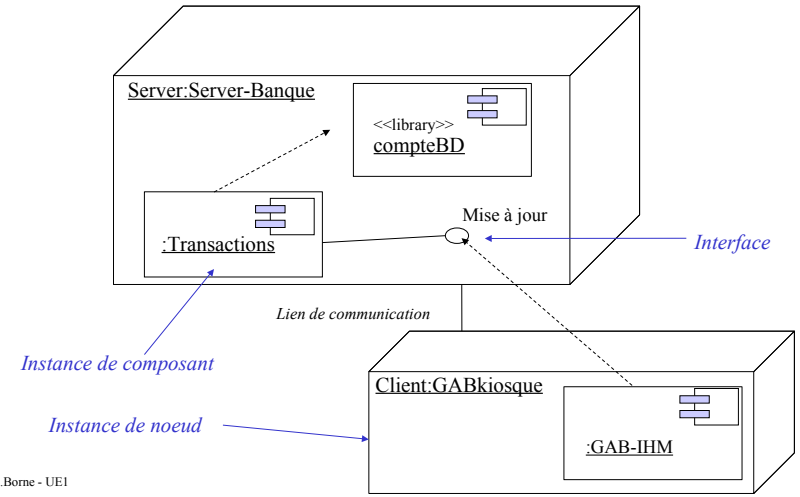


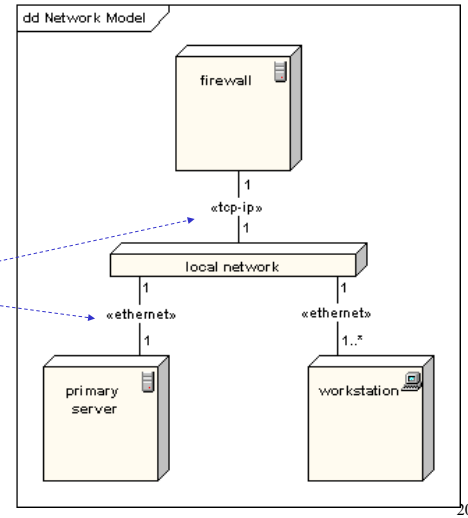
Diagramme avec des instances de composants



Communication (association) entre les noeuds

- Diagramme de déploiement pour un réseau

Stéréotypes pour Les protocoles du réseau



Utilisation des diagrammes de déploiement

Les diagrammes de déploiement modélisent la vue de déploiement statique d'un système.

Typiquement on utilise des diagrammes de déploiement pour :

- modéliser des systèmes embarqués.
- modéliser des systèmes client/serveur
- modéliser des systèmes entièrement distribués

Déploiement et configuration matérielle

Le modèle de configuration matérielle exprime les contraintes de mise en œuvre physique.

Il permet de spécifier, de documenter et de justifier tous les choix d'organisation physique en fonction des machines dédiées aux diverses fonctions techniques.

Développement du modèle de déploiement

- Poste de travail : représente un ou plusieurs acteurs pouvant être localisés sur une machine et remplissant une fonction identifiée.
- Le poste de travail ne représente pas forcément une machine physique, mais plusieurs machines avec même type de déploiement.
- Le modèle de déploiement aide à préciser la qualification des postes client, des réseaux et de leur sécurité physique, par rapport à des critères fonctionnels. Cela permet de justifier la localisation des BD et des environnements de travail.

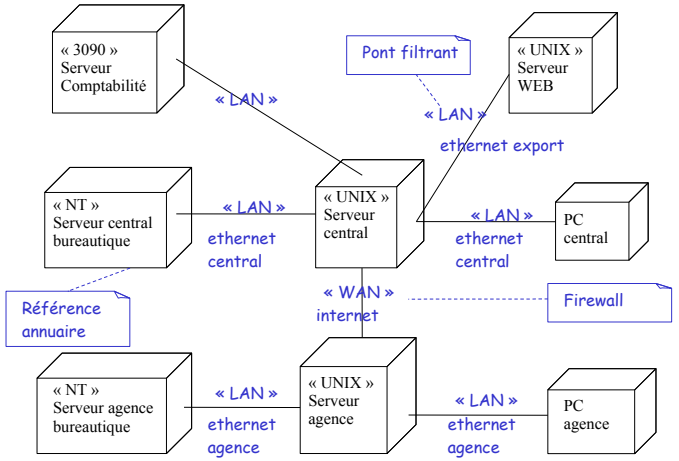
Capture des besoins techniques

- Par complémentarité avec la capture des besoins fonctionnels la capture des besoins techniques couvre toutes les contraintes qui ne traitent ni de la description du métier des utilisateurs, ni de la description applicative.
- Cette étape a lieu quand les architectes ont obtenu suffisamment d'informations sur les prérequis techniques : système matériel, outils pour le développement
- Les choix stratégiques de développement impliquent des contraintes relatives à la configuration du réseau matériel.

Eléments pour les besoins techniques

- Eléments utilisés :
  - Diagramme de déploiement (architecture à 3 niveaux)
  - Diagramme de composants, composants d'exploitation, architecture 3 tiers
  - Diagramme de cas d'utilisation, cas d'utilisation technique
  - Spécification logicielle détaillée

Exemple de configuration matérielle



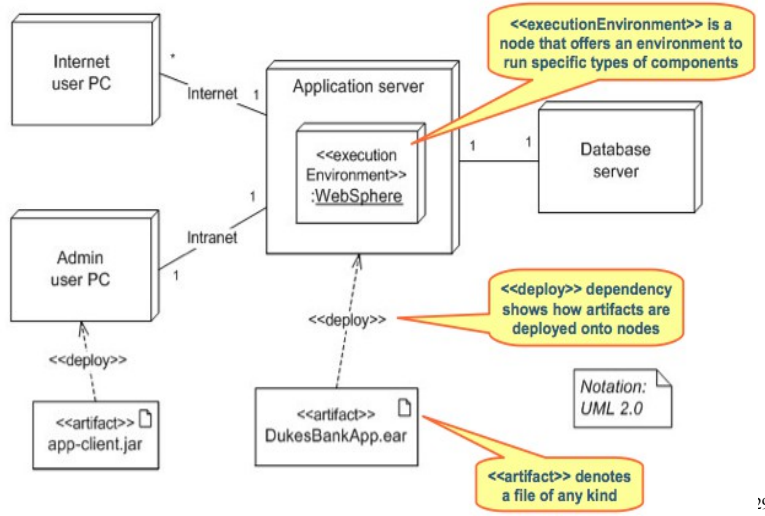
Contraintes d'exploitation de l'exemple

- Pour fonctionner avec le progiciel de comptabilité existant, le système doit nécessairement communiquer avec le serveur de comptabilité mis en oeuvre sur une machine de type « mainframe »
- La communication WAN prise en charge par Internet est subordonnée à l'usage des pare-feux (limite le jeu de protocoles entre clients et serveurs)
- L'export des informations sur Internet impose d'isoler le serveur Web du réseau Ethernet central
- L'annuaire centralisé de référence doit correspondre à celui qui existe sur le serveur de bureautique central

Architecture multi-vues

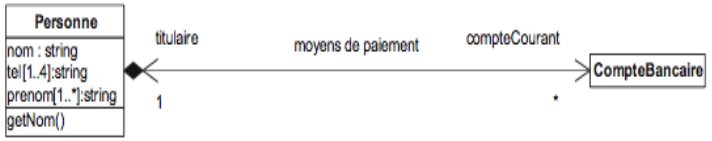
- Une architecture peut considérer le système d'au moins 4 manières :
  - Comment est-il structuré en un ensemble d'unité de code ?
    Vues de module
  - Comment est-il structuré en un ensemble d'éléments qui sont présents à l'exécution ?
    Vues d'exécution
  - Comment les composants sont organisés dans le système de fichier et comment le système est-il déployé sur le matériel ?
    Vues de déploiement
  - Quelle est la structure pour l'entrepôt de données ?
    Modèle de données

Exemple de vue de déploiement
Infrastructure matérielle et artefacts



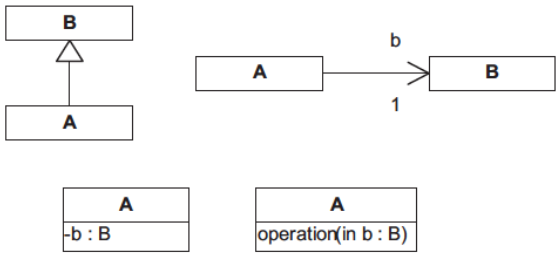
3 . Gestion des dépendances

- Dépendance : fait d'être lié organiquement ou fonctionnellement à un ensemble ou à un élément d'un ensemble.
- Dépendances entre classes.
- Dépendances mutuelles entre classes.



Dépendances entre deux classes

- Les causes d'un relation de dépendances entre deux classes

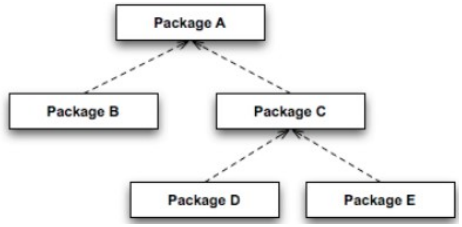


Dépendances entre composants

- Une dépendance est une relation entre deux composants.
- Types de dépendance :
  - Un composant peut dépendre d'un autre 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'implémente concrètement.

Dépendances dans les packages

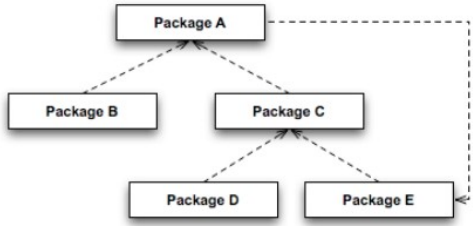
- Le but de la décomposition en packages est de temporiser la propagation des changements dans l'application.



- Point de synchronisation des changements

Effet d'un cycle dans le graphe de dépendances

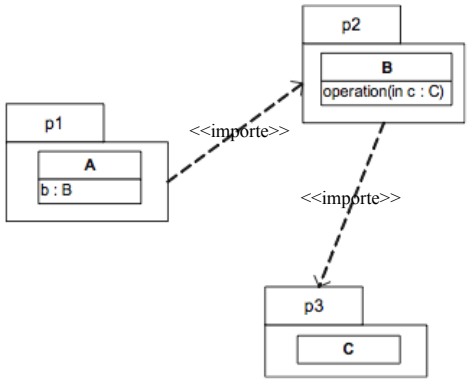
- Pour compiler A il faut utiliser une version de E qui s'appuie sur la version courante de A : A et E doivent être compilés en même temps



Il faut éviter tout cycle dans le graphe de dépendances

Autre exemple de dépendance entre packages

- Établir une relation d'import entre P1 et P2, et entre P2 et P3

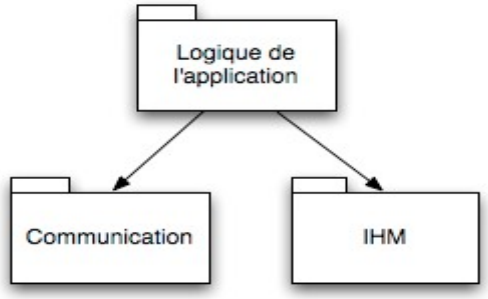


Principe d'inversion de dépendances

- Les modules de haut niveau ne doivent pas dépendre de modules de bas niveau. Tous deux doivent dépendre d'abstractions.
- Les abstractions ne doivent pas dépendre de détails. Les détails doivent dépendre d'abstractions.

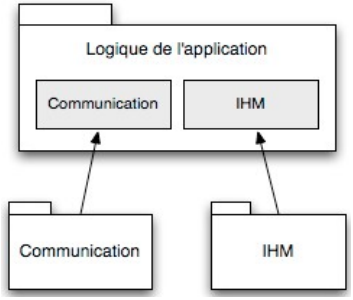
Exemple d'architecture classique

Module de haut niveau : la logique fonctionnelle (aspects métiers)
Modules de bas niveau : bibliothèques graphiques
ou de communication

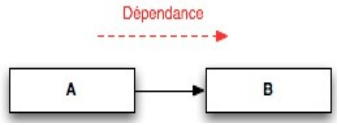


L'inversion de dépendance

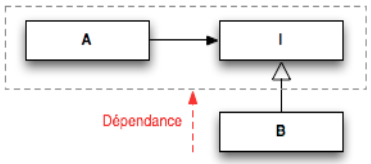
- Selon le principe, la relation de dépendance doit être inversée : les modules de bas niveau doivent se conformer à des interfaces définies et utilisées par les modules de haut niveau.



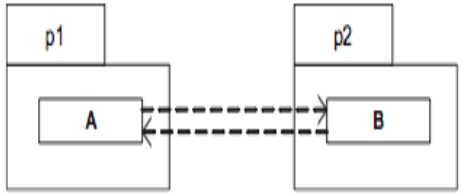
L'abstraction comme technique d'inversion des dépendances



Pour inverser la dépendance de A vers B, on introduit une classe d'interface I dont dérive B



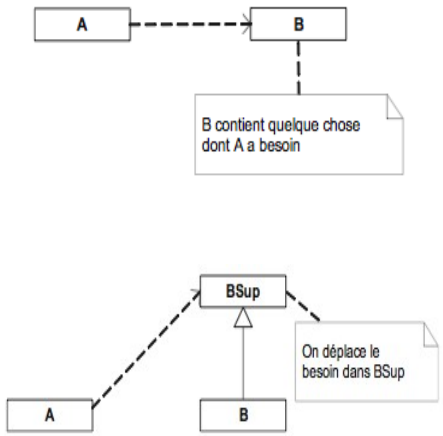
Application du principe d'inversion : Casser les cycles de dépendance



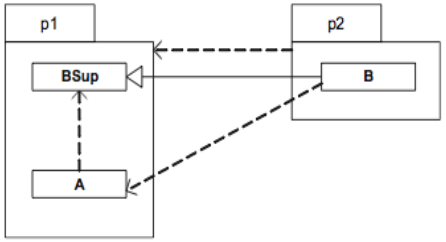
- On a besoin de définir deux packages et les deux classes ont des dépendances mutuelles.
- Le besoin mutuel entre deux classes ne peut pas être supprimé.

Mécanisme de suppression des cycles de dépendance

- Dépendance initiale que l'on veut supprimer
- Changer la dépendance en une indirection à l'aide d'une nouvelle classe et d'une relation d'héritage



Résultat après suppression du cycle



- On a toujours nos deux packages mais sans avoir d'import mutuel

Principe général

1. Identifier la dépendance sur laquelle peut se faire l'indirection
2. Isoler le besoin de la dépendance dans une superclasse afin de déplacer la dépendance. Par exemple si A dépend de B car A utilise une opération de B, il faut positionner cette opération dans la superclasse afin de pouvoir déplacer le lien de dépendance.
3. Etablir la relation d'héritage.
4. Positionner les classes dans les packages et mettre les bonnes relations d'import.