IFT6803:
Génie logiciel du commerce électronique

Chapitre 3: Conception orientée objet

Section 2: Conception architecturale

Julie Vachon,
Hiver 2003

Sommaire

Chapitre 3, Section 2

« Conception architecturale »

3.2.1 Architecture logicielle

Définition

- L'architecture logicielle décrit
  - l'organisation générales du système,
  - les éléments structurants et leurs interfaces,
  - le comportement du système, des sous-systèmes et des composants: leurs propriétés, leur composition (« comprend ») et leurs collaborations (« utilise »).

Architecture logicielle

Choix d'une architecture

- Dépend des besoins fonctionnels et non fonctionnels du système.
- Influencé par certains « modèles connus » de décomposition en composants et mode d'interactions.
- Utilité des architectures
  - **Modèle éprouvé** et enrichi par l'expérience de plusieurs développeurs.
  - **Mode d'interaction** établi entre modules via une sorte d'interface générique
  - Fournit une **plateforme d'intégration** pour connecter les différents sous-systèmes du système à développer.
  - Contribue à une **meilleure qualité du logiciel**: compréhensibilité, maintenance, évolubilité, réutilisation, performance, documentation, etc.

Architecture logicielle

Dimensions d'une architecture

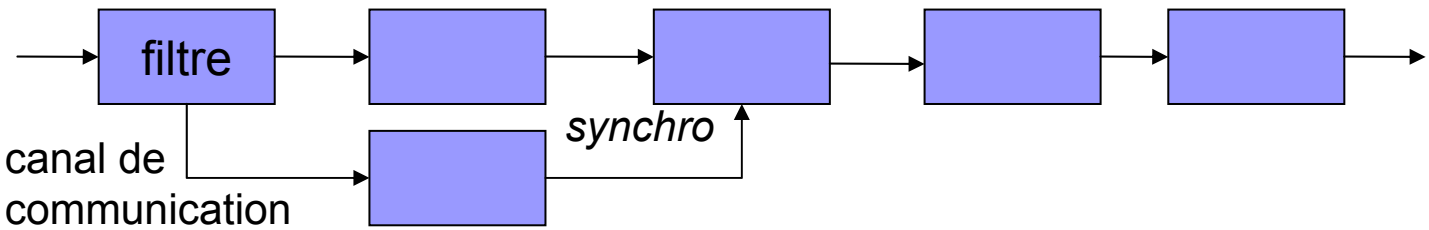
- Une architecture peut être vue sous plusieurs angles. Entre autres
  - Vue logique. Organisation conceptuelle (classes, interfaces, paquetages, sous-systèmes, composants, etc.)
  - Vue de déploiement. Organisation physique (allocation de processus aux unités de traitement, configuration réseau, etc.)

La décomposition proposée par la vue logique contribuera à définir la distribution de la vue de déploiement.

3.2.1.1 Architectures typiques

Architecture pipeline

- Sous-systèmes organisés en pipeline de filtres indépendants:
  - la sortie d'un filtre correspond à l'entrée de l'autre.
- Communication locale (voisins gauche et droit). Un filtre peut souvent commencer à opérer avant même d'avoir lu tout le flux d'entrée
- Utile pour les traitements en plusieurs étapes.
- Analyse facilitée: performance, synchronisation, goulot d'étranglement, etc…
- Exécution concurrente de filtres possible. Synchronisation des flux parfois nécessaire.
- Bon pour traitement en lot (batch). Mauvais pour traitement interactif.

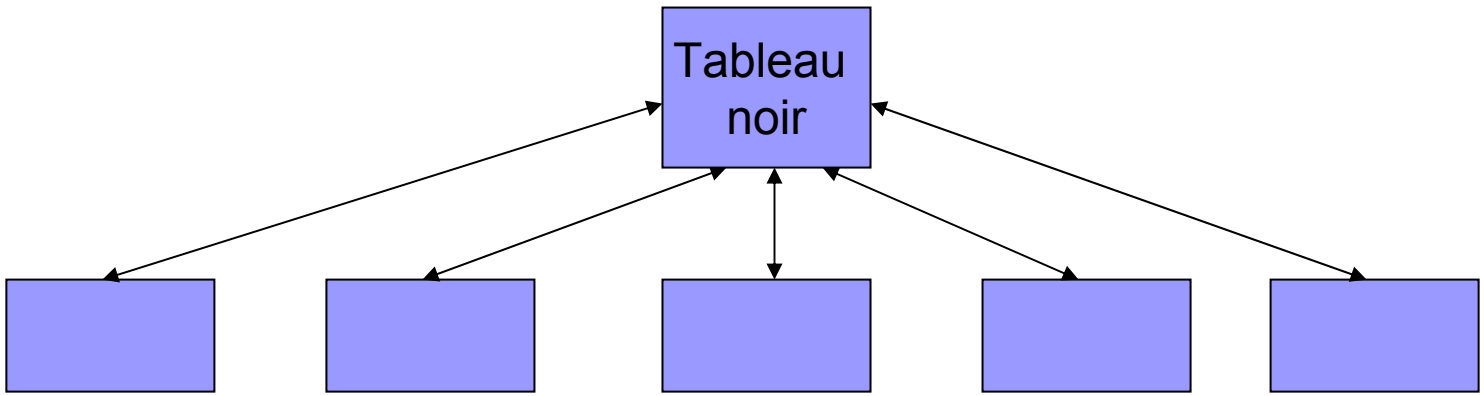


Ex.: Filtres employés pour la compilation d'un programme: analyseur lexical, syntaxique, sémantique, génération de code, …

Architectures typiques

Architecture de tableau noir

- Tableau noir: médium de communication entre les sous-systèmes.
- Communication étendue à tous les partenaires.
- Un des sous-système est désigné comme le tableau noir.
- On peut recevoir et transmettre des informations via le tableau.

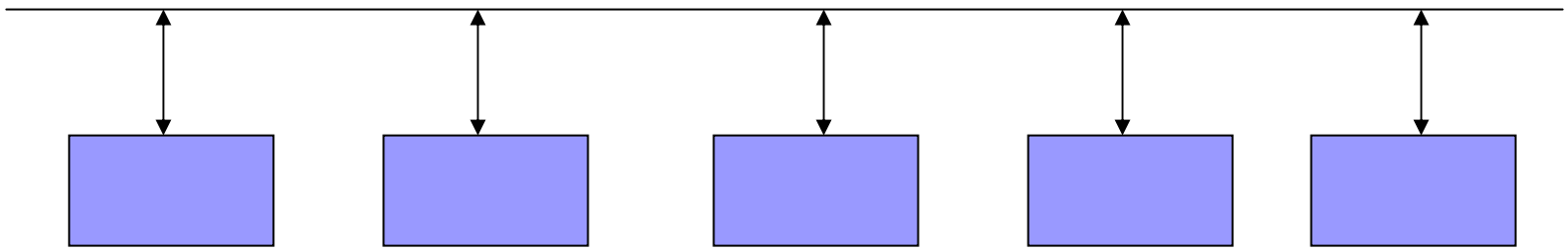


Ex. Système de vente aux enchères simplifié

Architectures typiques

Architecture basée événements

- Les sous-systèmes répondent aux événements.
  - Evénement = clic de souris, détection d'un signal par un capteur, arrivée d'un message, etc.
  - Représentation conceptuelle: l'annonce des événements est propagée sur un bus
- Appropriée pour les systèmes dont les composants doivent interagir avec l'environnement
- Les sous - composants s'abonnent pour recevoir les annonces de certains types d'événements.



Ex. Système d'interfaces utilisateurs

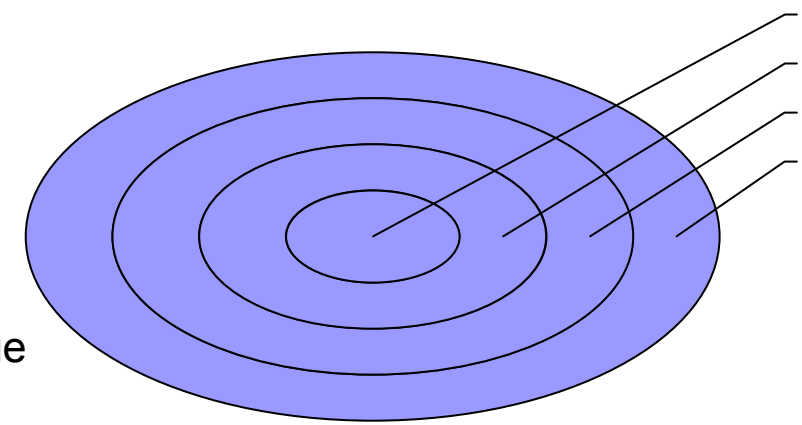
Architectures typiques

Architecture par couches

- Chaque couche offre un service (serveur) aux couches externes (client).
  - Service créé indépendamment du logiciel
  - Service crée pour son rôle particulier dans l'application.
- Le design doit expliquer le protocole d'interaction entre couches.
  - Toutes les couches ont accès à toutes les autres ?
  - Une couche n'a accès qu'aux couches adjacentes ?

  -- *Relation hiérarchique privilégiée.*
- Met à profit la notion d'abstraction. Les couches externes (ou supérieures) sont plus abstraites que les internes (ou inférieures)…
- Souvent utilisé pour les systèmes implémentant des protocoles.

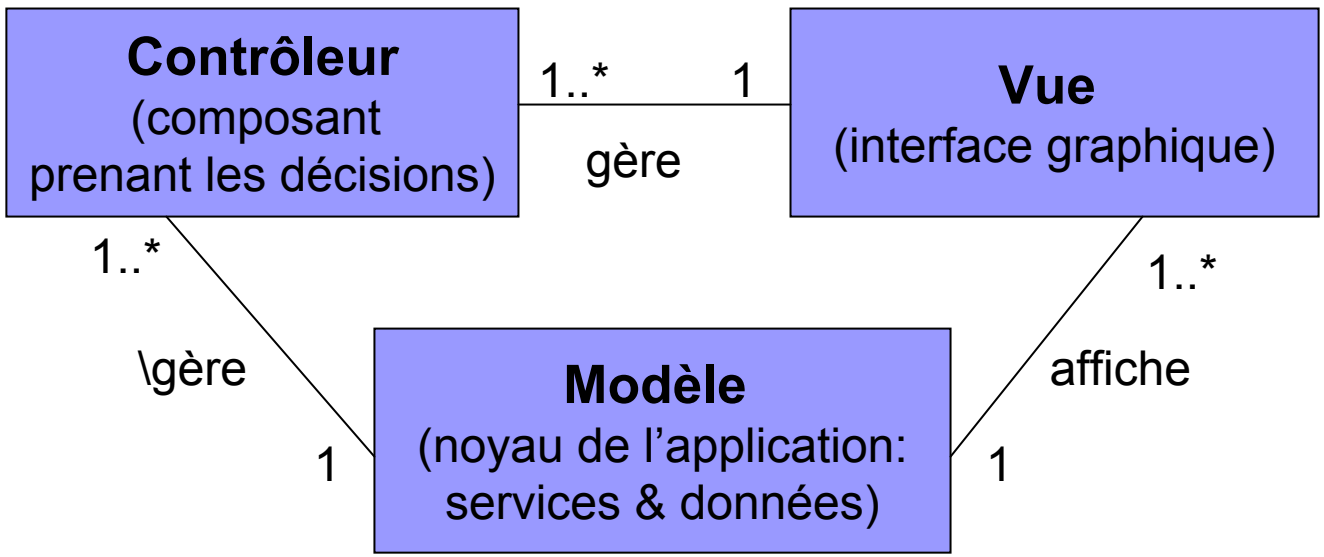
<u>Pour décrypter un fichier</u>
•Entrée d'un mot de passe
•Obtention d'une clef d'accès
•Encryptage/décryptage du fichier
•Fonctions d'encryptage/décryptage



Architectures typiques

Architecture Modèle-Vue-Contrôleur

- Approche architecturale (pattern) pour gérer les problèmes d'interface.
- Popularisée par l'environnement Smalltalk (Goldberg et Robson,1983)



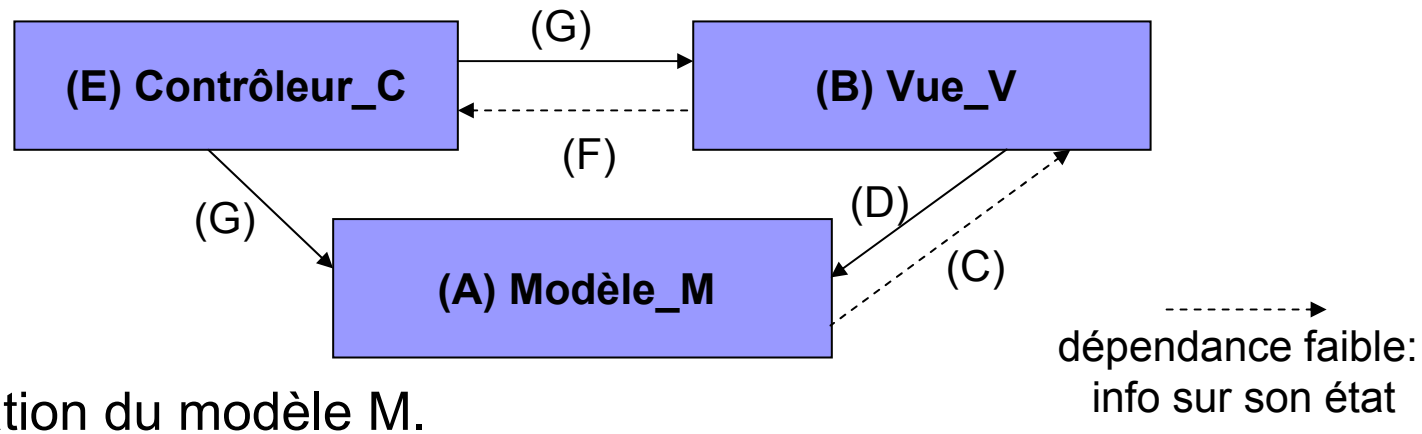
Architectures typiques

Architecture Modèle-Vue-Contrôleur

- Composants:
  - **Modèle = noyau de l'application**
    - ensemble des données et fonctions utilisées par l'application.
    - Connaît les composants « vue » qui dépendent de lui.
    - Informe les composants dépendants à propos des changements de données.
  - **Vue = composant graphique de l'interface**, façon de présenter les données du modèle
    - Crée et initialise son (ses) contôleur(s)
    - Affiche les informations pour l'utilisateur
    - Implémente sa propre procédure de mise à jour (fidélité au modèle)
    - Puise ses données du modèle
  - **Contrôleur = composant responsable des prises de décision**
    - Traite les événements résultant des entrées de l'usager
    - Traduit les événements (utilisateur) en requête de service pour le modèle OU en requête d'affichage pour la vue.
    - Implémente, si nécessaire, des procédures indirectes de mise à jour graphique

Architectures typiques

Architecture Modèle-Vue-Contrôleur

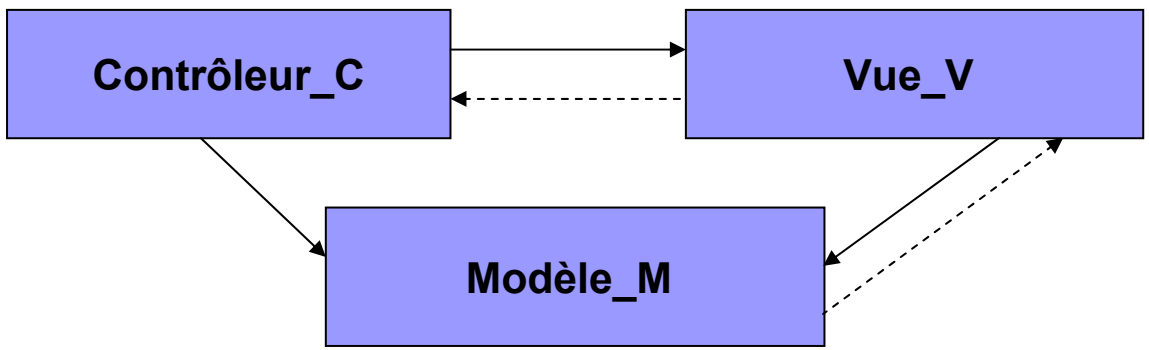


- (A) Création du modèle M.
- (B) Création d'une ou plusieurs vues.
- (C) Modèle initialisé avec ses vues.
- Initialisation de chaque vue V:
  - (D) La vue V est initialisée avec une référence sur le modèle M
  - (E) Création d'un (ou plusieurs) contrôleurs C.
  - (F) La vue V enregistre une référence sur le(s) contrôleur(s) C.
  - (G) Le contrôleur C est initialisé avec une référence sur le modèle M et une référence sur la vue V.

Architectures typiques

Architecture Modèle-Vue-Contrôleur

- Mode d'interaction « Modèle initiateur»
  - Lorsque le modèle change, il informe la(les) vue(s) qui dépend(ent) de lui. Les vues modifient leur affichage en conséquence.
  - Les vues informent leur contrôleur respectif du changement pour qu'il puisse prendre les actions conséquentes (ex. rendre inactif certains boutons de l'interface)
- Mode d'interaction « Contrôleur initiateur »
  - Lorsqu un événement utilisateur se produit, le contrôleur en est averti, décide du changement qui doit être apporté au modèle et l'en informe. À son tour, le modèle modifié informe les vues dépendantes (cf. modèle initiateur).

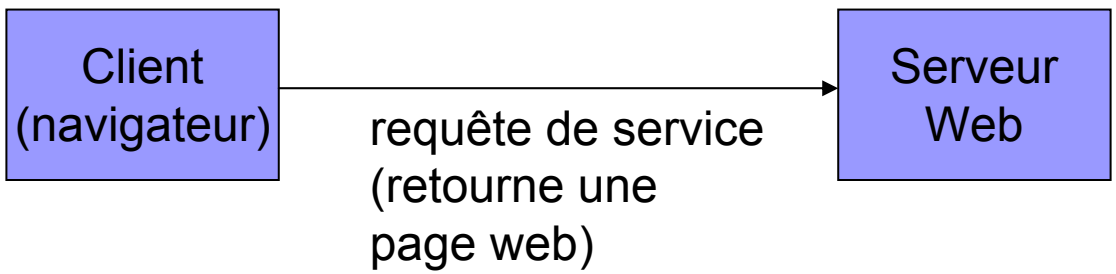


Architectures typiques

Architecture N-tiers

- Pour les systèmes distribués
- Architecture par couches
- Par abus de langage la notion de «tier» a pris le sens de couche distribuée.

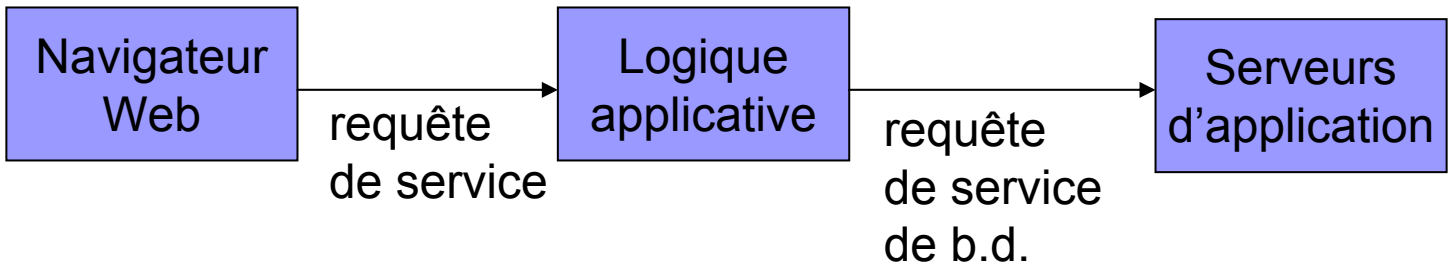
Architecture 2 tiers (thick client)



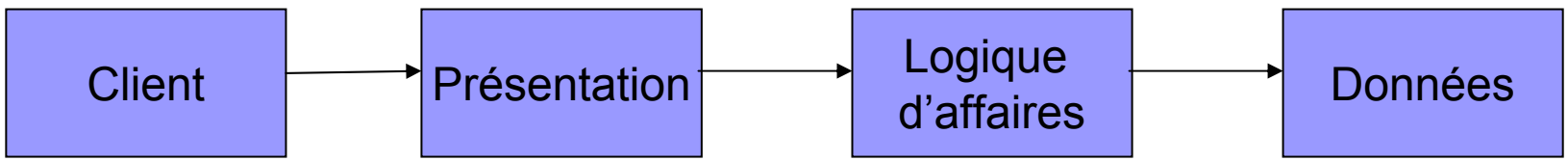
Architectures typiques

Architecture N-tiers

Architecture 3 tiers (thin client)



Architecture N tiers



Architectures typiques

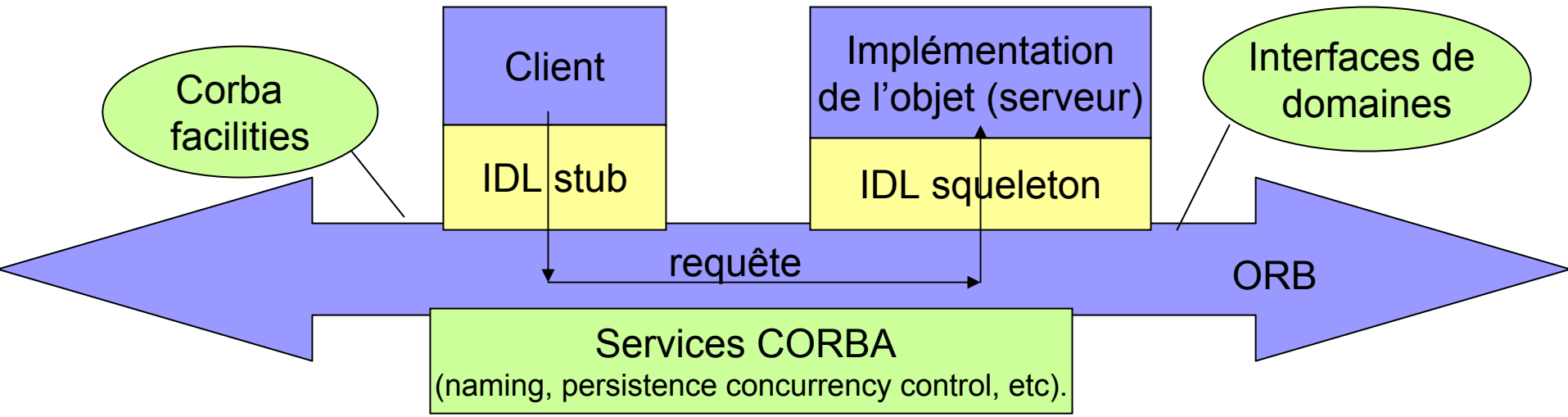
Modèle de composants

- Composant
  - Unité logicielle réutilisable.
  - Propriétés recherchées:
    - Interfaces claires
    - Services utiles et indépendants
    - Domaine d'application clair
  - Empaquetés dans des librairies, boîtes à outils, frameworks, etc.
- Modèle de composants
  - Infrastructure pour la composition et l'interaction de composants. Ex. EJB, JavaBean, COM.

Architectures typiques

Architecture comme cadre d'application pour l'intégration de composants (CORBA)

- Plateforme d'intégration de composants hétérogènes.
- Cadre d'application pour la réalisation de systèmes distribués.
- ORB (Object Request Broker):
  - bus qui permet à des clients et serveurs de se connecter.
  - les ORB sur différents réseaux peuvent communiquer ensemble.
- IDL: Interface Description Language



3.2.2 Stratégies de réutilisation

Granularité de la réutilisation

Granularité de la réutilisation	Stratégie
Classes (code)	Librairie
Composants et architecture (code et design)	Framework
Idée de solution (solution de design)	Pattern

Stratégies de réutilisation

Boîtes à outils

- Accent mis sur la **réutilisation du code** de classes définies dans une librairie.
- Il revient au programmeur de développer l'architecture et la logique de l'application.

- Types de boîtes à outils:
  - Foundation toolkits: Ensemble de classes mises à disposition dans un environnement de programmation: classes pour les données primitives, structurées, etc.

    Exemple: JDK 1.3, STL, etc.
  - Architecture toolkits: Ensemble de classes mis à disposition par un autre système (système d'exploitation, ODBMS (classes pour la persistence, les transactions, etc.), GUI toolkit)
    - Exemple. Swing, Cloudscape, EJB, etc.

Stratégies de réutilisation

Framework (cadre d'application)

Squelette de programme que le programmeur doit compléter…

- Ensemble intégré de **constructions logicielles** (classes, composants, etc.) qui collaborent et s'organisent autour d'une **architecture réutilisable** dans le but de faciliter le développement d'une **famille d'applications.**
  - Réutilisation d'une architecture.
  - Favorise la réutilisation de code (librairie de composants, de classes, etc.)
  - Favorise la réutilisation de solutions de conception (assemblage des composants, etc.)
- Constitue une architecture de base et une application partielle devant être spécialisée (classes abstraites doivent être implémentées)

Stratégies de réutilisation

Framework

- Ressemble à une librairie mais offre davantage:
  - Les composants d'un cadre d'application sont liés par des relations structurelles et de comportement prédéfinies.
- Offre une solution à la conception d'application dans un domaine spécifique donné:
  - Affaires, télécommunications, noyau de système d'exploitation, bases de données.
- Contrôle inversé
  - La boucle de contrôle principale est fournie par le framework au lieu d'être définie dans le code de l'application.
  - Le code du framework peut appeler le code défini dans l'application (liaison dynamique)

Stratégies de réutilisation

Framework

- Exemples:
  - **Brokat Financial Framework**: framework J2EE dédié au développement d'applications financières
    - Composants pour la sécurité, les requêtes interactives simples, les transactions, les services de confirmation avec l'utilisateur
  - **Struts:** framework J2EE dédié à la gestion de l'interaction homme-machine dans les applications Internet/intranet. Il propose de construire vos applications Internet autour d'une organisation de l'interaction Homme-Machine respectant le modèle d'architecture MVC II.
  - **Kona:** a Java/J2EE/JSP Framework and Tag Library for Rapid, Internet-Enabled, Business Application Development

Stratégies de réutilisation

Patterns

- Description d'une **solution** recommandée pour un **problème typique** dans un **contexte donné**.
  - Solution générique / problème générique.
- Solutions agrées par les bons programmeurs.« Bonnes pratiques », savoir-faire de conception.
- Facilite la communication entre développeur, peut réduire les risques.
- Permet de documenter un design.
- Regroupés dans un catalogue.

Stratégies de réutilisation

Patterns

- Le concept de « pattern » vient du monde des architectes:
  - Christopher Alexander 1977
- Constatation:

"Chaque modèle décrit un problème qui peut se produire plusieurs fois dans un environnement; il décrit l'essentiel d'une solution à ce problème, de telle sorte qu'on peut réutiliser cette solution de plein de façons différentes."

Stratégies de réutilisation

Patterns

- Types de patterns:
  - **Patterns d'analyse**
    - Permet de résoudre pbms liés à l'analyse. Ex. Comment utiliser l'agrégation ? Stratégies pour construire modèle objet dans un domaine donné.
    - Catalogue: Patterns de COAD (Fowler 95)
  - **Pattern de conception**
    - Patterns architecturaux: s'intéresse à la structure du système, sous-système et composants et aux relations entre eux.
    - Patterns de conception (design patterns): s'intéresse aux classes et aux relations entre elles.
      - Catalogue: pattern GoF (Gamma, Helm, Johnson, Vlissides)
  - **Idiomes (pattern d'implémentation)**
    - Explique comment implémenter certaines solutions en utilisant les features d'un langage spécifique.
    - Exemble: Solutions spécifiques à Java pour gérer les collections, les interfaces, les exceptions, la concurrence, etc.

      Java Idioms: http://c2.com/ppr/wiki/JavaIdioms/JavaIdioms.html

Stratégies de réutilisation

Patterns

- Format
  - **Nom**
  - Utilité
  - Alias
  - **Contexte**
  - **Problème**
  - **Solution**
  - Implémentation
  - Etc.

Stratégies de réutilisation

Patterns

- Patterns de conception
  - Particulièrement utiles lors du passage d'un modèle d'analyse à un modèle de développement parce qu'ils offrent aux développeurs un catalogue de solutions « clé en main ».
  - Types de patterns de conception:
    - **Creational patterns**: Proposent solutions aux problèmes de configuration et d'initialisation. Ex. Singleton pattern
    - **Structural patterns**: Proposent solutions aux problèmes de structure en organisant les interfaces et les relations entre classes de façon spécifique. Ex. Adapter pattern
    - **Behavioral patterns**: Proposent solutions aux problèmes de comportement en identifiant les façon dont un groupe de classe doit interagir pour réaliser un certain but. Ex. Observer pattern

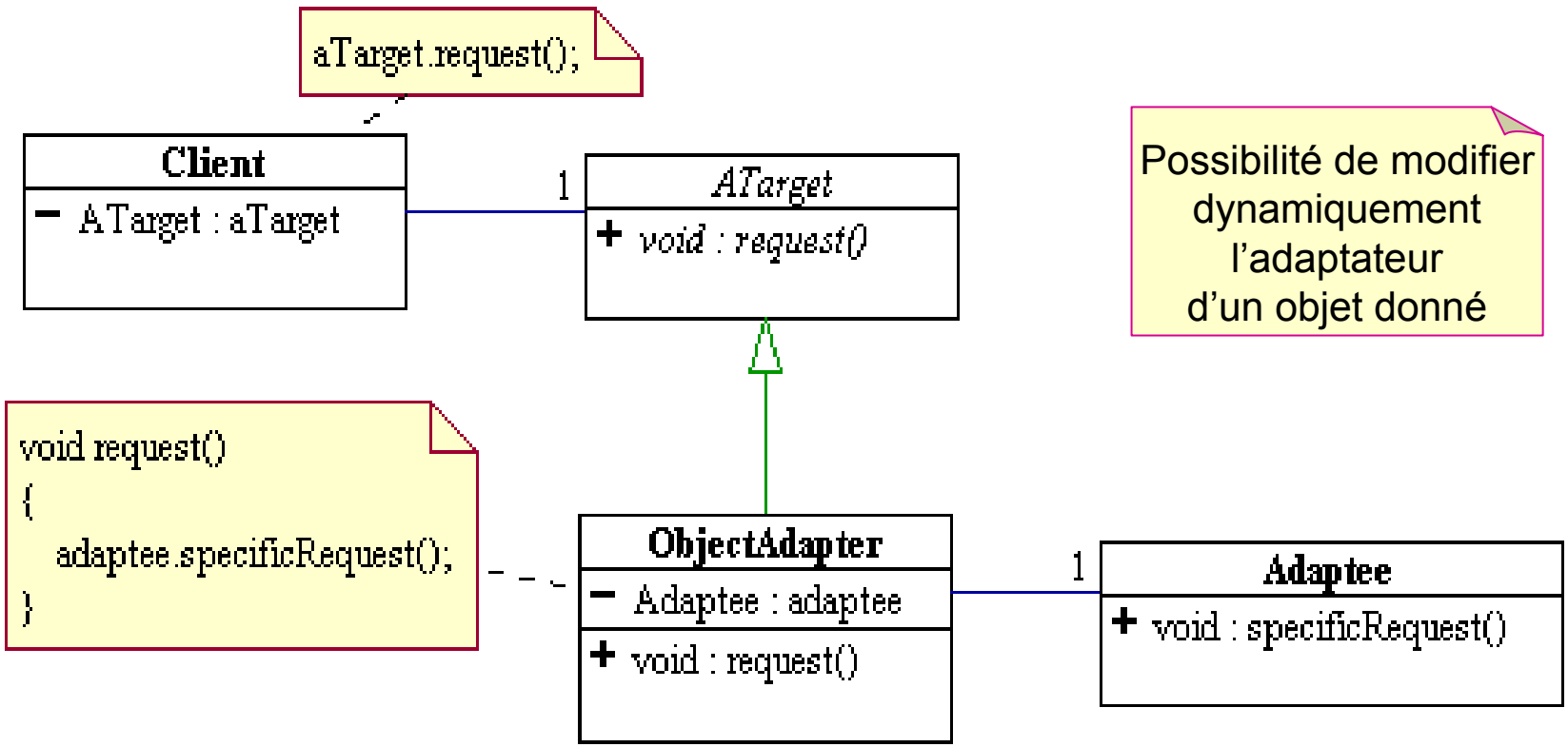
Stratégies de réutilisation

Patterns – Exemple

- **Nom**: Adaptateur
- **Alias**: Wrapper
- **Contexte**: Le client invoque une méthode d'un serveur.
- **Problème**: Le client s'attend à une interface différente de celle proposée par le serveur. Incompatibilité des interfaces.
- **Solution**: Utiliser un « adaptateur » pour convertir une interface et l'adapter à l'autre. On convertit l'interface A d'une classe en une interface B que le client comprend…
  - Adaptateur objet: s'appuie sur l'héritage simple et la délégation.
  - Adaptateur classe : s'appuie sur l'héritage multiple ou l'héritage d'interface.

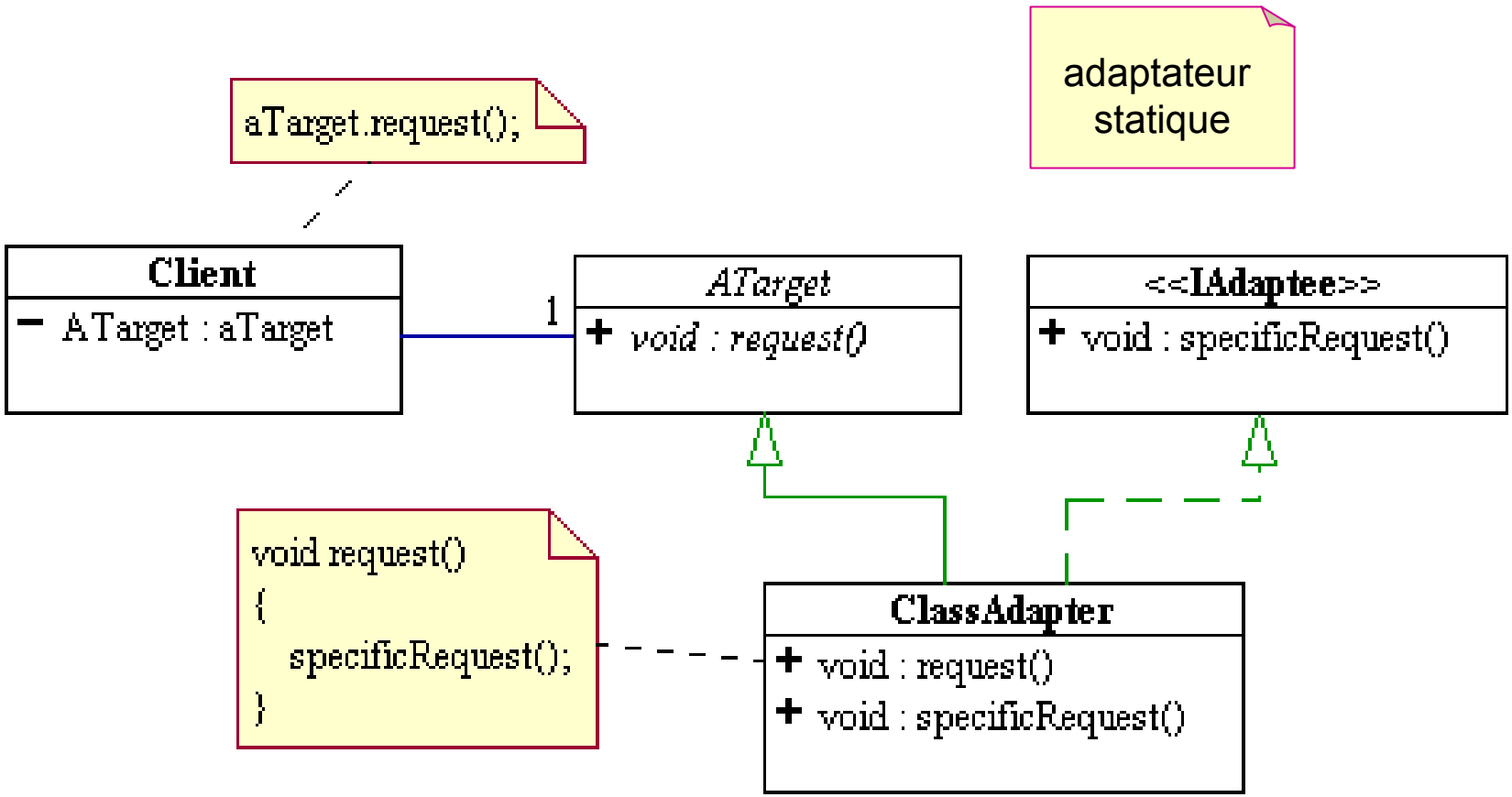
Stratégies de réutilisation

Patterns – Adaptateur objet



Stratégies de réutilisation

Patterns – Adaptateur classe



Stratégies de réutilisation

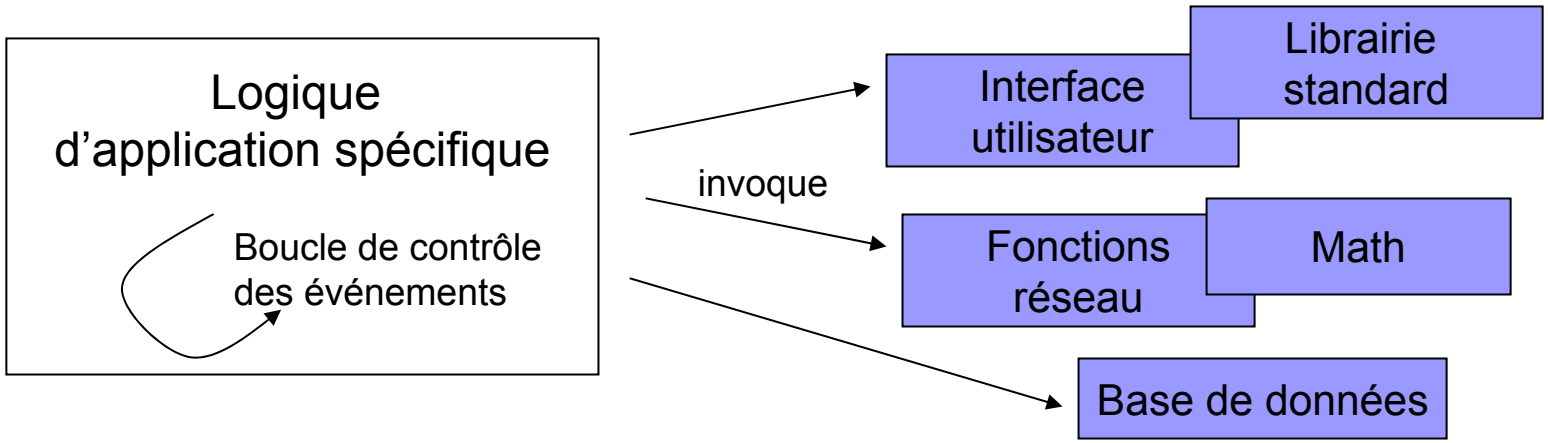
Pattern vs Framework

- Patterns
  - Réutilisation:design
  - Indique comment produire un bon design
  - Indépendant du langage d'implémentation
  - Couvre souvent des domaines généraux
  - S'intéresse à une petite partie de l'architecture.
- Framework
  - Réutilisation : design et code.
  - Est un design/conception architecturale (implémentation générique)
  - Langage d'implémentation spécifique.
  - Couvre souvent un domaine
  - S'intéresse à l'architecture générale.

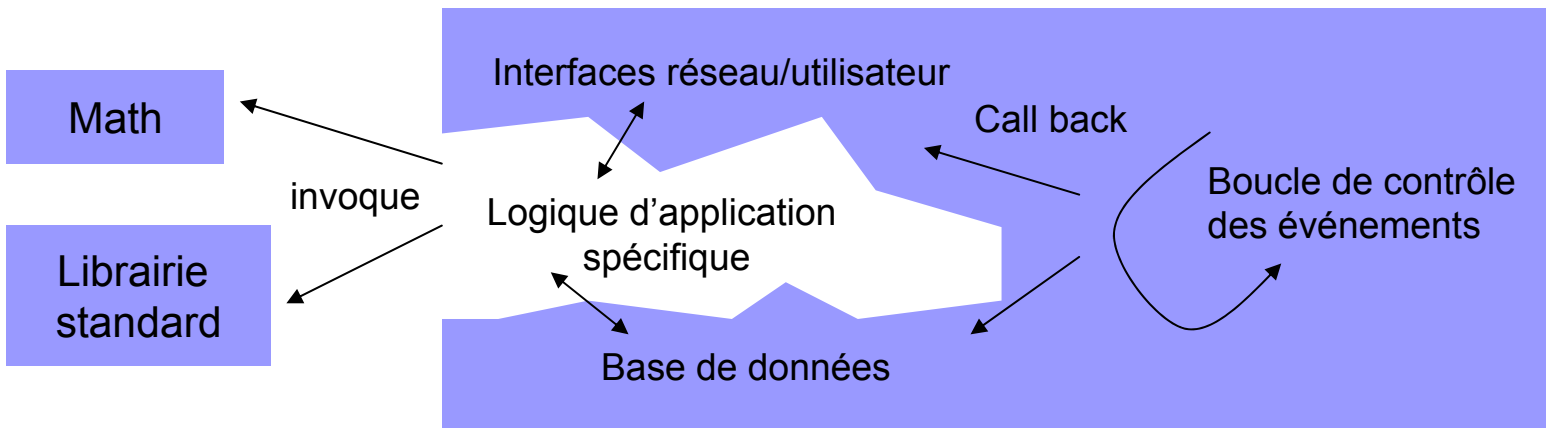
Stratégies de réutilisation

Boîte à outils vs Framework

Boîte à outils



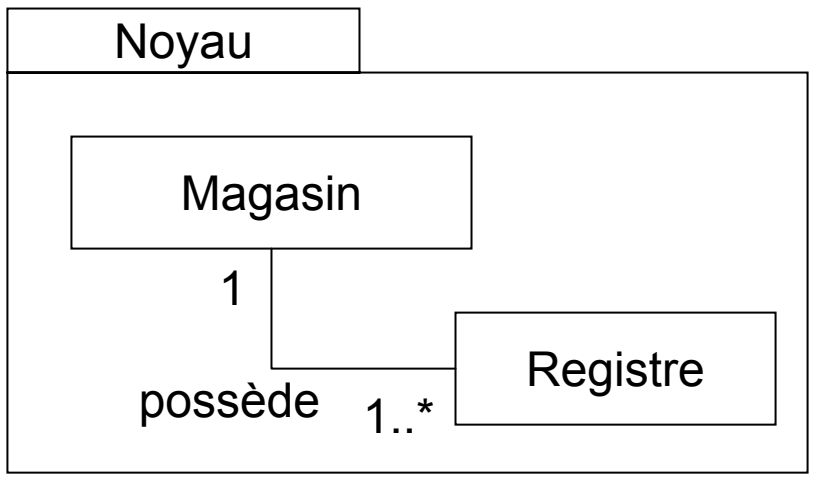
Framework



3.2.3 Conception des paquetages

Diagramme de paquetages

- **Paquetage**
  - Mécanisme général pour grouper **logiquement** les éléments de modélisation d'UML (classes, use case, composants, diagrammes, etc.)
  - Groupement logique d'éléments, sous-systèmes.



Conception des paquetages

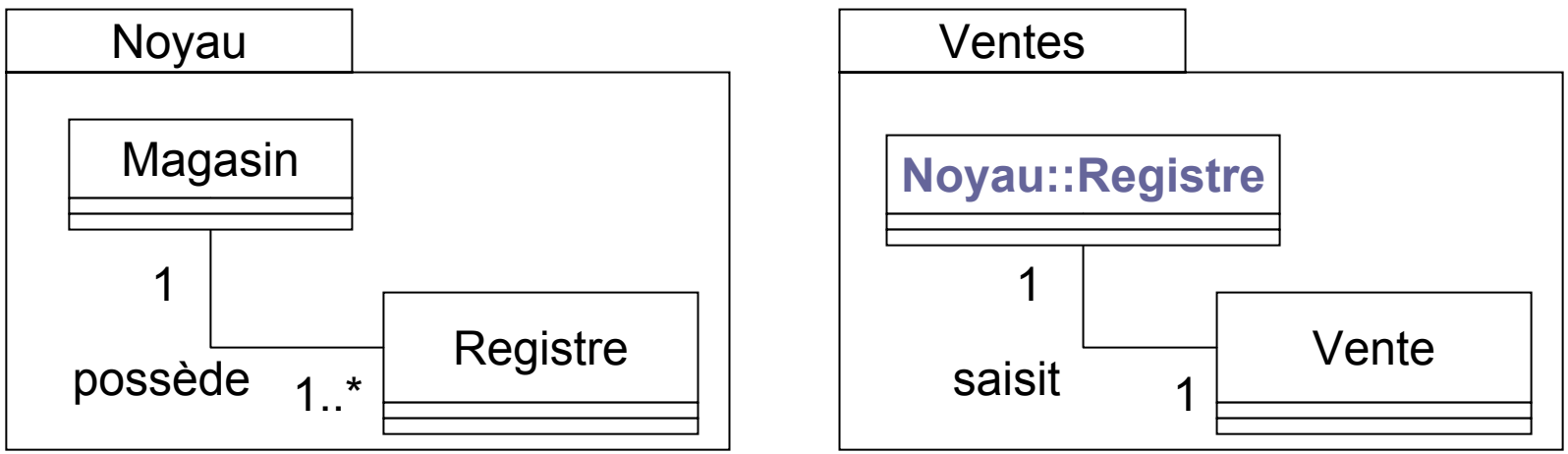
Diagramme de paquetages

- **Utilisation:**
  - Permet d'organiser et de structurer les grands systèmes.
  - Permet de contrôler les accès aux éléments d'un modèle, de gérer leur espace de stockage et leur configuration.
  - Un système peut être vu comme un paquetage unique contenant des sous-paquetages (représentant ses sous-systèmes).

Conception des paquetages

Diagramme de paquetages

- **Définition et référence**
  - Un élément de modélisation n'est **défini** que dans **un seul** paquetage.
    - Les paquetages peuvent être imbriqués.
    - Hiérarchie d'appartenance est un arbre strict.
  - Un élément **peut être référencé** dans plusieurs paquetage en utilisant son nom complet i.e. *NomPaquetage :: NomÉlément.*
    - Un paquetage constitue un espace de nommage.



Conception des paquetages

Diagramme de paquetages

- **Visibilité**

  On peut régler la visibilité du contenu d'un paquetage en employant les spécifications ci-dessous devant le nom du paquetage:

  - **private**: les éléments du paquetage ne sont pas visibles depuis l'extérieur du paquetage.
  - **protected**: les éléments du paquetage sont visibles à l'intérieur du paquetage et depuis les paquetages liés par un lien de généralisation.
  - **public**: les éléments du paquetages sont visibles par tous.

- **Stéréotype prédéfinis**: «system», «subsystem».

Conception des paquetages

Diagramme de paquetages

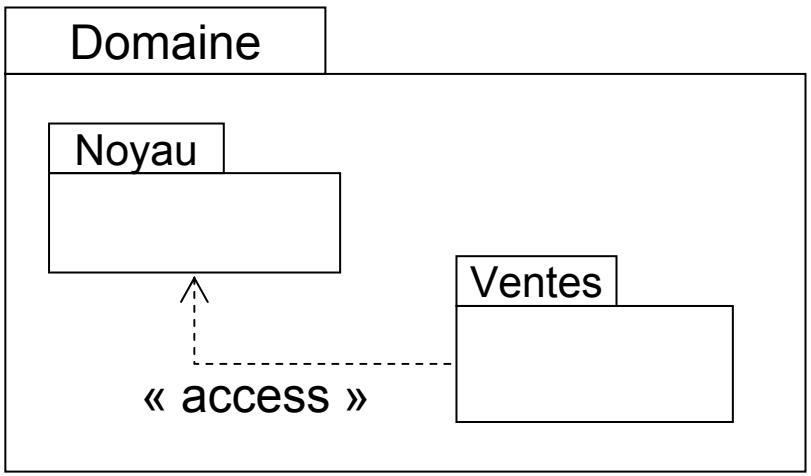
- **Relations entre paquetages**

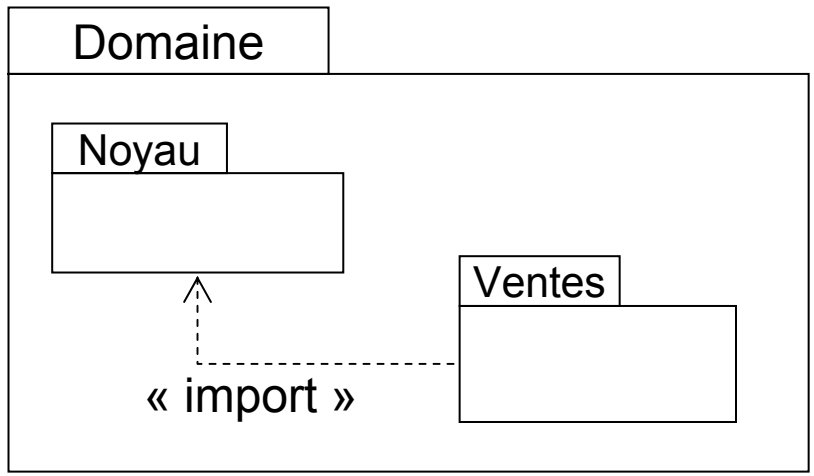
1. Généralisation (ex. spécialisation d'un paquetage abstrait)
2. Dépendance (c.f. relation UTILISE)
   - access dependency
     - P1 utilise (ou peut utiliser) au moins un élément de P2.
     - relation contrainte par les règles de visibilité.
     - les espaces de nommage entre P1 et P2 demeure distincts.
   - import dependency
     - Même objectif que access dependency
     - Sauf que l'espace de nommage est étendu: il n'est donc pas nécessaire d'utiliser le nom complet d'un élément pour le référencer.
     - Attention aux conflits de noms.

Conception des paquetages

Diagramme de paquetages

Dépendence « access » et « import »



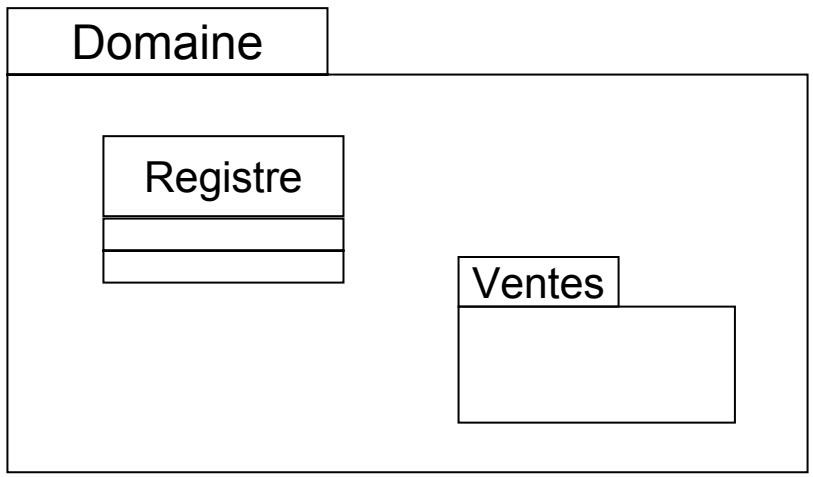


Dans ce cas, dans le package Ventes, *la classe* **Noyau::Registre** *peut être référencée simplement par* **Registre**

Conception des paquetages

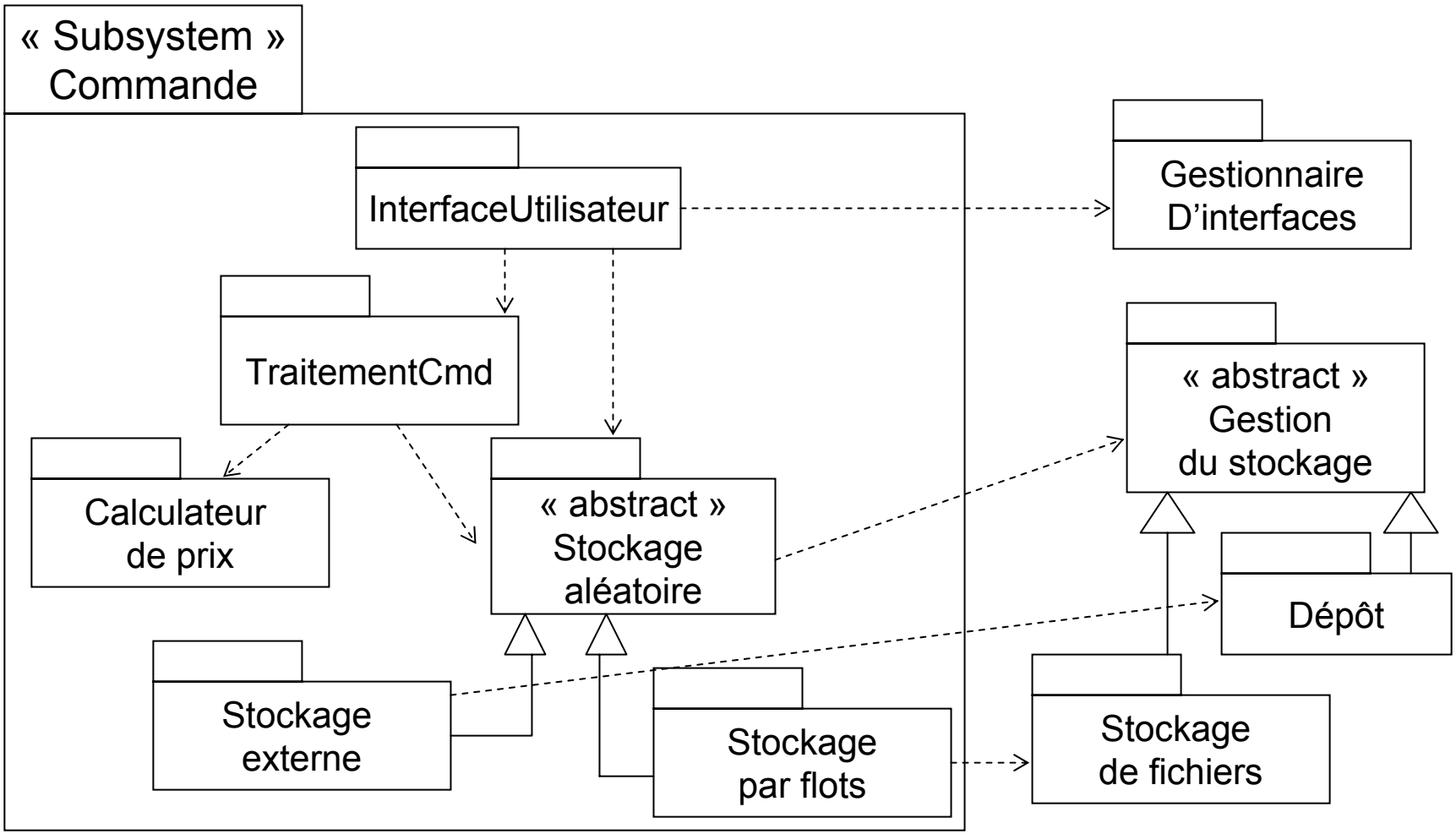
Diagramme de paquetages

- **Paquetages imbriqués** (cf. relation CONTIENT)
  - Un paquetage imbriqué est *implicitement* dépendant de tous les paquetages qui le contiennent et y a directement accès:
    - Sans égards aux contraintes de visibilité
    - Sans égards à la spécification d'une relation « import ».



Conception des paquetages

Diagramme de paquetages - exemple



Conception des paquetages

Diagramme de paquetages

- **Comment partitionner un ensemble d'éléments (classes, cas d'utilisation, etc.) en paquetages ?**

  Placer ensemble les éléments qui
  - Qui concernent le même sujet
  - Sont ensemble dans une hiérarchie de classes
  - Participent au même cas d'utilisation
  - Sont fortement liés

3.2.4 Conception des composants

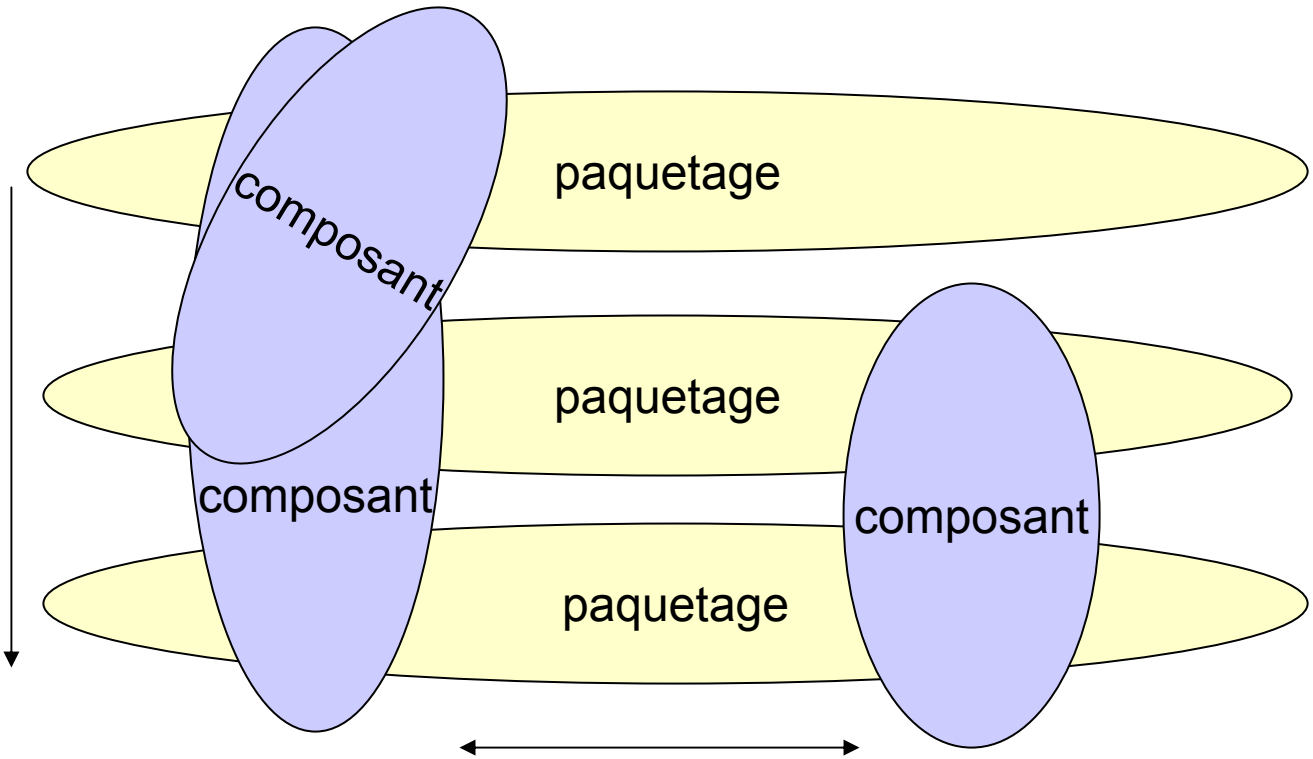
Composant

- Unité **physique** d'un système, une partie d'implémentation, un programme, un document compilé ou compilable (par une machine ou un humain).
- Unité logicielle **cohésive** qui fournit un ensemble de fonctionnalités. (cohésion fonctionnelle)
- Unité **indépendante** ayant une **interface bien définie** (contrat).
- Unité **réutilisable** et **remplaçable**.
- Unité pouvant s'**imbriquer** dans une autre.
- Unité atomique **instantiable** et **déployable en bloc** sur des nœuds.
- Unité à granularité variable:
  - à gros grain: utilisé pour représenter un sous-systèmes de haut niveau. Peu de dépendances.
  - à grain fin: utilisé pour représenter un groupe d'objets. Dépendances plus nombreuses. (ex. JavaBean).

Conception des composants

Composant vs paquetage

- **Regroupement physique =** proximité dynamique
- Une classe est implémentée par **au moins** un composant

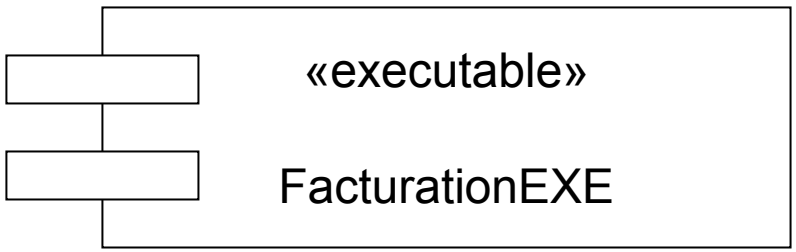


- **Regroupement logique**: proximité statique
- Une classe appartient à **un seul** paquetage

En général: taille paquetage > taille d'un composant

Conception des composants

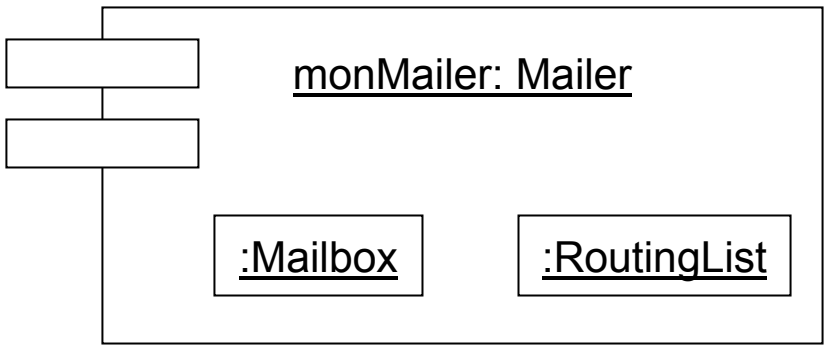
Composant



- Stéréotype prédéfinis en UML
  - « Executable » (Ex. module .exe)
  - « Library » (Ex. libraries d'objets liées statiquement ou dynamiquement)
  - « Tables » (table d'une base de données)
  - « File » (code source ou document de données)
  - « Document » (document lisible par un humain)

Conception des composants

Composant

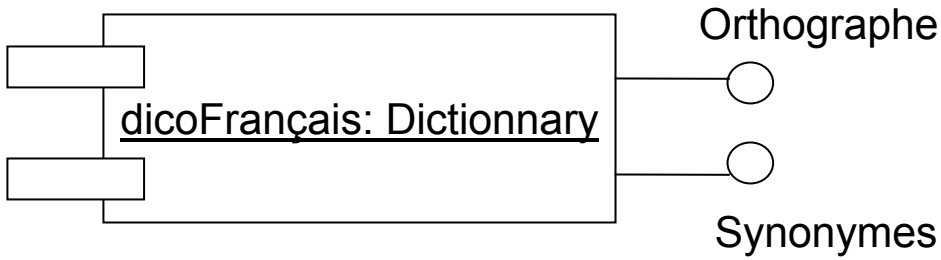


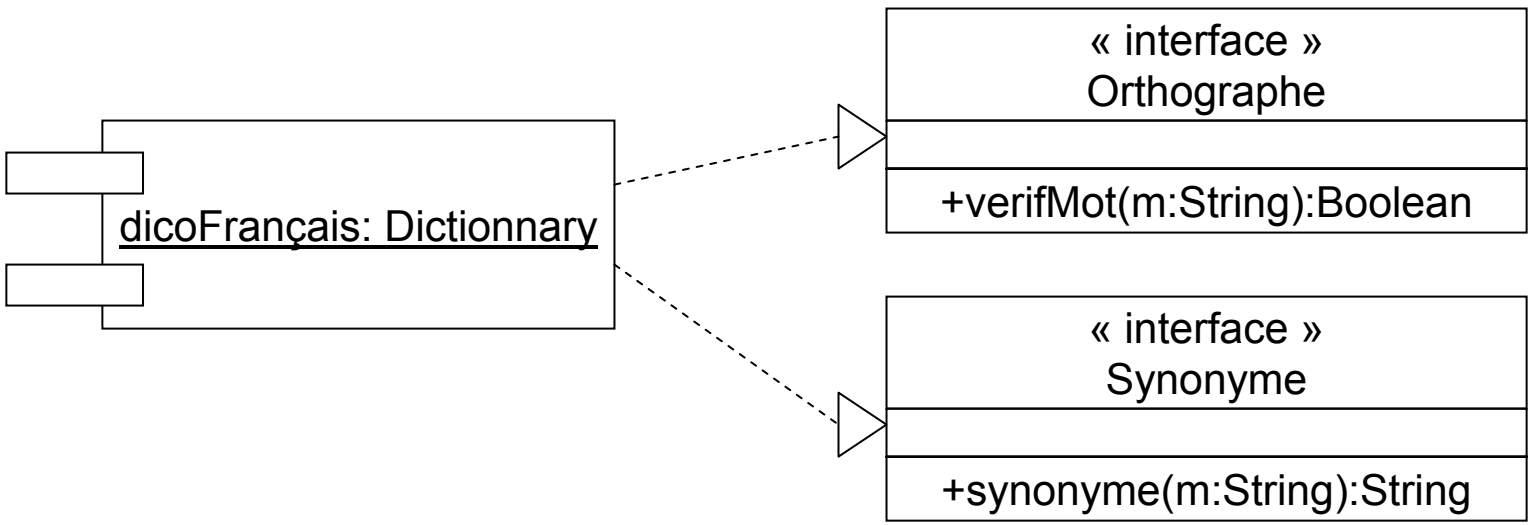
- Un composant est instanciable et peut avoir une identité.
  - L'état d'un composant est défini par l'état des objets physiques qui le compose.

Conception des composants

Composant - Interface

- Un composant réalise une ou plusieurs interfaces

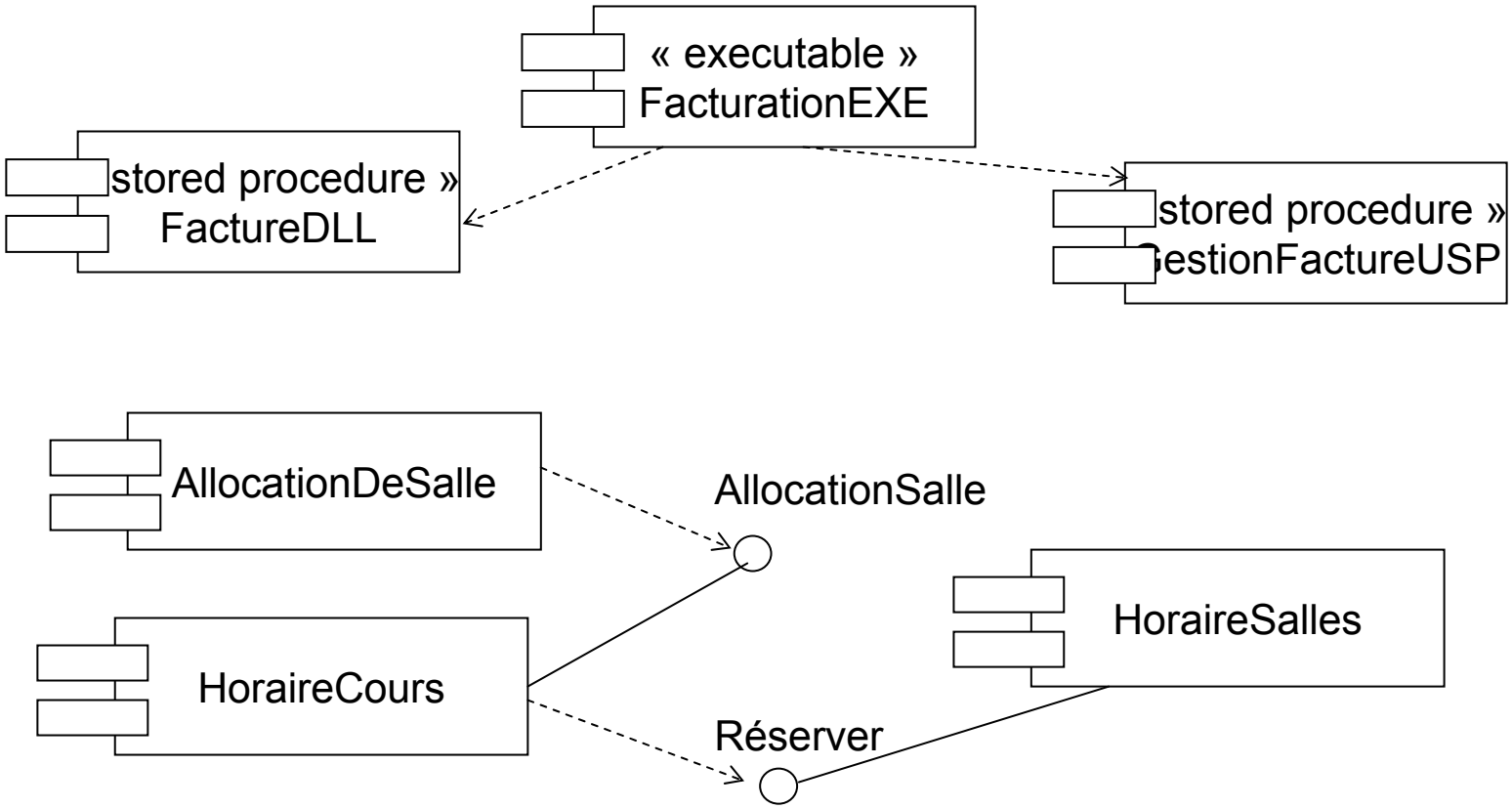




Conception des composants

Composant - Dépendance

- Un composant peut dépendre d'autres composants.



3.2.5 Conception du déploiement

Déploiement

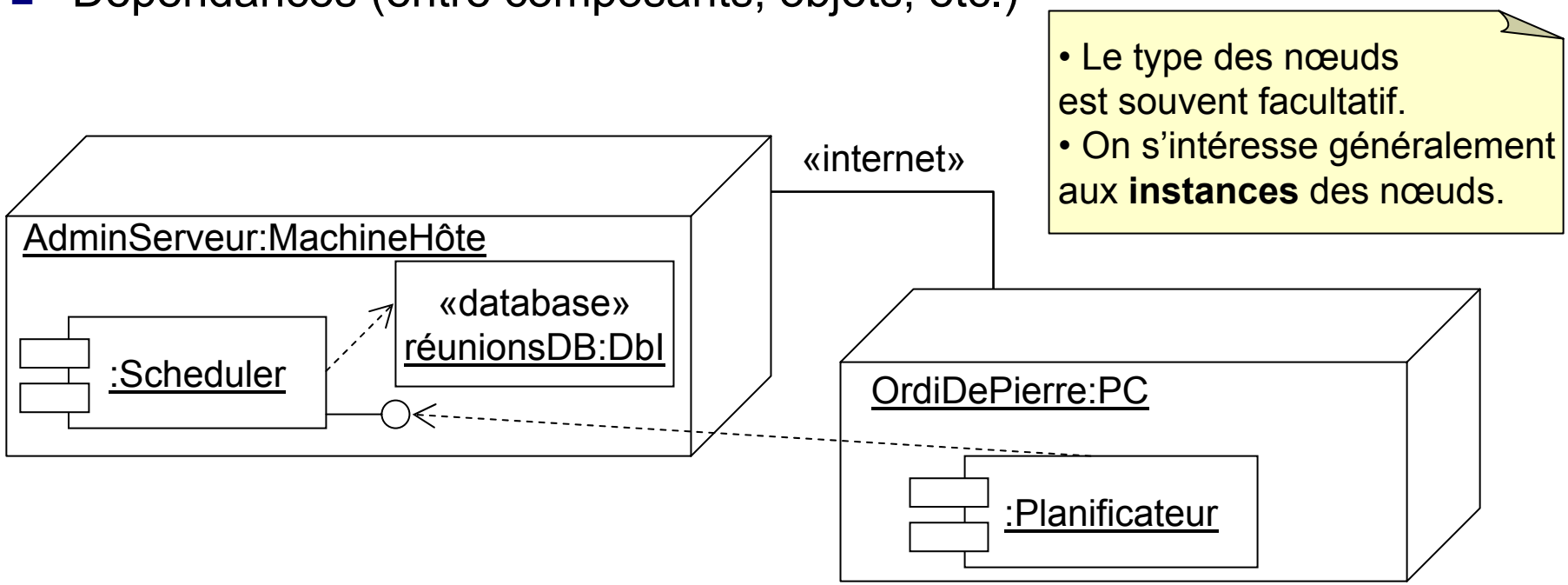
- Présente l'architecture d'un système et de son organisation physique en tenant compte des
  - **Nœud:** site d'implantation d'une ressource de calcul (informatique, mécanique ou humaine) qui disposent au minimum d'une mémoire et d'une capacité de traitement.
  - **Canal de communication:** association entre nœuds.
- Éléments de calcul distribués sur les nœuds:
  - instances des composants
  - objets exécutables
- Exemples de stéréotypes possibles pour les nœuds d'une application Internet: «WebServer», «ApplicationServer», «DataBaseServer»

Conception du déploiement

Déploiement

Élément d'un diagramme de déploiement:

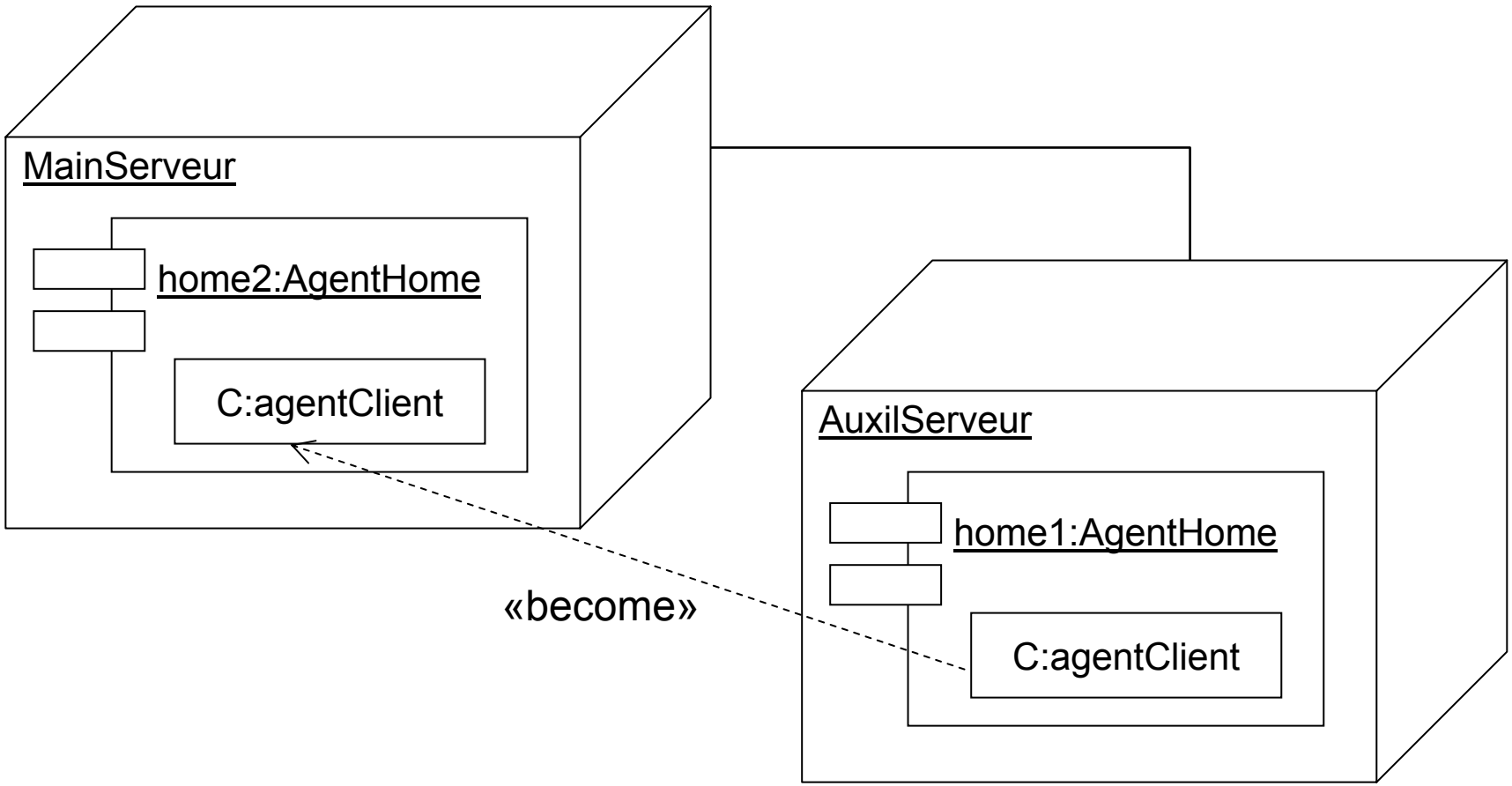
- Nœuds
- Liens de communications
- Instances de composants, interfaces, objets
- Dépendances (entre composants, objets, etc.)



Conception du déploiement

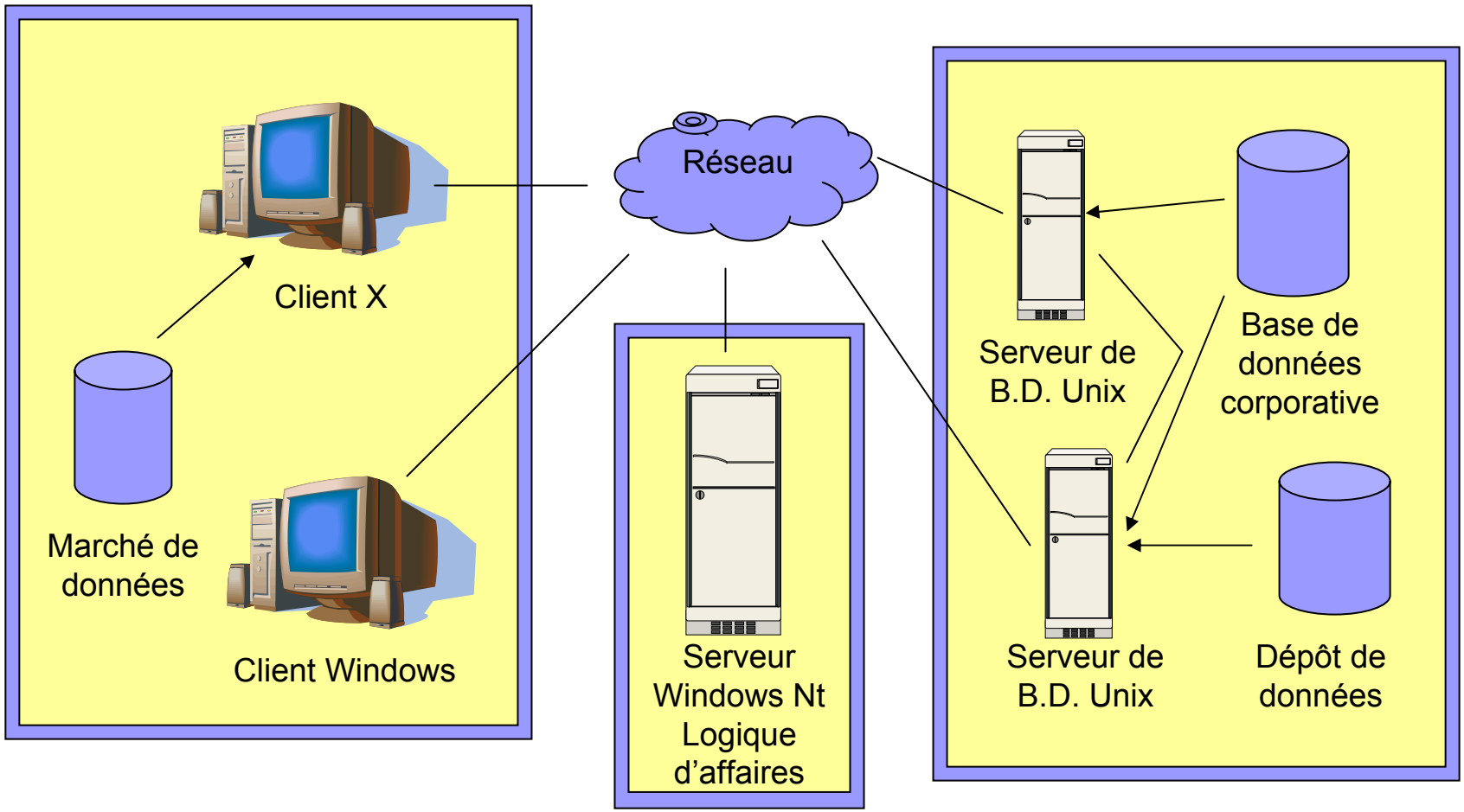
Déploiement

Migration



3.2.6 Design d'une architecture N- tiers

Architecture 3-tiers



Design d'une architecture N- tiers

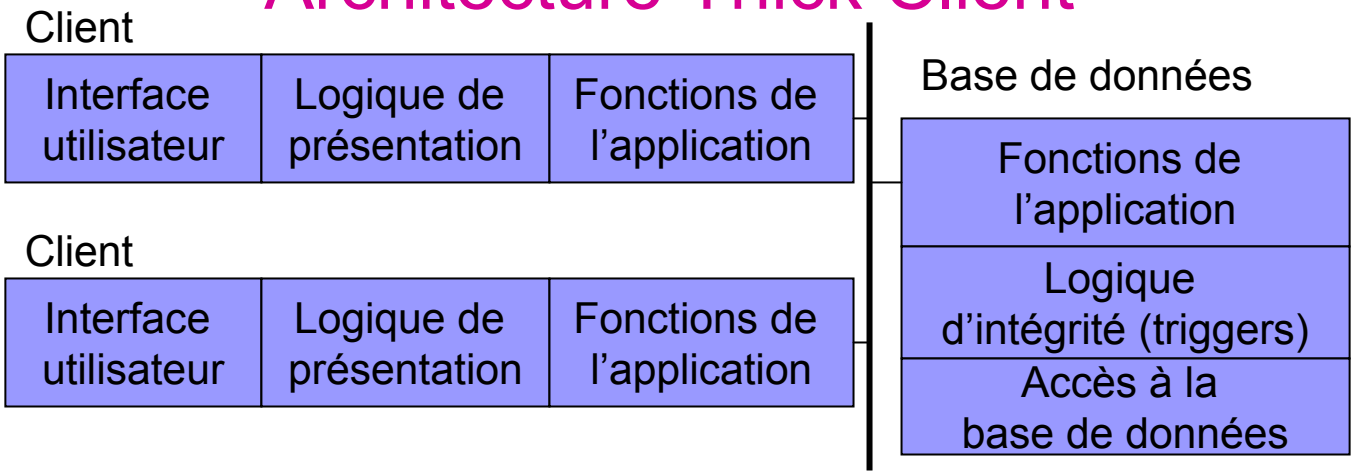
Interactions Application – Base de données

Quelles parties d'un système doit être déployées sur le client, sur le serveur et sur la base de données ?

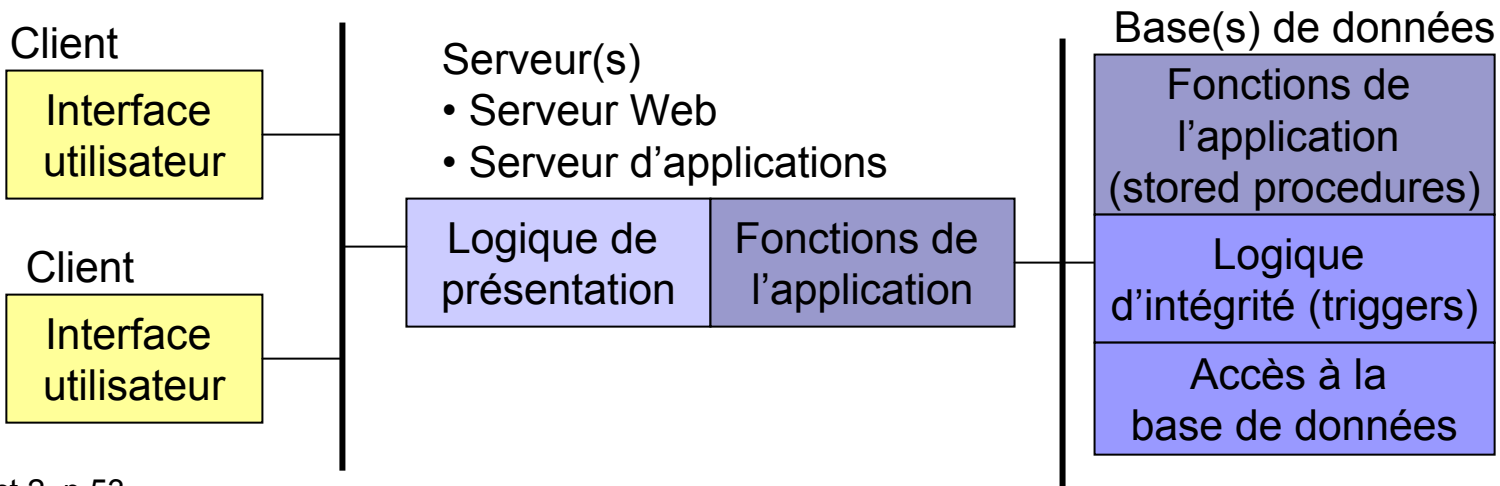
- Interface utilisateur:
  - Affichage graphique (GUI).
- Logique de présentation:
  - Gestion des GUI selon les fonctions de l'application.
- Fonctions de l'application
  - Logique qui décrit ce que l'application doit faire.
- Logique d'intégrité
  - Règles auxquelles doivent se conformer toutes les applications. Assurer la cohérence des opérations et l'intégrité des données en toute circonstance.
- Accès à la base de données
  - Fonctions d'accès aux données persistentes.

Design d'une architecture N- tiers

Architecture Thick Client

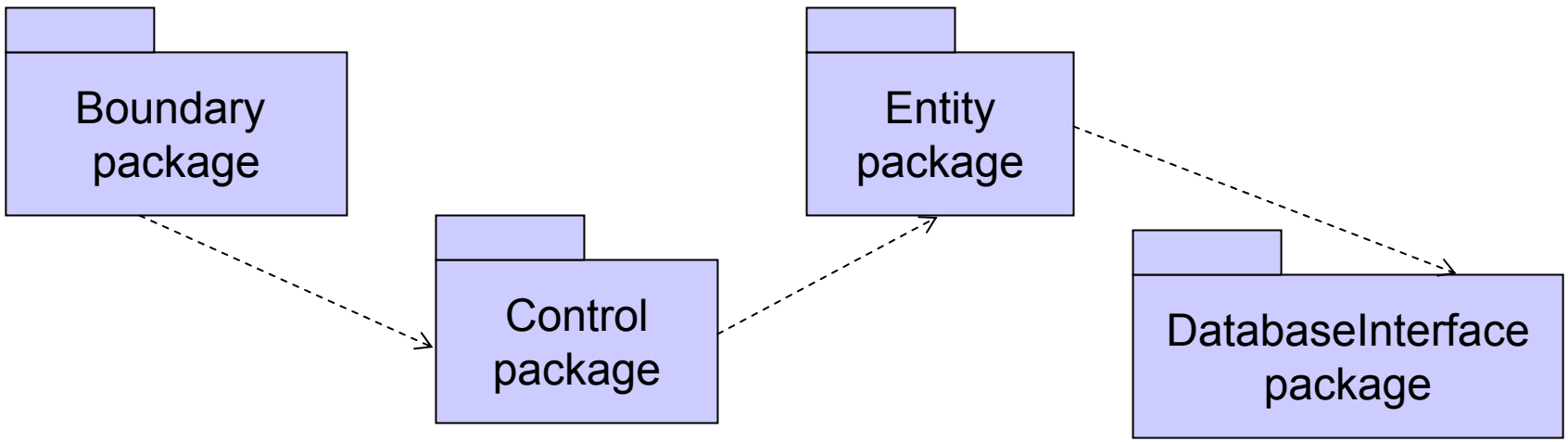


Architecture Thin Client



Design d'une architecture N- tiers

Approche de modélisation BCED

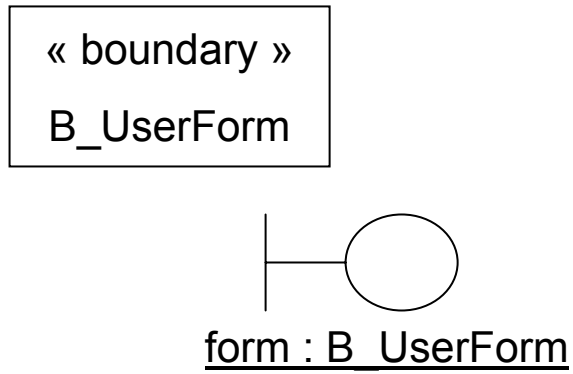


- S'inspire de l'approche MVC.
- Propose une factorisation des classes en 4 catégories.
- Prépare le terrain pour l'élaboration de composants se déployant naturellement une architecture N-tiers.

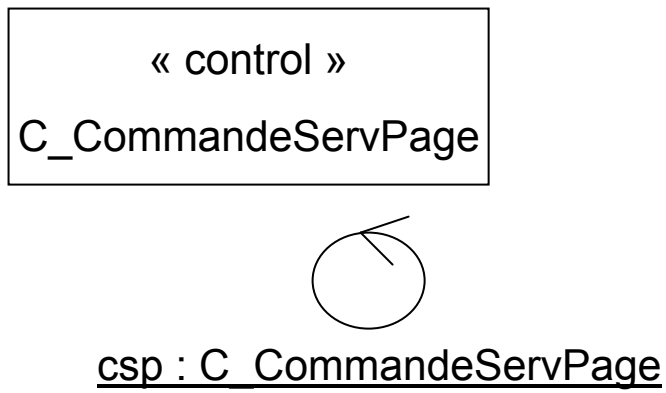
Design d'une architecture N- tiers

Approche de modélisation BCED

- **Classe boundary:**
  - Décrit les objets qui représente une interface entre un acteur et le système.
  - Permet de présenter une portion de l'état du système sous forme d'interface graphique, d'effets sonores, etc.
  - Nature généralement transitoire.



- **Classe control:**
  - Décrit les objets qui interceptent les événements et contrôle la logique de l'application.
  - Représente les actions et activités d'un cas d'utilisation.
  - Nature transitoire.



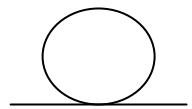
Design d'une architecture N- tiers

Approche de modélisation BCED

- **Classe entity :**
  - Décrit les objets qui représente la sémantique des entités dans un domaine d'application
  - Représente les structures de données la base de données du système.
  - Nature persistante.
- **Classe databaseInterface :**
  - Décrit les objets qui servent d'interface avec la base de données.
  - Implémente les opération de chargement et de sauvegarde des objets persistants
  - Offre d'autres services
    - Ouverture et de fermeture de connexion à la base de données. Validation/annulation de transactions. Configuration de la base de données, etc.

« entity »

E_compteBancaire



comptePaul : E_compteBancaire

« db interface »

D_DatabaseReader

Design d'une architecture N- tiers

Approche de modélisation BCED

Typiquement, le paquetage DatabaseInterface, sera composé de trois sous-paquetages contenant des classes permettant de gérer différentes opérations.

- **Paquetage CRUD** .(create – read – update – delete): Classes responsable de la création, lecture, mise à jour et destruction d'objets dans la base de données.
- **Paquetage Connexions**. Classes responsables de la fermeture/ouverture de connexions, de la gestion des transactions et des autorisations d'accès.)
- **Paquetage Schémas**: Classes responsables de la gestions des tables, colonnes, stored procedures de la base de données, etc.