Master Informatique

Génie Logiciel 2

178UD01

Cours n°1

Architectures logicielles et patron de conception

Plan du cours

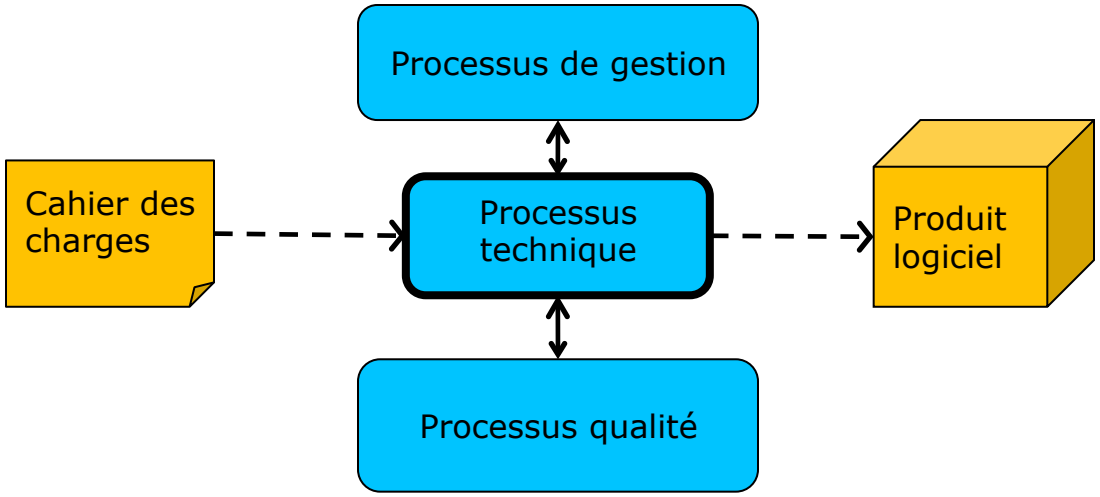
- Rappels sur les activités de développement
- La conception d'un produit logiciel
- Introduction aux architectures logicielles
- Quelques architectures connues
- Patrons de conception

Plan du cours

- Rappels sur les activités de développement
- La conception d'un produit logiciel
- Introduction aux architectures logicielles
- Quelques architectures connues
- Patrons de conception

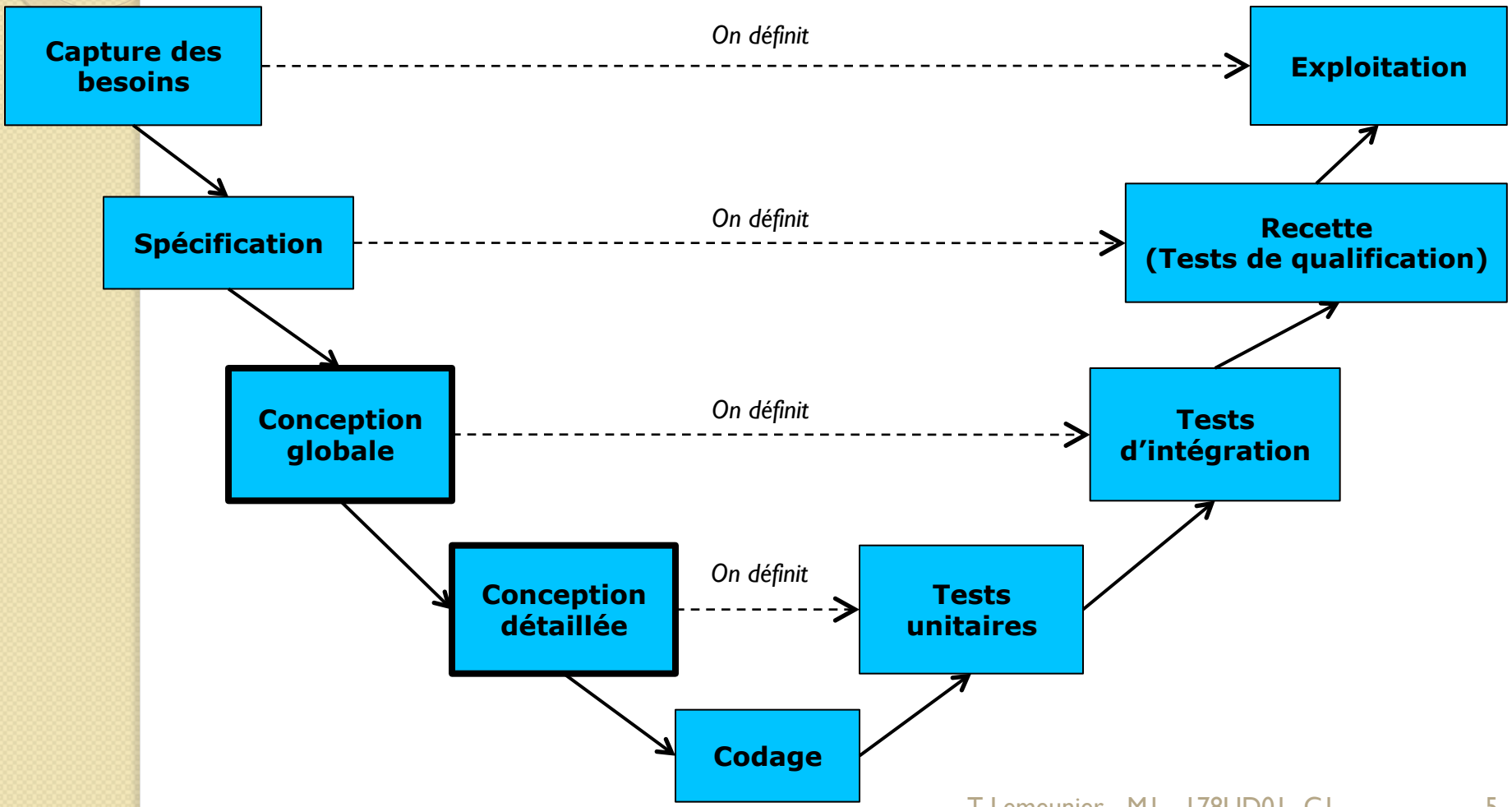
Rappel : les activités de développement (1/2)

- La production d'un produit logiciel s'appuie sur trois processus parallèles :
  - Le processus technique : développement
  - Le processus de gestion : gérer le développement
  - Le processus qualité : contrôler la qualité



Rappel : les activités de développement (2/2)

- Le cycle en V : un exemple de cycle de vie du processus technique



Plan du cours

- ✓ Rappels sur les activités de développement
- ➢ La conception d'un produit logiciel
- Introduction aux architectures logicielles
- Quelques architectures connues
- Patrons de conception

La conception en théorie

- La conception d'un produit logiciel :
  - Définir l'architecture logicielle
  - Définir le fonctionnement interne du produit
  - Définir le modèle des données (cf. cours BD)
- Mais les besoins changent :
  - Les besoins initiaux peuvent être incomplets
  - Les besoins peuvent comporter des erreurs
  - Les besoins sont souvent surestimés ou sous-estimés
  - Les besoins ne doivent pas être résolus dans une vision à court terme
- ➢ Nous devons écrire du code évolutif dans une architecture ouverte (c'est plus économique à long terme) ce qui est différent d'un code monolithique

La conception en pratique (1/3)

- Ecrire du code évolutif et ouvert :
  - Connaître/utiliser les différents types d'architectures
  - Connaître/utiliser des principes de conception et des règles de conception
- Dans le paradigme objet, la conception c'est :
  - Affecter des responsabilités à des classes en utilisant les principes et les règles
  - Les responsabilités sont de granularité variable :
    - la responsabilité de "*fournir un accès à des bases de données relationnelles*" peut nécessiter des dizaines de classes et des centaines de méthodes ;
    - la responsabilité "*créer une Vente*" ne demandera peut-être qu'une ou quelques méthodes.

La conception en pratique (2/3)

- Il y a deux types de responsabilités : faire et savoir
  - Faire :
    - Faire quelque chose par lui-même (créer un objet ou effectuer un calcul)
    - Déclencher une action d'un autre objet (déléguer)
    - Contrôler et coordonner les activités d'autres objets
  - Savoir :
    - Connaître les données privées encapsulées
    - Connaître les objets connexes
    - Connaître des éléments qu'il peut dériver ou calculer
- Exemples de principes :
  - Encapsuler les points de variations
  - Ecrire un code ouvert (à l'extension) et fermé (à la modification)
  - Faible couplage forte cohésion
  - Préférer la composition à l'héritage...
- Exemples de règles : patrons de conception (*design patterns*)
  - Les patterns GoF (Gang of Four)
  - Les patterns GRASP (proposés par Graig Larman)

La conception en pratique (3/3)

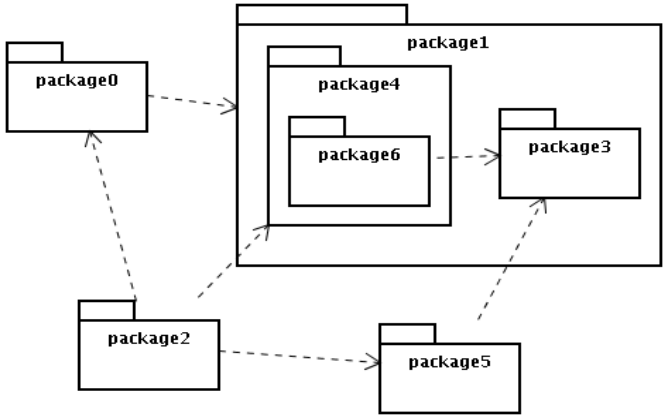
- Activité de conception :
  1. Définir l'architecture globale
  2. Concevoir des scénarios de plus en plus complets
     ① Partir des diagrammes de séquences niveau analyse
     ② Concevoir les opérations systèmes de ces diagrammes en s'aidant du modèle du domaine et des contrats d'opération
        - Construire des diagrammes d'interactions
        - En même temps que les diagrammes de classe
- Les classes de niveau conception ne sont pas des copies des classes de niveau analyse mais des classes logicielles
- On obtient le modèle de conception :
  - Diagrammes de classe plus ou moins détaillés
  - Diagrammes d'interactions

Plan du cours

- ✓ Rappels sur les activités de développement
- ✓ La conception d'un produit logiciel
- ➢ Introduction aux architectures logicielles
- Quelques architectures connues
- Patrons de conception

Intro. aux architectures logicielles (1/3)

- Une définition :



Architecture =

structuration du système à développer

en un ensemble de ***composantes***

ayant entre elles des ***relations*** bien définies

possédant certaines ***propriétés*** requises et

respectant certaines ***contraintes***

- Structuration : agencement des modules liés par des relations
- Composantes : notion de modules plus ou moins indépendants
- Relations : utilise / appelle / inclut / importe – exporte…
- Propriétés : validité / sécurité / performance…
- Contraintes : localisation / taille / organisationnel…

Intro. aux architectures logicielles (2/3)

- Qualités recherchées d'une architecture :
  - Une grande ***cohésion*** interne de chaque module :
    - Les éléments du module contribuent à un objectif commun
  - Un faible ***couplage*** entre les modules
    - Le moins d'interdépendance possible entre modules
    - Souhaitable pour l'adaptation au changement et la réutilisabilité
  - Être compréhensible et bien définie (*claire*)
  - Si possible être généralisable (*pour obtenir un framework*) ou issu d'une spécialisation (*à partir d'un framework*)
  - Être juste : permettre de répondre aux besoins fonctionnels et techniques
  - Être élégante et/ou harmonieuse (*critère subjectif*)
- Définition de *framework* : architecture générique et partielle qui fournit un template extensif dans un domaine particulier. Généralement, les éléments y sont modifiés, spécialisés et étendus pour s'adapter à l'architecture d'un système spécifique.

Intro. aux architectures logicielles (3/3)

- Choisir une architecture :
  - Appliquer une architecture finie standardisée :
    - Concevoir consiste à utiliser le standard défini par le standard
    - Par exemple : standard JEE pour les applications Java distribuées
  - Utiliser ou combiner plusieurs patterns d'architectures :
    - Décomposition fonctionnelle
    - Décomposition par les flux de données
    - Décomposition par niveaux (ou couches)
    - Décomposition par la distribution
      - Modèle du dépôt
      - Modèle client-serveur
    - Décomposition par composants
    - Par exemple : décomposition par niveaux et par composants distribués

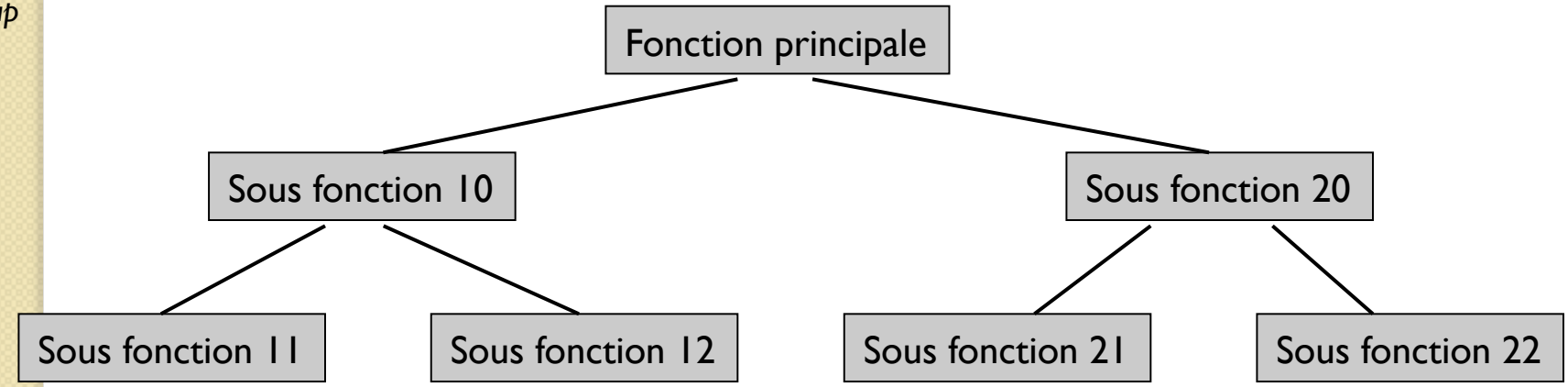
Plan du cours

- ✓ Rappels sur les activités de développement
- ✓ La conception d'un produit logiciel
- ✓ Introduction aux architectures logicielles
- ➢ Quelques architectures connues
- • Patrons de conception

Décomposition fonctionnelle (1/2)

- Approche descendante par décomposition (Exemple : SADT)
  - Stratégie classique de raffinements successifs
  - Décomposition répétée : problème $\Rightarrow$ sous problèmes
  - Arbre de modules avec une relation « se décompose en »
- Approche ascendante par composition
  - Arbre construit par le bas
  - Identification de modules qui résolvent des parties du problème puis composition de ces modules
- Approches mixtes

Bottom-up



Top-down

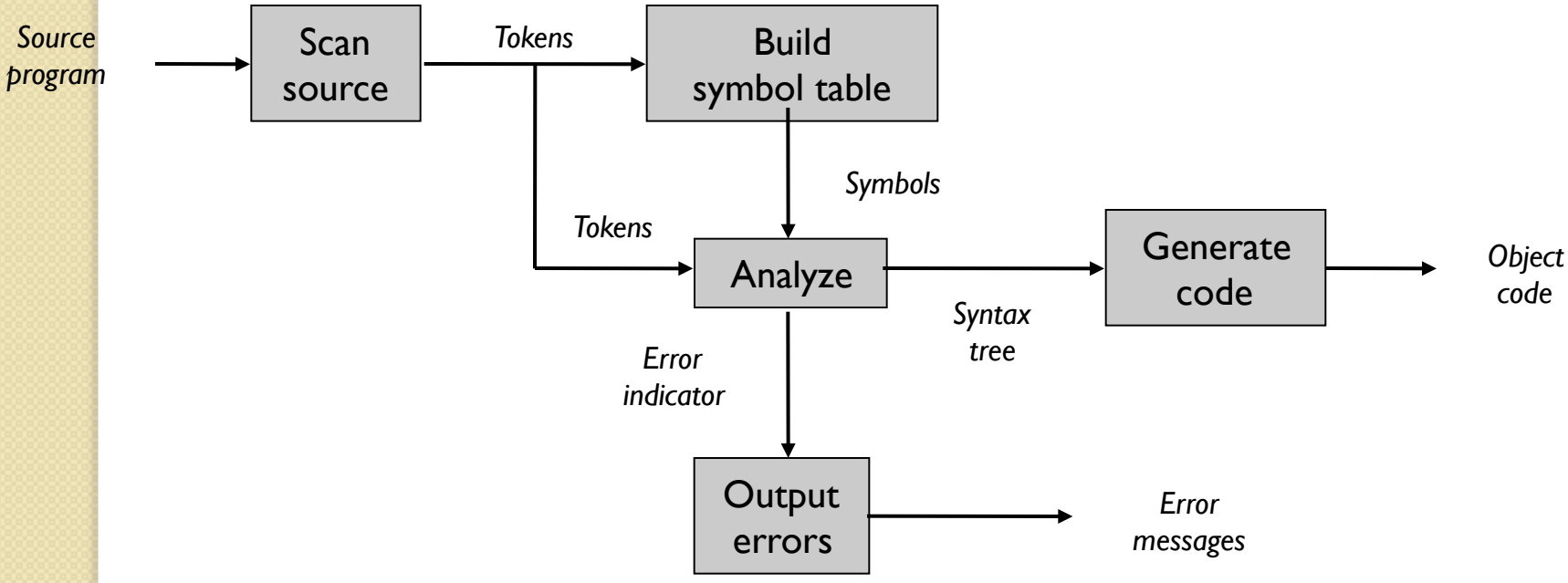
Décomposition fonctionnelle (2/2)

- Avantages :
  - Arbre de décomposition facile à comprendre
  - Efficace pour des petits problèmes qui ne vont pas évoluer
  - Approche « mécanique » bien rodée

- Inconvénients :
  - Modules à faible niveau d'abstraction
    (*modules définis pour le problème donné*)
    ⇒ peu ou pas de réutilisabilité des modules
  - Décomposition algorithmique procédurale
    ⇒ choix prématuré des structures de contrôle
  - Partage des structures de données
    (*définies dans le module de plus haut niveau ou globales*)
    ⇒ couplage très important (*peu ou pas de réutilisabilité*)

Décomposition par flux de données (1/2)

- Principe :
  - Décomposition à partir d'un diagramme de flux de données (traitements successifs des données)
  - Peut donner deux types de décomposition :
    - Décomposition par les données
    - Décomposition par les tâches/traitements



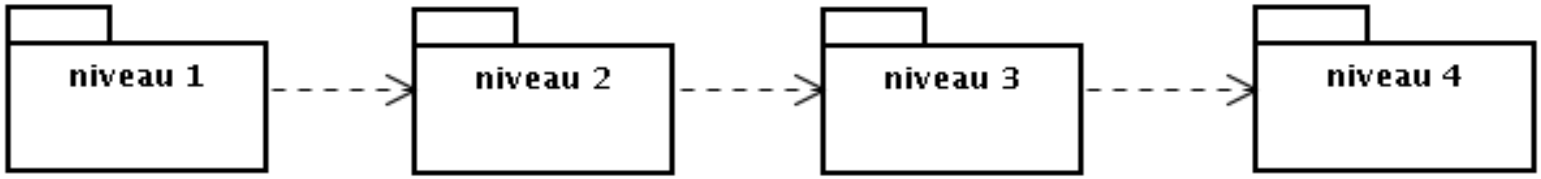
Décomposition par flux de données (2/2)

- Avantages :
  - A utiliser si peu d'abstraction possible (*cas d'une application très spécifique et/ou unique*)
  - A utiliser dans le cas d'une application qui ne doit pas évoluer
  - Architecture permettant la performance maximale
  - Exemples typiques : décodeurs audio et décodeurs vidéo

- Inconvénients :
  - Chaque module correspond à une étape de l'algorithme général de traitement ⇒ si l'algorithme change…
  - Peu d'autonomie des modules ⇒ fort couplage

Décomposition par niveaux (1/2)

- Principe :
  - Une série de couches (niveaux) offrant chacune un ensemble de services à sa couche supérieure :
    - Une couche ne communique qu'avec ses 2 voisines immédiates
    - Une couche délègue à la couche inférieure le travail qui ne la concerne pas
    - Une couche retourne à la couche supérieure son résultat
  - L'ensemble des modules d'une couche (niveau) correspond à un niveau d'abstraction (*qui cache des détails par rapport aux couches supérieures*)



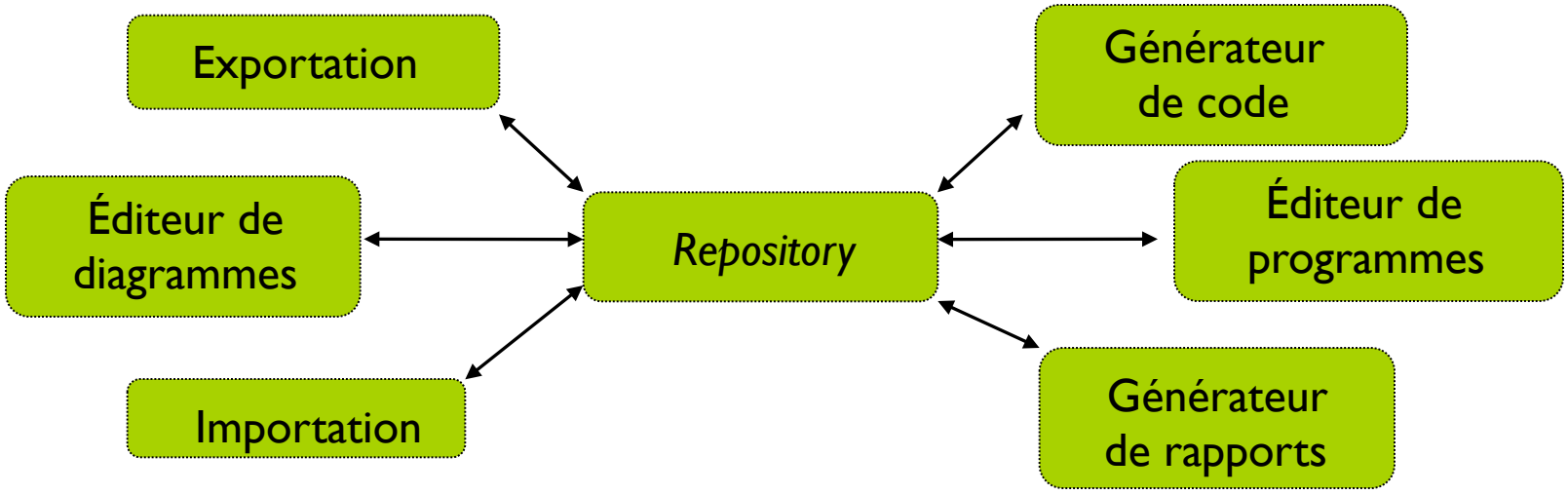
Décomposition par niveaux (2/2)

- Exemple : les 7 couches OSI pour les réseaux de communication

Niveau 7 : Application	Communication entre services
Niveau 6 : Présentation	Transformation de formats
Niveau 5 : Session	Session de communication
Niveau 4 : Transport	Processus communicants (adresses)
Niveau 3 : Réseau	Groupe (méthode de routage)
Niveau 2 : Data	Bits regroupés avec entête
Niveau 1 : Physique	Câble – (hardware)

Décomposition par distribution (1/6)

- Modèle du dépôt (*repository*)
  - Principe :
    - Les données sont partagées et se trouvent dans une BD centralisée qui gère les accès, la sécurité, l'archivage, etc. des données
    - Le reste du système se compose d'un ensemble de sous-systèmes actifs quasi-autonomes accédants aux données centralisées
  - Remarque : architecture typique des outils de travail collectif



Décomposition par distribution (2/6)

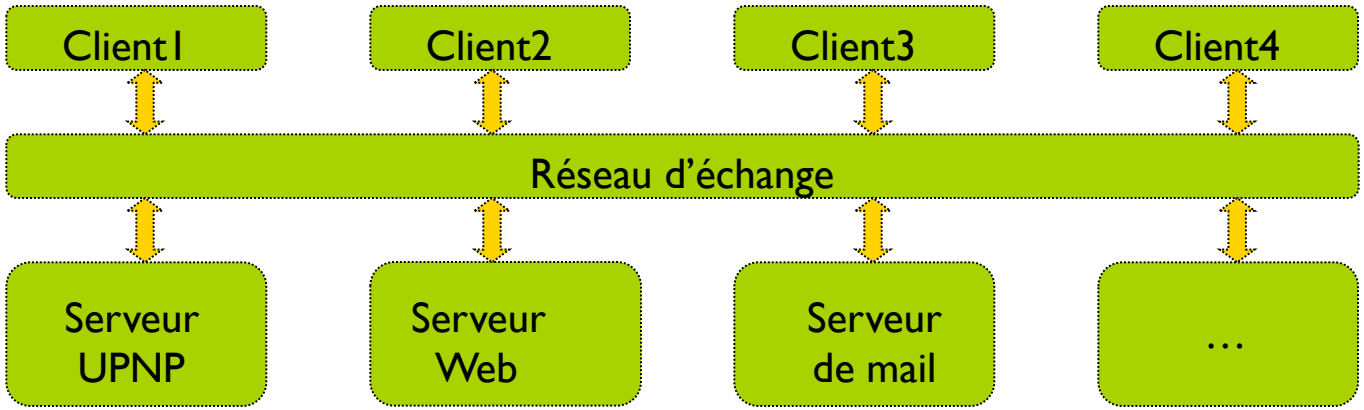
- Modèle du dépôt (suite)
  - Avantages :
    - Simple à comprendre
    - Centralisation des données :
      - Sécurité / maintenance / correction d'erreurs / backup
      - Partage des données entre utilisateurs
    - Efficace pour de grosses quantités de données
    - Quasi indépendances des sous-systèmes (remplacement « standard » ou ajout d'un outil possibles)
  - Inconvénients :
    - Dépendance des modules externes par rapport au dépôt central (*cas de crash, indisponibilité temporaire, etc.*)
    - Fonctionnement figée car dépendant du nœud centrale (*impossibilité de copie locale sous peine d'incohérence ou de perte ultérieur*)

Décomposition par distribution (3/6)

- Modèle client-serveur
  - Principe général :
    - Serveurs autonomes fournissant des services particuliers (*généralement distribués physiquement*)
    - Clients faisant appel à ces services (*on peut avoir plusieurs instances du même type de client*)
    - Composante(s) d'échange : réseau ou une machine (communication par fichier, par mémoire partagée, par pipe sous unix, etc.)
  - Avantages / inconvénients :

    +++ distribution favorisée

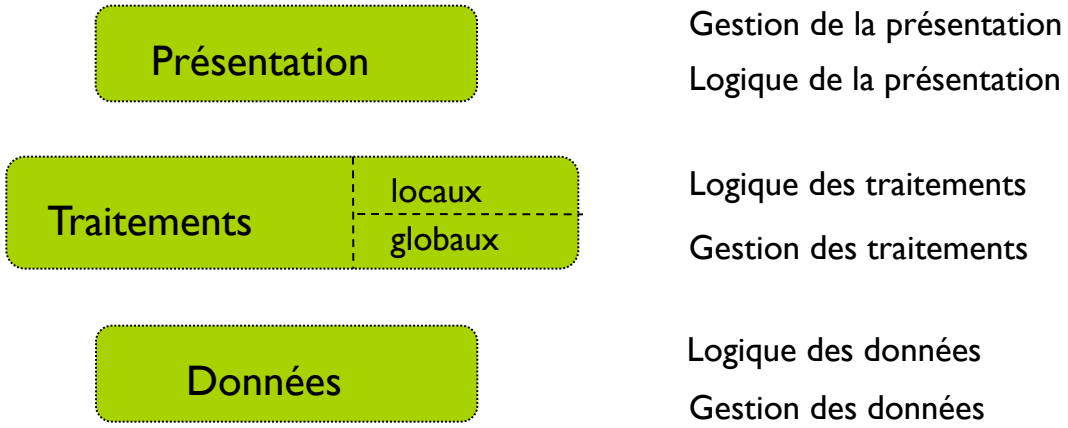
    - certains systèmes gèrent leurs données ⇒ risque de redondance



Décomposition par distribution (4/6)

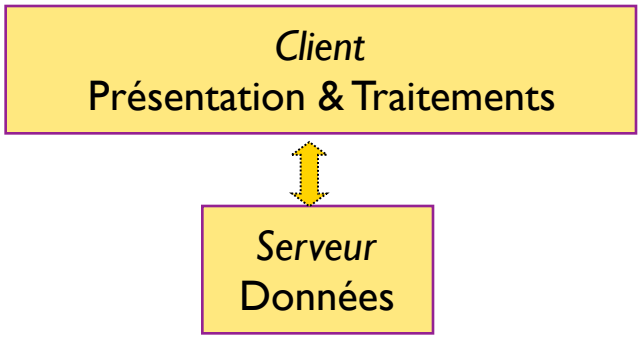
- Fondement « théorique » de l'architecture client-serveur :

  Une application peut se découper en trois niveaux logiques :
  - La *couche présentation* : toute interaction des utilisateurs avec les applications
  - La *couche logique applicative* : les traitements métiers
    - Les traitements locaux
      (le contrôle et l'aide à la saisie respectant la logique applicative)
    - Les traitements globaux
      (couche métier ou *business logic*, règles internes qui régissent l'application)
  - La *couche données* : ensemble des mécanismes permettant la gestion des informations stockées par l'application

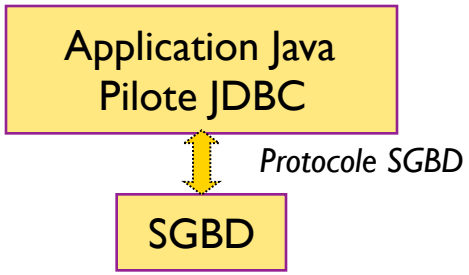


Décomposition par distribution (5/6)

- Modèle 2 tiers / 3 tiers / n tiers
  - Architecture 2 tiers (ou *client lourd ou épais*)
    - Client : couche présentation et couche traitements
    - Serveur : couche données
  - Avantages :
    - Utilisation d'une IHM utilisateur riche (*avec toutes les conséquence pour l'utilisateur*)
    - A permis d'introduire la notion d'interopérabilité
  - Inconvénients :
    - Charge du client importante
    - Demande une bande passante large entre *Client* et *Serveur*



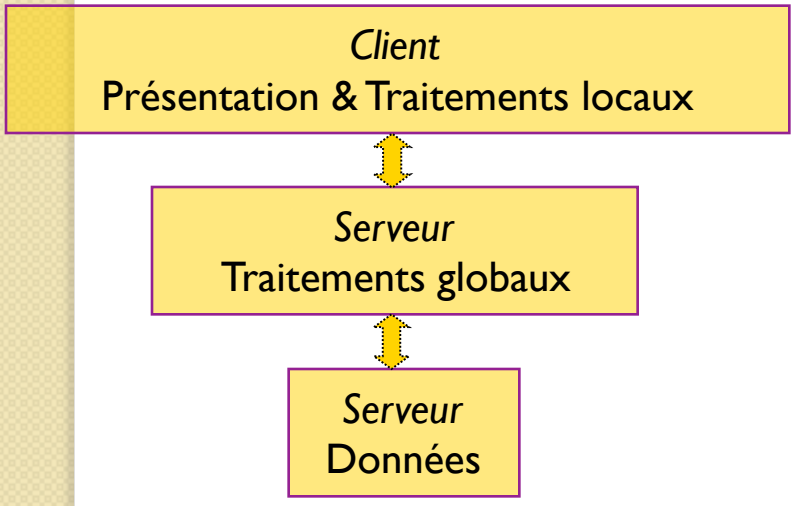
Architecture logique



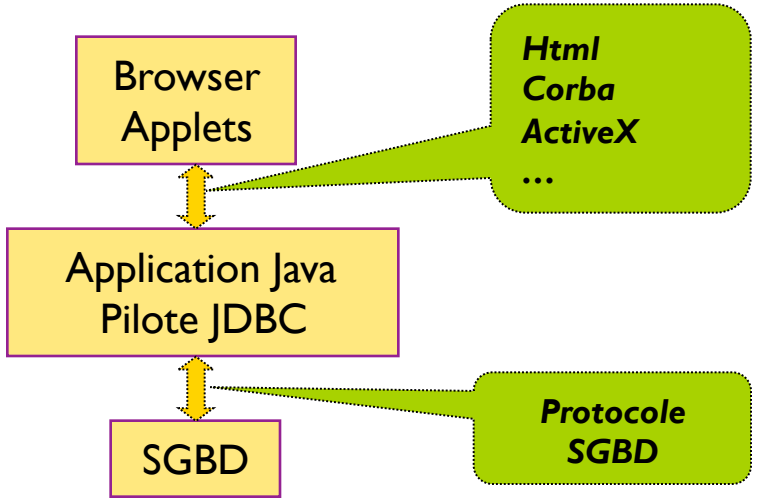
Architecture physique possible

Décomposition par distribution (6/6)

- Modèle 2 tiers / 3 tiers / n tiers (suite)
  - Architecture 3 tiers (ou *client léger*) :
    - Serveur de BD : couche données
    - Serveur : traitements globaux
    - Client : couche présentation et traitements locaux
  - Architecture n tiers : *traitements* globaux eux-mêmes multiples



Architecture logique



Architecture physique possible

Décomposition par composants

- Principes :
  - Le **composant** :
    - Une "brique" logicielle prestataire d'un ou plusieurs services
    - Un composant possède :
      - une ou plusieurs interfaces
      - un protocole de communication (distante ou locale)
      - une documentation complète d'utilisation

  - Deux composants proposant le même service doivent être interchangeables « rapidement »
  - Les composants sont installés dans un **serveur d'application** qui prend en charge un maximum de services techniques (sécurité, transaction, connexion aux ressources, etc.)

- Architecture :
  - Assemblage de composants
  - Architecture très modulable, évolutive et facilement maintenable
  - S'emploie avec une décomposition par niveau et/ou par distribution

- Exemples de technologies orientées composants :
  - Spécification *EJB* de la plateforme *Java EE* de Oracle
  - Le *framework WCF* de la plateforme *.Net* 3.0 de Microsoft

Plan du cours

- ✓ Rappels sur les activités de développement
- ✓ La conception d'un produit logiciel
- ✓ Introduction aux architectures logicielles
- ✓ Quelques architectures connues
- ➢ Patrons de conception

Intro. aux patrons de conception (1/3)

- Définition :
  - Un pattern décrit un problème devant être résolu, une solution et le contexte dans lequel cette solution est considérée
  - Il nomme une technique et décrit ses coûts et ses avantages
  - Il permet à une équipe de mettre un vocabulaire en commun pour parler des modèles de conception
- Généralement, quand on parle de design patterns, on fait référence au catalogue proposé par la bande des quatre Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1995)
- Ce catalogue :
  - fournit 23 patterns de conception organisés selon leur rôle et selon qu'ils s'appliquent sur des objets ou des classes
  - fournit également des liens entre patterns : telle application d'un pattern est lié ou permet d'utiliser telle(s) autre(s) pattern(s)

Intro. aux patrons de conception (2/3)

	Créateurs	Structuraux	Comportementaux
Classe	*Factory Method*	*Adapter* (class)	*Interpreter* *Template Method*
Objet	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Adapter* (object) *Bridge* *Composite* *Decorator* *Facade* *Flyweight* *Proxy*	*Chain of Responsability* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

- Les patterns créateurs portent sur le processus d'instanciation
- Les patterns structuraux portent sur l'organisation des classes et des objets
- Les patterns comportementaux portent sur les interactions entre objets et la répartition des responsabilités

Intro. aux patrons de conception (3/3)

- A quoi sert les patrons de conception ?
  - La plus part des patrons de conception permettent un code plus évolutif
  - Utiliser et réutiliser des solutions éprouvées
    - Donne un début de solution
    - Bénéficier de l'expérience des « aînés »
    - Ne pas réinventer des solutions pour des problèmes récurrents
  - Établir une terminologie commune
    - Langage commun pour le travail d'équipe
    - Point de vue commun sur un problème
  - Avoir un point de vue de haut niveau de la conception sans rentrer tout de suite dans les détails
  - En général, ce sont des outils de réflexion même si vous n'allez pas y trouver « la solution » toute faite