Chapitre 7 :
Communication

- Middlewares
- Modèles d'interaction
- Architecture client/serveur

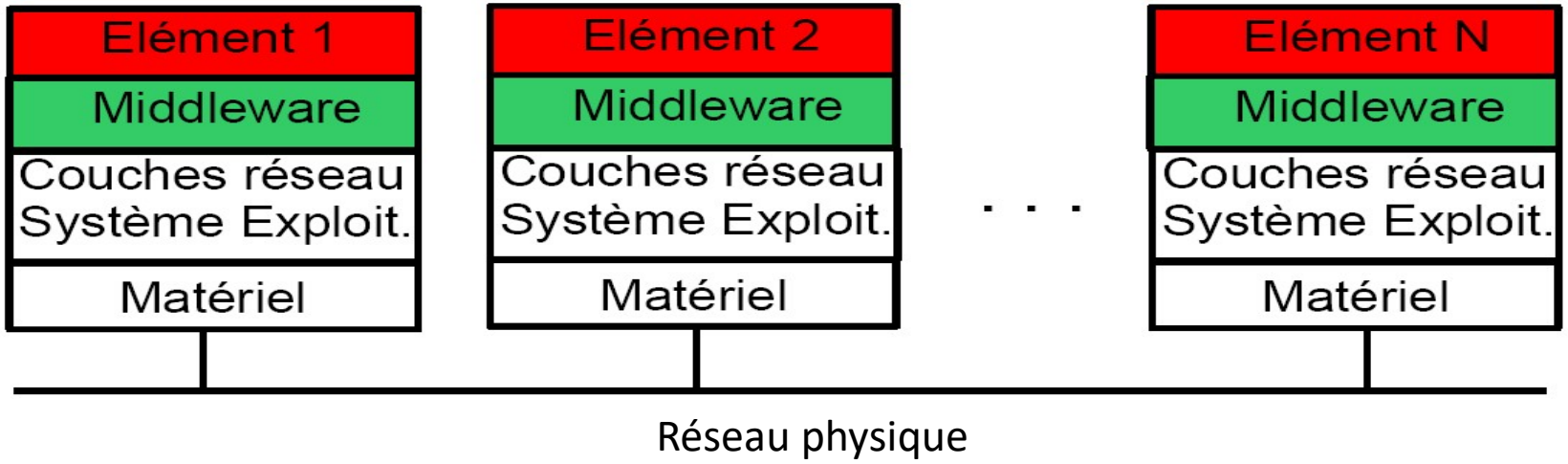
Communication

- Système distribué

  Ensemble d'entités logicielles communiquant entre-elles
- Entités logicielles s'exécutent sur des machines reliées entre elles par un réseau
- Communication entre entités logicielles
  - Le plus basique : directement en appelant les services des couches TCP ou UDP
  - Plus haut niveau : définition de couches offrant des services plus complexes
    - Couche réalisée en s'appuyant sur les couches TCP/UDP
    - Exemple de service : appel d'une procédure chez une entité distante
    - Notion de middleware (intergiciel)

Middleware

- Middleware ou intergiciel : couche logiciel
  - S'intercale entre le système d'exploitation/réseau et les éléments de l'application distribuée
  - Offre un ou plusieurs services de communication entre les éléments formant l'application ou le système distribué
    - Services de plus haut niveau que les communications via sockets TCP/UDP, augmentation du niveau d'abstraction



Familles de middlewares

- Plusieurs grandes familles de middleware
  - Appel de procédure/méthode à distance
    - Extension « naturelle » de l'appel local d'opération dans le contexte des systèmes distribués
      - Une partie serveur offre une opération appelée par une partie client
    - Permet d'appeler une procédure/méthode sur un élément/objet distant
    - Exemples (dans l'ordre historique d'apparition)
      - RPC (Remote Procedure Call) : solution de Sun pour C/Unix
      - CORBA (Common Object Request Broker Architecture) : standard de l'OMG permettant l'interopérabilité quelque soit le langage ou le système
      - Java RMI (Remote Method Invocation) : solution native de Java
  - Envoi ou diffusion de messages ou événements
    - Famille des MOM (Message Oriented Middleware)
    - Event Service de CORBA, JMS de Java
  - Mémoire partagée
    - Accès à une mémoire commune distribuée
    - JavaSpace de Java

Caractéristiques de middlewares

- Modèle RPC/RMI
  - Interaction forte entre parties client et serveur
    - Appel synchrone (pour le client) d'opération sur le serveur
    - Généralement, le client doit explicitement connaître le serveur
    - Peu dynamique (point de vue serveur) car les éléments serveurs doivent être connus par les clients et lancés avant eux
  - Mode 1 vers 1 : opération appelée sur un seul serveur à la fois
- Modèles à message ou mémoire partagée
  - Asynchrone et pas d'interaction forte entre éléments
    - Envoi de message est asynchrone
    - Pas de nécessité de connaître les éléments accédant à la mémoire
    - Permet une plus forte dynamicité : ajout et disparation d'éléments connectés au middleware facilités par les faibles interactions et l'anonymat
  - Mode 1 vers n
    - Diffusion de messages à plusieurs éléments en même temps
    - Accès aux informations de la mémoire par plusieurs éléments

But et fonctionnalités d'un middleware

- Gestion de l'hétérogénéité
  - Langage de programmation, systèmes d'exploitation utilisés ...
  - CORBA par exemple : fait interopérer via du RPC/RMI des éléments logiciels écrits dans n'importe quel langage
- Transparence à la localisation
  - Accès à des objets/ressources distantes aussi facilement que localement et indépendamment de leur localisation
- Offrir des abstractions de communication de plus haut niveau
  - Appel d'une procédure à distance sur un élément
  - Communication via une mémoire partagée
  - Diffusion d'événements
  - ...
- Offrir des services de configuration et de gestion du système
  - Service d'annuaire pour connaître les éléments présents
  - Services de persistance, de temps, de transaction, de sécurité ...

Service de nommage (d'annuaire)

- Cas des middlewares RPC/RMI
  - Service d'un élément = (interfaces d') opération(s) offerte(s) par un élément et pouvant être appelée(s) par d'autres
  - Un élément enregistre les opérations qu'il offre
  - Recherche d'une opération ou d'un élément, 2 modes
    - On demande à recevoir une référence sur un élément qui offre une opération compatible avec celle que l'on cherche
    - On sait identifier l'élément distant et on demande à récupérer la référence sur cet élément
  - Une fois la référence sur l'élément distant connu, on peut appeler l'opération sur cet élément via cette référence

Modèles d'interaction

- Les éléments distribués interagissent, communiquent entre eux selon plusieurs modèles possibles
  - Client/serveur
  - Diffusion de messages
  - Mémoire partagée
  - Pair à pair
  - ...
- Abstraction/primitive de communication basique
  - Envoi de message d'un élément vers un autre élément
  - A partir d'envois de messages, peut construire les protocoles de communication correspondant à un modèle d'interaction

Modèles d'interaction

- Rôle des messages
  - Données échangées entre les éléments
    - Demande de requête
    - Résultat d'une requête
    - Donnée de toute nature
  - Gestion, contrôle des protocoles
    - Acquittement : message bien reçu
    - Synchronisation, coordination ...

Modèle client/serveur

- Deux rôles distincts
  - Client : demande que des requêtes ou des services lui soient rendus
  - Serveur : répond aux requêtes des clients
- Interaction
  - Message du client vers le serveur pour faire une requête
  - Exécution d'un traitement par le serveur pour répondre à la requête
  - Message du serveur vers le client avec le résultat de la requête
- Exemple : serveur Web
  - Client : navigateur Web de l'utilisateur
  - Requêtes : récupérer le contenu d'une page HTML gérée ou générée par le serveur

Modèle client/serveur

- Modèle le plus répandu
  - Fonctionnement simple
  - Abstraction de l'appel d'un service : proche de l'appel d'une opération sur un élément logiciel
    - Interaction de base en programmation
- Particularités du modèle
  - Liens forts entre le client et le serveur
  - Un client peut aussi jouer le rôle de serveur (et vice-versa) dans une autre interaction
  - Nécessité généralement pour le client de connaître précisément le serveur (sa localisation)
    - Ex : URL du site Web
  - Interaction de type « 1 vers 1 »
    - 1 client communique avec 1 serveur à un moment donné

Diffusion de messages

- Deux rôles distincts
  - Émetteur : envoie des messages (ou événements) à destination de plusieurs récepteurs
    - Diffusion (broadcast) : à tous ceux qui sont présents
    - A un sous-ensemble de récepteurs : multicast
  - Récepteurs : reçoivent les messages envoyés
  - Peut être à la fois émetteur et récepteur
- Interaction
  - Émetteur envoie un message
  - Le middleware s'occupe de transmettre ce message à chaque récepteur

Diffusion de messages

- Deux modes de réception
  - Le récepteur va vérifier lui-même qu'il a reçu un message (pull)
    - Boîte aux lettres
  - Le récepteur est prévenu que le message est disponible et il lui est transmis (push)
    - Le facteur sonne à la porte pour remettre en main propre le courrier
- Particularités du modèle
  - Dépendance plus faible entre les participants
    - Pas besoin pour l'émetteur d'être directement connecté aux récepteurs ni même de savoir combien ils sont
  - Interaction de type « 1 vers N »

Mémoire partagée

- Les éléments communiquent via une mémoire partagée à l'aide d'une interface d'accès à la mémoire
  - Ajout d'une donnée à la mémoire
  - Lecture d'une donnée dans la mémoire
  - Retrait d'une donnée de la mémoire
- Le middleware gère l'accès à la mémoire pour chacun des participants
- Particularité du modèle
  - Aucun lien, aucune interaction directe entre les participants

Mémoire partagée

- Complexité du modèle : dans la gestion de la mémoire
  - On est dans un système distribué
    - Comment gérer une mémoire dans ce contexte ?
  - Plusieurs solutions
    - Déployer toute la mémoire sur un seul site
      - Accès simple mais goulot potentiel d'étranglement et fiabilité faible
    - Éclater la mémoire sur plusieurs sites
      - Avec ou sans duplication des données
      - Il faut mettre en place des algorithmes +/- complexes de gestion de mémoire distribuée

Modèle pair à pair (peer to peer)

- Un seul rôle : pas de distinction entre les participants
  - Chaque participant est connecté avec tous les participants d'un groupe et tout le monde effectue les mêmes types d'actions
  - Pour partager des données, effectuer un calcul commun ...
- Exemples
  - Modèles d'échanges de fichiers (bittorrent...)
    - Avec parfois un mode hybride client/serveur – P2P
    - Serveur sert à connaître qui possède un fichier ou faire des recherches
    - Le mode P2P est utilisé ensuite pour les transferts
      - Chacun envoie une partie du fichier à d'autres participants
  - Algorithme de consensus : choix d'une valeur parmi plusieurs
    - Chacun mesure une valeur (la même en théorie) puis l'envoie aux autres
    - Une fois reçues les valeurs de chacun, localement, chacun exécute le même algorithme sur ces valeurs pour élire la bonne valeur

Triptyque d'une application

- Trois parties sont intégrées et coopèrent pour le fonctionnement d'une application
  - Présentation : interfaces textuelles ou graphiques, interactions, entrée des données, validation, etc.
  - Traitements (Services métier) : traitements associés à l'application
  - Données (Persistance) : stockage et accès aux données
    - base de données
    - Fichiers (binaires, XML, ...)
- On les appelle aussi des « tiers » (étages)
- Application « 3 – tiers » quand les 3 parties sont clairement distinctes

Triptyque d'une application

- Modèle centralisé
  - Tout est sur la même machine

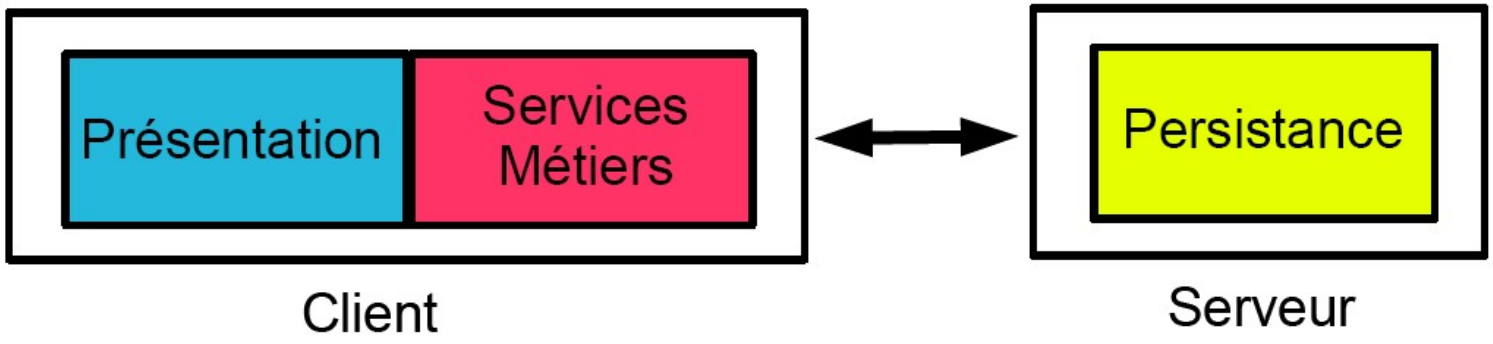
- Dans un contexte distribué
  - Les tiers sont / peuvent être exécutés sur des machines différente
  - Certains tiers peuvent être sous découpés
  - De nombreuses variantes de placement des tiers et de leur distribution

Architecture 2 – tiers

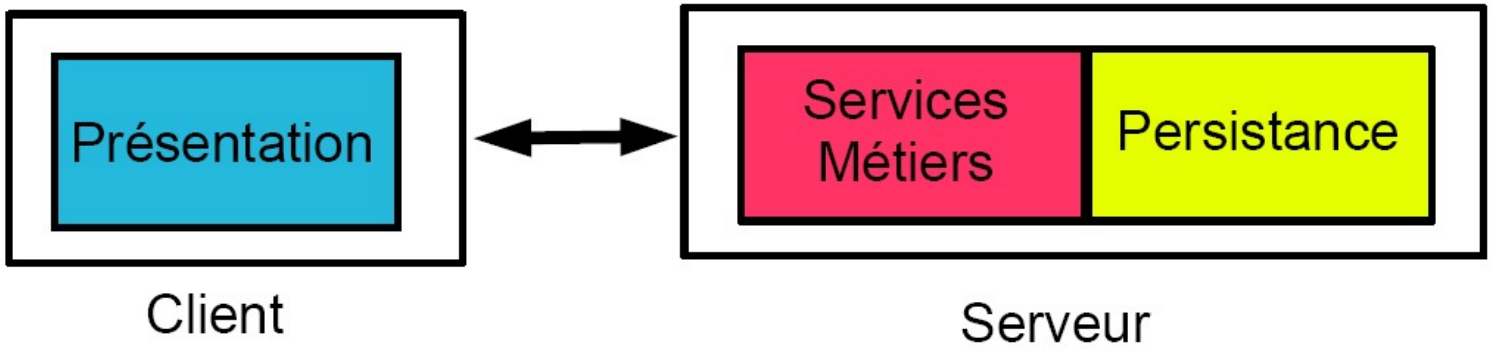
- Architecture 2 – tiers
  - Client / serveur de base, avec 2 éléments
    - Client : présentation, interface utilisateur
    - Serveur : partie persistance, gestion physique des données
  - Les services métier / la partie applicative peuvent être
    - Soit entièrement coté client, intégrés avec la présentation
      - La partie serveur ne gère que les données
        - » Ex : traitement de texte avec serveur de fichiers distants
        - » Ex : application accédant à une BDD distante
    - Soit entièrement coté serveur
      - La partie client ne gère que l'interface utilisateur
      - L'interface utilisateur peut même être exécutée sur le serveur
        - » Fonctionnement mode terminal / mainframe
        - » L'utilisateur a simplement devant lui écran / clavier / souris pour interagir à distance avec l'application s'exécutant entièrement sur le serveur
    - Soit découpés entre la partie serveur et la partie client

Architecture 2 – tiers

- Client : présentation + applicatif

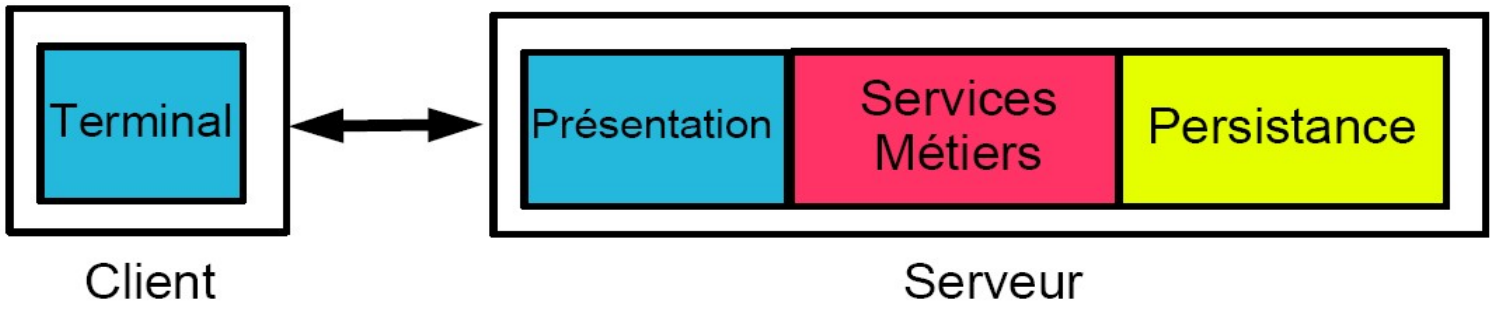


- Serveur : applicatif + gestion données

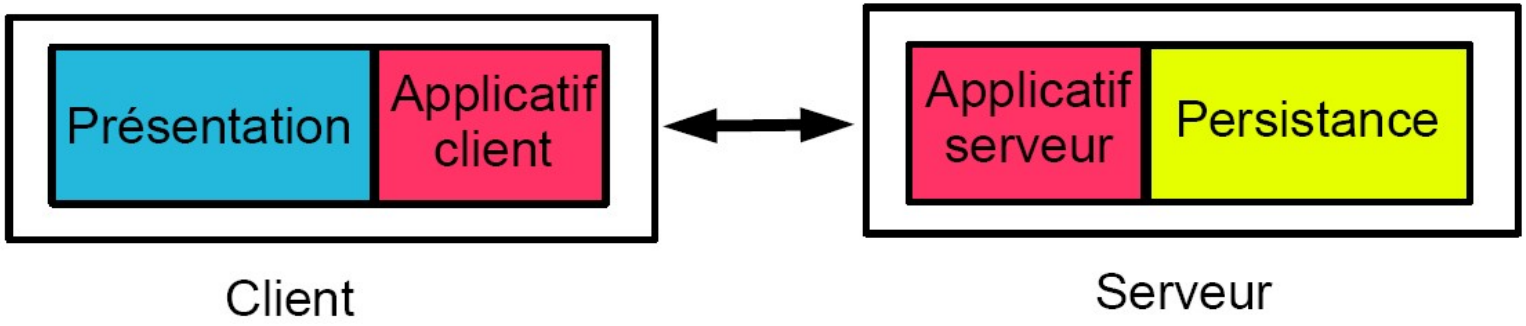


Architecture 2 – tiers

- Terminal : client intègre un minimum de la partie présentation

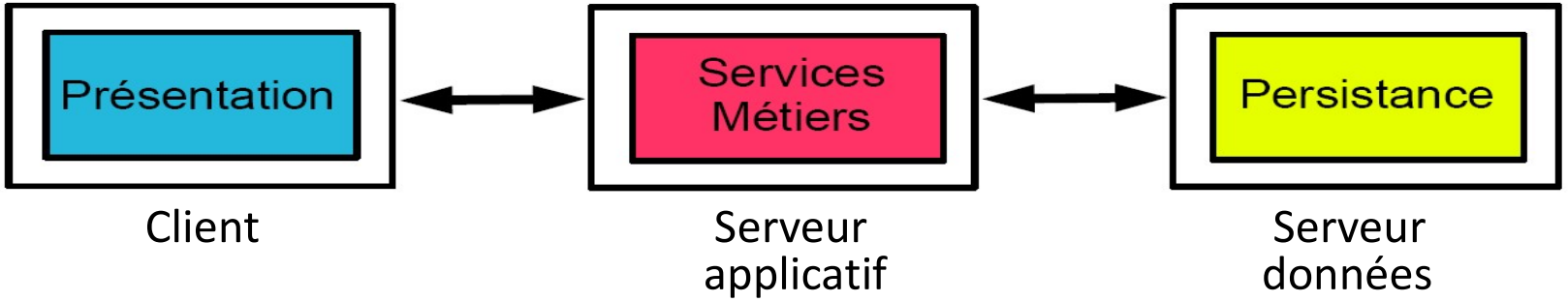


- Applicatif : découpé entre client et serveur



Architecture 3 – tiers

- Les 3 principaux tiers s'exécutent chacun sur une machine différente
  - Présentation
    - Machine client
  - Applicatif / métier
    - Serveur d'applications
  - Persistance
    - Serveur de (base de) données



Architecture 3 – tiers sur le web

- Très développée de nos jours
- Couche présentation
  - Navigateur web sur machine cliente
    - Client léger
  - Affichage de contenu HTML
- Couche applicative / métier
  - Serveur d'applications
    - Serveur HTTP exécutant des composants / éléments logiciels qui génèrent dynamiquement du contenu HTML
    - Via des requêtes à des BDD distantes
- Couche persistance
  - Serveur(s) de BDD de données

Architecture n – tiers

- Rajoute des étages / couches en plus
- La couche applicative n'est pas monolithique
  - Peut s'appuyer et interagir avec d'autres services
  - Composition horizontale
    - Service métier utilise d'autres services métiers
  - Composition verticale
    - Les services métiers peuvent aussi s'appuyer sur des services techniques
      - Sécurité
      - Transaction
      - …
- Chaque service correspond à une couche
  - D'où le terme de N-tiers

Architecture n – tiers

- Intérêts d'avoir plusieurs services / couches (3 ou plus)
  - Réutilisation de services existants
  - Découplage des aspects métiers et technique et des services entre eux : meilleure modularité
  - Facilite évolution : nouvelle version de service
  - Facilite passage à l'échelle : évolution de certains services
  - On recherche en général un couplage faible entre les services
    - Permet de faire évoluer les services un par un sans modification du reste de l'application
- Inconvénients
  - En général, les divers services s'appuient sur des technologies très variées : nécessite de gérer l'hétérogénéité et l'interopérabilité
  - Utilisation de framework / outils supplémentaires
    - Les services étant plus découpés et distribués, pose plus de problèmes liés à la distribution

Logique de présentation

- Les tâches liées à la présentation requièrent
  - L'interaction avec l'utilisateur
    - Réalisée par la GUI d'un client lourd : boutons, listes, menus, ...
    - Réalisée par le navigateur pour un client web
      - Via interaction plus « basique » : formulaires, liens vers des URLs ...
  - La logique de présentation
    - Le traitement des données retournées par les services métiers et leur présentation à destination de l'utilisateur
    - Réalisée par un client lourd qui généralement fait directement l'appel des services métiers sur le serveur
    - Pour un client web
      - Navigateur se contente d'afficher du code HTML qui ne peut pas être statique, vu que le contenu dépend des données retournées
      - Doit donc exécuter du code qui récupère les données retournées par les services métiers et génère dynamiquement du code HTML
      - Logique de présentation peut être exécutée coté client ou coté serveur mais se fait généralement dans un tiers à part coté serveur

Persistance

- Couche de persistance
  - Stockage et manipulation des données de l'application
  - Plusieurs supports physique
    - Fichiers binaires, textes « de base »
    - Fichiers XML
    - Une base de données ou un ensemble de bases de données
  - Pour ce dernier cas
    - Nécessité d'envoyer à distance des requêtes (types SQL) et d'en récupérer les résultats
    - Pour réaliser cela
      - Soit c'est natif dans le langage utilisé (ex : PHP)
      - Soit on passe par des frameworks ou des API dédiés

Persistance

- Quelques standards / outils d'accès à distance à des BDD
  - RDA (Remote Data Access) de l'ISO
  - ODBC (Open Data Base Connectivity) de Microsoft
  - JDBC (Java Data Base Connectivity) de Sun
    - Framework pour le langage Java
  - Fonctionnement général
    - Gestion de requêtes SQL mais avec indépendance du SGBDR utilisé (mySQL, PostgreSQL, Oracle ...)
      - En général, seule la phase de connexion au SGBDR est spécifique

Frameworks globaux

- Une application 3/N – tiers intégre un grand nombre de technologies
  - Présentation : HTML, GUI ...
  - Applicatif : objets, composants, scripts exécutables, services ...
  - BDD : fichiers XML, SGBDR, ...
  - Protocoles de communication : RPC / RMI, HTTP
- Pour faciliter l'intégration de ces technologies et le développement d'applications
  - Editeurs offrent des frameworks globaux
    - J2EE / Java EE chez Sun
    - .Net chez Microsoft
  - Serveur d'application
    - Serveur permettant d'exécuter les parties applicatives dans le contexte de ces framewoks