Technologie des applications client-serveur

UE RSX 102

Support de cours Tome 1

Anas ABOU EL KALAM

anas.abouelkalam@enseeiht.fr

Plan

Introduction / Notions générales

- De l'information centralisée au client-serveur
- Les "middlewares"
- Les architectures distribuées
- Application : architectures internet

Plan

Introduction / Notions générales

- De l'information centralisée au client-serveur
- Les architectures distribuées
- Les "middlewares"
- Application : architectures internet

Introduction : situation actuelle

Mutation permanente des **concepts**, des **techniques** et des **organisations** associées aux applications informatiques.

Effort de recherche et de production industrielle au niveau mondial :

- l'apparition de *composants* de **rapidité** et de **complexité** en croissance continue.

- la possibilité de développement de *solutions logicielles* et *organisationnelles* irréalistes quelques années auparavant.

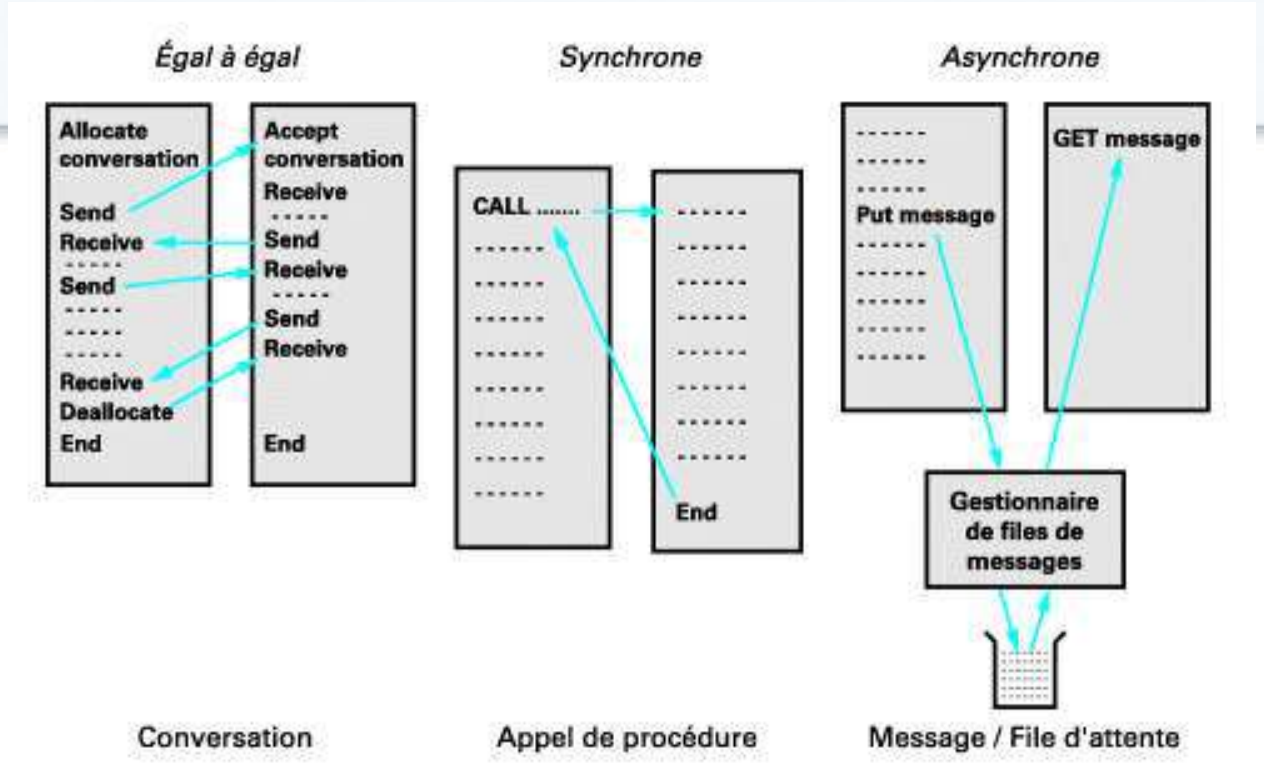
Introduction : systèmes disponibles

On dispose à prix accessible pour les entreprises et le grand public de calculateurs puissants munis :

- de **SE** *évolués*
- d'**IHM** *évoluées*
- de **capacités de stockage** *énormes*
- *de* **capacités de traitements** *rapides*
- de **moyens d'interconnexion** "**réseaux locaux**" très *performants*
- de **moyens d'interconnexion** à "**longue distance**" à *bas prix* et de *bande passante importante*.

Standardisation, C/S, ...

Introduction : Modes de comm entre progs



- Modèle conversationnel
  - e.g., conversation téléphonique, sockets (dans TCP / IP)
- Modèle à appel de procédure
  - e.g., Machine à sous, RPC
- Modèle asynchrone
  - e.g., poste, mail

Communication asynchrone par messages : avantages

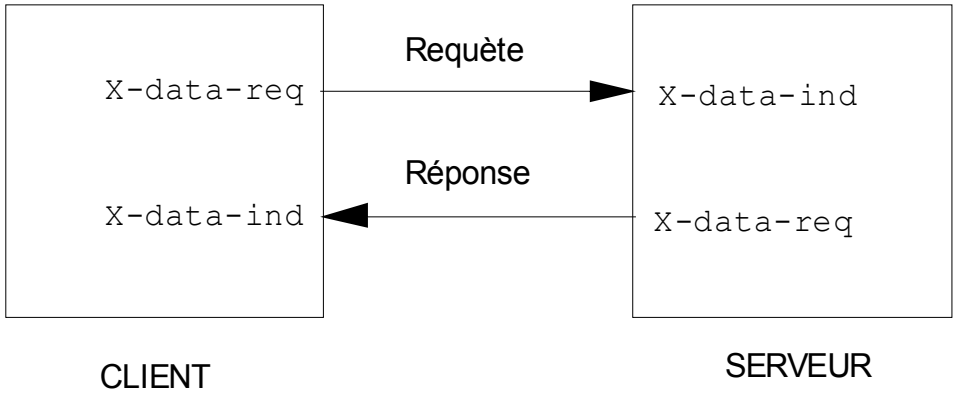
- Applis communicantes ne sont plus liées l'une à l'autre.
- Peuvent avoir des caractéristiques différentes de disponibilité et de débit
- Applis font appel à un gestionnaire de messages applicatifs
  - n'ont plus à se préoccuper des problèmes de comm inhérents aux échanges
  - Il leur suffit de placer / extraire messages dans des files d'attente
  - Rôle gestionnaires : transiter messages de files d'attente en files d'attente + (éventuellement intégrité, ...)
- Séparation nette entre développement d'applications et exploitation de système
- La standardisation (API commune d'accès) renforce utilisation ces techniques
- Asynchronisme permet de développer des applications qui traitent infos « au fil de l'eau », i.e. au fur et à mesure que les données sont disponibles, tout en écrêtant les pointes d'activité, si cela est nécessaire

Le C/S : définition (basique)

Architecture réseau dans laquelle :

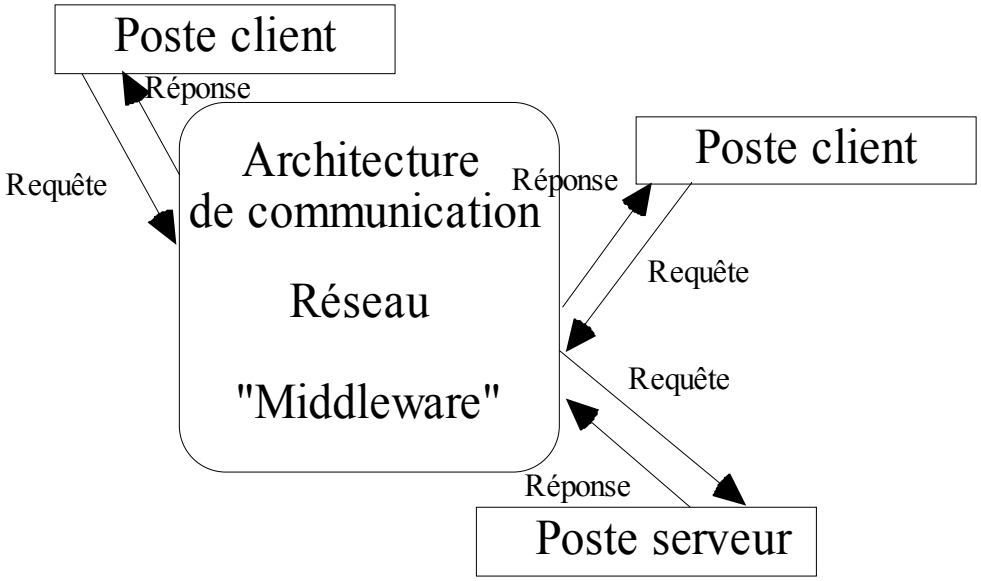
- données localisées et traités sur S // accessible aux C :
  - *client* émet des **requêtes**,
  - *serveur* rend le **service** demandé.

- ...utiliser communication par **messages** de données en mode **asynchrone** offerte par les protocoles de **transport**
  - 1er mssg : C envoie requête exécution traitement au S
  - 2nd mssg : S envoie réponse (après avoir effectué traitement)



Le C/S en mode message : Architecture générale

- Fonctionnement le plus fréquent en C/S :
  - mode requête réponse pour une population de C et de S



Le C/S : applications (1/2)

- informatique de **gestion**
- algorithmique **répartie**,
- informatique **industrielle**,
- Applications internet ....

- Dans son acception la plus complète: possibilité de définir n'importe quelle **architecture de communication**.
  - Chaque entité est ***à la fois client et serveur***
  - Chaque entité ***émet et traite*** des requêtes

C/S : applications (2/2)

Applications coopératives **("*Cooperative work*")**

• Ensemble d'*entités logicielles* **coopérant** au moyen d'un *réseau* à la réalisation d'une **tâche informatique**

Systèmes répartie **("*Distributed Computing*")**

• Applications *systèmes et réseaux* indispensables au fonctionnement des machines en réseau

Intelligence Artificielle Distribuée **("*Distributed AI*")**

• Application coopérative mettant en relation des *agents* qui fonctionnent selon des approches dérivées de l'IA

Le calcul massivement parallèle **("*Grid Computing*")**

• Application de calcul réalisé par le travail en *parallèle* et en coopération d'un nombre élevé de *processeurs*

Les systèmes répartis de contrôle commande de procédés industriels

Application de *contrôle en temps réel* de procédés tenant compte des contraintes de QoS et de *sûreté de fonctionnement*.

Le C/S en mode message : Architecture générale

- Notions & Questions à traiter …
- Modèles de répartition
  - *données*
  - *traitements*
  - Dans une *architecture réseau*
    - API
    - Middleware
    - …

Plan

Introduction / Notions générales

- De l'information centralisée au client-serveur

Les "middlewares"

Les architectures distribuées

Application : architectures internet

Niveaux d'abstraction d'une application

➔ ***Couche présentation (IHM)***

- interaction de l'application avec l'utilisateur.
  - *Gérer saisies clavier/souris, présentation info à l'écran*
    - Doit être conviviale et ergonomique

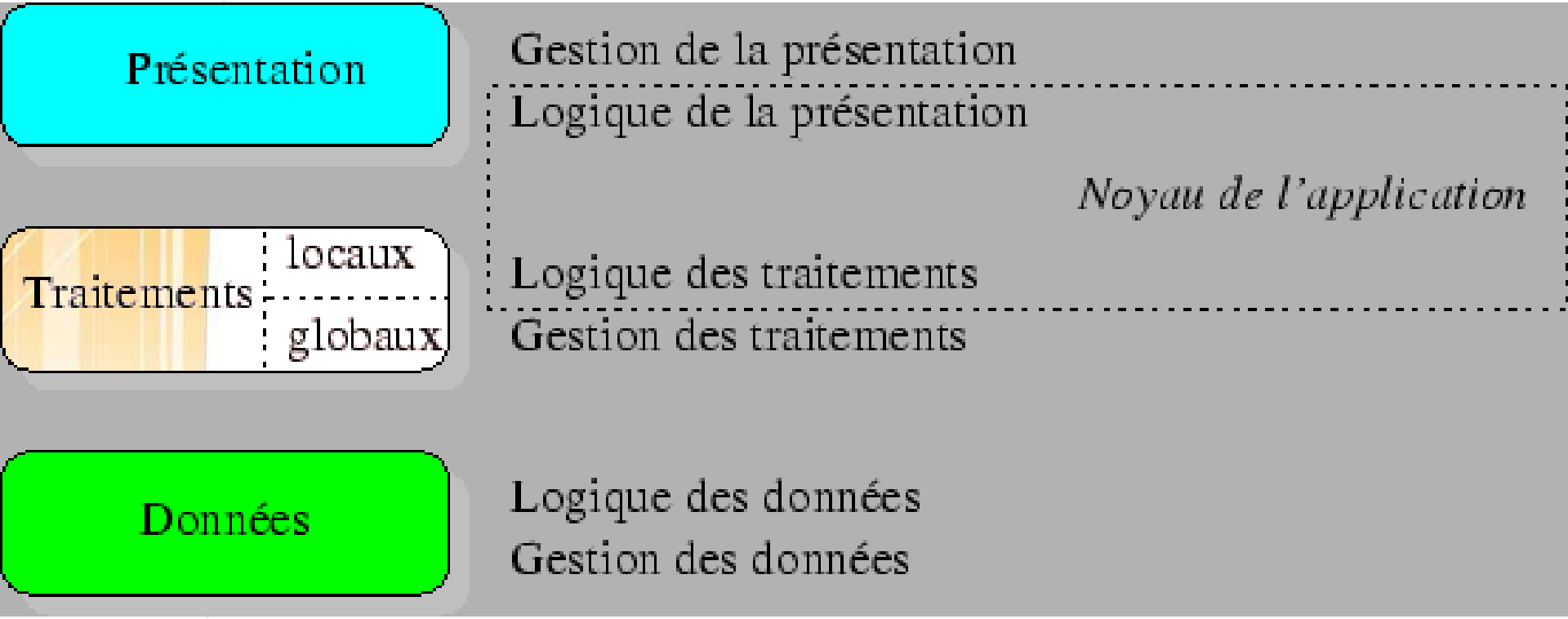
➔ ***Logique applicative, traitements***

- travaux à réaliser par l'application
  - *Traitements locaux* : contrôles dialogue au niveau IHM, aide à la saisie, …
  - *Traitements globaux* constituant l'appli elle même :
    - business logic : règles internes qui régissent l'entreprise

➔ ***Données***

- *mécanismes permettant la gestion des informations stockées par l'application*

Niveaux d'abstraction d'une application



Niveaux d'abstraction d'une application

➔ Ces trois niveaux peuvent être *imbriqués* ou *répartis* de différentes manières entre plusieurs machines physiques.

➔ Le noyau de l'application est composé de

- la logique de l'affichage/présentation
- la logique des traitements.

➔ Le découpage et la répartition de ce noyau permettent de distinguer les architectures applicatives suivantes :

- **1**-tiers *(étage)*,
- **2**-tiers ==> Client/Serveur
- **3**-tiers ==> Applications distribuées
- **n**-tiers.

L'architecture un-tiers : définition

➔ Les trois couches applicatives sont intimement liées et *s'exécutent sur le même ordinateur*

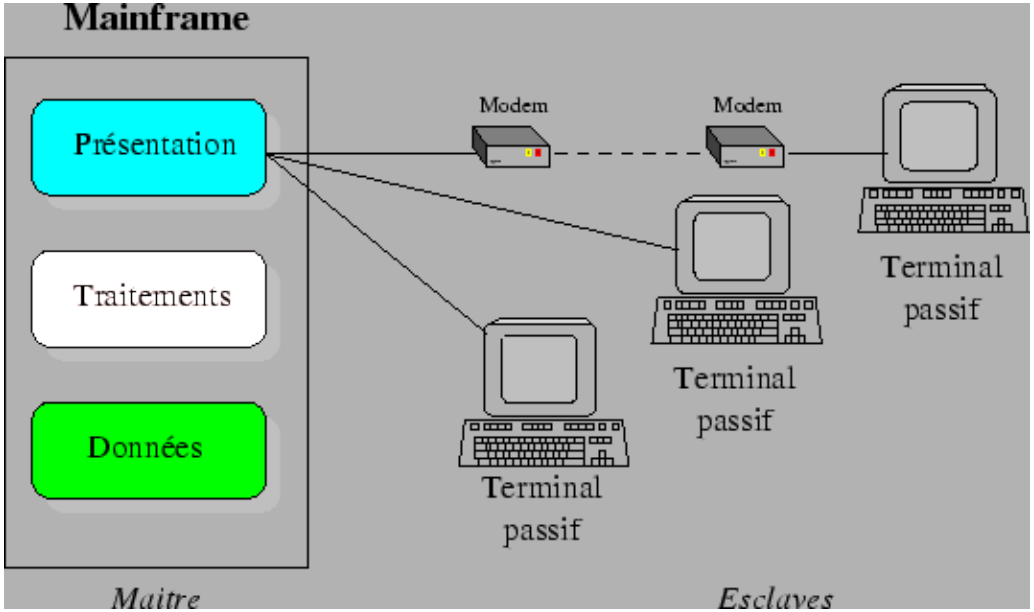
➔ On ne parle pas ici d'architecture client-serveur, mais d'informatique *centralisée*.

➔ Dans un contexte multi-utilisateurs, on peut rencontrer deux types d'architecture mettant en œuvre des applications un tiers :

- des applications sur site central
- des utilisateurs accédant depuis des sites distinctes (terminaux passifs) aux mêmes données
  - ==> partage de fichiers physiquement localisés sur site central

L'architecture un tiers : exemple du *mainframe* (1/2)

- **Accès multi-utilisateurs**
  - Les utilisateurs se connectent aux applis à l'aide de **terminaux passifs**
    - *esclaves.*
  - Les applis sont exécutées sur **Serveur central** (mainframe)
    - *qui prend en charge l'intégralité des traitements,*
      - y compris l'affichage qui est simplement déporté sur des terminaux passifs



L'architecture un tiers : *mainframe* (2/2)

Avantages

- Facilité d'administration
- Large palette d'outils de conception, de programmation et d'administration ayant atteint un niveau de maturité et de fiabilité
- Centralisation ➔ utilisation optimale des ressources

Mais

- Mainframe ==> Point dur ...
- Démodés
  - émergence des interfaces utilisateur
  - Interface utilisateur en mode caractère jugée obsolète par les utilisateurs <== émergence bureautique, ...
  - ré-habillage permet de rajeunir (*GUI*), mais au prix d'une lourdeur

L'architecture un tiers : Limitations

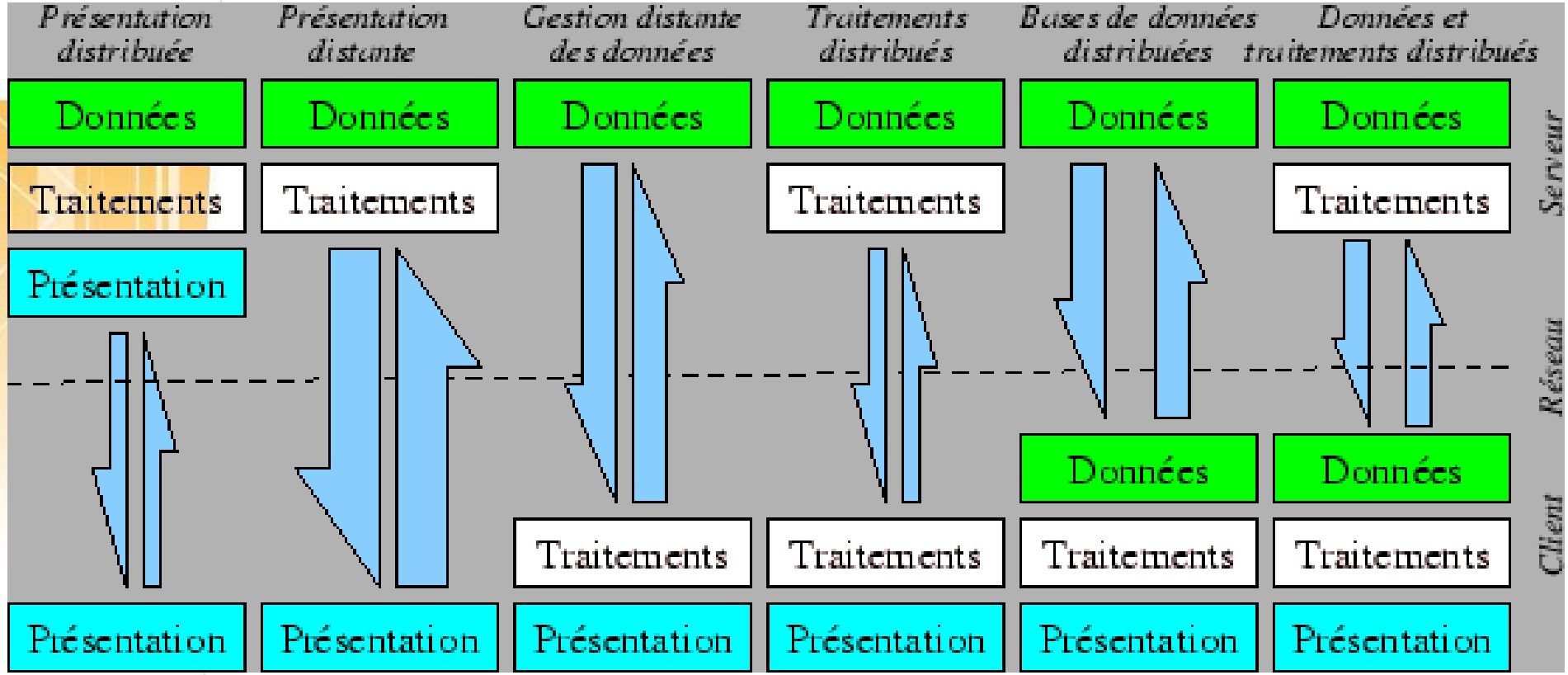
- Site central
  - Interface utilisateur en mode caractères
  - Cohabitation d'applications micro exploitant des données communes n'est pas fiable au delà d'un certain nombre d'utilisateurs.

- Solution doit concilier
  - l'interface utilisateur *moderne* des applications sur micro-ordinateurs.
  - *Fiabilité* des solutions sur site central, qui gèrent les données de façon centralisée,

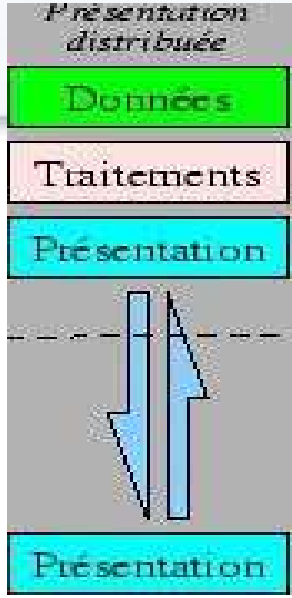
- Donc ...
  - il faut **scinder** les applications en plusieurs parties distinctes et coopérantes
    - gestion centralisée des données,
    - gestion locale de l'interface utilisateur.

***Ainsi est né le concept du* client-serveur ...**

Modèles de répartitions : schéma du *Gartner Group* (1/5)

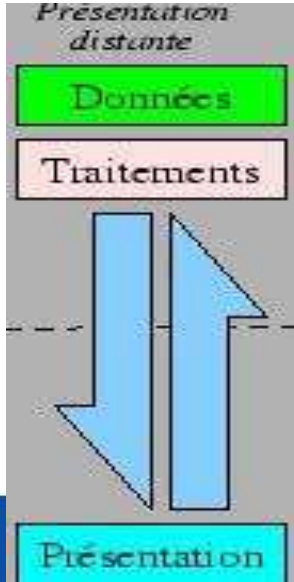


Le schéma du Gartner Group (2/5)



➔ **Présentation distribuée**

- habillage graphique de l'affichage en mode caractères d'applications fonctionnant sur site central
- Poste client conserve une position d'esclave / serveur



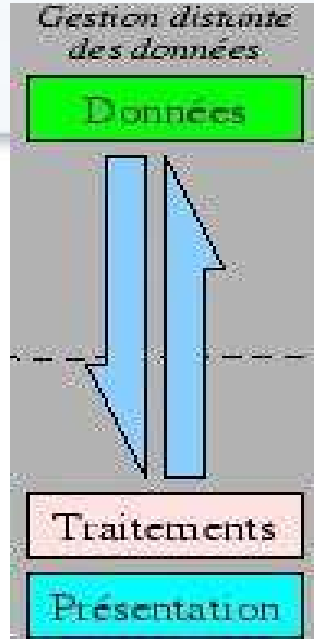
➔ **Présentation distante**

- Client serveur de présentation
- Serveur ➔ ensemble traitements
- Client ➔ que l'affichage
- Inconvénient
  - Génération trafic réseau fort
  - Aucune répartition charge entre C/S
- Exemples
  - *X-Window*, *DecTP* sur Vax, *APPC* sur IBM

Le schéma du Gartner Group (3/5)

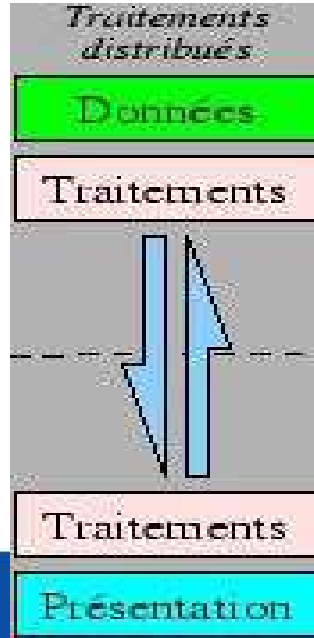
➔ Gestion distante des données

- Client serveur de données
- Client ➔ totalité de l'appli
- SGBD centralisé ➔ gestion données et contrôle d'intégrité
- Souvent pour applis type infocentre / banques de données
- Inconvénient
  - *Trafic réseau assez important*
  - *Ne soulage pas énormément le client*



➔ Traitement distribué

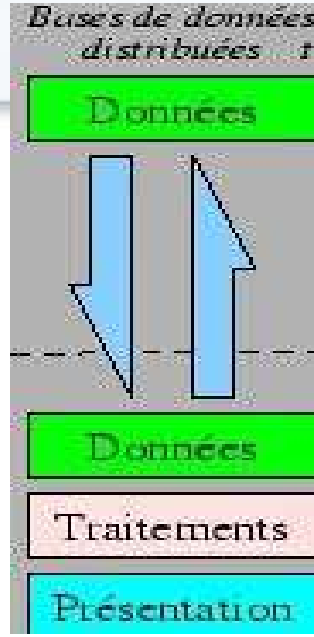
- Client serveur de traitement
- Découpage de l'appli se fait au plus près de son noyau et les traitements sont distribués entre le client et le serveur
- Appel de procédures
- Avantage
  - *optimiser répartition charge*
  - *Limite trafic réseau*
- Inconvénient
  - *C & S doivent à l'avance connaître et implémenter procédures à accomplir*



Le schéma du Gartner Group (4/5)

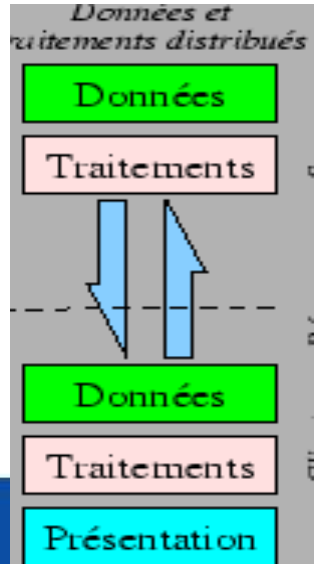
➔ Bases de données distribuées

- Variante du C/S de données dans laquelle une partie des données est prise en charge par le client
- Intéressant si
  - *l'appli doit gérer de gros volumes de données*
  - *On souhaite disposer de temps d'accès rapide*
  - *Contraintes de confidentialité*

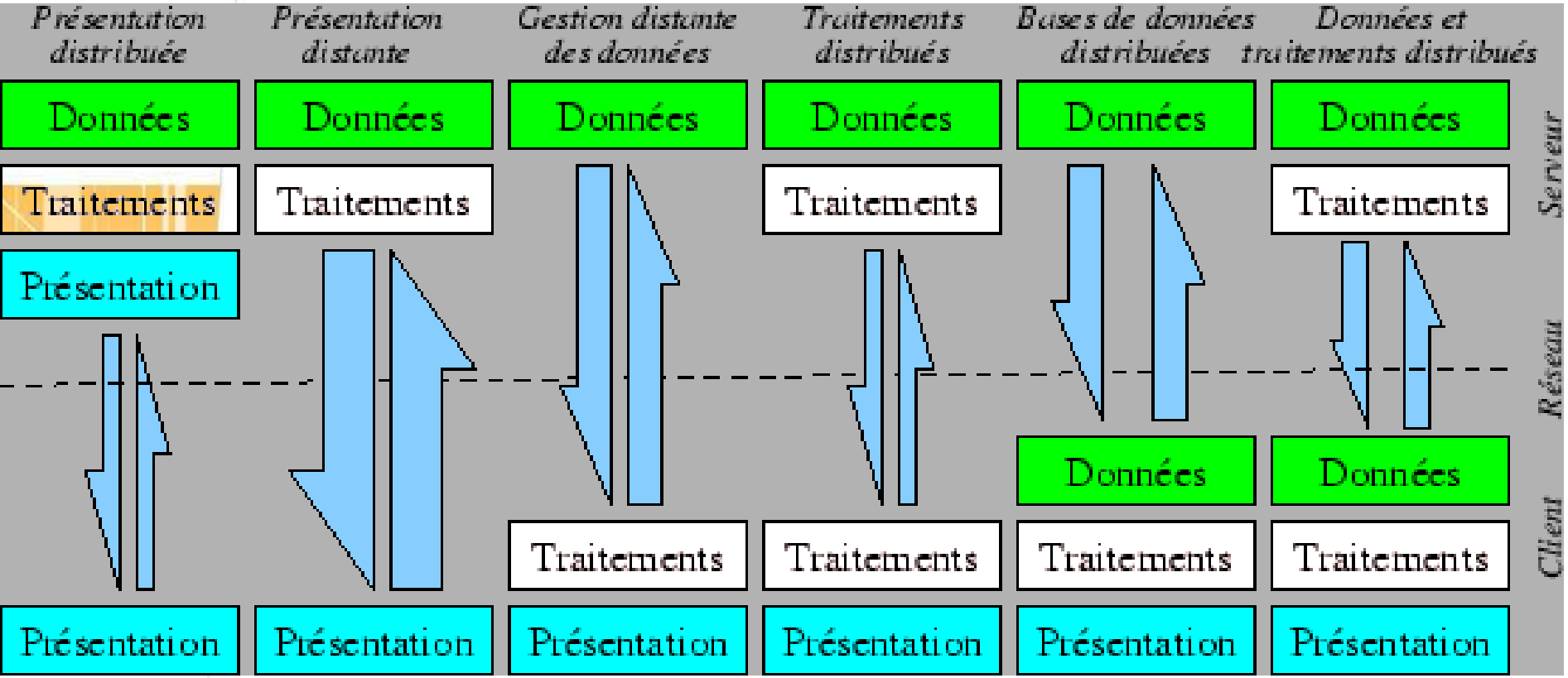


➔ Données et traitements distribués

- Tire partie de la notion de composants réutilisables et distribuables pour répartir au mieux la charge entre C/S
- Architecture complexe à MEV



Le schéma du Gartner Group (5/5)



Ne correspondent pas vraiment à des applis C/S

Archi 2-tiers

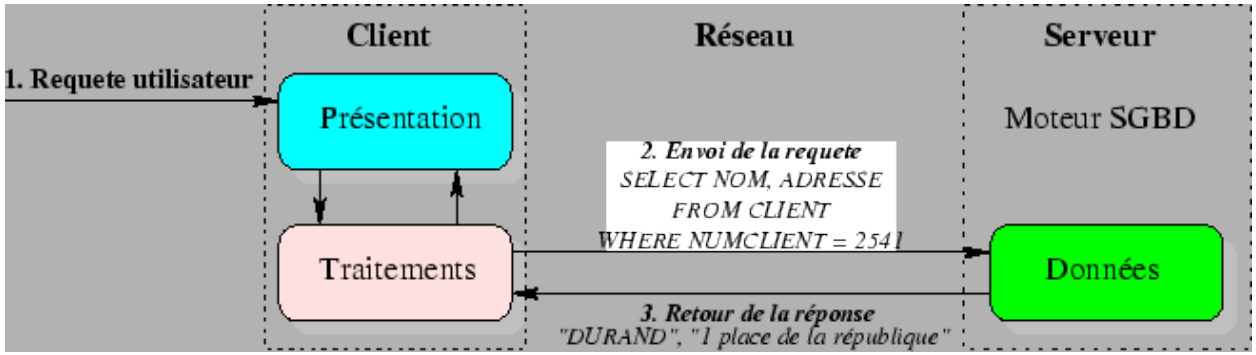
L'architecture deux tiers : Présentation

- le poste client se contente de ***déléguer la gestion des données*** à un service spécialisé.
  - E.g., appli de gestion fonctionnant sous Windows et exploitant *SGBD centralisé*
- Ce type d'application permet de tirer partie de
  - *Interface riche* des ordis déployés en réseau,
  - *Cohérence des données*, qui restent gérées de façon centralisée.
    - *Gestion des données prise en charge par SGBD centralisé, s'exécutant le plus souvent sur un serveur dédié*
- Client envoie requête (langage de requête e.g., SQL) au serveur
- Dialogue C/S
  - envoi de requêtes et retour des données correspondant aux requêtes.

Exemple de C/S en architecture deux tiers

- **client**
  - Provoque/initie l'établissement d'une conversation afin d'obtenir des données ou un résultat de la part du serveur.
  - Émet des requêtes
- **serveur**

Programme qui répond au client ==> rend le **service** demandé



- L'***échange*** de messages
  - transite à travers le réseau reliant les deux machines.
  - met en œuvre des mécanismes relativement complexes qui sont, en général, pris en charge par un ***middleware***.

Plan

Introduction / Notions générales

De l'information centralisée au client-serveur

- **Les Middlewares**

Les architectures distribuées

Application : architectures internet

Le Middleware : définition (1/2)

➔ Quoi ?

- **élément du milieu** = **interface de comm universelle entre processus**
- L'ensemble des *couches réseau* et *services logiciel* qui permettent le dialogue entre les différents composants d'une ***application répartie***.

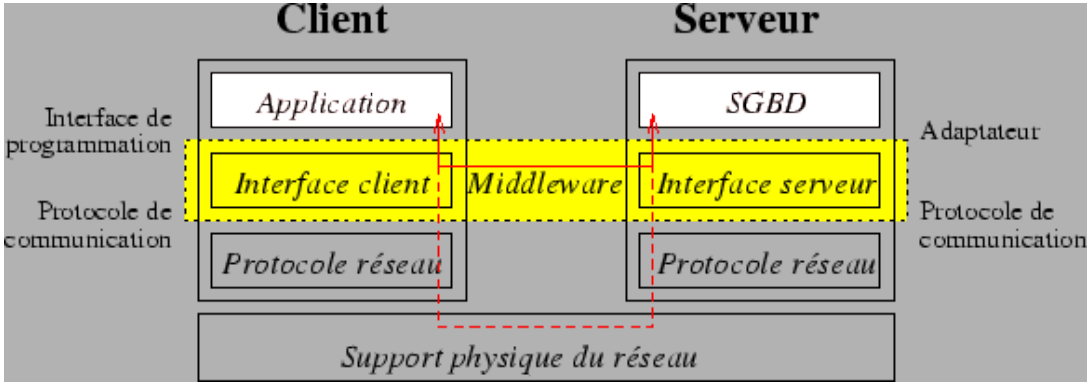
➔ Pourquoi ?

- Assurer échanges données entre C et S en ***masquant*** les différents *problèmes* potentiels liés à:
  - ***répartition*** *des données et traitements (accès distant, baisse performances)*
  - ***hétérogénéité*** *des matériels et des logiciels en opération.*
  - *…*

Le Middleware : définition (2/2)

envoyer des requêtes d'accès à des données (type SQL) d'un client vers un serveur et recevoir les résultats

- ***Ouvrir une connexion*** entre entités
- *Envoi de* ***requêtes SQL*** vers le serveur
- ***Conversion des formats*** de requêtes
- ***Envoi*** *des résultats*
- ***Conversion*** *des résultats*
- (*Gestion des erreurs*)
- ***Fermeture*** *de connexion*.

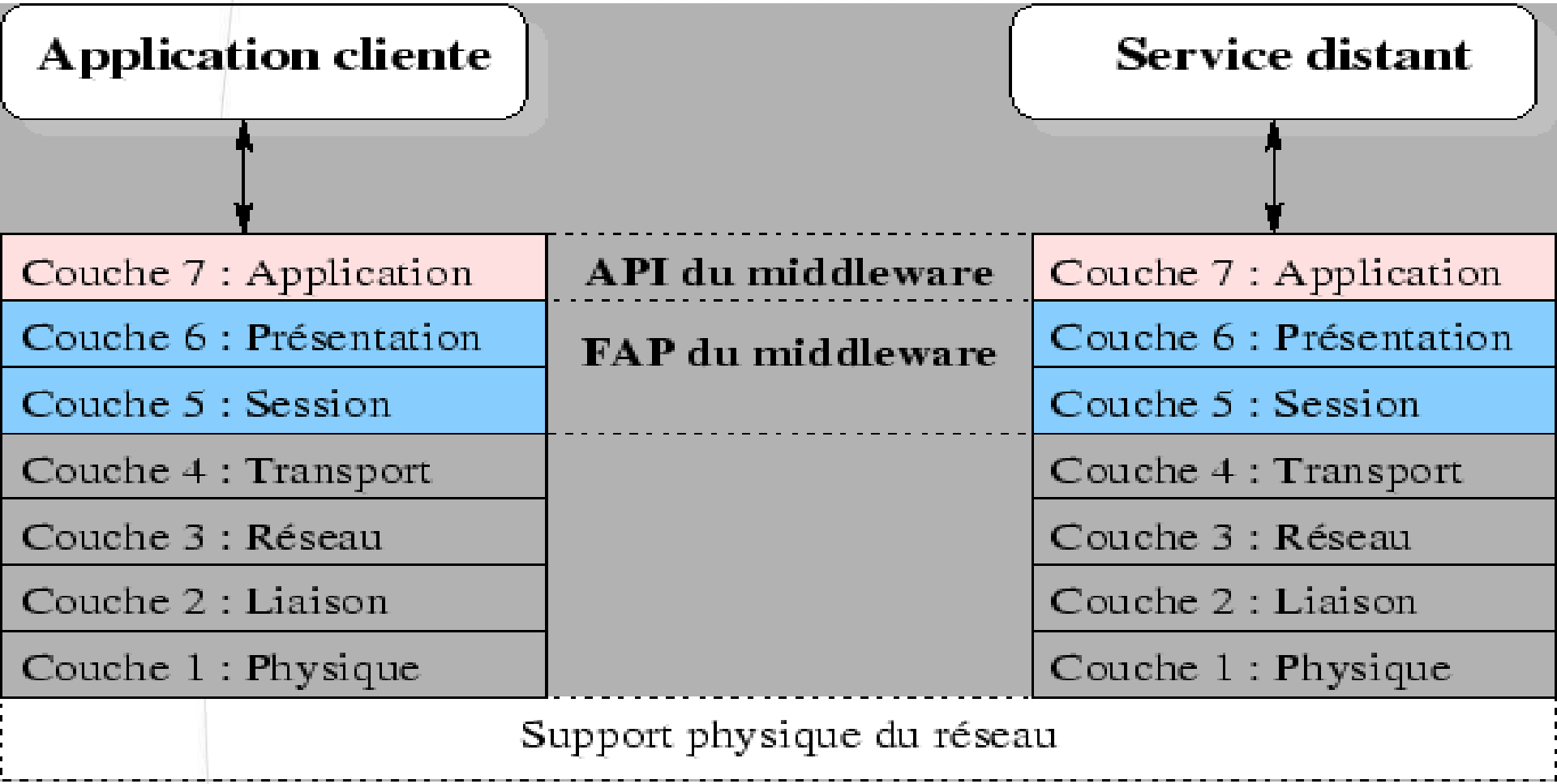


Le Middleware : services

Comment réaliser les fonctions classiques des systèmes centralisés que l'on rencontre maintenant en univers réparti ?

- **Conversion** : Service utilisé pour la communication entre machines mettant en oeuvre des *formats de données différents*, elle est prise en charge par la FAP
- **Adressage** : Permet d'identifier la machine serveur sur laquelle est localisé le service demandé afin d'en *déduire le chemin d'accès*.
  - Annuaire
- **Sécurité** : Permet de garantir la confidentialité et la sécurité des données à l'aide de mécanismes *d'authentification* et de *chiffrement* des informations.
- **Communication** : Permet la transmission des messages entre les deux systèmes sans altération.
  - connexion au serveur
  - préparation de l'exécution des requêtes
  - récupération des résultats
  - dé-connexion.

Le Middleware : services



Le Middleware : exemples

- **SQL*Net** : Interface permettant de faire dialoguer une application cliente avec une base de données Oracle.
  - Passage requête SQL, Appel procédures
  - indépendance vis à vis du réseau (topologie, protocole) et des OS
  - multi-thread = plusieurs connexions dans un seul process Oracle Serveur
  - journalisation et traçage, gestion des connexions avortées, ...

- **EDA (Enterprise Data Access)-SQL**
  - Hétérogénéité : interface commune pour clients s'exécutant sur +35 platformes
  - requêtes (simultanées) sur +80 BD
  - Vue uniforme indépendante du type plateforme, forme stockage, ...

ODBC: Interface standardisée isolant le client du serveur de données.
C'est l'implémentation par Microsoft du CLI (Call Level Interface) du SQL Access Group

- Elle se compose de :
  - gestionnaire de driver standardisé,
  - API s'interfaçant avec l'application cliente
  - driver correspondant au SGBD utilisé

Le Middleware : exemples

➔ **DCE** (*Distributed Computing Environment*) / RPC

Permet à processus tournant sur ++ ordis de communiquer comme s'il sont en local

- Dans prog client, ∃ fonction locale qui a le même nom que fonction distante et qui, en réalité, appelle fonctions de la bibliothèque RPC
  - gérent connexions réseaux, passage params, retour résultats
- De même, côté serveur, fonction/processus qui attend connexions clientes, appelle votre fonction avec bons param, renvoyer les résultats.
- Fonctions qui prennent en charge les connexions réseaux sont des "stub"
  - Il faut donc écrire stub C & stub S, en plus du prog C et fctn distante

Moniteur transactionnel (*Transaction Processing Monitor*)

➔ Sorte de "*scheduler* sophistiqué" pour *partager ressources* limitées (BD, mémoire) entre un *grand nombre d'utilisateurs* simultanés et plusieurs *serveurs*

- planification de l'exécution des transactions, synchronisation ...
- e.g., BEA Tuxedo pour Unix/windows, Top End de NCR pour Unix, IBM TPF, Bull TP8, ...

Le Middleware : Remarques

- Le choix d'un middleware est déterminant en matière d'architecture,
  - il joue un grand rôle dans la **structuration** du système d'information

- Pour certaines applis devant accéder à des services hétérogènes, il est parfois nécessaire de combiner plusieurs middlewares.
  - poste client doit connaître et mettre en œuvre plusieurs IPC (*Inter Process Communication*)
    - Mécanismes : queues de messages, mémoire commune, sémaphores
    - ➔ client lourd

C/S deux tiers : Avantages

- permet l'utilisation d'une interface utilisateur riche,
- Permet l'appropriation des applications par l'utilisateur,
- Introduit la notion d'interopérabilité.

C/S deux tiers : Limites

- Poste client supporte la grande majorité des traitements applicatifs
  - **Client lourd** ou ***fat client***

- le poste client est :
  - fortement sollicité,
  - devient de plus en plus complexe
  - doit être mis à jour régulièrement pour répondre aux besoins des utilisateurs,
    - coûts et la complexité de sa maintenance.

- la conversation entre client et serveur est assez bruyante et s'adapte mal à des bandes passantes étroites

- les applications se prêtent assez mal aux fortes montées en charge car il est difficile de modifier l'*architecture initiale*,

C/S deux tiers : Solution ?

- **But**
  - Résoudre limitations du C/S 2-tiers tout en conservant des avantages
- **Solution ?**
  - Architecture plus évoluée, facilitant les forts déploiements à moindre coût

➔ **architectures distribuées.**

Plan

Introduction / Notions générales

- De l'information centralisée au client-serveur
- **Les architectures distribuées**
- Application : architectures internet

L'architecture trois tiers : objectifs

Principe

➔ utilisation d'un poste client **simple** communicant avec le serveur par le biais d'un protocole **standard**.

- Données ➔ toujours gérées de façon *centralisée*,
- présentation ➔ toujours prise en charge par le poste *client*,
- logique applicative ➔ prise en charge par un *serveur intermédiaire*

Archi 3- tiers

Force des archi 3-tiers

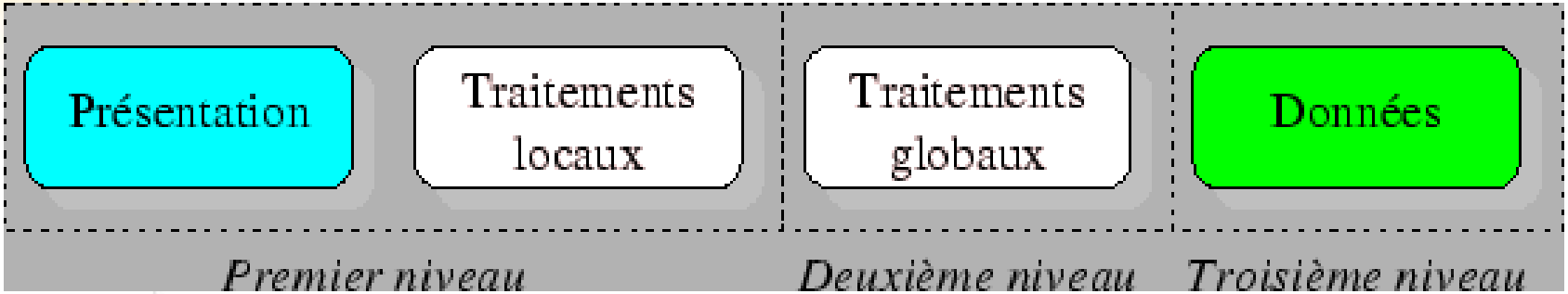
==> déploiement immédiat,

==> **évolutions** peuvent être **transparentes** pour l'utilisateur

==>caractéristiques du poste **client sont libres**

L'architecture trois tiers : répartition des traitements

- **premier niveau**
  - affichage et traitements locaux (contrôles saisie, mise en forme) pris en charge par client,
- **deuxième niveau**
  - traitements applicatifs globaux pris en charge par service applicatif
- **troisième niveau**
  - services de base de données pris en charge par un SGBD.



L'architecture trois tiers : premières tentatives

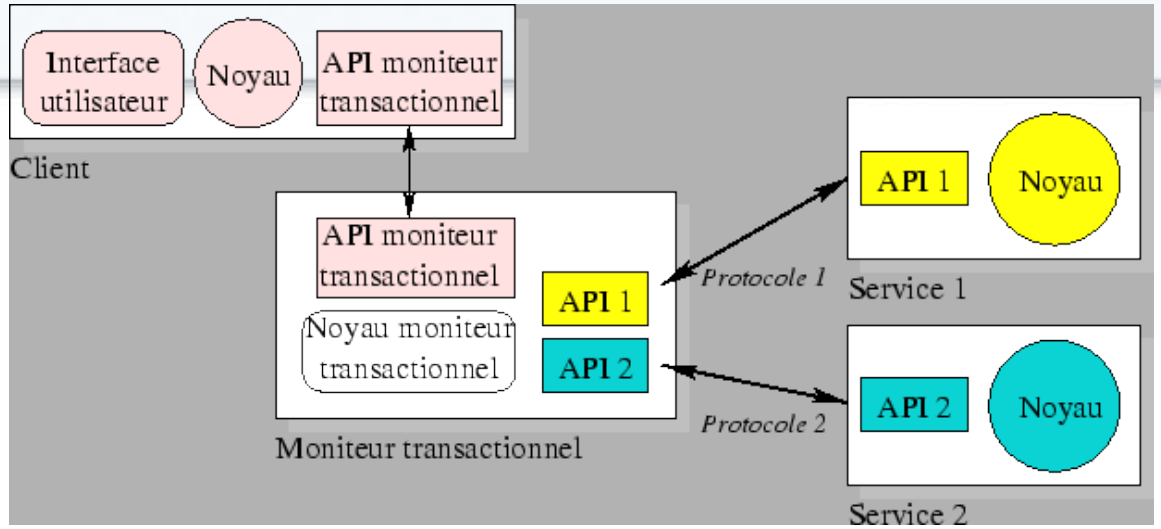
- Serveur d'application centralisé
- Communication C/S utilisant dialogue RPC

- Mécanismes pour dialogue RPC
  - Unix ➔ intégré au système NFS
  - Microsoft ➔ NetDDE puis DCOM ==> rendu obselète par .Net
  - Solutions propiétaires (ForT ou Implicite) permettent l'exploitation d'un serveur d'application par des clients simplement équipés d'un environnement d'exécution (*runtime*)

Mais

- **Absence de standards**
- **Mise en place Coûteuse**

Exemple d'archi trois tiers : serveur de transactions



- **Moniteur transactionnel** : mise en relation Client avec ∑ Serveurs de données
  - **Atomicité** : La transaction *ne peut être partiellement* effectuée,
  - **Cohérence** : transaction fait *passer la base d'un état cohérent à un autre*,
  - **Isolation** : transaction *n'est pas affectée par le résultat des autres*
  - **Durée** : *modifications* dues à une transaction sont *durablement garanties*.

    - **Soit transaction a définitivement eu lieu, soit elle n'a jamais existé**

- **API** : masque complexité de l'organisation des serveurs de données

L'architecture trois tiers : la révolution Internet

- ***World Wide Web*** (1989) :
  - permet de publier des informations richement mises en forme
  - affichage & traitements locaux (contrôles saisie, mise en forme) pris en charge par client
  - ARPANET ==> CERN fut l'acteur principal du WWW ==> W3C
  - Caractère universel, rendu possible par l'utilisation de **standards** reconnus
    - HTML, pour la description des pages disponibles sur le Web,
    - HTTP, pour la communication entre navigateur et serveur Web,
    - TCP/IP, le protocole réseau largement utilisé par les systèmes Unix,
    - CGI, l'interface qui permet de déclencher des traitements sur les serveurs Web
      - exécution d'un processus (sur Serveur) pour chaque invocation (par Client)
        - préférer extensions comme ISAPI, NSAPI ou Servlets java, …

  - ***Intranet***
    - Principes d'internet + technologies déployées dans entreprise
      - on utilise réseau local de l'entreprise
      - données gérées par un SGBD

L'architecture trois tiers : répartition des traitements

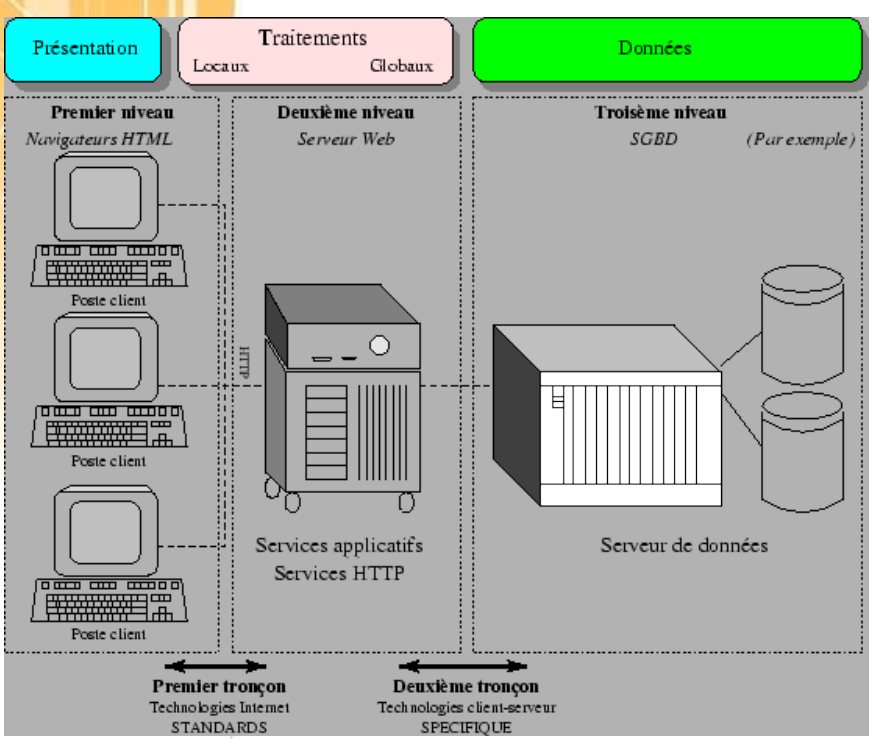
- ➔ Niveaux indépendants
  - ➔ peuvent être implantés sur *machines différentes*
  - poste client ne supporte plus Σ traitements

➔ moins sollicité, moins évolué, moins coûteux, …

- *ressources* présentes sur réseau sont *mieux exploitées*
  - puisque les traitements applicatifs peuvent être partagés ou regroupés
- *Fiabilité/performances* certains traitements se trouvent améliorées par centralisation
- relativement simple de faire face à une forte *montée en charge*, en renforçant le service applicatif

L'architecture trois tiers : exemple en Intranet

- poste ***client*** prend la forme d'un simple navigateur Web,
- service ***applicatif*** assuré par un serveur HTTP
- ***communication avec le SGBD*** met en oeuvre les mécanismes bien connus des applications client-serveur de la première génération.



- distinction entre deux tronçons de comm indépendants et délimités par serveur HTTP :
  - **circuit froid** : relie client au serveur Web pour permettre l'interaction avec l'utilisateur et la visualisation des résultats.
  - **circuit chaud** : permet collecte données ; peut évoluer sans impacter la configuration des postes clients

Architecture trois tiers : Le client léger *(Thin Client)*

- Ne prend en charge que
  - présentation de l'application avec, éventuellement
  - une partie de logique applicative permettant une vérification immédiate de la *saisie* et la *mise en forme* des données

- souvent constitué d'un simple *navigateur Internet*

- Client ne communique qu'avec la façade HTTP de l'application et ne dispose d'*aucune connaissance des traitements applicatifs ou de la structure des données exploitées*.

- Les évolutions de l'application sont donc possibles *sans nécessiter de modification de la partie cliente*.

  - Exemple de clients légers
    - poste Windows équipé d'un navigateur HTML
    - station réseau du type NC (Oracle, Sun et IBM)
    - terminal NetPC (windows)

Architecture trois tiers : Le client

Limites …

- Les pages HTML, même avec l'aide de langages de script, sont loin d'atteindre les possibilités offertes par l'environnement du client :
  - les pages affichées sont relativement *statiques*,
  - le développement *multi-plateforme* peut être contraignant
  - l'ergonomie de l'application est limitée aux possibilités du *navigateur*.
  - le *multi-fenêtrage* n'est pas facile à mettre en œuvre,
  - le déroulement de l'application doit se faire *séquentiellement*,

- Il est possible d'aller au delà des possibilités offertes par le langage HTML en y introduisant des **applets** Java ou des contrôles **ActiveX**.

- Java
  - Orienté objet, multi-plateforme
    - se prête bien à une utilisation sur Internet
      - JVM, Applet, API, JDBC, RMI, CORBA.

Architecture trois tiers : Les activeX

- réponse devant le phénomène Java (multi-plateforme)
- adaptation Internet des composants OCX constituant clef de voûte de l'archi DNA
- peuvent être *intégrés à une page HTML* mais, à la différence des applets Java, ils *persistent* sur le poste client après utilisation.
  - permet d'*optimiser les transferts de composants sur le réseau*
  - mais alourdissement du poste client, problèmes de *sécurité*, …
- Les composants ActiveX peuvent *communiquer* entre eux en utilisant la technologie *DCOM* ou avec des bases de données, en utilisant *ODBC*
- Profite de la richesse de *windows* mais rend les applications *dépendantes* de cette plateforme

- ActiveX dans application Intranet *vs* intérêt de cette architecture
  - ==> rappelle fortement le client-serveur deux tiers.

Architecture trois tiers : Le service applicatif

- architecture trois tiers
  - logique applicative prise en charge par le serveur HTTP.

(se retrouve dans la position du poste client d'une application deux tiers)

- Serveur HTTP peuvent mettre en œuvre technos basées sur :
  - **CGI** : mécanisme standard, mais grand consommateur de ressources,
  - **NSAPI** ou **ISAPI** : API de Netscape et Microsoft permettant l'écriture d'applications multi-thread intégrées au serveur HTTP,
  - scripts serveur comme **ASP** (*Active Server Page pour IIS*) ou **PHP** (*pour Apache*) sont interprétés par le serveur pour générer des pages dynamiquement,
  - **servlets Java** ou **JSP** : qui appliquent le mécanisme des applets aux traitements réalisés sur le serveur.

Architecture trois tiers : Limitations

- L'architecture trois tiers a corrigé les excès du client lourd
  - en centralisant une grande partie de la logique applicative sur un serveur
  - Le poste client, qui ne prend à sa charge que la présentation et les contrôles de saisie, s'est trouvé soulagé et plus simple à gérer.
- Par contre, le *serveur constitue la pierre angulaire de l'architecture et se trouve souvent fortement sollicité*
  - il est *difficile de répartir la charge* entre client et serveur.
  - On se retrouve confronté aux *problèmes de dimensionnement serveur* et de gestion de la montée en charge.
- Les contraintes semblent inversées par rapport à celles rencontrées avec les architectures deux tiers!!
  - *client est soulagé, mais serveur fortement sollicité …*

→ **Apparition des architectures n-tiers.**

Les architectures n-tiers : présentation

- pallier limitations architectures trois tiers & concevoir applis puissantes et simples à maintenir.
  - **distribuer** plus librement la **logique applicative**,
  - faciliter la **répartition de la charge** entre tous les niveaux.

- Mettre en œuvre une **approche objet** pour offrir
  - plus grande *souplesse d'implémentation*
  - faciliter la *réutilisation* des développements.
  - Capacités *d'extension*
  - permettre l'utilisation d'interfaces utilisateurs riches,
  - séparer nettement tous les niveaux de l'application,

Les architectures n-tiers : présentation

- L'appellation n-tiers pourrait faire penser que cette architecture met en œuvre un nombre indéterminé de niveaux de service, alors que ces derniers sont au maximum trois.
  - La ***distribution d'application entre de multiples services*** et non la multiplication des niveaux de service.

- **composants « métiers »** spécialisés et indépendants, introduits par les concepts orientés objets
  - composants métiers réutilisables
  - ➔ rendent service générique/réutilisable et clairement identifié.
  - sont capables de communiquer entre eux et peuvent donc *coopérer en étant implantés sur des machines distinctes*.

Les architectures n-tiers : présentation

➔ Distribution des services applicatifs sur plusieurs serveurs

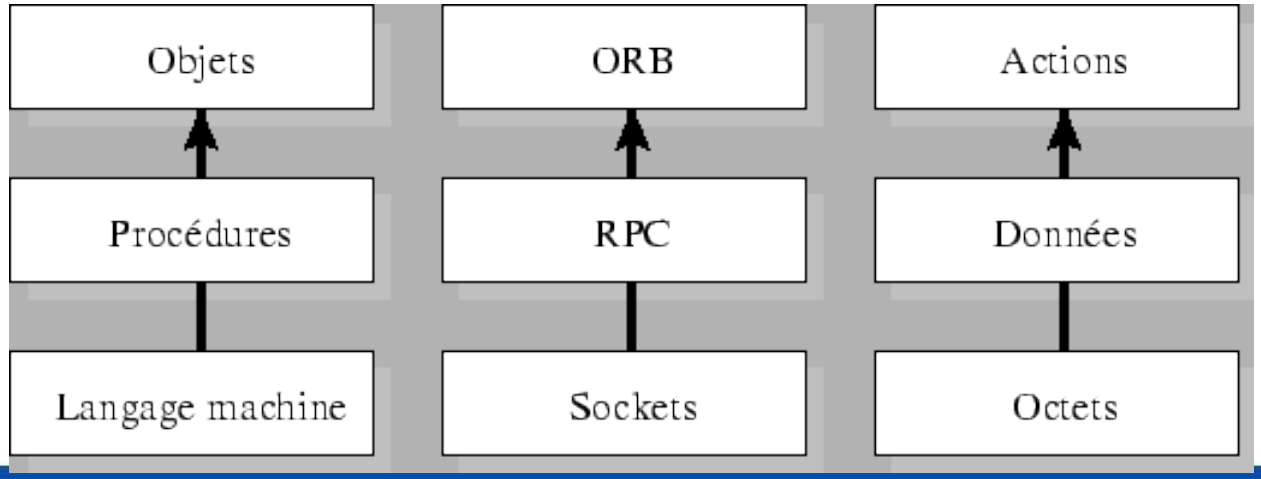
- facilite l'intégration de traitements existants dans les nouvelles applications....
  - Ex : connecter un programme de prise de commande existant sur le site de l'entreprise à une application distribuée en utilisant un middleware

➔ Objectif général

- évolutivité
- Maintenabilité
- Performances
  - Quantité de données stockée
  - Disponibilité du serveur
  - Nombre d'utilisateurs

Les architectures n-tiers : Le rôle de l'approche objet

- Les évolutions de l'informatique permettent de masquer la complexité des mécanismes mis en œuvre derrière une approche de plus en plus conceptuelle.
  - Programmation : langage machine ➔ OO (e.g., Java)
  - Protocoles réseau : couche physique ➔ ORB « Object Request Broker »
    - ensemble de fonctions (classes Java, bibliothèques C++...) qui implémentent un « bus logiciel » par lequel des objets envoient et reçoivent des requêtes et des réponses, de manière transparente et portable
      - permet une totale transparence des appels distants
        - » permet de manipuler un objet distant comme s'il était local
  - flux d'infos : octets ➔ messages



Plan

Introduction / Notions générales

De l'information centralisée au client-serveur

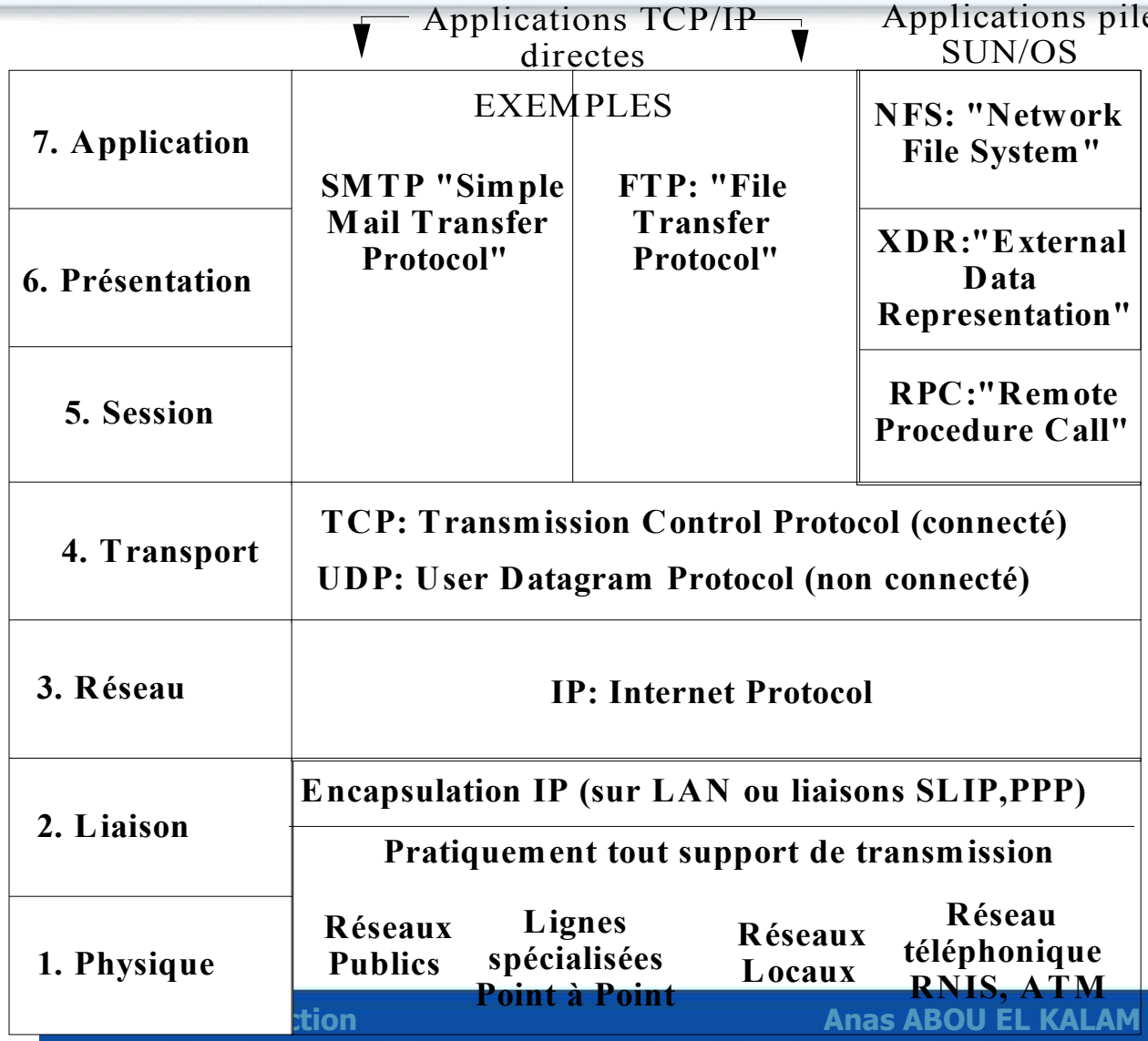
Les architectures distribuées

- **Application : architectures internet**

Architectures n-tiers pour le Web

- Rappels modèle OSI
- Les architectures 3-tiers classiques
- les architectures Web
- Le tiers client
- Le tiers Web
- Le tiers du milieu
- Le tiers ressource (EIS)

Couches OSI


Applications TCP/IP directes
Applications pile SUN/OS
7. Application
6. Présentation
5. Session
4. Transport
3. Réseau
2. Liaison
1. Physique
EXEMPLES
SMTP "Simple Mail Transfer Protocol"
FTP: "File Transfer Protocol"
NFS: "Network File System"
XDR:"External Data Representation"
RPC:"Remote Procedure Call"
TCP: Transmission Control Protocol (connecté)
UDP: User Datagram Protocol (non connecté)
IP: Internet Protocol
Encapsulation IP (sur LAN ou liaisons SLIP,PPP)
Pratiquement tout support de transmission
Réseaux Publics
Lignes spécialisées Point à Point
Réseaux Locaux
Réseau téléphonique RNIS, ATM

Couches OSI : niveau transport

- 1er des niveaux utilisable par user pour développer des applis C/S
- Réalise un service de transmission entre processus (transmission de bout en bout, "end to end").
- Selon les options de conception il assure:
  - Gestion des connexions.
    - protocoles en mode connecté
    - protocoles en mode non connecté.
- Négociation de qualité de service.
- Multiplexage/éclatement
- Contrôle d'erreur.
- Contrôle de flux.
- Contrôle de séquence.
- Segmentation.

Couches OSI ==> niveau transport ==> TCP

- Orienté connexion
  - permet à deux machines qui communiquent de contrôler l'état de la transmission.
- Remettre en ordre datagrammes en provenance du protocole IP
- Vérifier le flot de données afin d'éviter une saturation du réseau
- Formater les données en segments de longueur variable afin de les "remettre" au protocole IP
- Multiplexer les données
  - faire circuler simultanément des informations provenant de sources distinctes sur une même ligne
- Initialisation et fin d'une communication

Couches OSI ==> niveau transport ==> UDP

- transmission de données de manière très simple entre deux entités définies par @IP + N° port
- Mode non-connecté :
  - pas de moyen de vérifier si tous datagrammes envoyés sont bien arrivés à destination et ni dans quel ordre
- Non fiable : aucun contrôle de flux ni de congestion
- En revanche, pour un paquet UDP donné, l'exactitude du contenu des données est assurée grâce à un checksum

Couches OSI ==> niveau session

•structure et synchronise les dialogues point à point en mode message

- Synchronisation
  - qui peut émettre à tel moment
- gestion des « transactions »
  - mécanisme de correction des erreurs de traitement par restauration d'un état antérieur connu.

Remarques

• Une seule session peut ouvrir et fermer plusieurs connexions
- Pour travail : appel ami qui demande infos supplémentaires

• Plusieurs sessions peuvent se succéder sur la même connexion
- Coup de fil (avec votre épouse) à un de vos amis (avec sa femme)

Couches OSI ==> niveau session ==> RPC

- Le mode de communication du transport étant **le mode message asynchrone,** la session est définie pour offrir à l'usager un mode de dialogue de type synchrone.
- => Permettre à un usager d'exécuter une procédure ou une fonction sur un autre site:
- **Problème délicat :** Assurer en environnement réparti une sémantique pour l'appel de procédure distante voisine de celle connue en univers centralisé.

- Exemples : Systèmes d'objets répartis
  - **JAVA RMI** :"Remote Method Invocation"
  - **CORBA** :"Common Object Request Broker Architecture"
  - **WS-SOAP** :"Web Services - Simple Object Access Protocol"

Couches OSI ==> niveau présentation

- Traite du codage des données échangées
  - différents sites ayant des représentations différentes peuvent utiliser les données
- Exemple
  - "mots" = suite d'octets (mot de 32 bits=collection de 4 octets)
  - Intel ==> octets numérotés de droite à gauche
  - Motorola ==> de gauche à droite
    - Si machine à base Intel <==> machine à base Motorola ?

- ==> Les conversions de types : caractères, numériques, construits
- **Syntaxe abstraite**
  - analogue de syntaxe définition de types dans langage évolué
- **Syntaxe de transfert**
  - Représentation unique dans le réseau des valeurs échangées pour transférer les données

Couches OSI ==> niveau présentation ==> standards de conversion

Réseaux publics

Syntaxe abstraite ASN1: **X208**

Syntaxe de transfert :**X209**

SUN-OS/ Internet

XDR : "eXternal Data Representation".

Systèmes d'objets répartis CORBA

IDL : "Interface Definition Language".

CDR : Common Data representation.

Web services

WSDL : "Web Services Definition Language".

RPC/Encoded : "Syntaxe de transfert XML"

Couches OSI ==> niveau présentation

- Compression des données
  - Transferts avec application d'algorithmes de compression
  - e.g., son, l'image ou la vidéo...
  - Dans OSI, ces algos sont fournis par couche présentation
- Sécurité
  - Cryptographie, ...
- ...

Couches OSI ==> niveau application

•cette couche propose également des services fournir à l'utilisateur des fonctions dont il a besoin couramment

•Principalement des services de transfert de fichiers, (FTP), de messagerie (SMTP) de documentation hypertexte (HTTP) etc.

Accès aux informations distantes : le WWW

- Objectif
  - Permettre l'accès à des données distantes pour des personnes.
- Bases
  - Systèmes de désignation **URL** ("Uniform Resource Locator"),
  - Protocole de communication **HTT**P (Hyper Text Transfer Protocol"),
  - Initialement format **HTML=>** format **MIME** Extension

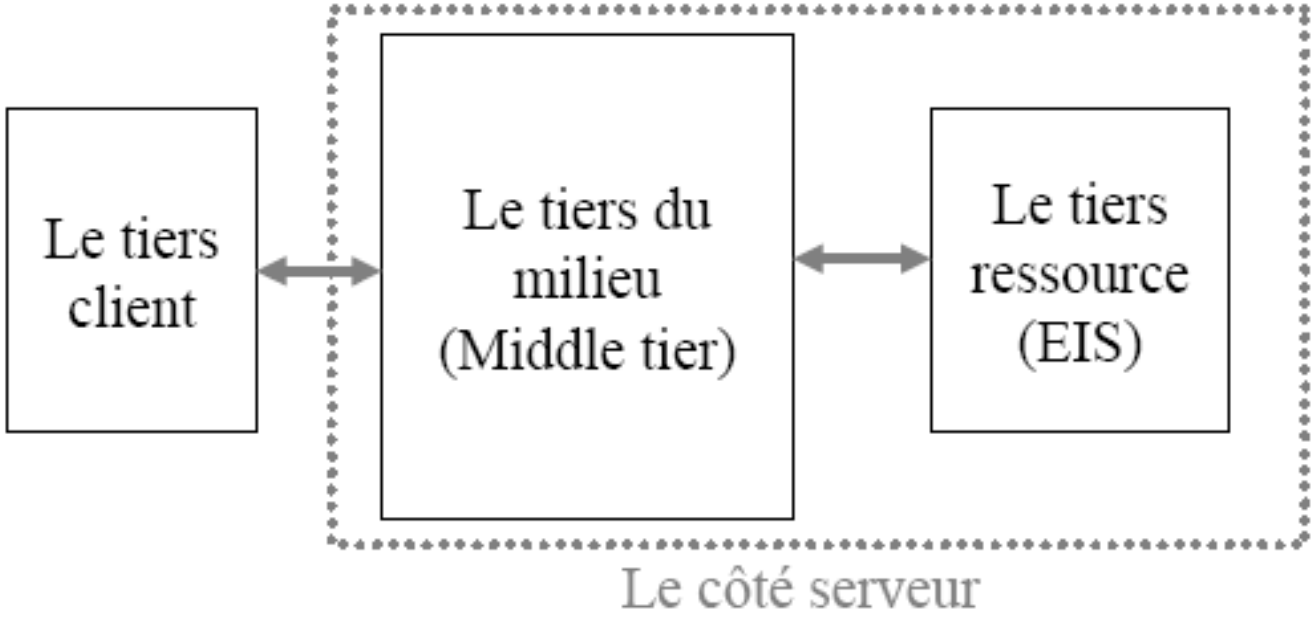
==> format **XML**

  - Extension : permettre communications entre programmes: notion de services web
    - **SOAP** ' Simple Object Access Protocol',
    - **WSDL** 'Web Services Definition language'
    - **UDDI** 'Universal Description Discovery and Integration'

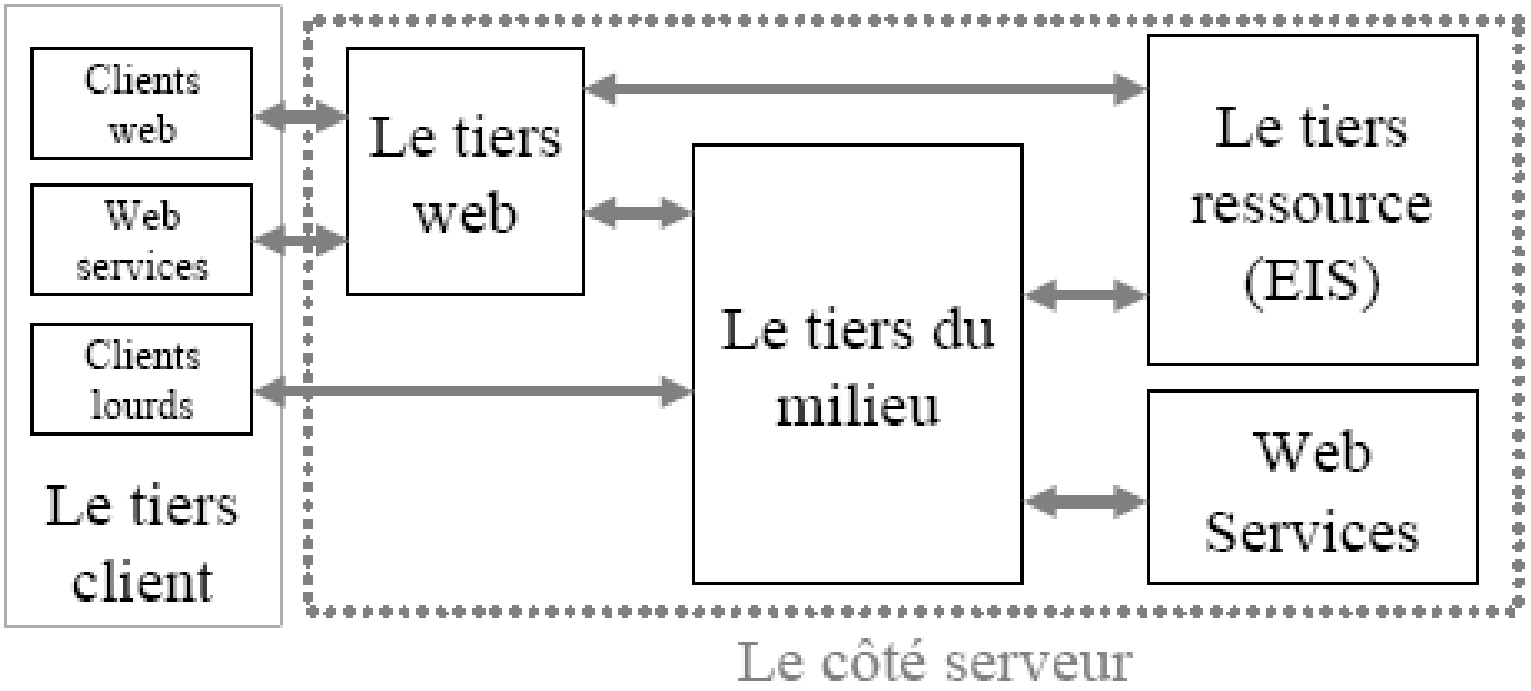
Architectures n-tiers pour le Web

- Rappels modèle OSI
- Les architectures 3-tiers classiques
- les architectures Web
- Le tiers client
- Le tiers Web
- Le tiers du milieu
- Le tiers ressource (EIS)

Rappel : architectures 3-tiers classiques

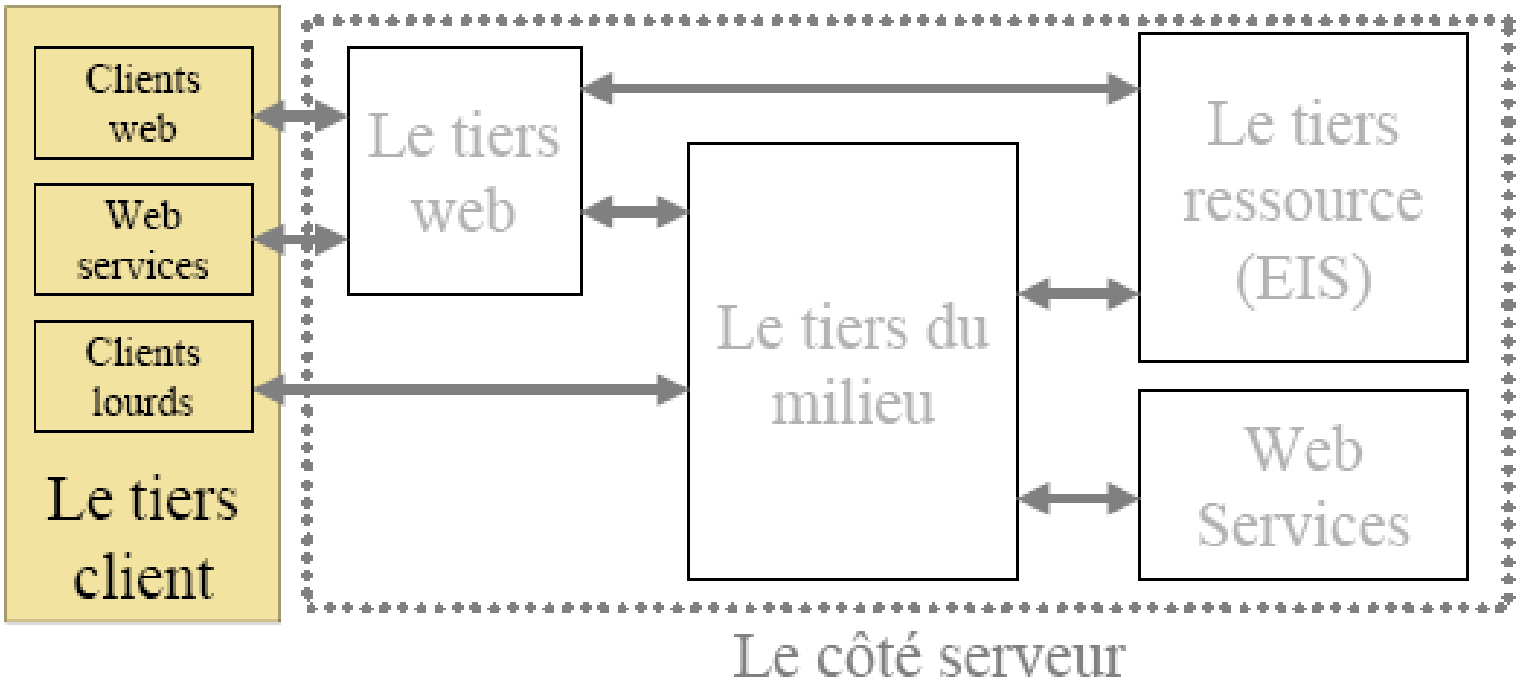


Les architectures web



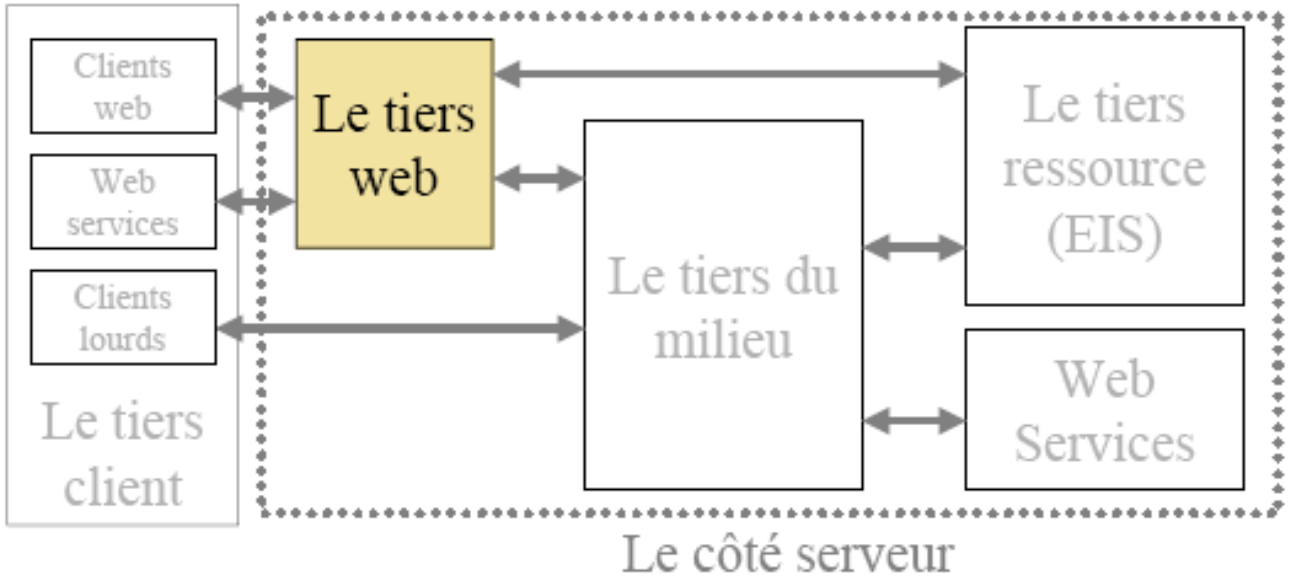
Les architectures web : le tiers client

- web browser
- PDA
- client lourd (fat client), applets, apps
- Web-service



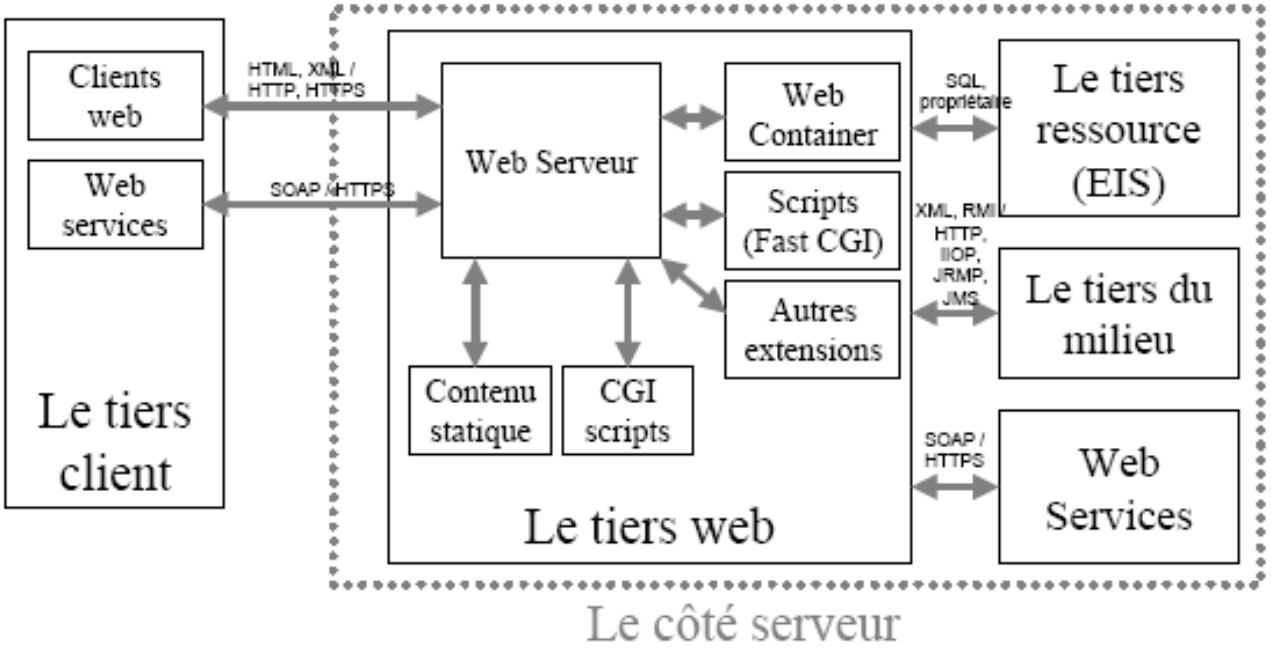
Les architectures web : le web tiers

- reçois les requêtes HTTP des clients et renvois les réponses
- permet la séparation entre présentation et « business logic »
- génère du contenu dynamiquement
- transforme des requêtes HTTP dans un format compris par l'application
- contient la logique du flot de présentation
- identifie la session de l'utilisateur
- supporte plusieurs types de clients



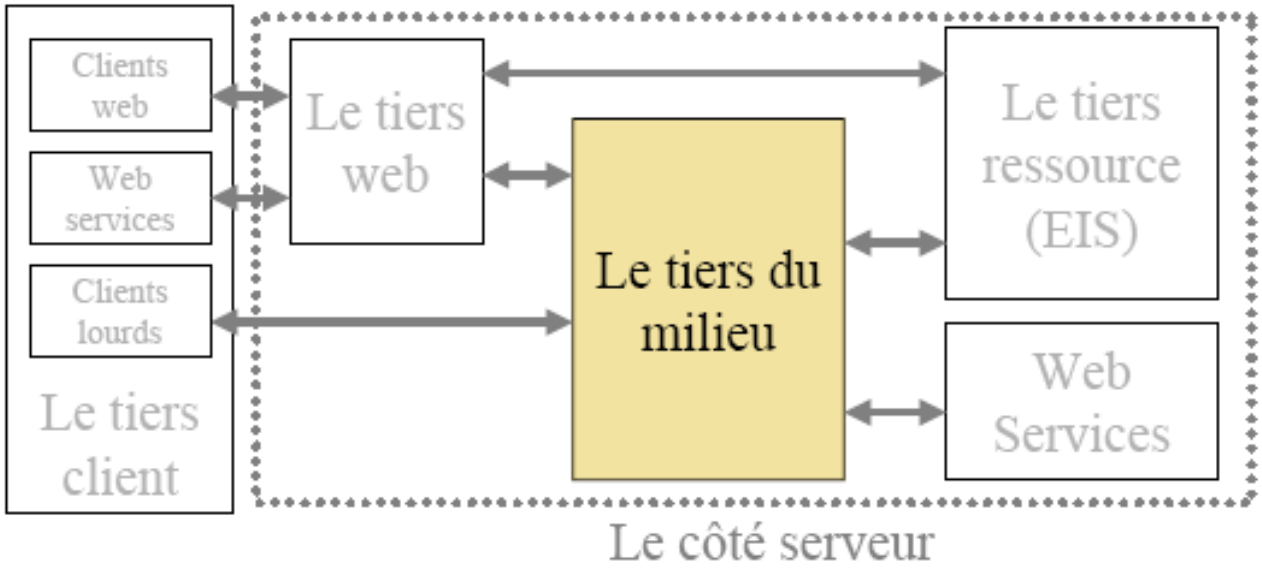
Le tiers web : Technoligies & architecture

- CGI/FastCGI
- ASP
- Java Servlets
- JSP
- PHP, Python
- JavaScript



Les architectures web : le tiers du milieu

- Gestion de composants
- Tolérance de fautes, haute disponibilité
- Passage à l'échelle
- Balance de charge
- Ressources spooling
- Transaction Management
- Console de management
- Sécurité



Les architectures web : le tiers ressource

Base de données (databases)

- JDO, SQL/J, JDBC, ADO.NET
- ERP (Enterprise Resource Planning) & EAI (Enterprise Application Integration)
- J2EE Connector, protocoles propriétaires

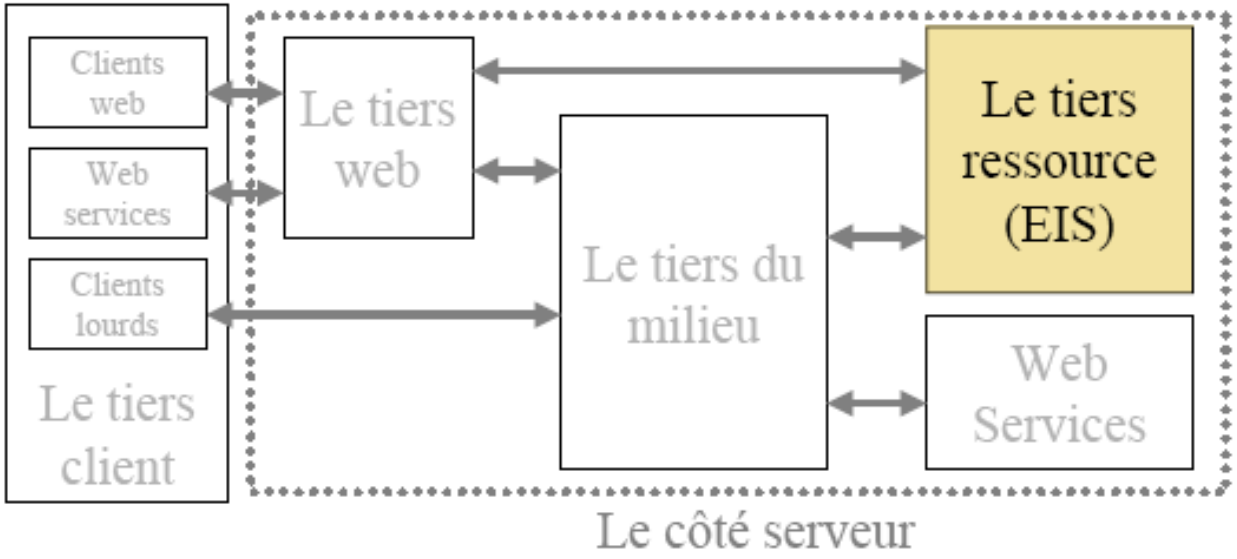


Plate-formes

Deux mondes/logique ….

Plan

- De l'information centralisée au client-serveur
- Les architectures distribuées
- Application : architectures web

- **Plate-forme .NET**

- Plate-Forme J2EE
- Les services Web

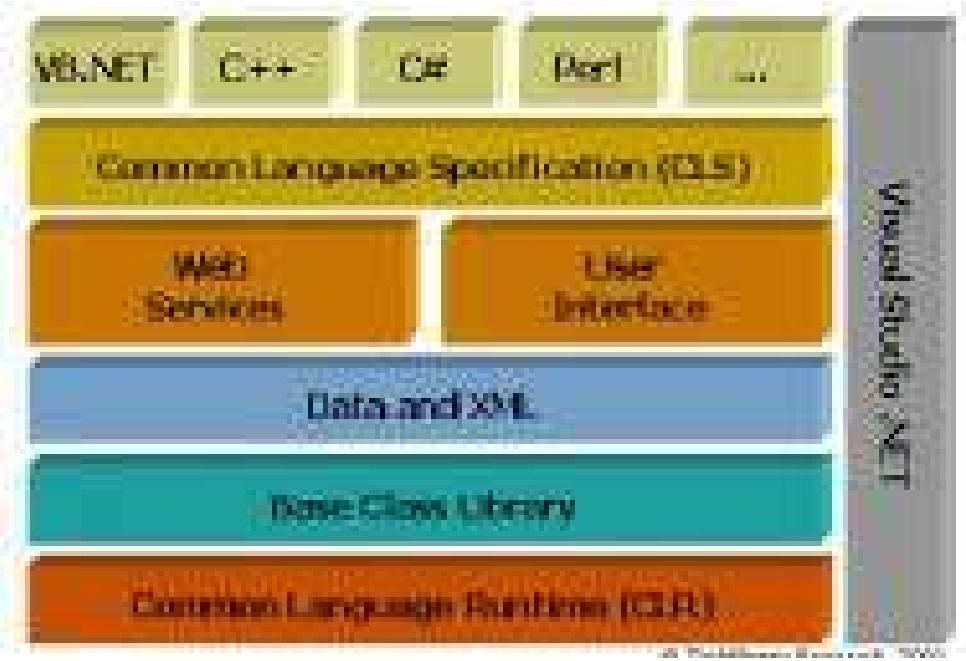
. NET

➔Technologie/spécification Microsoft

- Présentation : IE
- Serveur HTTP / FTP : IIS
- Communication : COM, MSMQ, COM+
- Composants : COM, MTS
- Business <-> Data : ActiveX Data Object, ODBC
- Accès aux données : OLEDB
- Persistance : SQL Server, Exchange, Active Directory et NTFS
- …

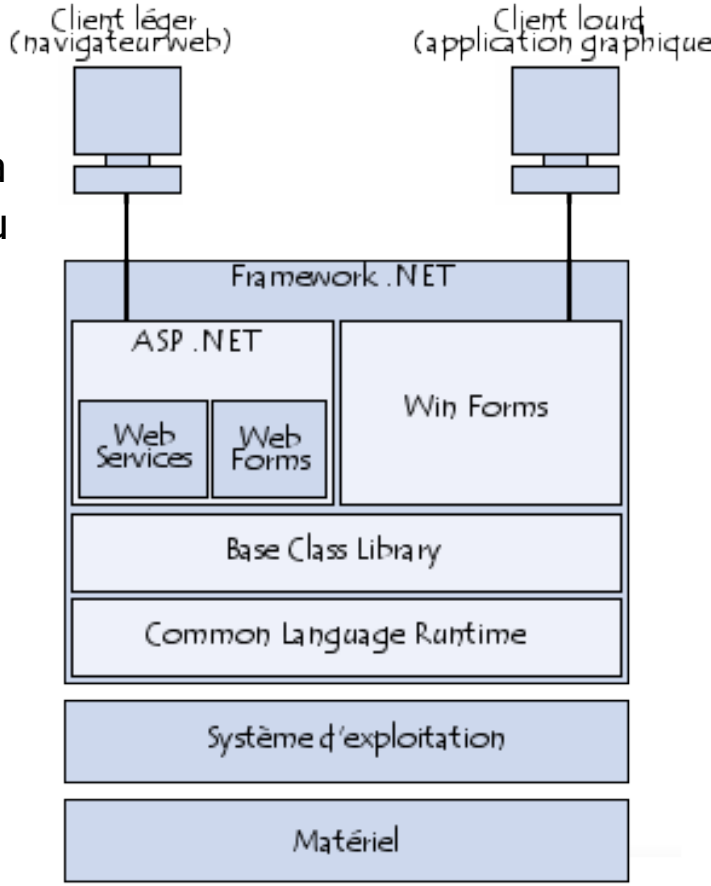
. NET

- Réponse de Microsoft à J2EE
  - pour Dvpt d'applis d'entreprises multi-niveaux, basées sur des composants
- correspond à un ensemble Microsoft qui comporte un framework, des langages de développement, des spécifications techniques et des systèmes (Systèmes d'exploitation, Logiciels serveurs...)
- 3 parties
  - CLR
  - BCL
  - ASP.NET
- CLS
- CTS
- MSIL



Framework .NET

- **environnement d'exécution**
  - ***CLR*** : moteur d'exécution,
    - compiler le code source en **MSIL** (*MS Intermediate Language*)
    - 1ère exécution, code MSIL est compilé à la volée en code spécifique au système grâce au compilateur JIT (*Just In Time*).
  - ***ASP .NET*** : environnement d'exécution d'applications et de services web
  - ***WinForms*** : environnement d'exécution d'applications lourdes.
- **Services**
  - ***FCL*** : (**Framework Class Library**) : librairie OO
  - ***SDK*** (Software Development Kit) : implémentation de ces classes



ASP.NET / WebForms

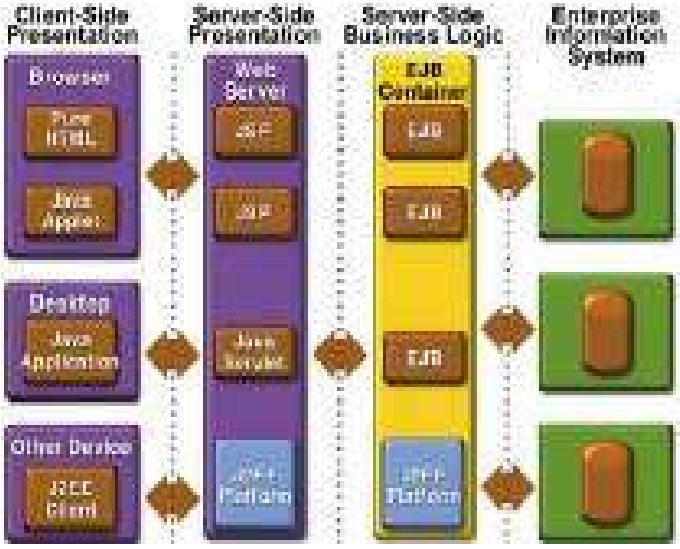
- **ASP.NET est une abstraction de HTTP**
  - ***Pages compilées et exécutées dans le CLR***
    - Modèle de développement basé sur les WebForms …
  - ***Séparer traitements et présentations***
    - formulaire représente la page web
    - Traitements contenus dans 2de page appelée « code behind »
    - Cette page peut être codée dans n'importe quel langage du Framework .Net et implémente événements …
    - Page finale (générée au client) intègre présentation et Code « Behind »

Plan

- De l'information centralisée au client-serveur
- Les architectures distribuées
- Application : architectures web
- Plate-forme .NET
- **Plate-Forme J2EE**
- Les services Web

J2EE

- **Standard international**
- **Une appli J2EE assemble des composants**
  - ***Composants client*** : appelts
  - ***Composants Web*** : Servlets, JSP
  - ***Composants Business*** : EJB
- Notion de ByteCode
- API : EJB, JDBC, Java Servlet Technology, JSP Technology, JMS, JTA, JavaMail Technology, JAF, JAXP, JAAS, ….



Architecture n-tiers : Les Java Beans

- **Les Java Beans**
  - **composant logiciel réutilisable**.
  - notion large englobe aussi bien un simple bouton qu'une appli complète.
  - Concrètement, les Java Beans sont des **classes Java utilisant des interfaces particulières**.
  - peut être utilisé indépendamment, sous la forme d'une simple applet, ou intégré à un développement Java
  - Les beans sont prévus pour pouvoir **inter-agir avec d'autres beans** au point de pouvoir développer une application simplement en assemblant des beans avec un outil graphique dédié.
    - Sun fournit gratuitement un tel outil : B.D.K. (*Bean Development Kit*)

Architecture n-tiers : Les Java Beans

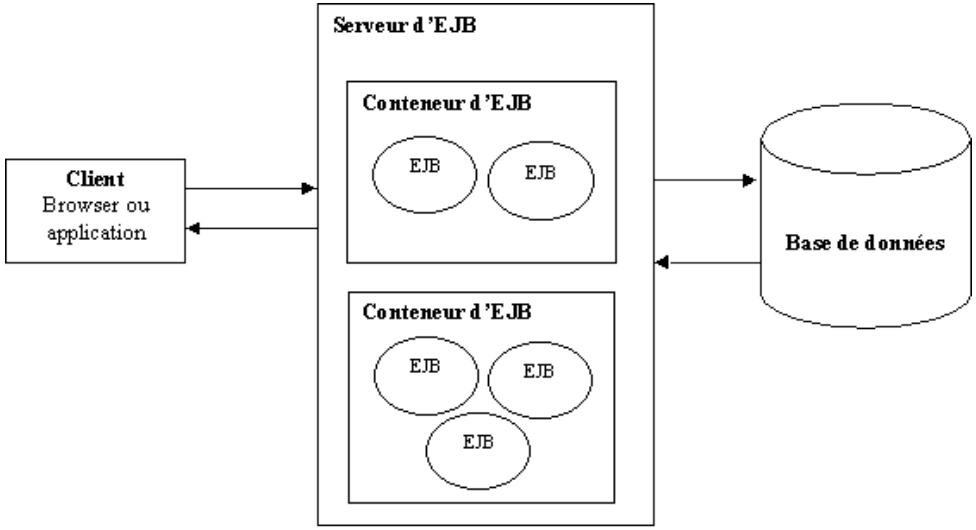
- caractéristiques :
  - persistance : elle permet grâce au mécanisme de *sérialisation* de *sauvegarder* l'état d'un bean pour le *restaurer* ainsi si on assemble plusieurs beans pour former une application, on peut la sauvegarder.
  - communication grâce à des *événements* qui utilise le modèle des *écouteurs*
  - peut dévoiler son comportement à ses futurs utilisateurs à l'aide :
    - *propriétés* qu'il expose et rend accessible à l'aide d'*accesseurs*,
    - des *méthodes* qu'il permet d'*invoquer*, comme tout objet Java,
    - des *événements* qu'il peut générer pour *avertir d'autres composants*.
  - introspection : ce mécanisme permet de *découvrir de façon dynamique* l'ensemble des *éléments* qui compose le bean (attributs, méthodes et événements) sans avoir le source.
  - possibilité de paramétrer le composant : les données du paramétrage sont conservées dans des propriétés.
  - sont livrés sous forme de fichiers JAR

Architecture n-tiers : Les EJB

- but : **faciliter la création d'applications distribuées** pour les entreprises.
- Les EJB sont des composants et en tant que tel, ils possèdent certaines caractéristiques comme la **réutilisabilité**, la possibilité de **s'assembler** pour construire une application, etc ...
- Les EJB et les beans n'ont en commun que d'être des composants.
  - Les java beans sont des composants qui peuvent être utilisés dans toutes les circonstances.
  - Les **EJB doivent obligatoirement s'exécuter dans un environnement serveur dédié.**
  - Les EJB **ne comportent pas forcément de partie visible**,
- Les EJB permettent aux développeurs de *se concentrer sur traitements orientés métiers*
  - Les *EJB et l'environnement dans lequel ils s'exécutent prennent en charge certain traitements* : gestion des transactions, persistance des données, ...
  - peu*vent communiquer avec des Java Beans côté client de façon indépendante du protocole* (IIOP, RMI, DCOM),

Architecture n-tiers : Les EJB

- Les EJB sont adaptés pour être intégrés dans une architecture 3-tiers ou n-tiers.
- Les EJB *s'exécutent dans un environnement particulier* : le **serveur d'EJB**
  - fournit des fonctionnalités utilisées par un ou ++ conteneurs d'EJB qui constituent le serveur d'EJB.



- *serveurs d'EJB commerciaux*
  - BEA Weblogic, IBM Webpshere, Sun IPlanet, Macromedia JRun, Borland AppServer,...
- *serveurs d'EJB open source* :
  - JBoss, Jonas

Architecture n-tiers : Les Microsoft Transaction Server : (MTS)

COM : Components Objects Model

- mécanisme au niveau du système d'exploitation dans lequel il est possible d'installer des composants serveurs afin de fournir des services variés à des logiciels clients
- développement de logiciel est facilité, puisqu'il est possible d'écrire un programme en faisant appel à des traitements effectués par un serveur COM développé par un tiers.

MTS

- Serveur destinée à simplifier la gestion et le suivi des composants distribuées (COM)
- Fonctionnalités associées aux composants : prise en charge automatique des
  - transactions pour protéger l'intégrité des données,
  - sécurité à base de rôles,
  - accès aux bases de données, aux logiciels de mise en file d'attente des messages.
  - optimisation des performances telles que les pools de connexion.

Communication entre objets : les RMI

- mettre en œuvre facilement des objets distribués
  - permettre l'*appel*, l'*exécution* et le *renvoi* du *résultat d'une méthode exécutée dans une machine virtuelle différente* de celle de l'objet l'appelant.
- Serveur : machine sur laquelle s'exécute la méthode distante
- Client : appel méthode
  - à obtenir une **référence** sur l'objet distant
  - appeler la méthode à partir de cette référence.
  - **Stub**: représentation locale de l'interface de l'objet serveur (**Skeleton**)

- Un objet distribué se caractérise par son *interface* et son *adresse* (*URL*).

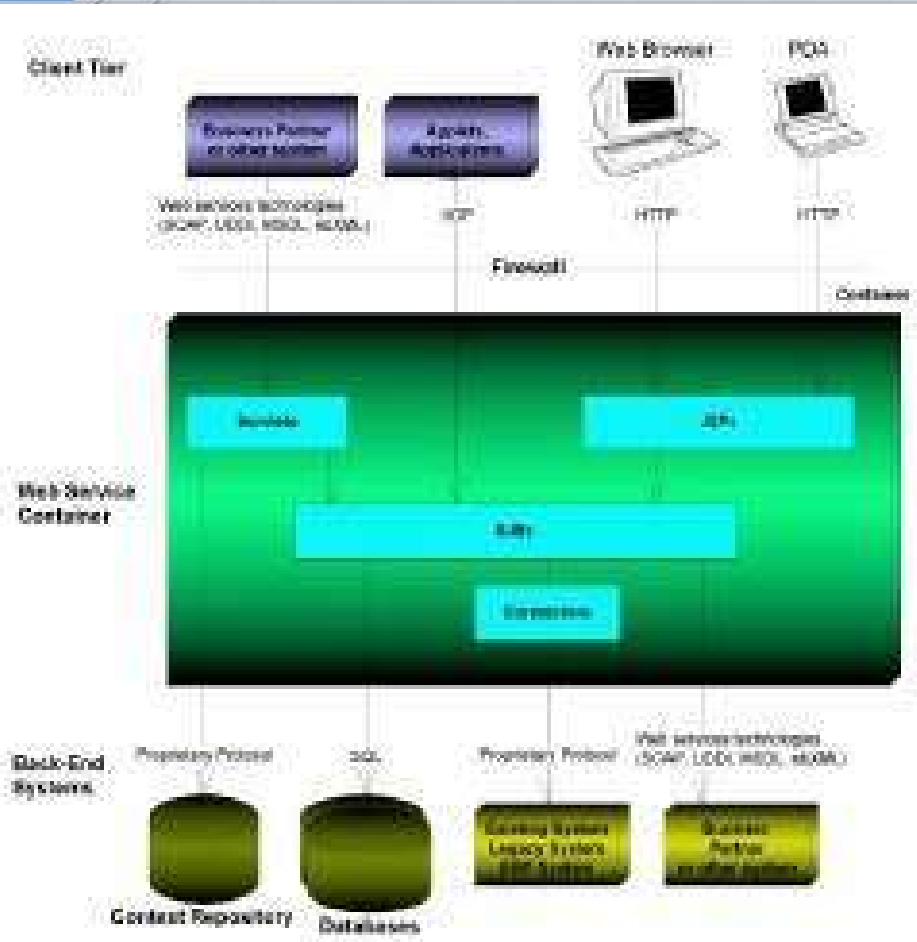
Communication entre objets : CORBA

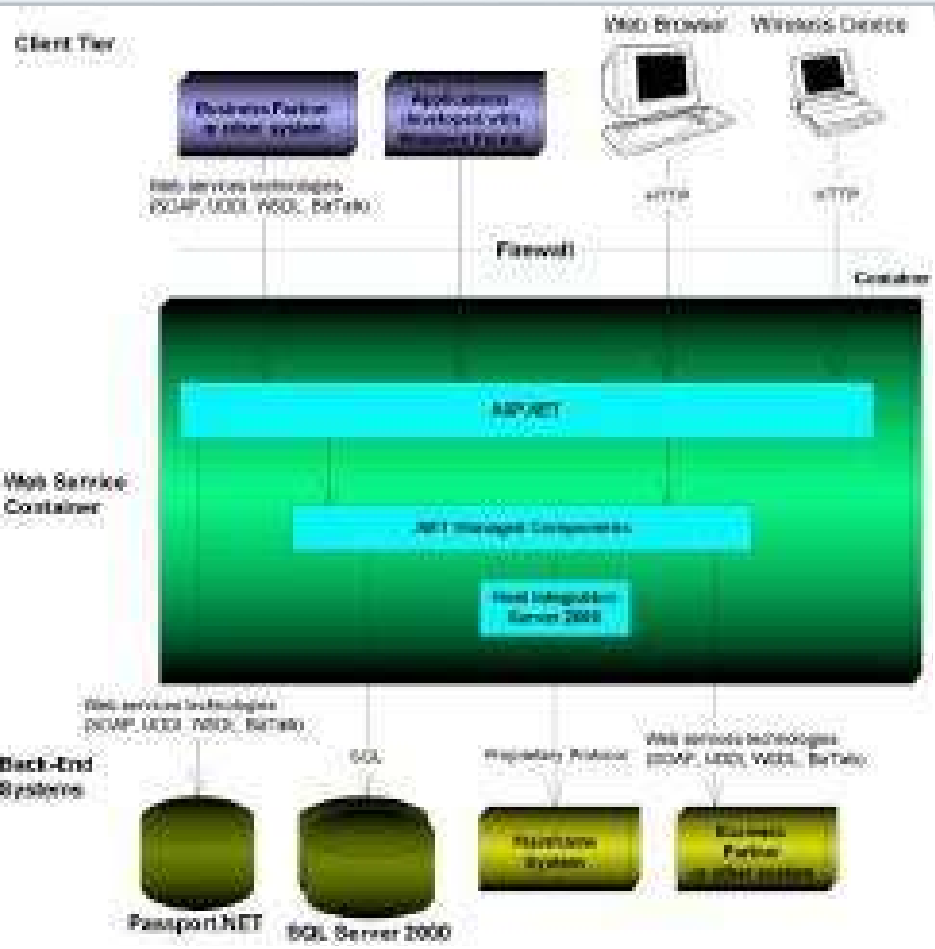
- Standard défini par l'OMG pour faire communiquer des objets, au sens logiciel du terme, quels que soient le langage de programmation utilisé et la machine sur laquelle s'exécute le programme.

- Une application accède à un objet distant en utilisant une télécommande locale, appelée *proxy*. Ce proxy lui permet de déclencher les méthodes de l'objet distant à l'aide de primitives décrites avec le langage IDL.

- L'OMG effectue toute une série de recommandations connues sous le nom de CORBA services visant à proposer des interfaces génériques pour chaque type de service usuel (nommage, transaction, cycle de vie, ...).

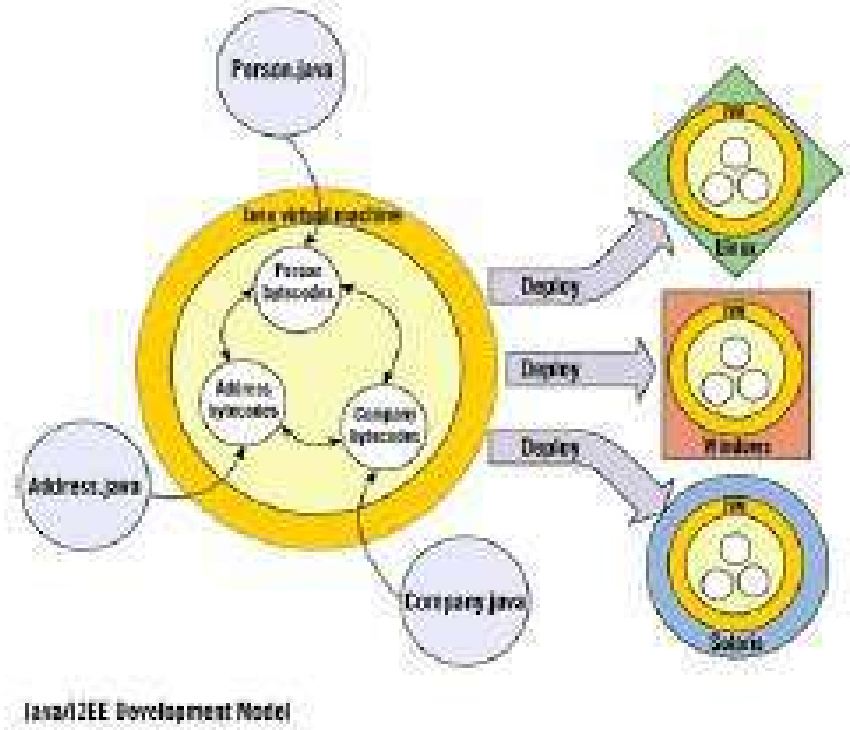
Comparaison J2EE et .NET





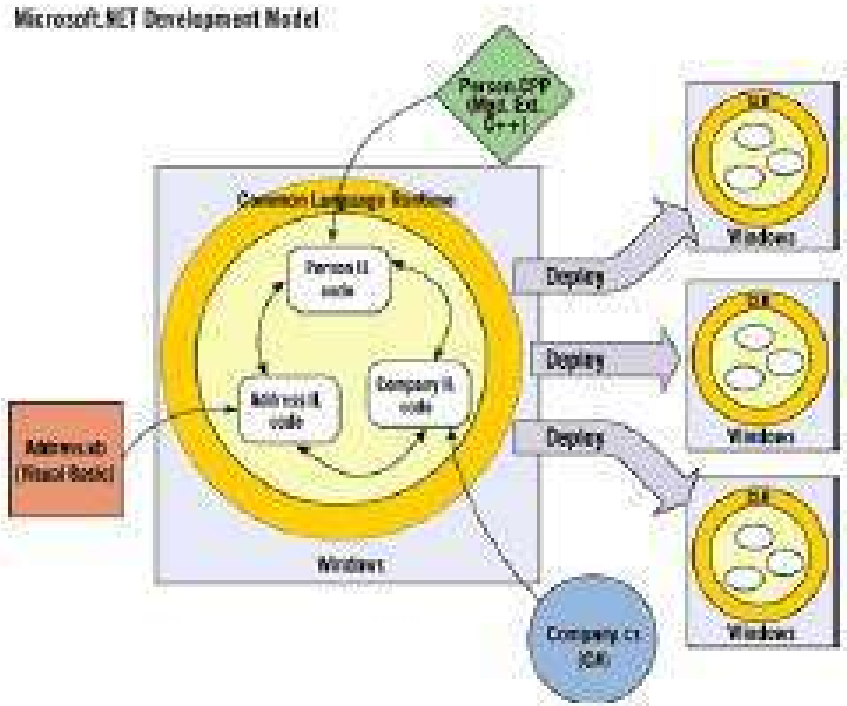
Comparaison J2EE et .NET

Un langage
Plusieurs plate-formes



Java/J2EE Development Model

Plusieurs langages
Une plate-forme



Comparaison J2EE et .NET

	Microsoft. NET	J2EE	différences essentielles
Langage	C#, Multi-Langage	Java	C# a certains des JavaBeans et ajoute les metadata tags. L'intégration dans la syntaxe est différente. J2EE est plate-forme indépendant mais langage spécifique, .NET est langage indépendant mais plate-forme spécifique.
Services	BCL	Java core API	Similaire services
Présentation	ASP.NET	Servlet JSP	ASP.NET utilise tout les langages supportes dans .NET et est compile en code natif par le CLR. JSPs utilisent Java code (snippets, ou JavaBean références), compile en bytecodes.
Interprète	CLR	JVM	CLR permet a du code de plusieurs langages d'utiliser un ensemble de composants partages.
GUI composants	Win Forms Web Forms	Swing	Composants Web similaire ne sont pas disponible en Java. WinForms et WebForms sont complètement intègre a VisualStudio .net
DB accès	ADO.NET	JDBC, JDO, SQLJ	ADO.NET est construit a partir d'une architecture XML
WebServices	oui	oui	.NET web services supposent un model de message base sur SOAP tandis que J2EE laisse le choix au developpeur.
Implicit middleware	oui	oui	
Technologie	Produit	Standard	J2EE est une specification, .NET est une strategie de produits

Comparaisons

	JSP	PHP	ASP	Asp.Net
Langage	Java	PHP	VBScript ou JScript	Tous les langages supportés par .Net (C#, VB.Net, Delphi, ...)
mode d'exécution	Compilé en pseudo code (byte code)	Interprété	Interprété	Compilé en pseudo code (MSIL)
principaux avantages	Repose sur la plate-forme Java dont elle hérite des avantages	Open source Nombreuses bibliothèques et sources d'applications libres disponibles Facile à MEV	Facile à mettre en oeuvre	Repose sur la plate-forme .Net dont elle hérite des avantages Wysiwyg et evénementiel Code behind pour séparation affichage / traitements
principaux inconvénients	Débogage assez fastdieux	Débogage assez fastdieux	Débogage fastdieux Beaucoup de code	Fonctionne essentiellement sur plate-formes Windows. (Voir le

Comparaison J2EE et .NET

- Architecture n-tiers à base d'objets est le grand GAGNANT !
  - De nos jours, l'important n'est pas de maîtriser Java, VB, ou C#, mais de comprendre en quoi ces langages sont utiles pour implémenter des archi solides, évolutives et maintenables.

- **Intégration à l'existant : 1 point partout, la balle au centre**
  - Choisir .NET quand tout le système informatique repose sur un environnement Microsoft - et inversement - c´est possible
    - J2EE → connecteur universel (JCA)
    - . NET → Web services

Comparaison J2EE et .NET

- **Le pragmatisme de Microsoft face à l'ouverture de J2EE**
  - Un développeur Visual Basic passera sans difficulté à .Net. Mais plus difficilement à Java, l´inverse étant également vrai
  - *chez Microsoft, tout est packagé : à un problème correspond une solution*
  - *sur J2EE, ∃ souvent ≠ solutions, parfois sous forme de briques à assembler*
    - *L´approche .Net est donc plus simple et adaptée à la PME.*
      - *Mais la plate-forme manque de maturité, Microsoft ne couvre pas tous les besoins d´une PME*
      - *faire le choix .Net, c´est accepter de dépendre de Microsoft là où …*
    - *avec J2EE on a le choix entre différents éditeurs et mêmes des solutions gratuites ou quasiment gratuites en Open Source".*
  - *argument avant tout marketing : .Net accepte certes plus de 25 langages de développement là où J2EE n´accepte que Java*
    - *Mais je vois mal un développeur Cobol développer pour .Net. Ce n´est pas impossible, mais la culture, les architectures, les applications... tout est différent avec .Net ou J2EE*

Comparaison J2EE et .NET

- **.Net moins cher sur le court terme, mais après ?**
  - *Microsoft : solutions étant plus packagées, elles nécessitent moins de développements, moins de ressources humaines*
    - *la phase d´acquisition des compétences est plus courte, la mise en oeuvre est plus rapide et tout cela coûte forcément moins cher que l´assemblage de briques de J2EE.*
      - *Le retour sur investissement est donc plus rapide sur le court terme.*
      - *Avec le temps, les choses s´équilibrent et …*
      - *c´est même plus délicat sur le long terme, car il n´est pas impossible que la phase initiale de J2EE, certes plus fastidieuse, s´avère plus avantageuse sur le long terme".*

Architecture cohérente

Architecture n-tiers

- vision cohérente du système d'information.
- Tous les services sont représentés sous la forme d'objets interchangeables qu'il est possible d'implanter librement, en fonction des besoins.
- potentiel très important et permettrait utilisation objets métiers réutilisables
- moindre coût et évolution en fonction des besoins

- à voir …
  - comment il sera utilisé,
  - à quel coût
  - comment sera gérée la transition depuis les environnements actuels.

Récapitulatif & exemples

- **Architecture 1-tiers**
  - Application mainframe
- **Architecture 2-tiers**
  - C/S avec SGBDR et middleware orienté message, transactionnel ou objet
- **Architecture 3-tiers**
  - Clients légers, serveurs web, serveurs d'application et middleware
- **Architecture n-tiers**
  - Répartition de charge sur différents serveurs, mise en œuvre de fermes de serveurs, …

Plan

- De l'information centralisée au client-serveur
- Les architectures distribuées
- Application : architectures web
- Plate-forme .NET
- Plate-Forme J2EE

- **Les services Web**

Services web

Pourquoi ?

- L'intégration de modules distribués est très difficile à cause de l'hétérogénéités des systèmes
- dialoguer à distance via Internet, indépendamment des plates-formes et des langages sur lesquelles elles reposent
  - CORBA était une solution mais sa complexité a freiné son développement

Quoi ?

- composant logiciel représentant une fonction applicative.
- peut être accessible depuis une autre application
  - client, serveur ou autre service Web
- Utilise réseaux Internet & protocoles de transports disponibles.
- peut être implémenté comme une application autonome ou comme un ensemble d'applications
- Basés sur XML
- Protocoles simples et standards

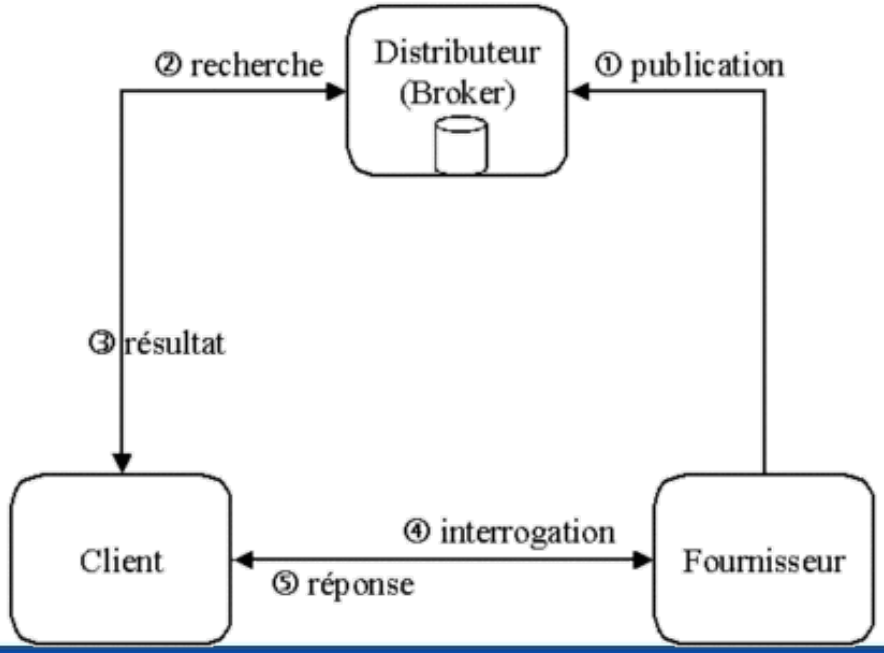
Services web

Comment ?

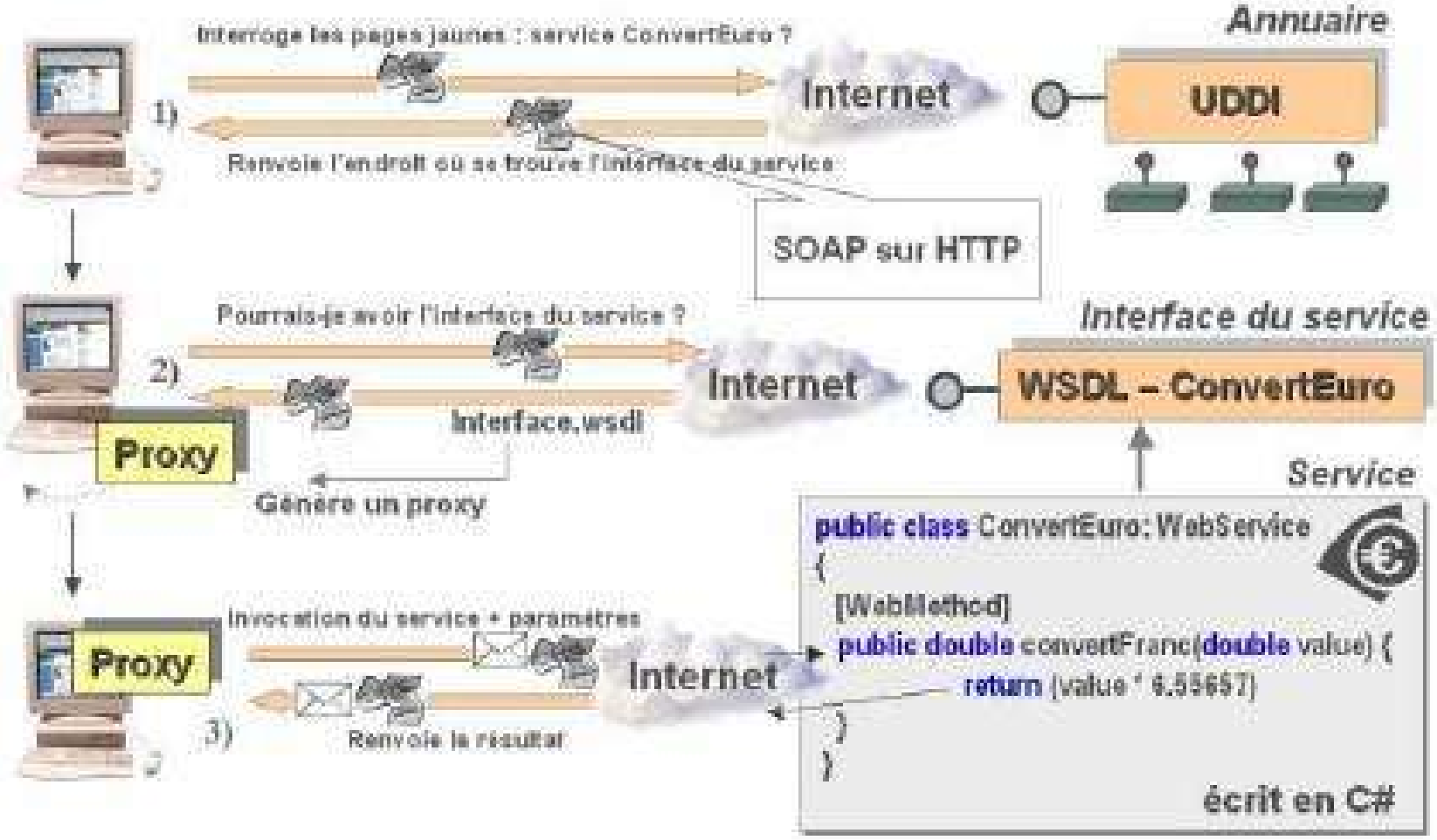
➔ protocoles standardisant les modes d'invocation mutuels de composants applicatifs

- Couche de transport
  - *HTTP, FTP ou SMTP, …*
- Messages XML
  - *SOAP*
- Description des services
  - *WSDL*
- Recherche de service
  - *UDDI*

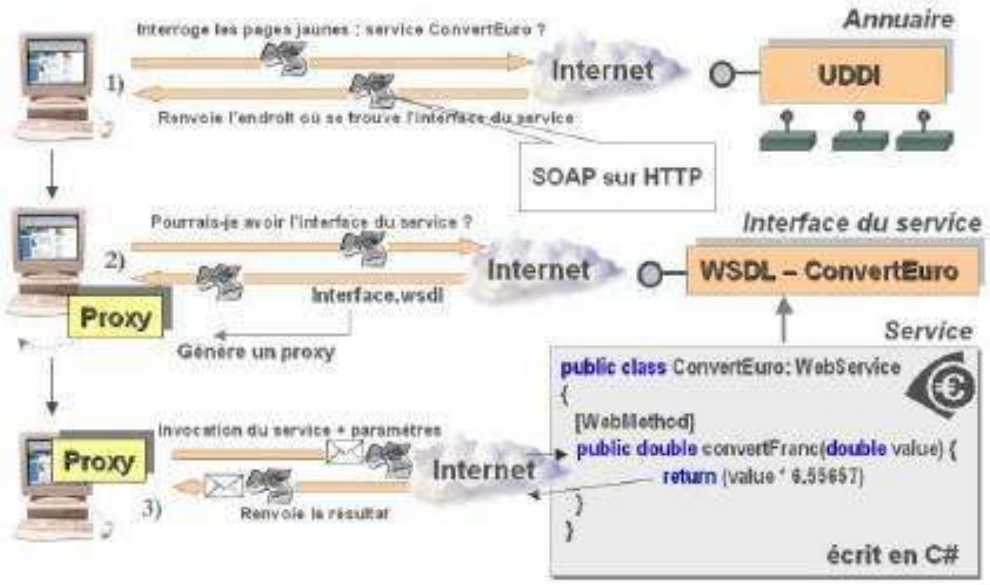
Scénario



Exemple

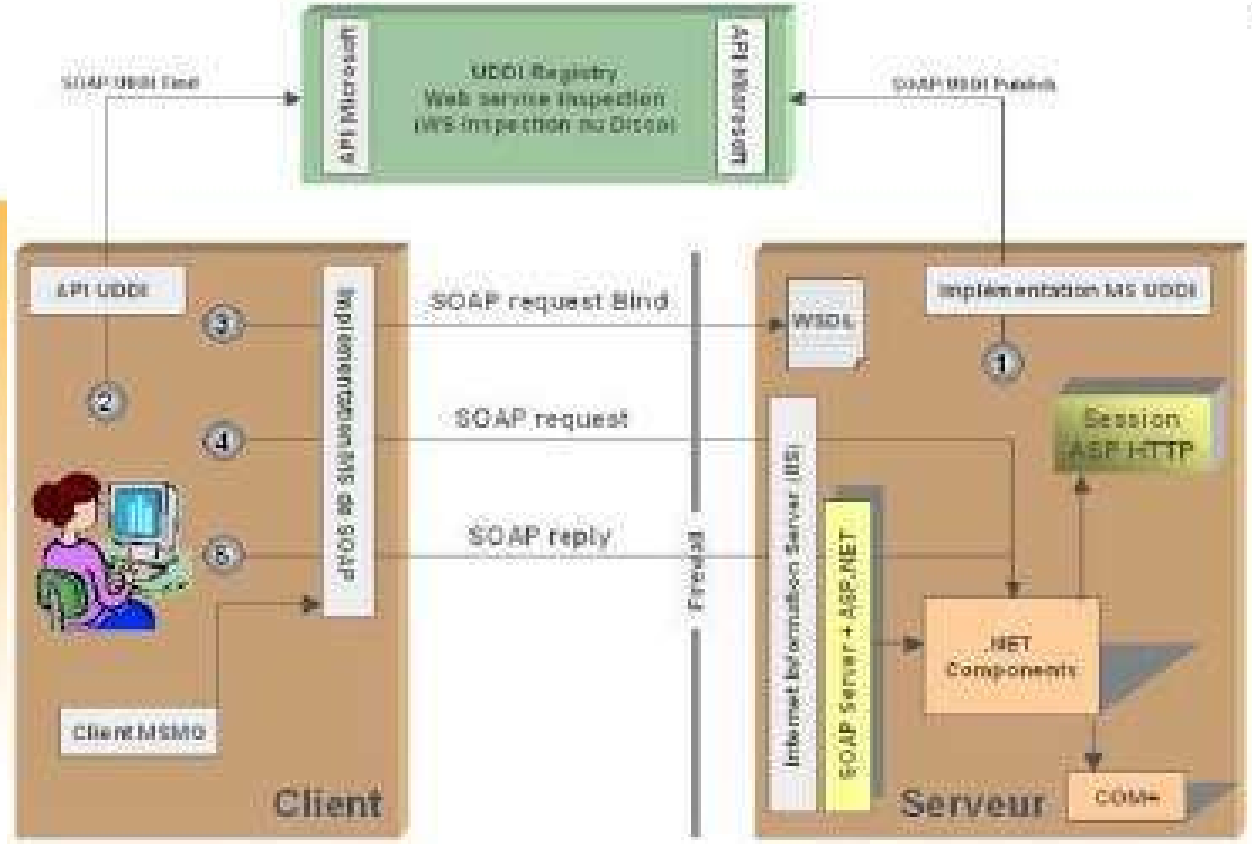

Annuaire
Interroge les pages jaunes : service ConvertEuro ?
1)
Internet
UDDI
Renvoie l'endroit où se trouve l'interface du service
SOAP sur HTTP
Interface du service
Pourrais-je avoir l'interface du service ?
2)
Internet
WSDL – ConvertEuro
Interface.wsdl
Proxy
Génère un proxy
Service
public class ConvertEuro: WebService
{
[WebMethod]
public double convertFranc(double value) {
return (value * 6.55957)
}
}
écrit en C#
Invocation du service + paramètres
Proxy
Internet
3)
Renvoie le résultat

Exemple



- Client
  - Demande service
  - Fait recherche sémantique dans annuaire UDDI qui donne la liste des prestataires habilités à répondre à la requête
  - Réception réponse (en XML)
  - Recherche interface du composant référencé dans annuaire
- Interface WSDL
  - Décrit l'ensemble des services implémentés par l'objet distribué et il est possible de vérifier si l'interface correspond à la demande
- Proxy SOAP
  - invocation service

L'architecture Web Services .NET

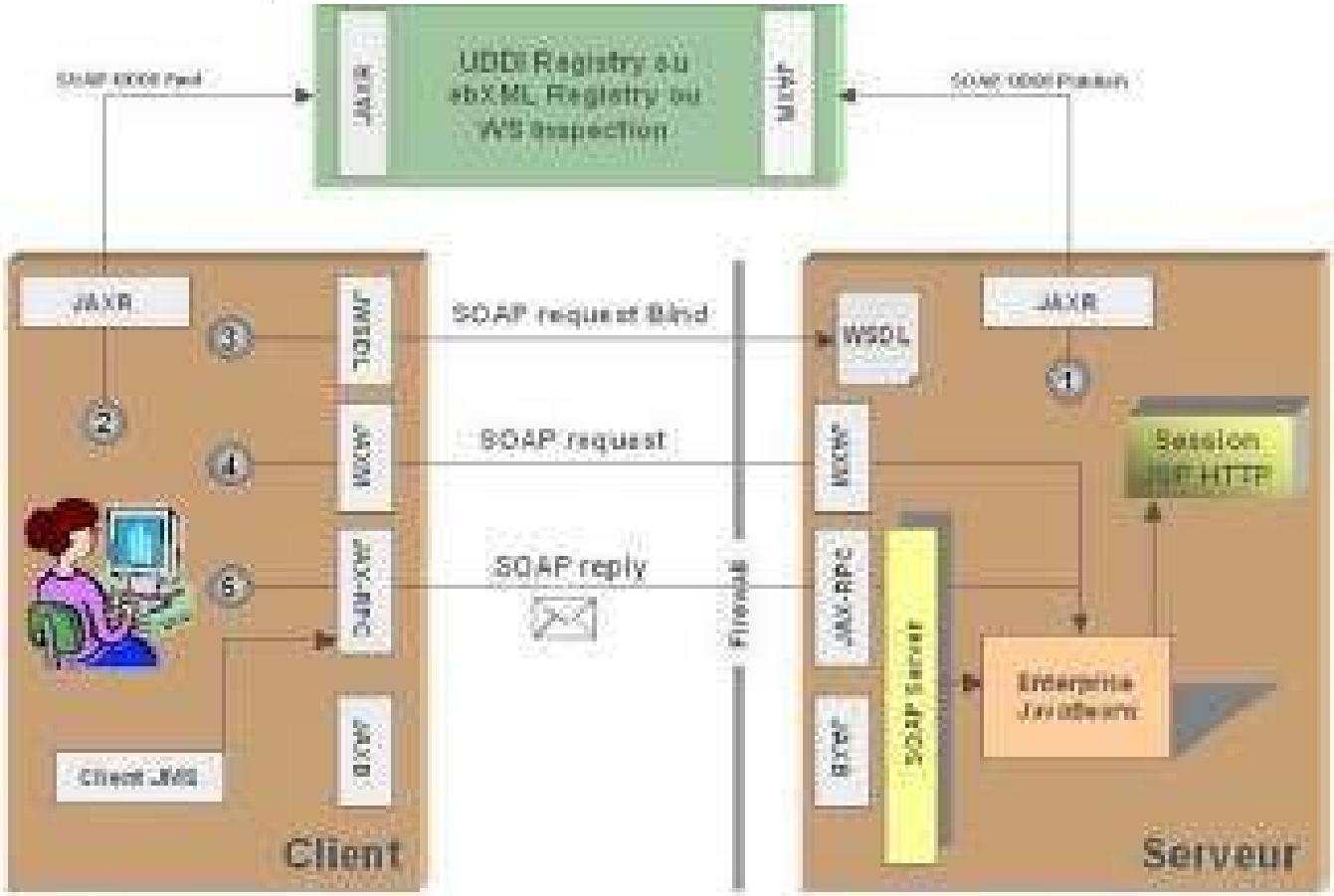


Environnement totalement intégré

Avantages :
- homogénéité de l'ensemble de l'architecture permettant d'utiliser pleinement les ressources du système Windows et du Framework .NET
- impact positif sur les performances et sur l'intégration de l'outil Visual Studio avec la plate-forme

Inconvénients
implique de posséder l'ensemble des produits de la gamme Microsoft .NET dont IIS

L'architecture Web Services J2EE



J2EE fournie un ensemble d'APIs et d'interfaces dans le but de proposer plusieurs implémentations différentes. A l'heure actuelle, excepté le WebService Pack de Sun faisant office de RI (Reference Implementation), il n'existe aucun produit intégrant toutes ces APIs.

C/C

- **Avantages**
  - Simple, présent sur toutes plates-formes / langages
  - Déployable partout
  - Couche légère sur systèmes existants
- **Problèmes**
  - Nouveau
  - Applis indépendantes et contrôlées par organisations différentes
  - Fiabilité non garantie
  - …

BIBLIO

- Micromax Information Services Ltd. 1999.
  N-Tiers Background Articles.
  www.n-tiers.com, Octobre 1999
- Frédéric NAJMAN Cédric NICOLAS, Christophe AVARE.
  Java client-serveur.
  Eyrolles, 1998.
- Thierry RUIZ Emmanuel LIGNE.
  Corba : Objectifs et architecture.
  www.etu.info.unicaen.fr/ cliquet/dess/corba/doc-fr/node3.html, Avril 1997.
- Philippe USCLADE Jean-François GOGLIN.
  Du client-serveur au web-serveur.
  Hermès Sciences, 1999.
- Daniel MARTIN.
  Architecture des applications réparties.
  http://worldserver2.oleane.com/, Octobre 1999.
  /dmartin/Architecture applications reparties.htm.