Programmation client/serveur

IFT 6802

Par Laurent Magnin

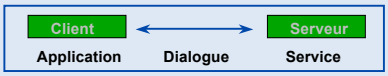
Université de Montréal

Le modèle Client / serveur

- **Repose sur une communication d'égal à égal entre les applications**
- **Communication réalisée par dialogue entre processus deux à deux**
- **Un processus est le client, l'autre le serveur**
- **les processus ne sont pas identiques mais forment plutôt un système coopératif**
- **Le résultat de cette coopération se traduit par un échange de données, le client réceptionne les résultats finaux délivrés par le serveur**

Le modèle (suite)

- **Le client initie l'échange,**
- **le serveur est à l'écoute d'une requête cliente éventuelle.**



Le service rendu = traitement effectué par le serveur, modèle client/serveur ==> répartition des services plutôt que de l'application elle-même.

Introduction

- **L'architecture Client/Serveur est l'aboutissement d'un ensemble d'évolutions technologiques survenues dans les dernières décennies :**
  - capacités mémoires,
  - performances des processeurs et des réseaux,
  - évolutions des logiciels : interfaces graphiques, multimédia, des interfaces de communications.

Introduction (suite)

- **Architecture d'abord utilisée dans les systèmes « *Time Sharing* »**
- **S'étend de plus en plus vers tous les domaines d'activités :**
  - gestion de base de données,
  - les systèmes transactionnels,
  - les systèmes de messagerie, Web, Intranet,
  - les systèmes de partage des données,
  - le calcul scientifique
  - etc.

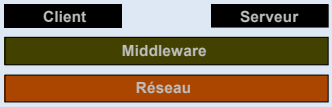
Introduction (suite)

- **Les freins**
  - difficulté de concevoir des applications distribuées,
  - manque de cohérence entre les applications clientes et serveurs,
  - manque d'outils d'administration des serveurs au niveau des services et des réseaux.
  - réticences des responsables pour des raisons de sécurité, de dispersion des données jugées sensibles,
  - incompatibilité avec les systèmes existants.

Le middleware

- **Complément de services du réseau permettant la réalisation du dialogue client/serveur :**
  - prend en compte les requêtes de l'application cliente,
  - les transmet de manière transparente à travers le réseau jusqu'au serveur,
  - prend en compte les données résultat du serveur vers l'application.

Le middleware (suite)



- **L'objectif essentiel du middleware est d'offrir aux applications une interface unifiée permettant l'accès à l'ensemble des services disponibles sur le réseau: l' API**
- **API du middleware = ciment entre les protocoles du réseau et les applications.**

Le middleware (suite)

- **Couche dite FAP (*Format and Protocols*) se superpose aux couches constitutives du réseau :**
  - réalise la synchronisation du dialogue entre client et serveur,
  - définit le format des données échangés,
  - fait le lien avec la couche transport.

➢**Selon le modèle OSI, la couche FAP s'identifie aux couches session et présentation**

Application	Api
Présentation	FAP
Session	FAP
Transport	Ex: TCP
Réseau	Ex: IP
Liaison	Ex: Ethernet
Physique	Ex 10Base5

La conception en trois tiers

- **But : structurer les applications en clients et serveurs**
- **Une application informatique est représentée selon un modèle en trois couches:**
  - la couche présentation (interface Personne / Machine)
  - La couche de traitement
  - La couche de données
- **Se compare au modèle de programmation MVC (modèle, vue, contrôleur)**

La couche de présentation

- **gestion de l'affichage (exemple Windows, X-window, etc.),**
- **logique de l'affichage, partie intrinsèque de l'applicatif qui transmet à la gestion de l'affichage, les éléments de présentation.**

La couche de traitement

- **la couche traitements qui constitue la fonctionnalité intrinsèque de l'application :**
  - la logique des traitements : l'ossature algorithmique de l'application,
  - la gestion des traitements déclenchés par la logique de traitements qui réalise la manipulation des données de l'applicatif (ex: procédures SQL).

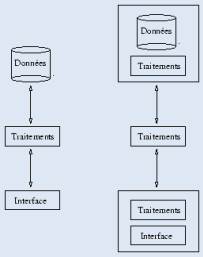
La couche de données

- **la couche données qui assure la gestion des données applicatives:**
  - la logique des données constituant les règles régissant les objets de la base de données,
  - la gestion des données (consultation et mise à jour des enregistrements). Un système de type SGBDR, habituellement, est responsable de cette tâche.

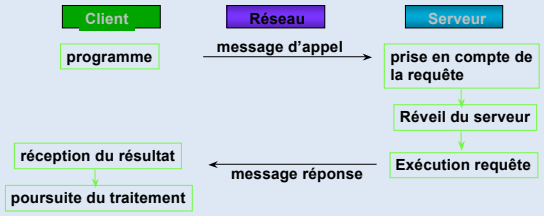
La conception trois tiers distribuées

- **Ce découpage permet de structurer une application en mode client/serveur;**
- **Exemple :**
  - le module de gestion des données peut être hébergé par un serveur distant,
  - le module de gestion de l'affichage peut également être géré par un serveur distant (un Terminal X par exemple).

Architectures trois tiers



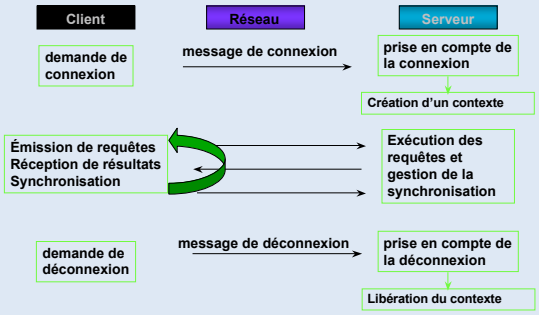
Communication : non-connecté



Communication : non-connecté

- **L'arrivée des données + ordonnancement + non duplication ne sont pas garantis par le protocole ; ==> à gérer par l'application**
- **l'approche non-connecté implique généralement une connexion synchrone**
- **En général, mode *stateless* (sans état)**
  - Aucune mémorisation des données utilisées par un client

Communication : connecté



Communication : connecté

- **Permet une plus grande fiabilité du service**
- **Plus lourd**
- **Mode *stateful* (avec état)**
  - Sauvegarde des données des clients par le serveur
  - Interactions plus complexes

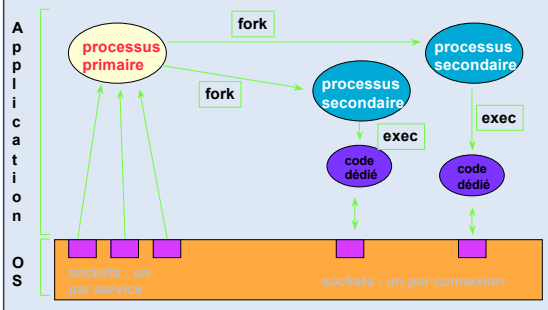
Conception : architecture serveur

- **Processus serveur:**
  - Offre une connexion sur le réseau,
  - Entre indéfiniment dans un processus d'attente de requêtes clientes,
  - Lorsqu'une requête arrive, le serveur déclenche les processus associés à cette requête, puis émet la ou les réponses vers le client.
  - Problème : gérer plusieurs client simultanément.
  - Les types de serveurs
    - serveurs itératifs: ne gèrent qu'un seul client à la fois
    - serveurs parallèles : fonctionnent « en mode concurrent ».

Modèles de serveurs

	Connecté	Non connecté
Itératif	Services qui nécessitent très peu de traitement par requête mais requièrent un transport fiable de type TCP. Peu utilisé.	Services qui nécessitent très peu de traitement par requête (pas de concurrence). Exemple: serveur TIME
Parallèle	Offre un transport fiable et est capable de gérer plusieurs requêtes de différents clients simultanément	Très peu utilisé

Serveurs multi-services



Avantages des serveurs multi-services

- **le code réalisant les services n'est présent que lorsqu'il est nécessaire,**
- **la maintenance se fait sur la base du service et non du serveur : l'administrateur peut gérer le serveur par service au lieu de le gérer globalement.**
- **Ce schéma est retenu en standard : le « super serveur » (inetd en BSD) consistant en un processus multi-services multi-protocoles offrant une interface de configuration (fichier systèmes) permettant à l'administrateur système d'ajouter de nouveaux services alors qu'aucun processus supplémentaire n'est nécessaire.**

Conception : architecture cliente

- **Une application cliente est moins complexe que son homologue serveur car :**
  - la plupart des applications clientes ne gèrent pas d'interactions avec plusieurs serveurs,
  - la plupart des applications clientes sont traitées comme un processus conventionnel ; au contraire, un serveur nécessite des accès privilégiés de connexion au middleware.
  - la plupart des applications clientes ne nécessitent pas de protection supplémentaires, le système d'exploitation assurant les protections élémentaires suffisantes.

Références

- **http://www.centralweb.fr/download/**
- **http://www.pps.jussieu.fr/Livres/ora/DA-OCAML/book-ora188.html**
- **http://cui.unige.ch/db-research/Enseignement/analyseinfo/JAVAF/cliser.html**
- **http://www-bi.imag.fr/Infos/Personnes/Sacha.Krakowiak/Enseignement/ti-deug/TP/client-serveur.html**

Références (suite)

- **http://www.infres.enst.fr/~domas/TPthr-resjava.html**
- **http://www.grappa.univ-lille3.fr/polys/frime/sortie002.html**
- **http://www.crim.ca/rd/ti-dinners/tidinner270301.pdf**
- **http://www.softwired-inc.com/people/maffeis/articles/research/client_server.pdf**

http://www.iro.umontreal.ca/~pift3880/cs.html

Architectures de sites web dynamiques

Exemple d'application Clients/Serveur
(©Adnane Benjelloun, CRIM)

Plan de la présentation

- **Pourquoi parler d'architectures dynamiques ?**
- **Architecture de base**
- **Différentes solutions**
- **XML – XSL**

Pourquoi parler d'architectures dynamiques ?

- **Statique : pages créées une fois pour toute**
- **Dynamique : pages générées lors de la requête HTTP**
  - Flexibilité de développement
  - Simplicité de mise en production
  - Facilité de la maintenance

Développement

- **Rendre les tâches de programmation indépendantes les unes des autres**
- **Accélérer le développement par la réutilisation**
- **Mieux profiter des expertises**
- **Rester indépendant de l'environnement de développement et ne dépendre que des standards**

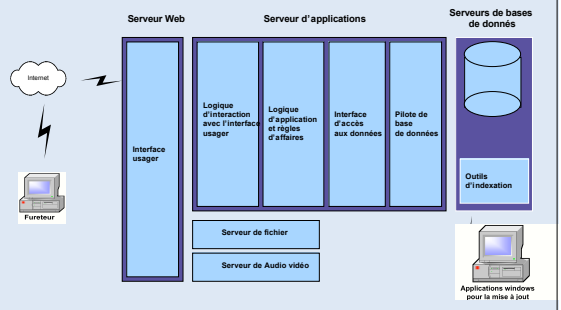
Production

- **Garantir un temps de réponse acceptable.**
- **Mettre en place un système stable et robuste.**
- **Assurer la sécurité adéquate aux données.**

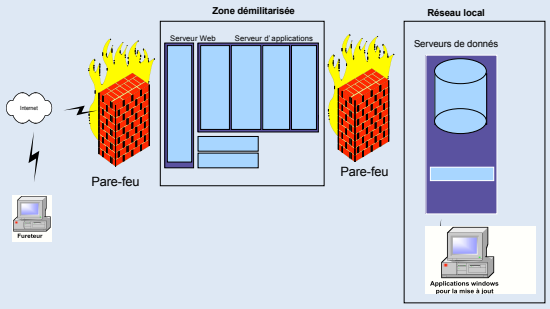
Maintenance

- **Détecter et corriger facilement les failles du système**
- **Pouvoir ajouter de nouvelles fonctionnalités**
- **Ouverture vers d'autres systèmes**

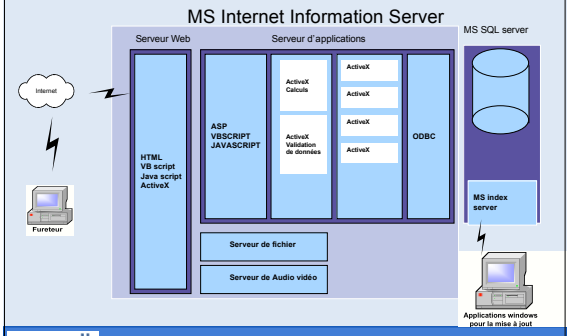
Architecture de base



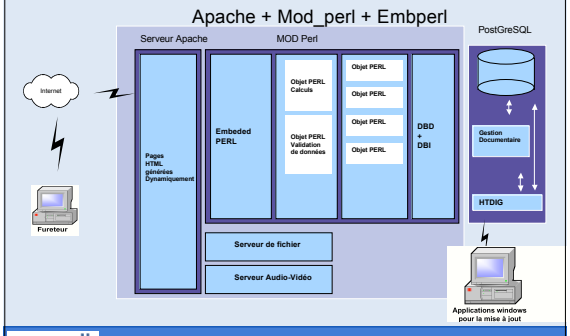
Sécurité



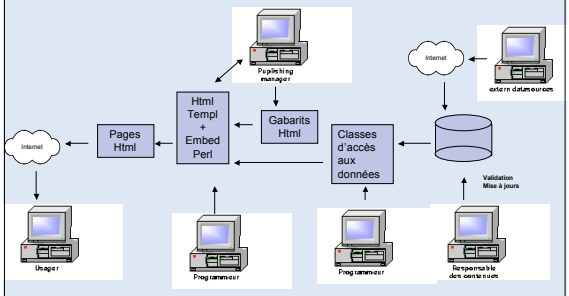
Solution Microsoft



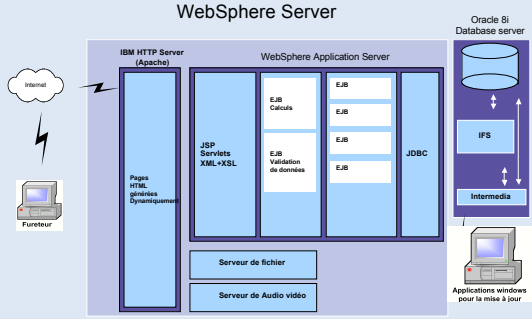
Solution PERL (0$)



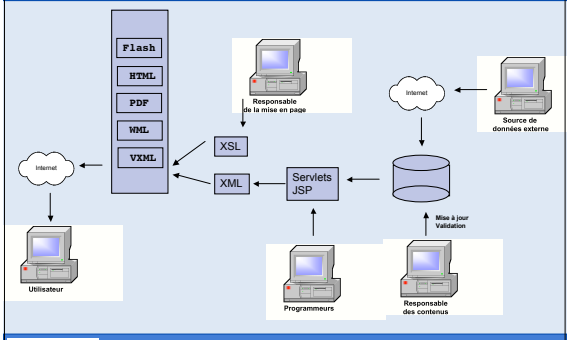
Qui fait quoi ?



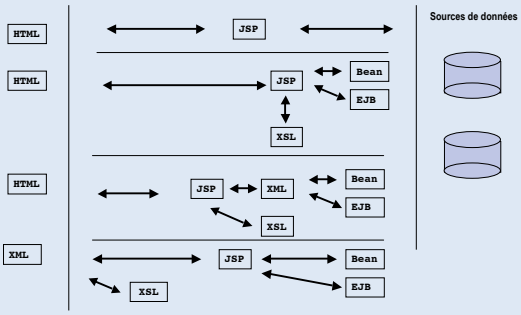
Solution JAVA (EX : IBM + Oracle)



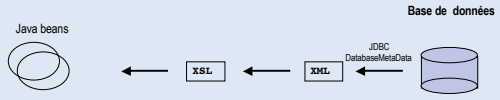
Qui fait quoi ?



JSP - XML - XSL



XML + XSL = java beans



XML + XSL = java beans suite

XML

```
<table name='regions' class_name='Regions' obj_name='Region'>
    <field name='id' col_name_fstu='Id' col_name_allu='ID' type='4' func='getInt'
    java_type='int'/>
    <field name='name' col_name_fstu='Name' col_name_allu='NAME' type='1'
    func='getString' java_type='String' />
    <primaryKey name='id' col_name_fstu='Id' col_name_allu='ID' key_seq='1' />
</table>
```

XML + XSL = java beans suite

XSL

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text"/>
<xsl:template match="/tables">
<xsl:for-each select="table">
package db;

public class <xsl:value-of select="@obj_name"/> {
        <xsl:for-each select="field">
        private <xsl:value-of select="@java_type"/> <xsl:value-of select="@name"/>;
        </xsl:for-each>
        public <xsl:value-of select="@obj_name"/>() {
                super();
        }
        <xsl:for-each select="field">
        public <xsl:value-of select="@java_type"/> get<xsl:value-of select="@col_name_fstu"/>() {
                return <xsl:value-of select="@name"/>;
        }
        </xsl:for-each>
        <xsl:for-each select="field">
        public void set<xsl:value-of select="@col_name_fstu"/>(<xsl:value-of select="@java_type"/> new_<xsl:value-of select="@name"/>) {
                <xsl:value-of select="@name"/> = new_<xsl:value-of select="@name"/>;
        }
        </xsl:for-each>

}
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```


XML + XSL = java beans suite

Java

```
package db;

public class Region {
        private int  id;
        private String  name;
        public Region() {
                super();
        }
        public int getId() { return id; }

        public String getName() { return name; }

        public void setId(int new_id) { id = new_id; }

        public void setName(String new_name) { name = new_name; }
}
```