

# INTERNET ET LE WCF

François-Xavier BAILLET



# Internet, espace $\infty$ ?

- Les couches OSI
- TCP/IP : son adressage, ses protocoles, ses applications...

=> Univers connu ...

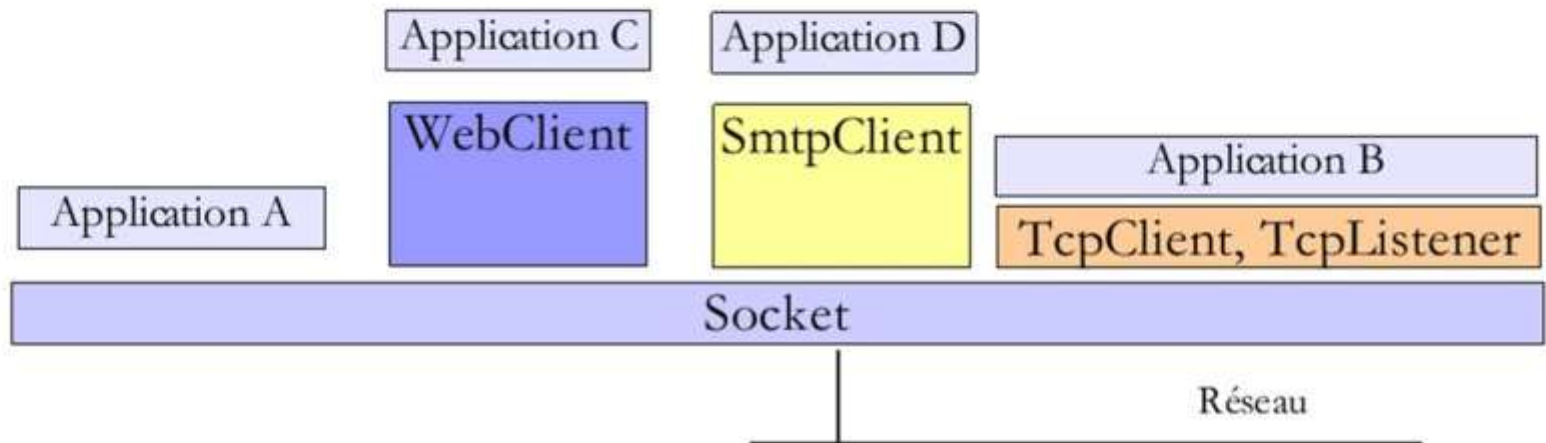
- .NET dans tout cela ?

➔ Des classes

➔ Les technos du WCF

# Internet

- Diverses classes pour travailler avec le réseau



- La base : Classe *IPAddress* avec méthodes
- L'essentiel : *Socket* pour gérer totalement et finement une communication en TCP/IP
- Simplement : *TcpClient*, *TcpListener* inclus la gestion des sockets



# Services Web

- Accéder à l'information dans le WWW
  - Le navigateur c'est bien, mais interactif
  - La programmation : complexe mais efficace
- Impératif :
  - Localisation de l'information
  - Format des données, simple et précis

=> La solution : Webservice (RPC, Soap, Rest,...)
- Un Web Service est un composant logiciel identifié par une URI, dont les interfaces publiques sont définies et appelées en XML.

Sa définition peut être découverte par d'autres systèmes logiciels.

Les services Web peuvent interagir entre eux d'une manière prescrite par leurs définitions, en utilisant des messages XML portées par les protocoles Internet (**W3C**) : WSDL



# Services Web

- Intérêts :
  - Les données peuvent être présentes uniquement sur le serveur distant
  - Le serveur distant peut disposer d'une puissance de calcul ou de capacités de stockage dont l'utilisateur local ne dispose pas
  - L'application distante peut être utilisée simultanément par un grand nombre d'utilisateurs et sa mise à jour n'intervient qu'à un seul endroit
  - Vous avez besoin des données, mais vous n'en êtes pas responsable/propriétaire
  - Délimite facilement les domaines de responsabilité

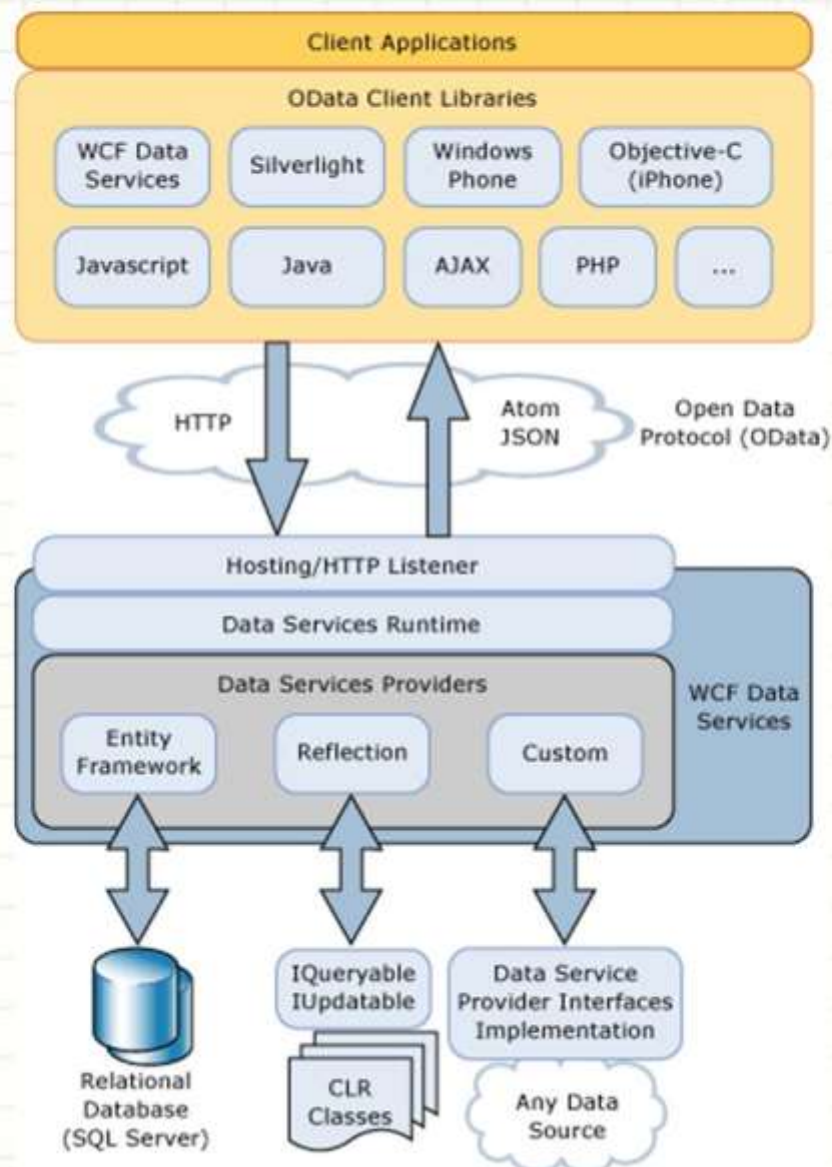




# Services Web

- Nécessite une connexion permanente à Internet
- Basé sur des protocoles standards, indépendants des constructeurs et éditeurs
  - HTTP(S) => Passer à HTTPS est un standard
  - XML
  - ...
- Ecriture et lecture des Web Services dans des langages différents et hétérogènes

# Services Web

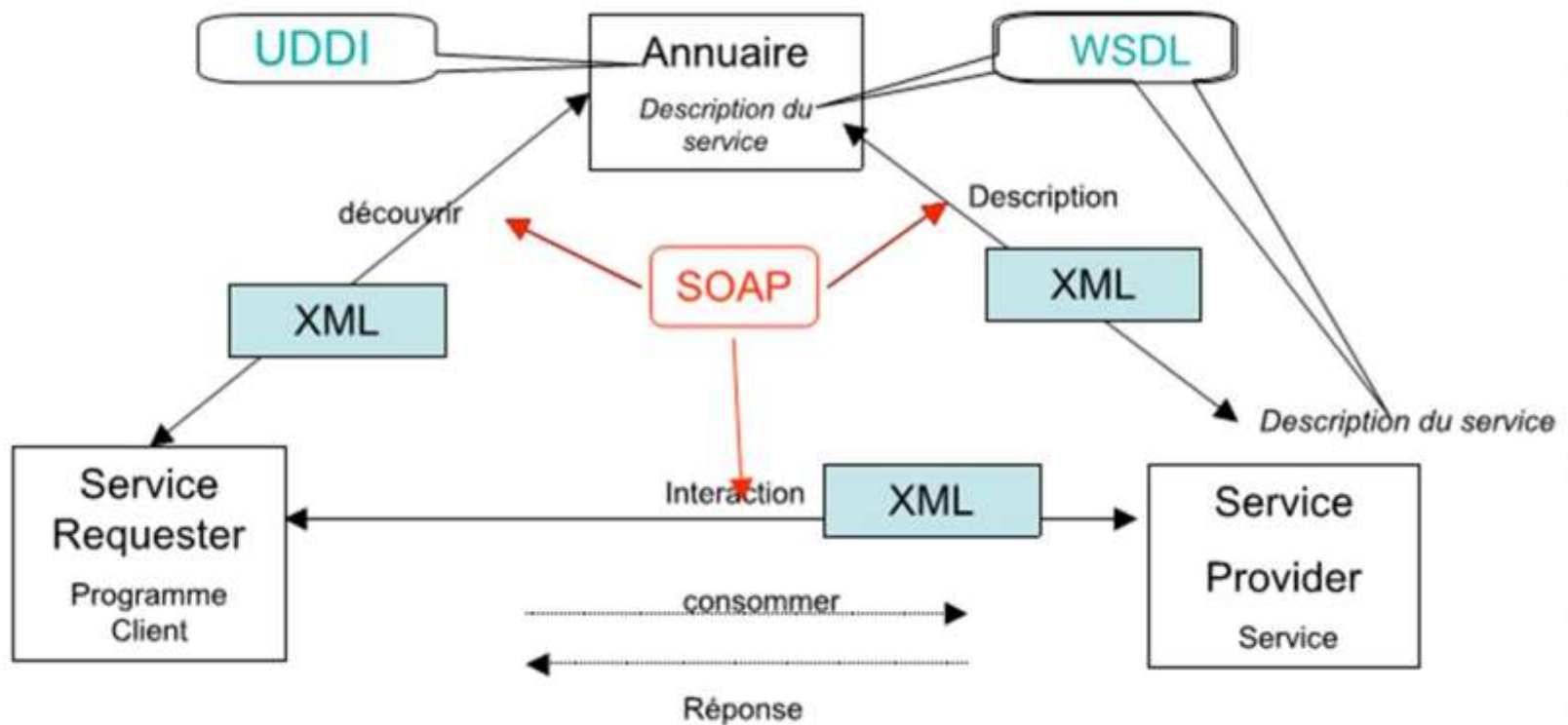


# Services Web : fonctionnement

- Couches
  - **Invocation**, visant à décrire la structure des messages échangés par les applications
  - **Découverte**, pour permettre de rechercher et de localiser un service web particulier dans un annuaire de service.
  - **Description**, dont l'objectif est la description des interfaces des services web.



# Services Web : Architecture



# Services Web : Architecture

**WSDL** (Web Services Description Language) Donne la description en XML des W.S. en précisant les méthodes pouvant être invoquées, leur signature et les points d'accès (URL, port, etc...)

**UDDI** (Universal Description, Discovery and Integration) normalise une solution d'annuaire distribué de W.S., permettant la publication et l'exploration. UDDI se comporte comme un W.S. dont les méthodes sont appelées en SOAP.

**SOAP** (Simple Object Access Protocol) est un protocole d'échanges de message **XML** entre des clients et des fournisseurs de services web.

Accès à un Web Service :

**REST** (Representational State Transfer)

Consommer un W.S. REST revient à appeler une simple URL en http (GET ou POST, le serveur renvoie la réponse, en ...XML... la plupart du temps)

Communiquer par fichier XML

XML-RPC

SOAP



# Services Web

- Acteurs :
  - Client : Programme Windows, Console, d'un SE quelconque
  - Serveur : Quelque part, disponible sur le réseau (Internet ou local)
  - Appel de fonction du client vers le serveur





# Services Web : SOA

- Architecture orientée services : (SOA : Services Oriented Architecture) architecture logicielle s'appuyant sur un ensemble de services
- Objectif : décomposer une fonctionnalité en un ensemble de fonctions basiques, appelées services, fournies par des composants et de décrire finement le schéma d'interaction entre ces services.
- But : **cesser de construire la vie de l'entreprise autour d'applications pour faire en sorte de construire une architecture logicielle globale décomposées en services correspondant aux processus métiers de l'entreprise.**
- Lorsque l'architecture SOA s'appuie sur des web services, on parle alors de WSOA, pour Web Services Oriented Architecture.

# Services Web : SOA => Intérêts

- Une modularité permettant de remplacer facilement un composant (service) par un autre
- Une réutilisabilité possible des composants (par opposition à une système tout-en-un fait sur mesure pour une organisation).
- De meilleures possibilités d'évolutions (il suffit de faire évoluer un service ou d'ajouter un nouveau service)
- Une plus grande tolérance aux pannes
- Une maintenance facilitée



# SOAP

- Un message SOAP est composé de :
  - Une déclaration XML (comme tout document XML)
  - Une enveloppe SOAP contenant
    - Un « header »
    - Un « body »
- Processus RPC
- Exemple : Requête d'une fonction sur un serveur en [php \(Client\)](#) & [php \(Serveur\)](#)...
- Et en C# ???, c'est moins simple...



# SOAP

- Exemple d'échange : [ICI](#)

- ```
<?xml version="1.0" encoding="UTF-8"?>
  <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="http://uri-
    fxb/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-
    ENC="http://schemas.xmlsoap.org/soap/encoding/" SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
      <ns1:perroquet>
        <param0 xsi:type="xsd:string">
          Avec un param&#232;tre en entr&#233;e et les
          &#233;changes SOAP &#58;
        </param0>
      </ns1:perroquet>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```



# REST

- REST n'est pas un protocole ou un format, c'est un style d'architecture => le WEB !
- Les principes :
  - URI : adresse du serveur et paramètres de la requête
  - HTTP : protocole du transfert (vous connaissez)
  - Données sous formes simples : XML, JSON ...
- W.S. REST  $\approx$  appel URL en http (post ou get), réponse du serveur en XML (souvent)

# REST

- Requête :

<http://monsite.fr/getInfo.asp?par1=xxxx&par2=58>

- Réponse :


```
<RETURN>
  <ERROR>
    <NUMBER>[n° erreur]</NUMBER>
    <DESCRIPTION>[détail de l'erreur]</DESCRIPTION>
  </ERROR>
  <RESULT>
    <ROWS>
      <ROWCOUNT> 1</ROWCOUNT>
      <ROW>
        <REP_1>[contenu du 1° champ]</REP_1>
        <REP_2>[contenu du 2° champ]</REP_2>
        <REP_3>[contenu du 3° champ]</REP_3>
        <REP_4>[contenu du 4° champ]</REP_4>
      </ROW>
    </ROWS>
  </RESULT>
</RETURN>
```





# WCF

- WCF est sorti en Novembre 2007. C'est l'un des trois piliers du .Net Framework 3.0
- WCF est une technologie qui permet de faciliter la mise en place des applications distribuées en servant de socle commun aux architectures orientées services (SOA : Service Oriented Architecture).
- WCF se veut être un modèle uniforme de distribution de services :
  - Faiblement couplé pour plus de souplesse
  - Adapté à toutes situations et besoins de communication :
    - Service WEB
    - Communication inter-processus
    - Gestion de message entre applications
    - ...



# WCF : Intérêts

- Communication entre programmes unifiés :
  - programmation identique quel que soit le protocole de communication
  - C'est au déploiement de l'application que l'on va spécifier le protocole
- Développement de services simplifiés à l'extrême (trop peut-être ?)



# WCF : Etapes de mise en œuvre

- Première étape : définition du contrat. On définit les signatures des méthodes composant notre service WCF, et du format des données à échanger.
- Seconde étape : implémentation du contrat. On implémente les services.
- Troisième étape : configuration du service. On définit les endPoints, autrement dit les points d'accès au service.
- Quatrième étape : hébergement des services dans une application .NET.
- Puis, il ne reste plus qu'à créer une application cliente (quelque soit sa forme, Windows, Web, ou tout autre composant) qui pourra consommer le service WCF.



# WCF : détail de mise en œuvre

- Créer un projet "Application de service Web"
- Définir l'interface *IService* définissant les méthodes et objets exposées par le service
- Définir l'implémentation de l'interface du service (code métier du service)
- Créer le fichier de configuration d'accès au code métier, utilisé par le serveur d'application (.svc)

# WCF : détail de mise en œuvre

- Pour définir le [contrat de service] :
- La métadonnée *[ServiceContract(Namespace = "xxxx")]* doit être attaché à l'interface
- La métadonnée *[OperationContract]* doit être attaché à chaque méthode de l'interface
- Si on utilise des objets dans les échanges :
  - utiliser *[DataContract]* pour la classe
  - utiliser *[DataMember]* pour les propriétés



# WCF : hébergement du service

- Dépend de l'OS hébergeant le service et du protocole de transport de données choisi
- L'auto hébergement (ServiceHost) :
  - Hébergement du service au sein d'une application (Windows Forms, WPF, Service Windows ou application console...)
  - Géré par l'utilisateur, très utile pour la communication pair à pair
- Sous IIS
  - Très simple à mettre en œuvre,
  - Géré par IIS ! (Démarrage, recyclage...)
  - Limité à http et https



# WCF : hébergement du service

- Fichier de configuration XML *.config* (*App.config* ou *Web.config*) contient :
- Les paramètres d'hôte *Host* : déterminent l'emplacement du service sur le réseau
- Le *EndPoint* : le point de connexion des clients sur le service
- Le *Behavior* : détermine le comportement du serveur vis-à-vis du client

# WCF : Consommation du service

- Le fichier *wsdl* contient les méthodes et le mode d'utilisation
- Visual Studio génère une "classe proxy"
- => masque la complexité de la com. avec le W.S.
- Génération du proxy:
  - En ajoutant une référence de service dans le projet consommant le service WCF
  - En utilisant l'utilitaire svcutil.exe (méthode manuelle) avec W.S. actif.



# WCF : Consommation du service

- Utilitaire svcutil :

- sous : C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin
- svcutil.exe /language:cs /out:generatedProxy.cs /config:app.config  
http://localhost:8000/ServiceModelSamples/service

- Fournit

- le app.config pour l'application cliente
- la classe proxy avec les types utilisés par le W.S.

- Créer le client :

- On importe les deux fichiers générés précédemment qui se trouvent dans le répertoire de l'utilitaire svcutil.exe (clic droit sur le projet → ajouter un élément existant)
- On renomme ensuite le fichier de code generatedProxy.cs et le fichier output.config en App.config
- On harmonise le nom de la classe ou du namespace du proxy.
- On rajoute les références nécessaires au projet (System.ServiceModel ...)



# WCF : Exemple...

- Lancement du W.S.
- Le fichier WSDL
- Le client consommateur
- Le Projet ;-)
- Interroger directement un WS

