

# ARCHITECTURE ORIENTÉE SERVICES

Jemal Ahmed, MT à ISET Sfax  
Ahmed.jemal.sfax.iset@gmail.com

# Organisation

2

- Cour magistral
- Exemples concrets
- Travaux pratiques

# Présentation générale du cours

3

## □ **Volume Horaire**

- ▣ 30h cours

## □ **Objectifs**

- ▣ Comprendre les fondements de l'architecture orientée services
- ▣ Comprendre les fondements de XML, les parseurs XML, les espaces de nommage, et XML schema
- ▣ Pouvoir développer des services Web
- ▣ Orchestrer les services Web
- ▣ Maîtriser les microservices
- ▣ Exploiter la plateforme Spring cloud

# Evaluation

4

- Travaux pratiques, Manipulation
- Mini Projet
- Examen Théorique

# Plan du cours

6

- Evolution des langages et des modèles de développement
- Introduction aux Web Services
- Le protocole SOAP
- WSDL : Web Service description Language



# Partie 1

Evolution des langages et des modèles de développement

# Paradigmes de programmation

8

- Différents paradigmes :
  - ▣ Programmation Procédurale
  - ▣ Programmation Orientée Objet
  - ▣ Programmation Orientée Composants
  - ▣ Programmation Orientée Service
  - ▣ Programmation avec les Micro-Services

# Paradigmes de programmation

9

- Différents paradigmes :
  - ▣ Programmation Procédurale

(Source wikipédia)

La programmation procédurale est un paradigme de programmation basé sur le concept d'appel de procédure. Une procédure contient simplement une série d'étapes à réaliser. N'importe quelle procédure peut être appelée à n'importe quelle étape de l'exécution du programme.

- ▣ Programmation Orientée Objet
- ▣ Programmation Orientée Composants
- ▣ Programmation Orientée Service
- ▣ Programmation avec les Micro-Services



# Paradigmes de programmation

10

- Différents paradigmes :
  - ▣ Programmation Procédurale
  - ▣ Programmation Orientée Objet

(Source wikipédia)

Un objet représente un concept, une idée ou toute entité du monde physique. Il possède une structure interne et un comportement, et il sait communiquer avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; la communication entre les objets via leurs relations permet de réaliser les fonctionnalités attendues, de résoudre les problèmes.

- ▣ Programmation Orientée Composants
- ▣ Programmation Orientée Service
- ▣ Programmation avec les Micro-Services

# Paradigmes de programmation

11

- Différents paradigmes :
  - ▣ Programmation Procédurale
  - ▣ Programmation Orientée Objet
  - ▣ Programmation Orientée Composants

(Source wikipédia)

La programmation orientée composant (POC) consiste à utiliser une approche modulaire au niveau de l'architecture d'un projet informatique, ce qui permet d'assurer au logiciel une meilleure lisibilité et une meilleure maintenance. Les développeurs, au lieu de créer un exécutable monolithique, se servent de briques réutilisables.

- ▣ Programmation Orientée Service
- ▣ Programmation avec les Micro-Services

# Paradigmes de programmation

12

- Différents paradigmes :
  - ▣ Procédures
  - ▣ Programmation Orientée Objet
  - ▣ Programmation Orientée Composants
  - ▣ Programmation Orientée Service

## Besoins ?

Support de l'hétérogénéité: plusieurs plate-formes logicielles et matérielles

Accès et manipulation des données de partout

- ▣ Programmation avec les Micro-Services

# Paradigmes de programmation

13

- Différents paradigmes :
  - ▣ Procédures
  - ▣ Programmation Orientée Objet
  - ▣ Programmation Orientée Composants
  - ▣ Programmation Orientée Service
  - ▣ Programmation avec les Micro-Services

(Source Martin Fowler)

« Le style architectural des micro-services est une approche permettant de développer une application unique sous la forme d'une suite logicielle intégrant plusieurs services. Ces services sont construits autour des capacités de l'entreprise et peuvent être déployés de façon indépendante. »

# Architecture orientée Service

14

## Architecture orientée Service

Une architecture orientée services (notée SOA pour Services Oriented Architecture) est une architecture logicielle s'appuyant sur un ensemble de services simples.

Elle permet de 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.

Lorsque l'architecture SOA s'appuie sur des web services, on parle alors de WSOA, pour Web Services Oriented Architecture.

The slide features a decorative header with two horizontal bars. The top bar is a thin green line. Below it is a thicker green bar that contains the text 'Partie 2'. To the left of this green bar is a vertical orange bar. The text 'Partie 2' is centered within the green bar in a white, sans-serif font.

## Partie 2

### Introduction aux Web Services

# Serveur Web

16

- Un serveur est un ordinateur connecté en permanence à l'Internet. Il met à disposition des informations et / ou des services répondant aux requêtes des clients web.
- Souvent appelé serveur http
- 2 types de ressources
  - ▣ *Statiques*: ne nécessitent pas de traitement côté serveur,
  - ▣ *Dynamiques*: chaque demande de page nécessite des opérations spécifiques du serveur.
- Nombreux logiciels disponibles:
  - ▣ Apache,
  - ▣ Internet Information Service (IIS),
  - ▣ Sun Java System Web Server .

# Service Web : les origines

17

- Le Web d'hier
  - ▣ conçu pour une interaction client-application
  - ▣ Développé autour de 2 standards http et html
  - ▣ permet de supporter le B2C
- Le Web d'aujourd'hui
  - ▣ Le Web est partout. Que peut-on faire avec ?
    - Supporter des places de marché électronique (E-market places)
    - intégrer des procédés métiers
    - partager des ressources
  - ▣ Les approches existantes sont ad hoc
    - e.g. interactions application-application avec des formulaires html
  - ▣ But: permettre une interaction application-application



# Trois générations d'applications Web

18

## □ 1<sup>ère</sup> **génération**

- Ressources statiques (rôle de serveur)
- Documents
- e.g. pages Web statiques (HTML)

## □ 2<sup>ème</sup> **génération**

- Ressources dynamiques (rôle de serveur)
- Applications
- e.g. pages Web dynamiques, ASP, JSP, PHP, ...

## □ 3<sup>ème</sup> **génération**

- Fonctionnalités métiers
- Services
- e.g. liste de noms de ville à ajouter dans un page web

# Service Web : définition

19

- A Web service is a software application identified by a URI, whose interfaces and binding<sup>(1)</sup> are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols. (W3C definition)
- Un service Web est une application logicielle
  - ▣ identifiée par un URI dont les interfaces et les liaisons sont définies, décrites et découvertes avec des mécanismes XML, et
  - ▣ supporte une interaction directe avec les autres applications logicielles en utilisant des messages XML via un protocole Internet.

(1) An association between an Interface, a concrete protocol and a data format

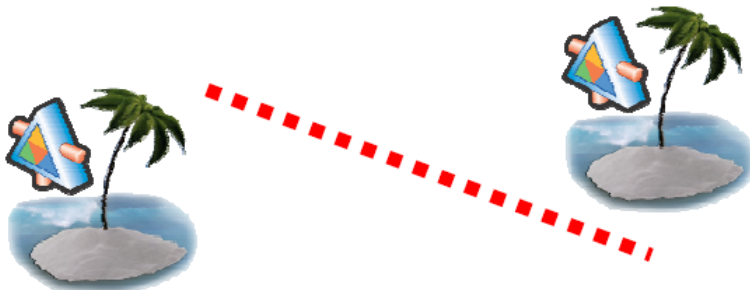
# Les web services

20

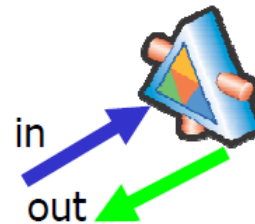
→ Un Service est Autonome  
(et sans état)



→ Les Frontières entre services  
sont Explicites



→ Un Service expose un Contrat



Conditions Générales de Vente  
Règlement Intérieur  
Vos droits/Vos devoirs

→ Les services communiquent par  
messages



# Service Web : caractéristique

21

- Les services Web sont
  - ▣ des services
    - autonome
    - exposant des contrats
    - dont les frontières sont explicites
    - qui communiquent par messages
  - ▣ accessible sur le Web
    - Les messages sont véhiculés par des protocoles Web

# Avantages des WS

22

- Les services Web fournissent l'interopérabilité entre divers logiciels fonctionnant sur diverses plates-formes.
- Les services Web utilisent des standards et protocoles ouverts.
- Les protocoles et les formats de données sont au format texte dans la mesure du possible, facilitant ainsi la compréhension du fonctionnement global des échanges.
- Basés sur le protocole HTTP, les services Web peuvent fonctionner au travers de nombreux pare-feu sans nécessiter des changements sur les règles de filtrage.
- Les outils de développement, s'appuyant sur ces standards, permettent la création automatique de programmes utilisant les services Web existants.

# Inconvénients des WS

23

- ❑ Les normes de services Web dans certains domaines engendrent l'exécution de beaucoup de code
- ❑ Les services Web souffrent de performances faibles comparées à d'autres approches de l'informatique répartie telles que le RMI, CORBA, ou DCOM.
- ❑ Par l'utilisation du protocole HTTP, les services Web peuvent contourner les mesures de sécurité mises en place au travers des pare-feu.

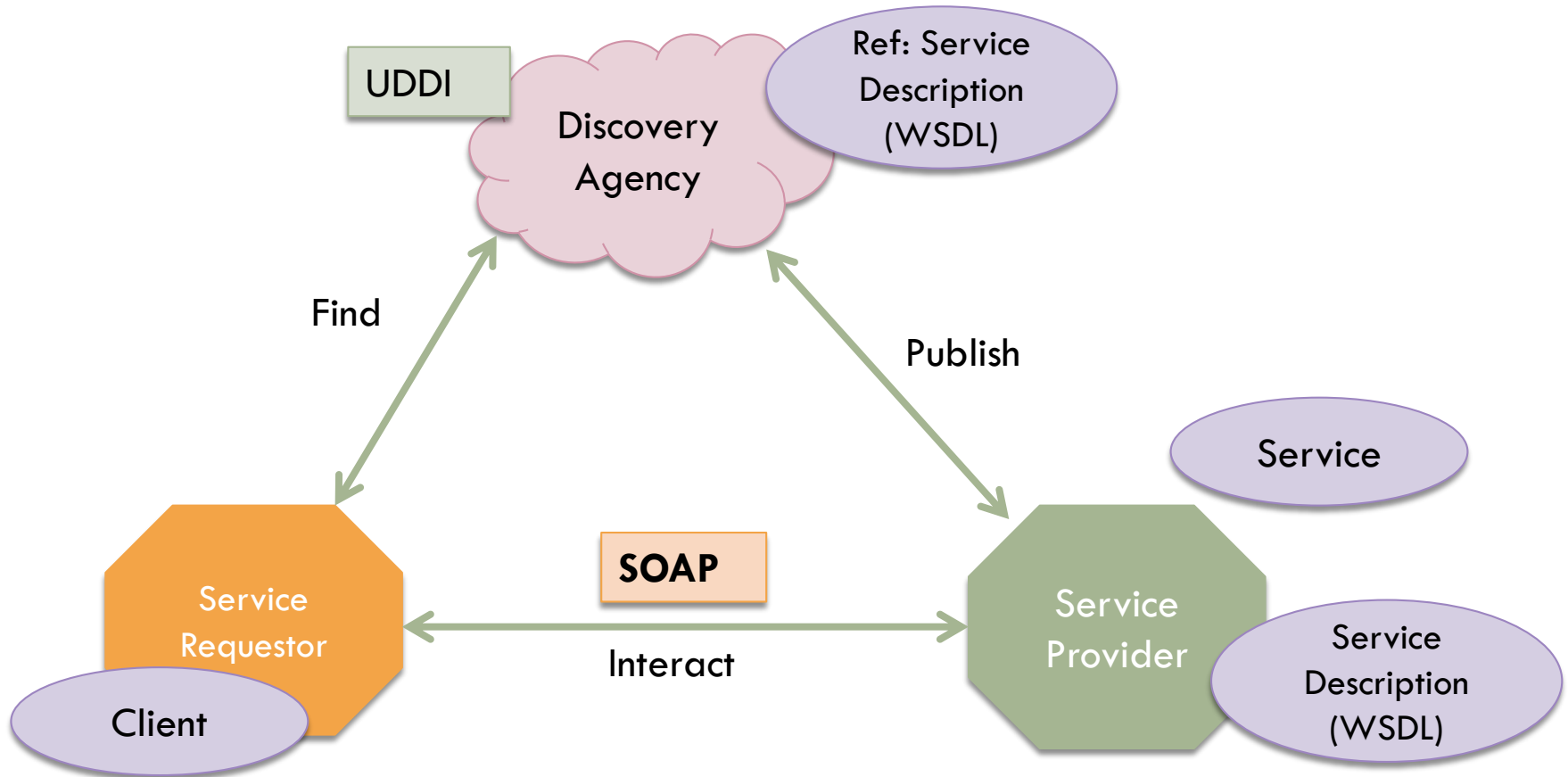
# Types de service web

24

- Deux types se présentent :
  - REST qui est un ensemble de principes architecturaux
  - SOAP, WSDL et UDDI qui font partie d'une « pile » de spécifications décrivant très concrètement des formats de messages et des protocoles.
- Les services Web RESTful (Representational State Transfer) sont dédiés fonctionner au mieux sur le Web. C'est un style architectural qui spécifie des contraintes pour assurer la performance, l'évolutivité et la modifiabilité.
- SOAP (Simple Object Access Protocol) est un protocole de transmission de messages. Il définit un ensemble de **règles** pour structurer des messages qui peuvent être utilisés dans de simples transmissions unidirectionnelles, mais il est particulièrement utile pour exécuter des dialogues requête-réponse **RPC** (Remote Procedure Call).

# Architecture Web service par W3C (SOAP-based) (1 / 2)

25



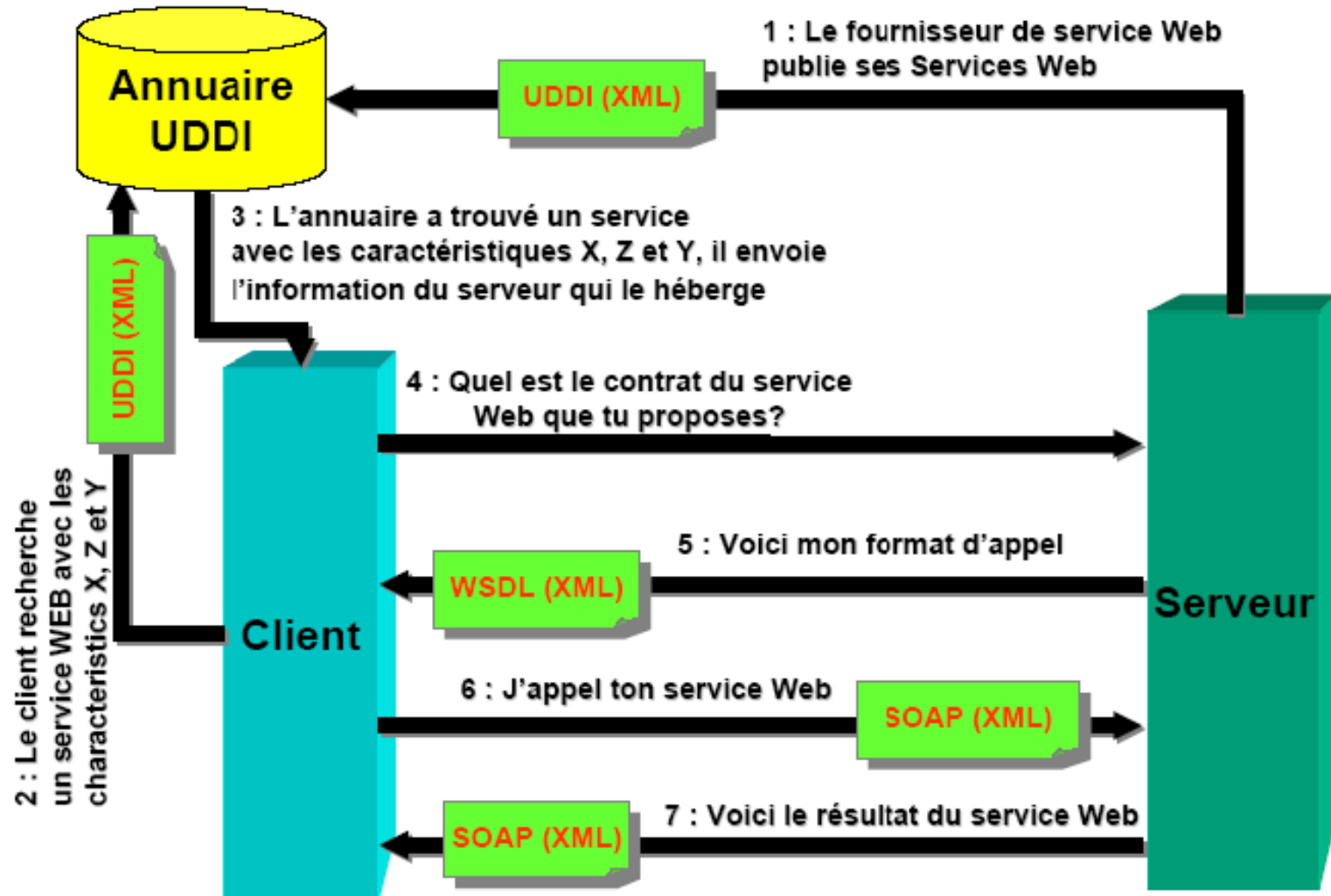
**UDDI** = **U**niversal **D**escription, **D**iscovery, and **I**ntegration

[ <http://uddi.xml.org/> ]



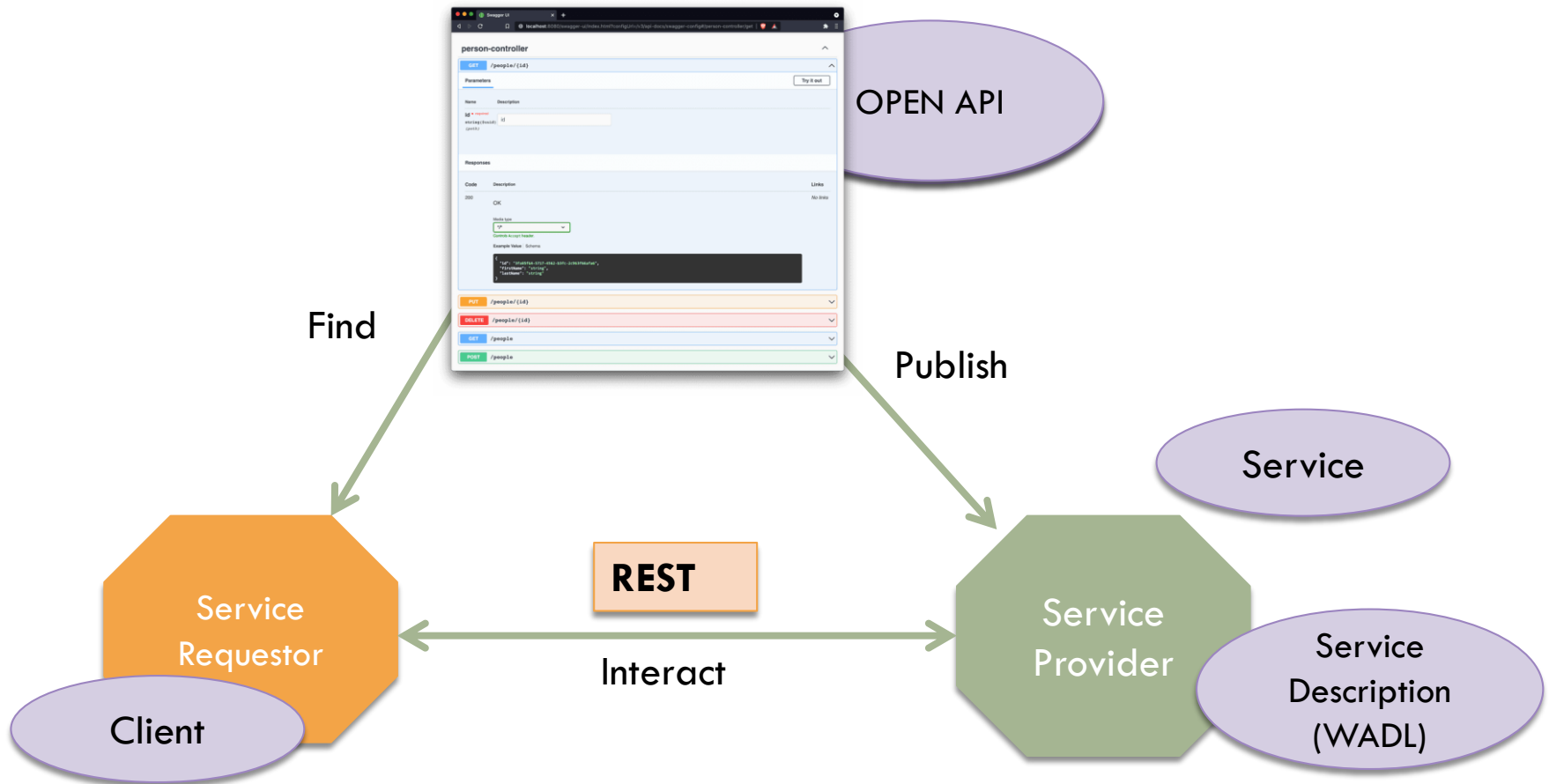
# Architecture Web service (2/2)

26



# Architecture Web service (REST-based)

27



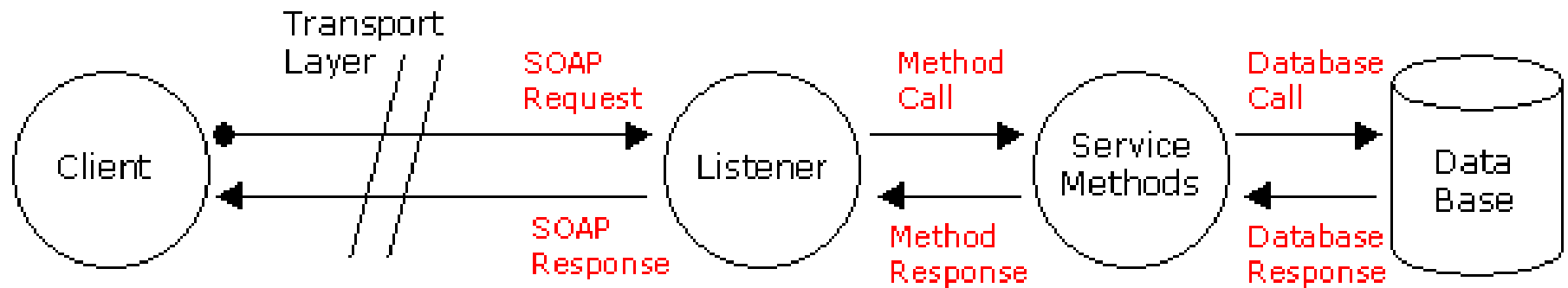
# Architecture Web service

28

- Les partenaires
  - Le fournisseur de services crée le service Web, puis publie son interface ainsi que les informations d'accès au service, dans un annuaire de services Web.
  - L'annuaire de services rend disponible l'interface du service ainsi que ses informations d'accès, pour les demandeurs potentiel de service.
  - Le demandeur de services accède à l'annuaire de service pour effectuer une recherche afin de trouver les services désirés. Ensuite, il se lie au fournisseur pour invoquer le service.
- Les standards
  - Protocole SOAP : Achemine les messages entre fournisseur, annuaire et demandeur de services
  - Langage WSDL : Décrit les interfaces des services
  - Norme UDDI : spécifie la structure des annuaires de services

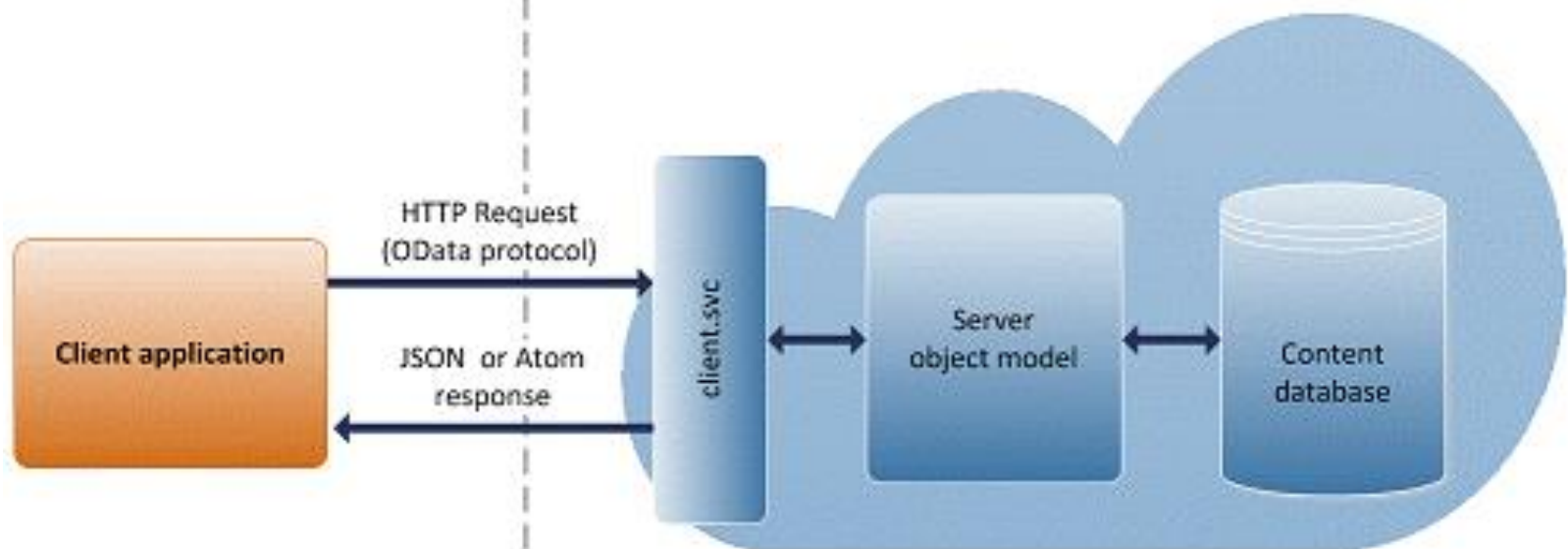
# Rest Vs Soap

29



Client

SharePoint





# Partie 3

## Le protocol SOAP

# SOAP (1)

31

**Simple Object Access Protocol** est un **Protocole** de dialogue entre applications basées sur du XML.

Deux objectifs à la base :

- Interopérabilité entre applications d'une même entreprise (Intranet)
- Interopérabilité interentreprises entre applications et services web

Similaire au protocole RPC, SOAP utilise le protocole HTTP pour les échanges de données mais aussi SMTP et POP.

Spécifications du W3C:

SOAP 1.1 : <http://www.w3.org/TR/2000/NOTE-SOAP-20000508>

SOAP 1.2 : <http://www.w3.org/TR/soap12>

# SOAP (2)

32

	<b>RMI</b>	<b>DCOM</b>	<b>CORBA</b>	<b>SOAP</b>
<b>Défini par</b>	SUN	Microsoft	OMG	W3C
<b>Plate-forme</b>	Multi	Win32	Multi	Multi
<b>Langage de Développement</b>	Java	C++, VB, VJ	Multi	Multi
<b>Langage de définition</b>	Java	ODL	IDL	WSDL
<b>Transport</b>	TCP, HTTP, IIOP	IP/IPX	GIOP, IIOP	HTTP, HTTPS SMTP
<b>Transaction</b>	Non	Oui	Oui	Oui
<b>Sécurité</b>	SSL, JAAS	SSL	SSL	SSL

# SOAP (3) : Concept des messages

33

- Les messages SOAP sont utilisés pour envoyer (requête) et recevoir (réponse) des informations d'un récepteur
- Un message SOAP peut être transmis à plusieurs récepteurs intermédiaires avant d'être reçu par le récepteur final (➔ chaîne de responsabilité)
- Le format SOAP peut contenir des messages spécifiques correspondant à des erreurs identifiées par le récepteur
- Un message SOAP est véhiculé en utilisant un protocole de transport (HTTP, SMTP, ...)



# Structure

34

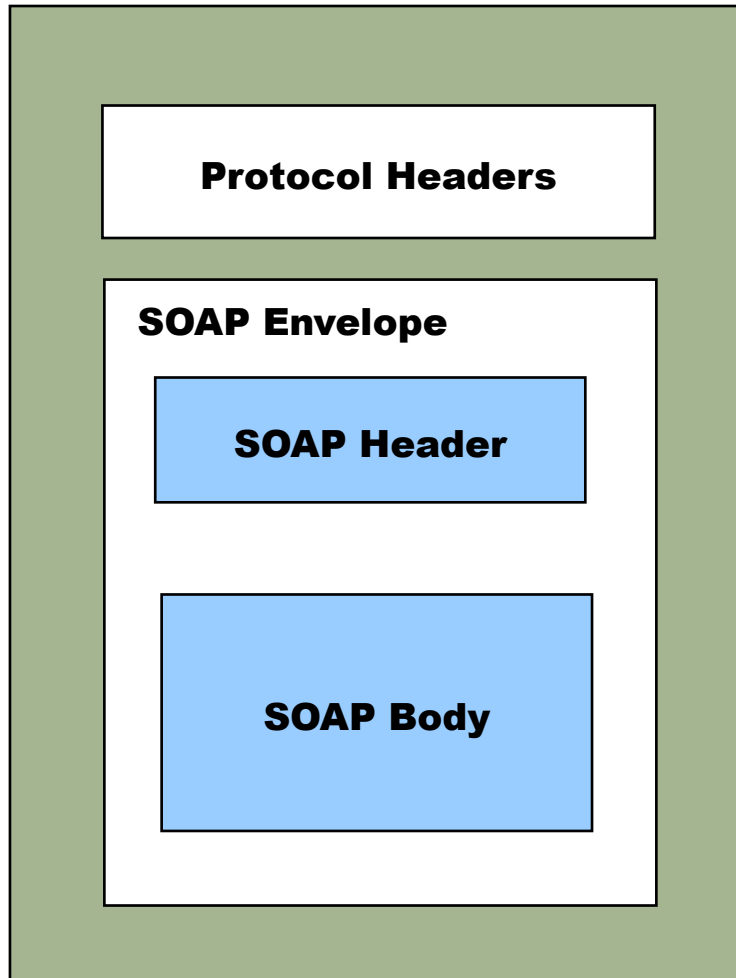
<Envelope> est la racine

<Header>, <Body> et <Fault> sont les fils :

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
  <soap:Header>
    ... Header information...
  </soap:Header>
  <soap:Body>
    ... Body information...
    <soap:Fault>
      ...Fault information...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

# Structure

35



- **Éléments d'un message SOAP :**
  - Enveloppe
    - Élément pouvant contenir des déclarations d'espaces de noms ou des sous-éléments
  - Header (optionnel)
    - Permet des extensions telles que authentication, session...
  - Body (obligatoire)
    - Définit la méthode appelée, contient les paramètres
    - Peut contenir un élément Fault en cas d'erreur

# SOAP Header

36

- SOAP Header : Mécanisme d'extension du protocole SOAP
  - La balise Header est optionnelle
  - Si la balise Header est présente, elle doit être le premier fils de la balise Enveloppe
  - La balise Header contient des *entrées*
  - Une *entrée* est n'importe quelle balise incluse dans un namespace. Les *entrées* contenues dans la balise Header sont non applicatives.

# SOAP Header

37

## □ Exemple

```
<SOAP-ENV:Header>
```

```
<t:Transaction xmlns:t="some-URI" SOAP-ENV:mustUnderstand="0" >
```

**5**

```
</t:Transaction>
```

```
</SOAP-ENV:Header>
```

## □ L'attribut mustUnderstand

- si Rien ou =0 : l'élément est optionnel pour l'application réceptrice
- si =1 : l'élément doit être exploité de l'application réceptrice. Sinon le traitement du message par le récepteur doit échouer
- Dans notre cas, L'id 5 de la transaction est facultatif pour l'application réceptrice

# SOAP Body

38

- SOAP Body : Le Body contient le message à échanger
  - La balise Body est obligatoire
  - La balise Body doit être le premier fils de la balise Enveloppe (ou le deuxième si il existe une balise Header)
  - La balise Body contient des *entrées* qui sont des données applicatives.
  - Une *entrée* est n'importe quelle balise incluse optionnellement dans un namespace
  - Une *entrée* peut être une Fault.

# SOAP Fault

39

- ❑ SOAP Fault : Balise permettant de signaler des cas d'erreur.
- ❑ SOAP 1.1 : La balise Fault contient les balises suivantes:
  - Faultcode (Obligatoire): un code permettant d'identifier le type d'erreur.
  - Faultstring (Obligatoire): une explication en langage naturel.
  - Faultactor : une information identifiant l'initiateur de l'erreur.
  - Detail : Définition précise de l'erreur.

# SOAP Fault

40

- Faultcode : 4 groupes de code d'erreur

- Client,
- Server,
- MustUnderstand,
- VersionMismatch

- Exemple SOAP 1.1

```
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <s:Fault>
      <faultcode xmlns="">s:Client</faultcode>
      <faultstring xml:lang="fr-FR" xmlns="">
        Une opération invalide s'est produite.
      </faultstring>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

# Exemple

41

## Requête HTTP invoquant une méthode *GetOrders* :

```
POST /Orders HTTP/1.1
Host: xxx.xxx.xxx.xxx
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://www.someorders.com/GetOrders"
```

```
<?xml version="1.0"?>
```

```
<SOAP-ENV:Envelope
```

```
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
    <SOAP-ENV:Body>
```

```
      <orders:GetOrders
```

```
        xmlns:orders="http://www.someorders.com/orders">
```

```
          <CustomerID>ALFKI</CustomerID>
```

```
          <SalesRepID>85</SalesRepID>
```

```
        </orders:GetOrders>
```

```
      </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```



# Exemple

42

## Réponse HTTP à la requête précédente :

```
HTTP/1.1 200 OK
MessageType: CallResponse
Content-Type: text/xml
```

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <orders:GetOrdersResponse
      xmlns:orders="http://www.someorders.com/orders">
      <OrderID>102</OrderID>
      <OrderDesignation>Cofe</OrderDesignation>
    </orders:GetOrdersResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# SOAP – Type

43

- Les données contenues dans un message SOAP étant typées il faut définir un moyen de les encoder.
- L'encodage c'est la représentation de valeurs sous forme XML.
- Le décodage est la construction de valeurs à partir d'XML
- L'XML représentant des valeurs a une structure qui dépend du type de ces valeurs
- Il faut donc définir ce type
  - Soit par un mécanisme définit par l'utilisateur
  - Soit en utilisant des Schéma XML (préconisé)

# SOAP – Type

44

## □ Définition

### ▣ Value (valeur d'une donnée)

- Simple value (string, integers, enumeration, etc)
- Complex value (array, struct, ...)

### ▣ Type (d'une value)

- Simple Type
- Complex Type

# SOAP – Type

45

- Types simples
  - ▣ Types définis dans XML Schéma
  - ▣ Entier, flottant, entiers négatifs, chaînes caractères, ...
  - ▣ Énumérations
  
- Types complexes
  - ▣ Tableaux
  - ▣ Partie d'un tableau
  - ▣ Objet

# Exemple: SOAP – Simple Type

46

## □ Types xml simples

```
<element name="age" type="int" />
```

```
<element name="taille" type="float" />
```

```
<age>23</age>
```

```
<taille>1.87</taille>
```

## □ Enumeration

```
<element name="couleur" />
```

```
<simpleType base="xsd:string">
```

```
  <enumeration value="Rouge">
```

```
  <enumeration value="Bleu">
```

```
</simpleType>
```

```
</element>
```

```
<couleur>Bleu</couleur>
```

# Exemple: SOAP – Complex Type

47

```
<element name="Personne">
```

```
<complexType>
```

```
  <element name="Nom" type="xsd:string">
```

```
  <element name="Prenom" type="xsd:string">
```

```
  <element name="Age" type="xsd:float">
```

```
</complexType>
```

```
</element>
```

```
<Personne>
```

```
  <Nom>Durand</Nom>
```

```
  <Prenom>Michel</Prenom>
```

```
  <Age>34.7</Age>
```

```
</Personne>
```

# Exemple: SOAP – Complex Type

48

## □ Tableau

```
<element name="TabPersonne">  
  <complexType base="SOAP-ENC:Array">  
    <element name="Personne" type="Personne" maxOccurs="unbounded">  
    </complexType>  
  </element>
```

```
<xyz:TabPersonne SOAP-ENC:arrayType="Personne[2]">  
  <Personne>  
    <Nom>Durand</Nom>  
    <Prenom>Michel</Prenom>  
    <Age>34.7</Age>  
  </Personne>  
  <Personne>  
    <Nom>Dupond</Nom>  
    <Prenom>Serge</Prenom>  
    <Age>40</Age>  
  </Personne>  
</xyz:TabPersonne>
```

The slide features a decorative header with two horizontal bars. The top bar is thin and olive green, while the bar below it is thicker and orange. The text 'Partie 4' is centered within the olive green bar.

## Partie 4

**WSDL : Web Service description Language**



# Rôle du langage WSDL

50

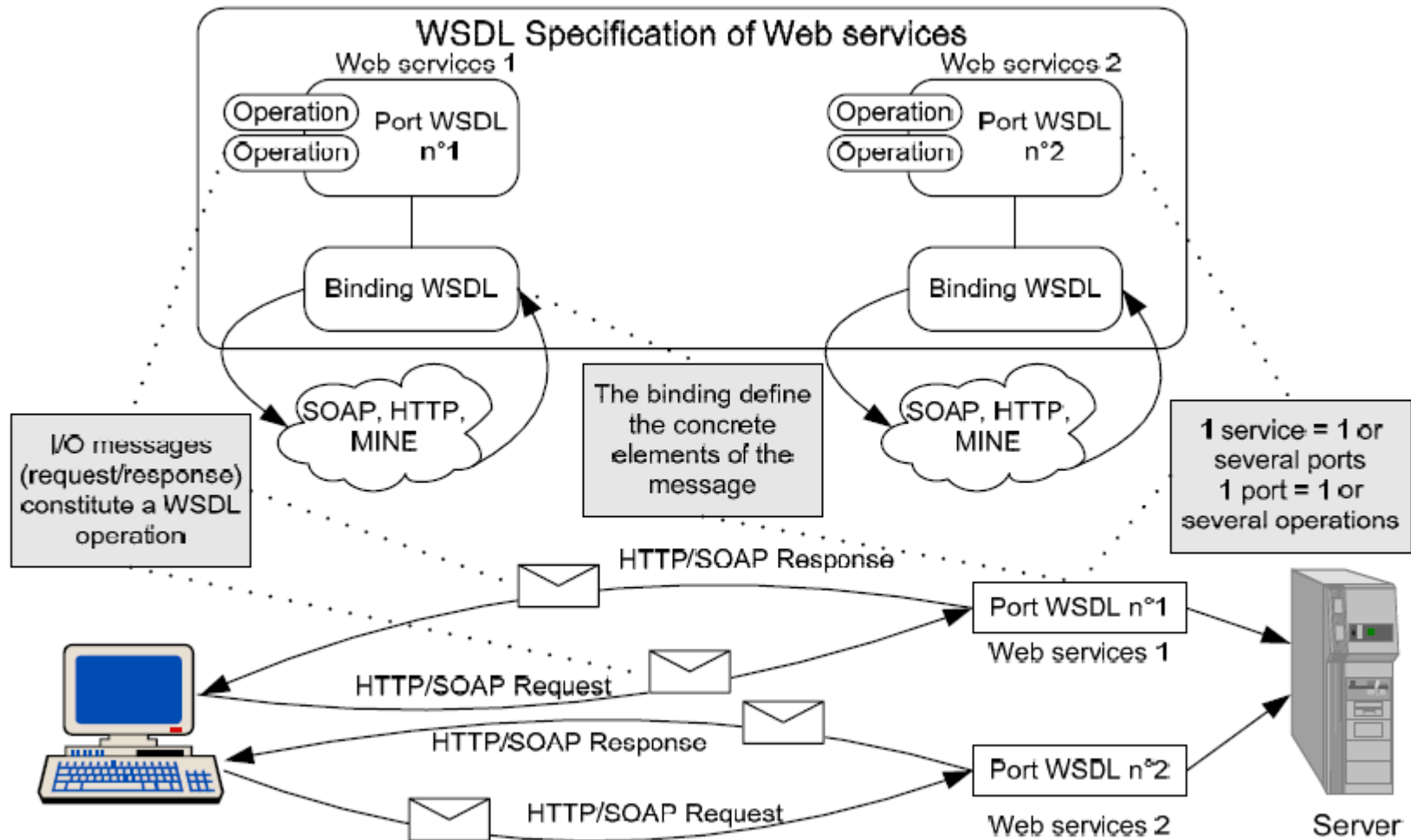
- WSDL est un langage qui permet de décrire :
  - ▣ l'interface d'un service Web (types de données, opérations, entrées, sorties)
  - ▣ comment invoquer un service Web
- Objectif :
  - ▣ Décrire les services comme un ensemble d'opérations et de messages abstraits reliés à des protocoles et des serveurs réseaux
  - ▣ Permet de décharger les utilisateurs des détails techniques de réalisation d'un appel
- WSDL est un langage qui standardise les schémas XML utilisés pour établir une connexion entre émetteurs et récepteurs.

# Web Services – WSDL

- Deux version du wsdl sont définies par le W3C
  - ▣ WSDL 1 est prévu pour fonctionner avec SOAP 1.1  
<http://www.w3.org/TR/wsdl>
  - ▣ WSDL 2 est prévu pour fonctionner avec SOAP 1.2  
<http://www.w3.org/TR/wsdl20>

# Description des Web services via WSDL

52



# Structure

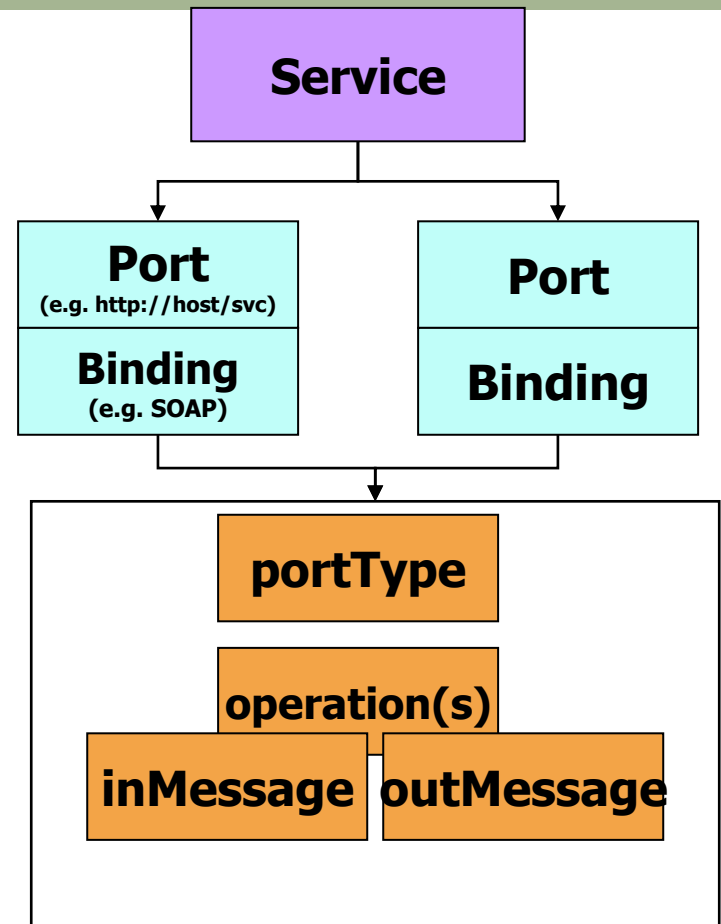
- Un document WSDL est constitué de plusieurs éléments permettant la plus grande abstraction possible dans la définition des services
- Ces différents éléments permettent de séparer les briques habituellement mises en jeu dans l'utilisation des services
- Ces briques sont pour l'essentiel :
  - L'adresse où est situé le service
  - Le protocole associé à l'utilisation du service
  - L'ensemble des opérations accessibles
  - Les type de messages à utiliser en entrée et en sortie des opérations
  - Les types de données à véhiculer dans les messages
  - ...

# Concepts

- Un service : une collection de ports ( port ou endpoints )
- Un port : une adresse réseau et un binding
- Un binding : un protocole et un format de données associé à un type de port (port type)
- Un type de port : un ensemble d'opérations (proche d'une interface au sens Java)
- Une opération : une action proposée par un service web, décrite par ses messages (proche d'une méthode au sens Java)
- Un message : un ensemble de données
- Une donnée : une information typée selon un système de type comme celui des schémas du W3

# Description

- ▶ **Élément Type**
  - ▶ Types des paramètres (schéma XML)
- ▶ **Élément Message**
  - ▶ Appel et retour d'opération
- ▶ **Élément Port type**
  - ▶ Groupe d'opération
- ▶ **Élément Binding**
  - ▶ URL de l'opération
  - ▶ Type de protocole



# Structure d'un document WSDL

56

<definitions>

<types>définition des types</types>

<message>définition des messages</message>

<portType>définition des interfaces </portType>

<binding>définition des bindings</binding>

<services>définition de endpoint</service>

</definitions>

# HelloWord.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloWorld"
  targetNamespace=http://hello.jaxrpc.samples/
  xmlns:tns=http://hello.jaxrpc.samples/
  xmlns=http://schemas.xmlsoap.org/wsdl/
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/ >
  <types />
    <message name="sayHello">
      <part name="String_1" type="xsd:string" />
    </message>
    <message name="sayHelloResponse">
      <part name="result" type="xsd:string" />
    </message>
  <portType name="Hello">
    <operation name="sayHello" parameterOrder="String_1">
      <input message="tns:sayHello" />
      <output message="tns:sayHelloResponse" />
    </operation>
  </portType>
```



# HelloWord.wsdl

```
<binding name="HelloBinding" type="tns:Hello">
  <operation name="sayHello">
    <input>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        use="encoded" namespace="http://hello.jaxrpc.samples/" />
    </input>
    <output>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        use="encoded" namespace="http://hello.jaxrpc.samples/" />
    </output>
    <soap:operation soapAction="" />
  </operation>
</binding>
<service name="HelloWorld">
  <port name="HelloPort" binding="tns:HelloBinding">
    <soap:address location="http://localhost:8080/axis/Hello" />
  </port>
</service>
</definitions>
```

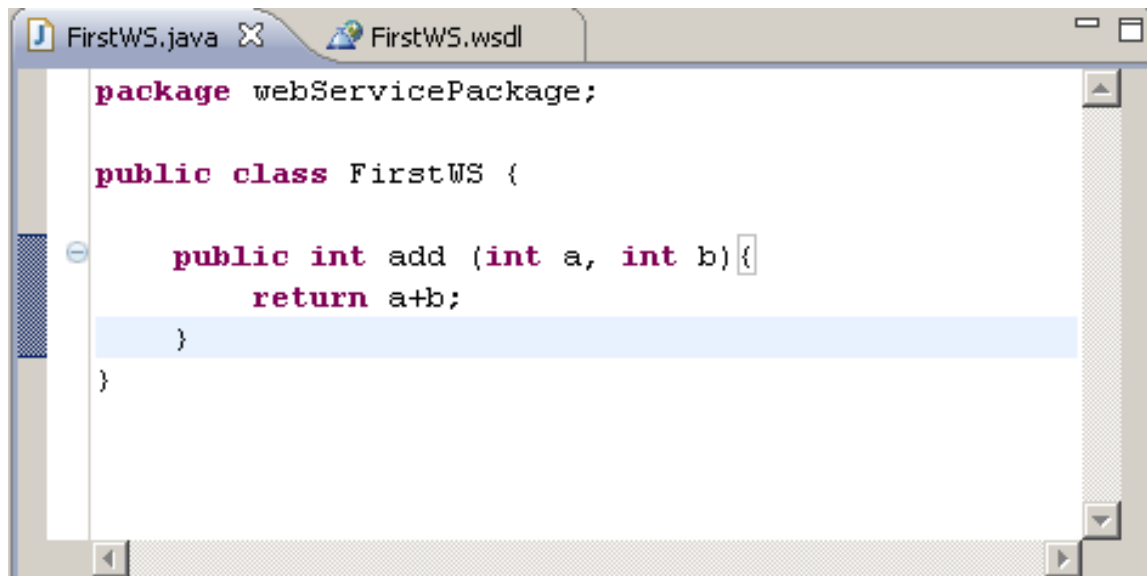
# HelloWord



- Le service HelloWorld
- Propose l'opération sayHello (qui prend comme paramètre une chaîne de caractères et renvoie une chaîne de caractères)
- Est accessible sur <http://localhost:8080/axis/Hello>
- Par des messages SOAP
- Sous forme RPC au dessus de HTTP

# Exemple de service : Addition

- Exemple (Service Web - Java)
  - Nous commençons par un service Web simple qui consiste à calculer la somme de deux valeurs entiers.



```
package webServicePackage;

public class FirstWS {

    public int add (int a, int b){
        return a+b;
    }
}
```

- Les WSDL présentés dans la suite sont générés automatiquement par Java2WSDL

# Types

- L'élément **types** contient les définitions de types utilisant un système de typage (comme XML schéma)
  - `<wsdl:types>`
    - `<schema elementFormDefault="qualified" targetNamespace="http://webServicePackage" xmlns="http://www.w3.org/2001/XMLSchema">`
      - `<element name="add">`
        - `<complexType>`
          - `<sequence>`
            - `<element name="a" type="xsd:int" />`
            - `<element name="b" type="xsd:int" />`
          - `</sequence>`
        - `</complexType>`
      - `</element>`
      - `<element name="addResponse">`
        - `<complexType>`
          - `<sequence>`
            - `<element name="addReturn" type="xsd:int" />`
          - `</sequence>`
        - `</complexType>`
      - `</element>`
    - `</schema>`
  - `</wsdl:types>`

# message

- Les **messages** sont envoyés entre deux interlocuteurs (exemple: une opération reçoit et envoie des messages).

```
+ <wsdl:types>
- <wsdl:message name="addResponse">
    <wsdl:part name="parameters" element="impl:addResponse" />
</wsdl:message>
- <wsdl:message name="addRequest">
    <wsdl:part name="parameters" element="impl:add" />
</wsdl:message>
+ <wsdl:portType name="FirstWS">
+ <wsdl:binding name="FirstWSSoapBinding" type="impl:FirstWS">
+ <wsdl:service name="FirstWSService">
+ <!-- -->
</wsdl:definitions>
```

# PortType

- Un **portType** permet d'identifier (nommer) de manière abstraite un ensemble d'opérations.
  
- WSDL définit 4 types d'opération :
  - One-Way : un client envoie un message à l'opération, mais ne s'attend à aucune réponse
  - Request-response : le client envoie la demande, et l'opération répond
  - Solicit-response : lorsque les opérations envoient des messages et le client répond.
  - Notification : lorsque les opérations envoient des messages au client, mais n'attend pas de réponse

# Operation

- Quelque soit le type d'opération la définition est sensiblement la même.
- Une opération :
  - Reçoit des messages : `<wsdl:input ...>`
  - Envoie des messages : `<wsdl:output ...>` ou `<wsdl:fault ...>`
- La présence et l'ordre des input/outputs/fault dépendent du type de l'opération.

```
- <wsdl:portType name="FirstWS">  
  - <wsdl:operation name="add">  
    <wsdl:input name="addRequest" message="impl:addRequest" />  
    <wsdl:output name="addResponse" message="impl:addResponse" />  
  </wsdl:operation>  
</wsdl:portType>
```

# Binding SOAP

- WSDL permet de lier une description abstraite (portType) à un protocole.

```
<wsdl:binding name="binding_name" type="portType_name" >  
...  
</wsdl:binding>
```

- Pour préciser que le binding est de type SOAP, il faut inclure la balise suivante :

```
<soap:binding transport="uri" style="soap_style" />
```

- ▣ Transport définit le type de transport
  - <http://schemas.xmlsoap.org/soap/http>  
pour utiliser SOAP/HTTP
- ▣ Style définit la façon dont sont créés les messages SOAP de toutes les opérations
  - rpc : Encodage RPC défini par SOAP RPC
  - document : Encodage sous forme d'élément XML



# Binding: exemple

```
- <wsdl:binding name="FirstWSSoapBinding" type="impl:FirstWS">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="add">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="addRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="addResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

# Service

- Un service est un ensemble de ports
- Un port a un portType
- Dans le cadre de SOAP, un port à une adresse (qui correspond à l'adresse http)

```
<wsdl:service name="Service_name">  
  <wsdl:port name="port_name" binding="binding_name">  
    <soap:address location="adress_Of_Deployed_Service"/>  
  </wsdl:port>  
</wsdl:service>
```

```
- <wsdl:service name="FirstWSService">  
- <wsdl:port name="FirstWS" binding="impl:FirstWSSoapBinding">  
  <wsdlsoap:address location="http://localhost:8080/FirstWebService/services/FirstWS" />  
  </wsdl:port>  
</wsdl:service>
```

68

## Partie 2.1

### SOAP Web Services

# Création d'un Web Service SOAP

69

- Top-Down :
  1. Ecriture du WSDL
  2. Ecriture du code associé
  
- Bottom-Up :
  1. Ecriture du code du service
  2. Génération automatique du WSDL à partir de ce code

# Approche Top-Down

70

- Génération du code Java à partir du WSDL décrit
  - ▣ Exemple en java :
    - `wsimport -d src/generated`  
<http://example.org/stock?wsdl>
    - `wsdl2java -d src/generated -server -`  
`client` <http://example.org/stock?wsdl>

# Approche Bottom-Up

71

- Une fois le code des services écrit, on génère le WSDL
  - ▣ Exemple en java avec code annoté :
    - `wsgen -cp . ws.Hello`
  - ▣ Avec un conteneur web et JAX-WS, ceci est automatique

# Création d'un Web Service

72

- Technologies existantes :
  - Apache Axis
  - Axis 2
  - Java JAX-WS
  - Metro
  - Apache CXF (Permet l'implémentation de service rest)
  - JAX-RS (Restful WS)

73

## Axis 2



# Axis 2 : Concept

74

- Axis est une librairie de Apache permettant la création et le déploiement des web services. Elle fournit :
  - ▣ Un serveur SOAP indépendant
  - ▣ Une API pour développer des services web SOAP
  - ▣ Le support de différentes couches de transport : HTTP, FTP, SMTP, POP et IMAP, ...
  - ▣ La sérialisation/désérialisation automatique d'objets Java dans des messages SOAP
  - ▣ Des outils pour créer automatiquement les WSDL correspondant à des classes Java ou inversement pour créer les classes Java sur la base d'un WSDL.
  - ▣ des outils pour déployer, tester et surveiller des web-services.

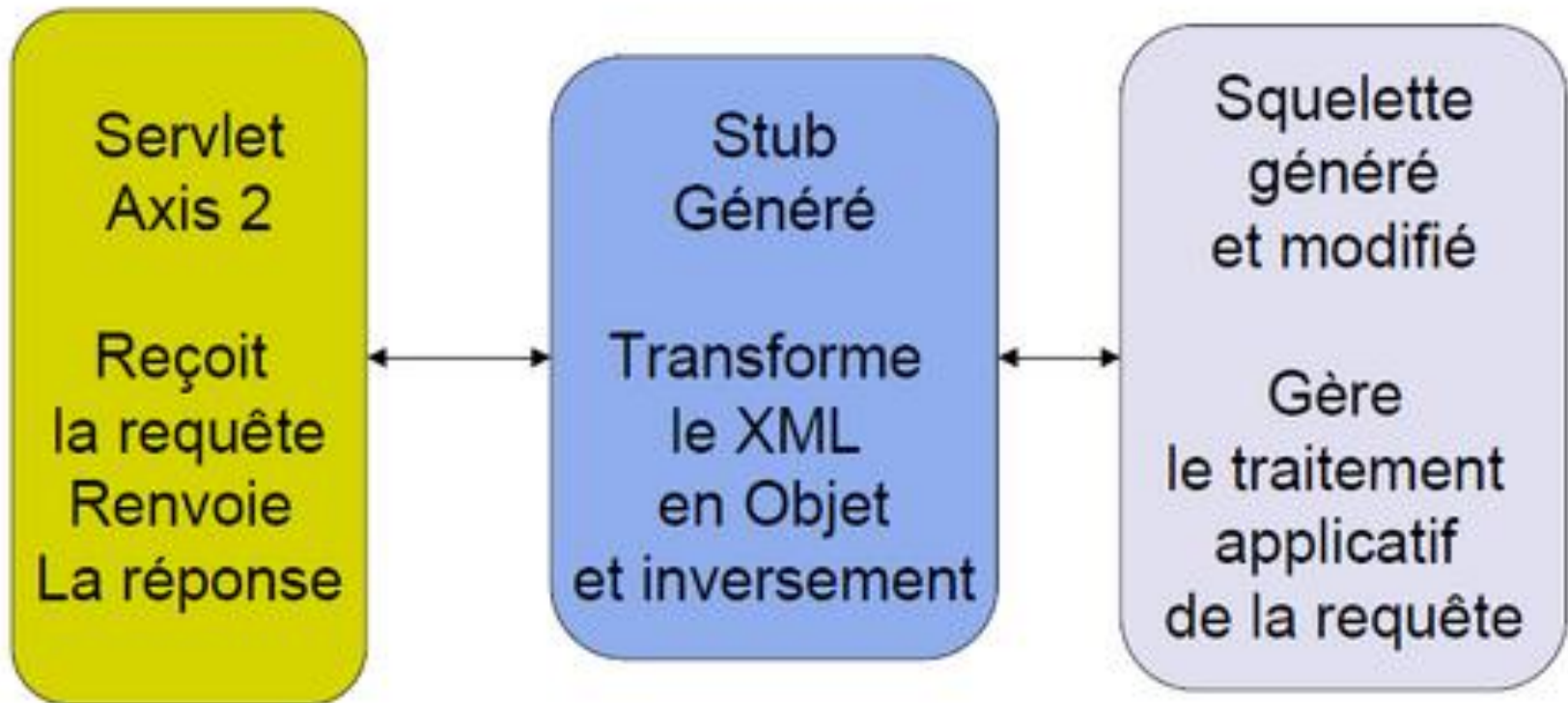
# Architecture (Serveur)

75

- ❑ Axis fournit une Servlet (AxisServlet) qui reçoit des message SOAP sur http et qui transforme l'appel en un appel de méthode classique Java
- ❑ Développer un Web Service revient alors à développer un objet Java et à enregistrer ses méthodes auprès de la Servlet AxisServlet.
- ❑ Les clients envoient alors leurs messages SOAP sur http à AxisServlet.
- ❑ Pour SMTP les clients envoient leurs messages par mail à un démon. Le démon reçoit ces messages et les renvoie sur http à AxisServlet.

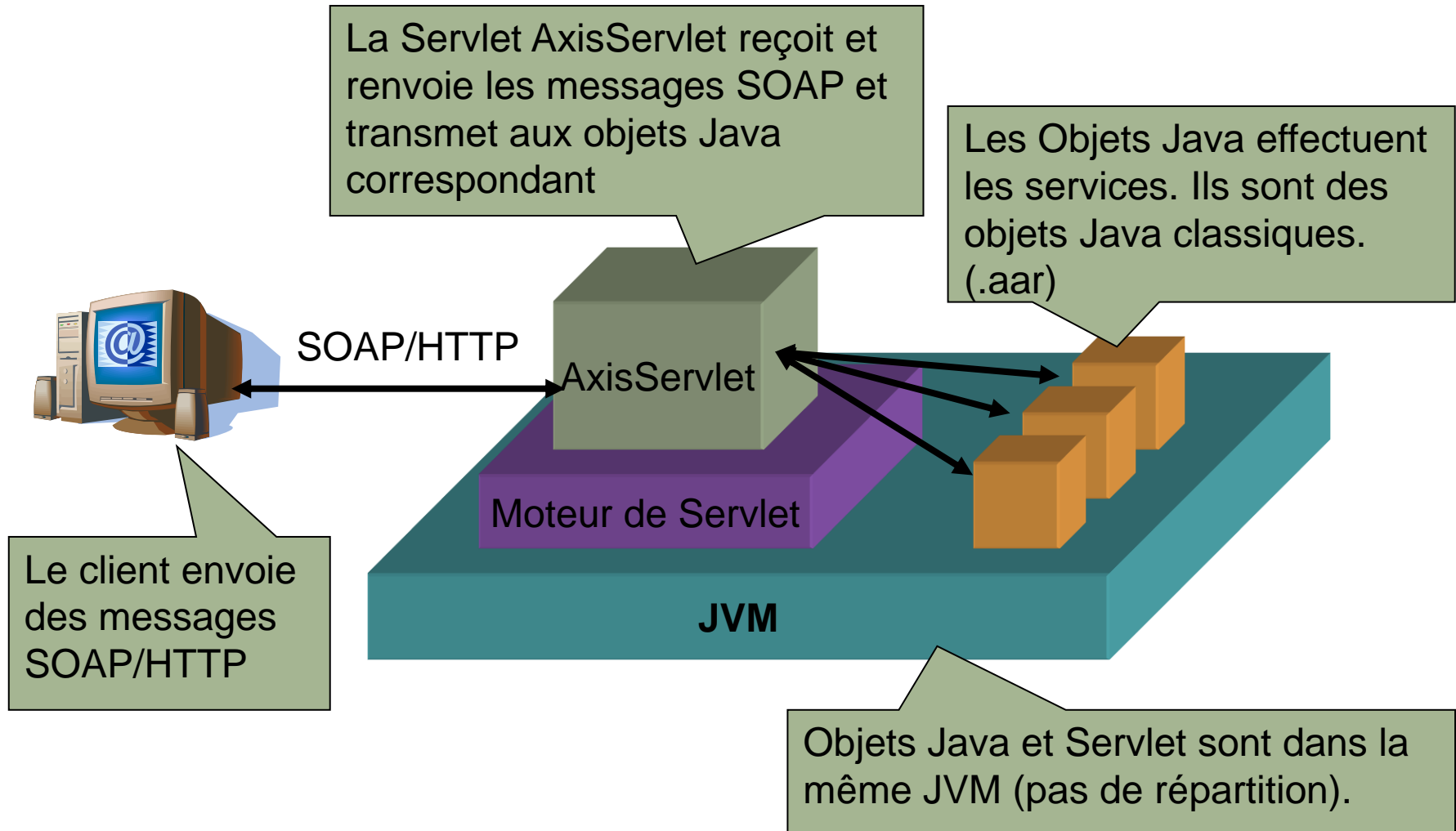
# Fonctionnement : runtime

76



# Fonctionnement : runtime

77



# Développement d'un Web Service 1 / 3

78

- La création d'un nouveau service Web avec Apache Axis2 comporte quatre étapes:
  - ▣ Écrire la classe d'implémentation.
  - ▣ Écrire un fichier services.xml pour expliquer le service Web.
  - ▣ Créer une archive \* .aar (Axis Archive) pour le service Web.
  - ▣ Déployer le service Web.

# Développement d'un Web Service 2/3

79

```
public class MyFirstWebService {  
    public final String BOOK1 = "La méthode";  
    public final String BOOK2 = "Le Macroscopie";  
  
    public int getPrice(String bookTitle) {  
        if (bookTitle.compareTo(BOOK1)==0) {  
            return 15;  
        }  
        else if (bookTitle.compareTo(BOOK2)==0) {  
            return 20;  
        }  
        else return 300;  
    }  
}
```

# Services.xml

80

```
<service name=" MyFirstWebService" >
  <Description> Please Type your service description here
</Description>
  <messageReceivers>
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
      class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
      class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
  </messageReceivers>
  <parameter name="ServiceClass" locked="false">
    com.troisieme. MyFirstWebService
  </parameter>
</service>
```

# Déploiement d'un Web Service

81

- ❑ Exporter le projet ainsi créé sous forme d'archive java (.aar)
- ❑ Pour déployer le service il suffit de placer le fichier .aar généré sous le répertoire Webservices \$TomeCat\webapps\axis2\WEB-INF\services et redémarrer le serveur d'application.

- ❑ L'url du fichier WSDL est :

`http://localhost:8080/axis2`  
`/services/MyFirstWebService?wsdl`

```
-MyFirstWebService
  -META-INF
    -services.xml
  -lib
  -com
    -Troisieme
      -MyFirstWebService.class
```





# JAX-WS

# JAX-WS

83

- **Java API for XML Web Services**
- **JAX-WS 2.0 : Standard dans Java EE 5**
- **JAX-WS utilise des annotations Java pour créer les services**

# JAX-WS

84

- JAX-WS définit un ensemble d'API et d'annotations permettant de construire et de consommer des services web en Java.
- Elle fournit les outils pour envoyer et recevoir des requêtes de services web via SOAP en masquant la complexité du protocole.
- Ni le consommateur ni le service n'ont donc besoin de produire ou d'analyser des messages SOAP car JAX-WS s'occupe de traitement de bas niveau.
- JAX-WS dépend lui-même d'autres spécifications comme JAXB.

# Annotations JAX-WS

85

- `@WebService` : La classe annotée est déclarée comme étant un service
- `@WebMethod` : La méthode annotée doit être exposée en tant qu'opération du service
- `@WebParam` : Permet de spécifier les propriétés d'un paramètre
- Par défaut, les méthodes publiques de la classe sont exposées

# Appeler un Web Service

86

- Doit toujours se baser sur le WSDL (Top-Down)
- Génération du code Java à partir du WSDL
- La plupart des IDE savent générer le code Java à partir d'un WSDL automatiquement



Type 1 : Service déployé sur tomcat

# Méthodologie de développement de web service

88

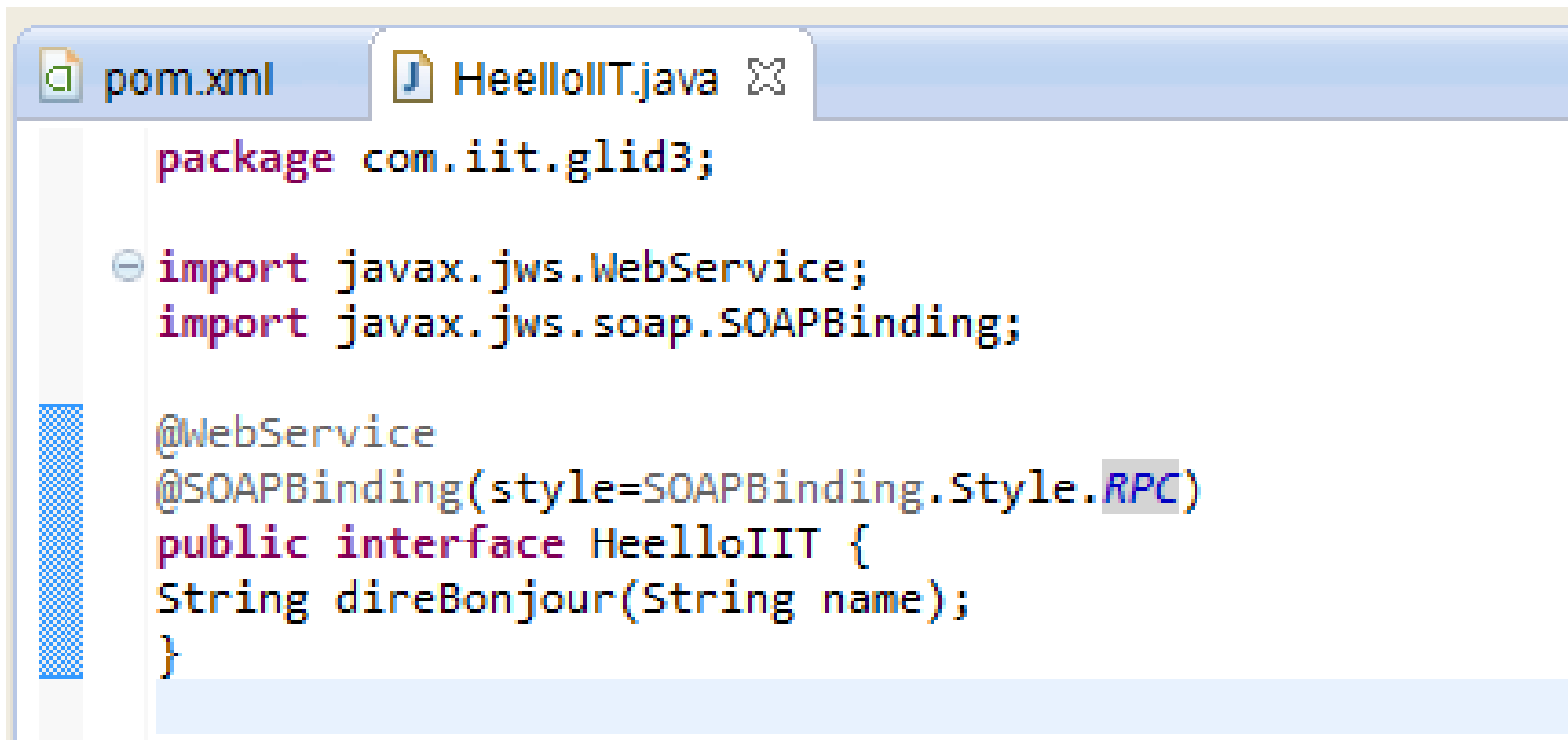
- Nous allons ici mettre en œuvre l'approche 'bottom-up' :
  1. Création de l'interface java du service (optionnel)
  2. Création de l'implémentation du service (obligatoire)
  3. Publication du service web (obligatoire)
- Nous utilisons un dynamic web project
- Nous utilisons la dépendance maven suivante:

```
<dependency>  
    <groupId>com.sun.xml.ws</groupId>  
    <artifactId>jaxws-rt</artifactId>  
    <version>2.2.5</version>  
</dependency>
```

# Partie 1 : interface

89

- Le web service que nous allons créer expose une seule opération : **ditBonjour()**



```
package com.iit.glid3;


import javax.xml.ws.WebService;
import javax.xml.ws.soap.SOAPBinding;

@WebService
@SOAPBinding(style=SOAPBinding.Style.RPC)
public interface HeelloIIT {
    String direBonjour(String name);
}
```



# Partie 2 : implémentation

90



The screenshot shows an IDE with four tabs: pom.xml, HeelloIIT.java, HelloIITWS.java (active), and HeelloIITPub. The code in the active tab is as follows:

```
package com.iit.glid3;

import javax.ws.WebService;

@WebService(endpointInterface="com.iit.glid3.HeelloIIT")
public class HelloIITWS implements HeelloIIT{

    @Override
    public String direBonjour(String name) {
        // TODO Auto-generated method stub
        return "hello "+name+" !!! ";
    }
}
```

# Partie 3 : publication (1 / 3)

91

Nous ajoutons un fichier de configuration Web-Inf/***sun-jaxws.xml*** dont le contenu est le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<endpoints xmlns="http://java.sun.com/xml/ns/jaxws/ri/runtime" version="2.0">
```

```
  <endpoint
```

```
    name="hello"
```

```
    implementation="com.iit.glid3.HelloIITWS"
```

```
    url-pattern="/hello" />
```

```
</endpoints>
```

# Partie 3 : publication (2/3)

92

Nous modifions le fichier de configuration Web.xml en ajoutant la portion suivante

```
<listener>
```

```
    <listener-class> com.sun.xml.ws.transport.http.servlet.WSServletContextListener </listener-class>
```

```
</listener>
```

```
<servlet>
```

```
<servlet-name>WSServlet</servlet-name>
```

```
    <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
```

```
    <load-on-startup>1</load-on-startup>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
    <servlet-name>WSServlet</servlet-name>
```

```
    <url-pattern>/hello</url-pattern>
```

```
</servlet-mapping>
```

# Partie 3 : publication (3/3)

93

**com.sun.xml.ws.transport.http.servlet.WSServletContextListener** est une classe d'écoute (listener) qui lit sur le fichier `/WEB-INF/sun-jaxws.xml`

Nous déployons le projet sur Tomcat et nous visitons l'adresse suivante:

[http://localhost:5055/TP2\\_Service/hello](http://localhost:5055/TP2_Service/hello)

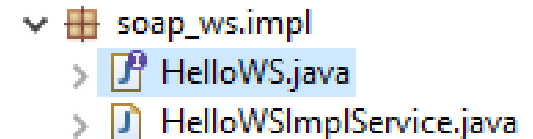
## Web Services

Endpoint	Information
Service { <a href="http://glid3.iit.com/">http://glid3.iit.com/</a> Name: HelloIITWSService Port Name: { <a href="http://glid3.iit.com/">http://glid3.iit.com/</a> HelloIITWSPort	Address: <a href="http://localhost:5055/TP2_Service/hello">http://localhost:5055/TP2_Service/hello</a> WSDL: <a href="http://localhost:5055/TP2_Service/hello?wsdl">http://localhost:5055/TP2_Service/hello?wsdl</a> Implementation class: com.iit.glid3.HelloIITWS

# Client du Web Service

94

- Ouvrir une invite de commande et exécuter la commande suivante:
  - `java -cp "lib\*" com.sun.tools.ws.WsImport -s  
E:\Ahmed\WS_Client\client2  
http://localhost:9999/soap_ws/hello?wsdl`
- Les stubs client seront générés
  - Deux dans notre cas : HelloWS et HelloWSImplService



# Consommation des Web Services

95

- La consommation des Web Services se fait moyennant les stubs générés:

```
/**  
 * @see HttpServlet#doGet(HttpServletRequest request,  
 *      HttpServletResponse response)  
 */  
protected void doGet(HttpServletRequest request,  
    HttpServletResponse response) throws ServletException, IOException  
{  
    HelloServiceImplService service = new HelloServiceImplService();  
    HelloWS port = service.getHelloServiceImplPort();  
    response.getWriter().append("Served at:  
").append(port.direBonjour("Ahmed"));  
}
```



Type 2 : Service autonome avec  
provider

# Dépendances

97

- `<dependencies>`
- `<dependency>`
  - `<groupId>com.sun.xml.ws</groupId>`
  - `<artifactId>jaxws-rt</artifactId>`
  - `<version>2.3.1</version>`
  - `<type>pom</type>`
- `</dependency>`
- `<dependency>`
  - `<groupId>com.sun.xml.ws</groupId>`
  - `<artifactId>rt</artifactId>`
  - `<version>2.2.10</version>`
  - `<scope>compile</scope>`
- `</dependency>`



# Dépendances

98

- `<dependency>`
  - `<groupId>com.sun.org.apache.xml.internal</groupId>`
  - `<artifactId>resolver</artifactId>`
  - `<version>20050927</version>`
- `</dependency>`
- `</dependencies>`

# Interface du service

99

```
package com.iit.glid3;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.DOCUMENT)
public interface HelloWorld{
    @WebMethod String direBonjour(String name);
}
```

# Implémentation du service

100

```
package com.iit.glid3;

import javax.ws.WebService;

//Service Implementation
@WebService(endpointInterface = "tn.esps.HelloWorld")
public class HelloWorldImpl implements HelloWorld{

    @Override
    public String direBonjour(String name) {
        return "Hello World JAX-WS " + name;
    }

}
```

# Ajout d'un provider

101

```
package com.iit.glid3;
import javax.xml.ws.Endpoint;

public class HelloWorldPublisher{

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:7779/ws/hello", new
        HelloWorldImpl());
    }

}
```

# Client du WS

102

```
package com.iit.glid3;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;

public class HelloWorldClient{
    public static void main(String[] args) throws Exception {
        URL url = new URL("http://localhost:7779/ws/hello?wsdl");
        QName qname = new QName("http://esps.tn/", "HelloWorldImplService");
        Service service = Service.create(url, qname);
        HelloWorld hello = service.getPort(HelloWorld.class);
        System.out.println(hello.direBonjour("WS document"));
    }
}
```

# Interface du service coté client

103

```
package com.iit.glid3;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.DOCUMENT)
public interface HelloWorld{
    @WebMethod String direBonjour(String name);
}
```

104

# Apache CXF

# Définition

105

- Apache CXF est un framework Open Source de développement de services simplifiant:
  - ▣ la création,
  - ▣ la configuration,
  - ▣ le déploiement
  - ▣ et l'utilisation de Services Web
- Apache CXF supporte les standards (SOAP, HTTP, JMS...)
- CXF propose une approche Code-First. Cette approche consiste à développer d'abord le code « métier » du Web Service et de se baser ensuite sur apache CXF pour générer automatiquement le « contrat » nécessaire au dialogue avec le Web Service.



# Dépendance de CFX

106

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-frontend-jaxws</artifactId>
  <version>2.2.3</version>
</dependency>
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-transports-http</artifactId>
  <version>2.2.3</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring</artifactId>
  <version>2.5.5</version>
</dependency>
```

# publication (1 / 2)

107

Nous ajoutons un fichier de configuration Web-Inf/**cxf-servlet.xml** dont le contenu est le suivant :

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xmlns:soap="http://cxf.apache.org/bindings/soap"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://cxf.apache.org/jaxws
    http://cxf.apache.org/schemas/jaxws.xsd">
  <jaxws:endpoint
    id="helloCXF"
    implementor="com.iit.glid3.HelloIITWS"
    address="/helloCXF">
  </jaxws:endpoint>
</beans>
```

# Partie 3 : publication (2/2)

108

Nous modifions le fichier de configuration Web.xml en ajoutant la portion suivante

```
<servlet>
```

```
<servlet-name>cx</servlet-name>
```

```
<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
```

```
<load-on-startup>1</load-on-startup>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
<servlet-name>cx</servlet-name>
```

```
<url-pattern>/services/*</url-pattern>
```

```
</servlet-mapping>
```

# WSDL File

109

- [http://localhost:5055/TP3\\_Service\\_CXF/services/helloCXF?wsdl](http://localhost:5055/TP3_Service_CXF/services/helloCXF?wsdl)

# Application

110

- Les taux de change en cours permettant de faire la conversion d'un montant en euros dans d'autres monnaies : Dollars américains/canadiens, Yen... sont disponibles dans l'URL suivante :  
`http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml`
- On vous demande d'écrire un services Web selon l'implementation qui vous convient qui :
  1. retourne la liste des monnaies présentes sur ce fichier xml
  2. exploite les informations fournies sur ce fichier xml pour permettre la conversion d'un montant en euro en une autre monnaie fournis par le client du service.

# Application

111

- On vous demande également d'écrire :
  - ▣ une page Form.jsp qui sollicite le service ListMon pour récupérer la liste des monnaies et les afficher dans une liste déroulante
  - ▣ Une servlet qui appelle le service Conv pour convertir la monnaie et afficher le résultat de conversion.
- Tester l'application sur votre navigateur Web.

126

# Web Service REST

# Web Service REST

127

## Définition

- ❑ Acronyme de **RE**presentational **S**tate **T**ransfert défini dans la thèse de Roy Fielding en 2000.
- ❑ REST n'est pas un protocole ou un format, contrairement à SOAP, HTTP ou RCP, mais un style d'architecture inspiré de l'architecture du web fortement basé sur le protocole HTTP
- ❑ Il n'est pas dépendant uniquement du web et peut utiliser d'autre protocoles que HTTP



# Web Service REST

128

Ce qu'il est :

- Un système d'architecture
- Une approche pour construire une application

Ce qu'il n'est pas

- Un protocole
- Un format
- Un standard

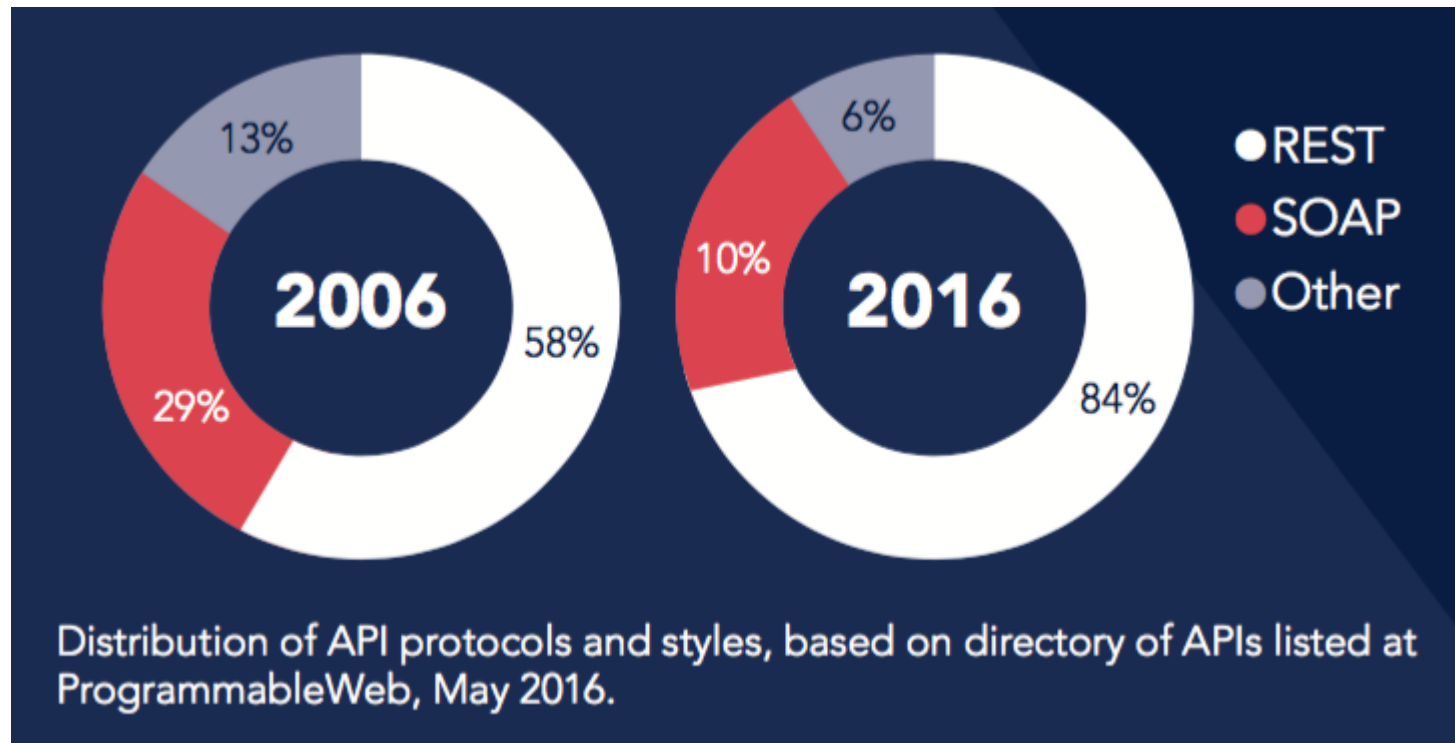
# REST → utilisation

129

- Utiliser dans le développement des applications orientés ressources (ROA) ou orientées données (DOA)
- Les applications respectant l'architecture REST sont dites RESTful

# REST → Statistics

130



# REST → Caractéristiques

131

- Les services REST sont sans états (Stateless)
  - Chaque requête envoyée au serveur doit contenir toutes les informations relatives à son état et est traitée indépendamment de toutes autres requêtes
  - Minimisation des ressources systèmes (pas de gestion de session, ni d'état)
- Interface uniforme basée sur les méthodes HTTP (GET, POST, PUT, DELETE)
- Les architectures RESTful sont construites à partir de ressources uniquement identifiées par des URI(s)

# Requêtes REST

132

- Ressources
  - ▣ Identifiée par une URI  
(<http://unice.fr/cursus/master/miage>)
- Méthodes (verbes) permettant de manipuler les ressources (identifiants)
  - ▣ Méthodes HTTP : GET, POST, PUT, DELETE
- Représentation : Vue sur l'état de la ressource
  - ▣ Format d'échanges entre le client et le serveur (XML, JSON, text/plain,...)

# Ressources

133

- ❑ Une ressource est un objet identifiable sur le système

→ Livre, Catégorie, Client, Prêt

Une ressource n'est pas forcément un objet matérialisé (Prêt, Consultation, Facture...)

- ❑ Une ressource est identifiée par une URI : Une URI identifie uniquement une ressource sur le système → une ressource peut avoir plusieurs identifiants

→ <http://ntdp.miage.fr/bookstore/books/1>

Clef primaire de la  
ressource dans la BDD

# Methodes (Verbes)

134

- Une ressource peut subir quatre opérations de bases CRUD correspondant aux quatre principaux types de requêtes HTTP (GET, PUT, POST, DELETE)
- REST s'appuie sur le protocole HTTP pour effectuer ces opérations sur les objets
  - ▣ CREATE → POST
  - ▣ RETRIEVE → GET
  - ▣ UPDATE → PUT
  - ▣ DELETE → DELETE

# Méthode GET

135

- La méthode GET renvoie une représentation de la ressource tel qu'elle est sur le système

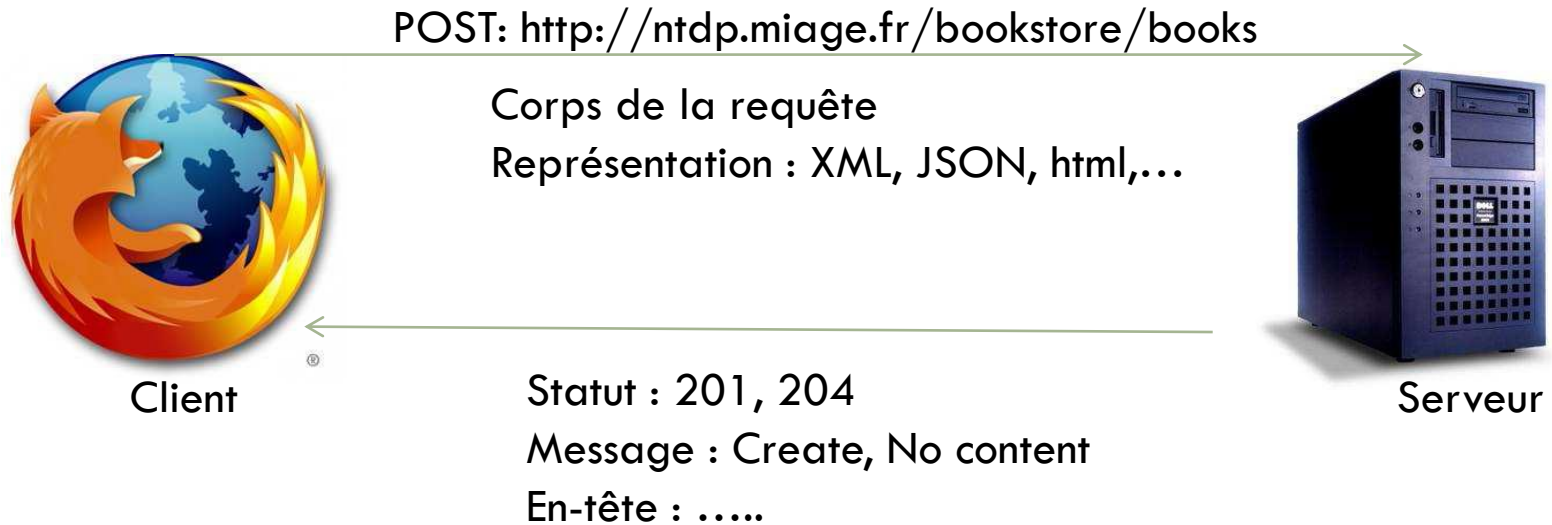




# Méthode POST

136

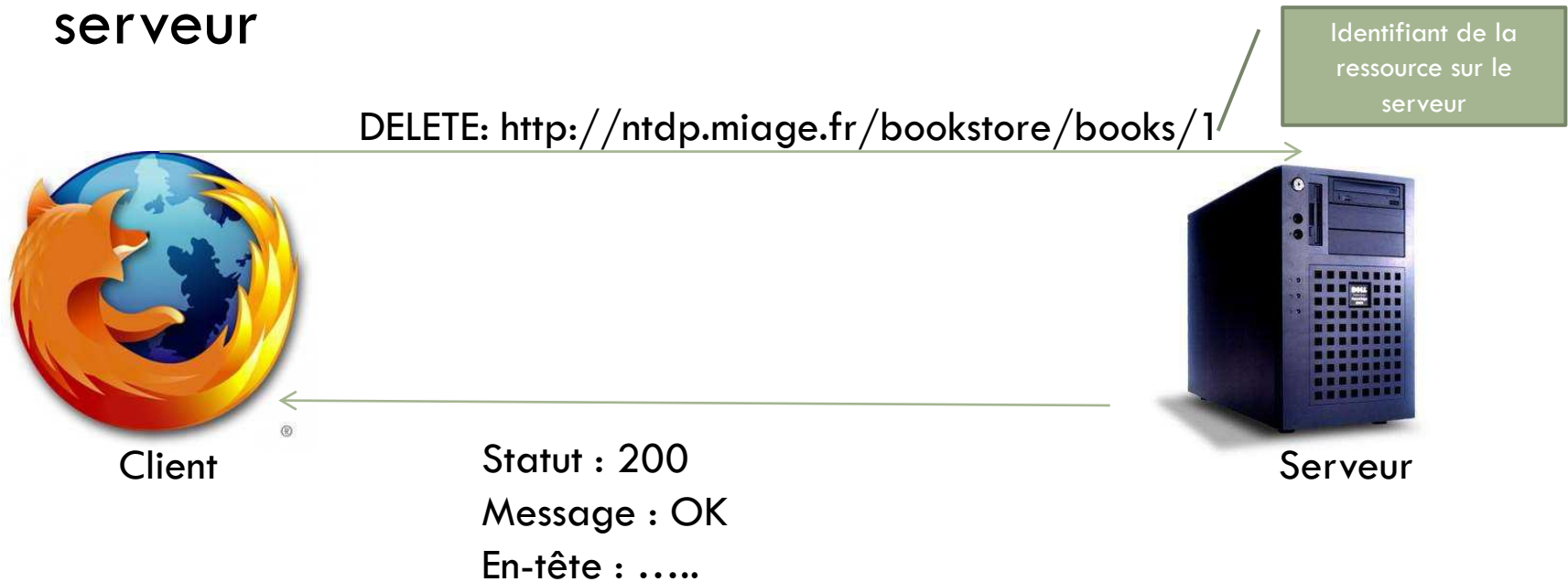
- La méthode POST crée une nouvelle ressource sur le système



# Méthode DELETE

137

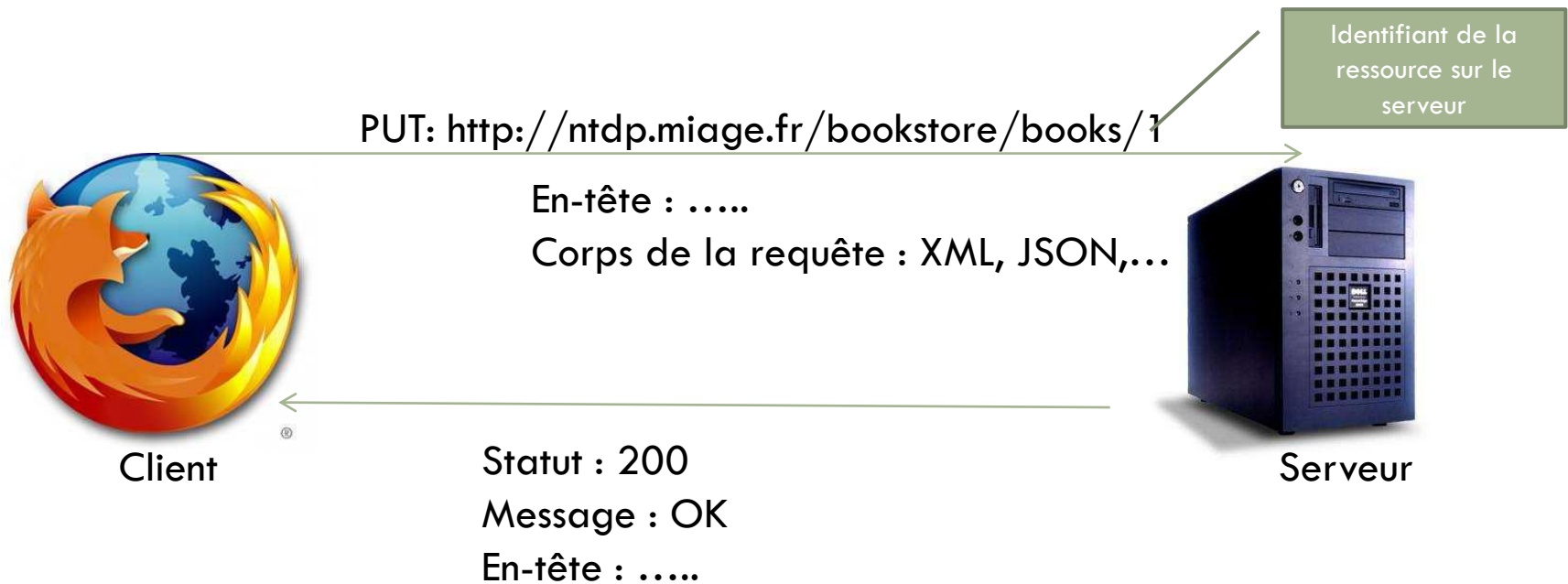
- Supprime la ressource identifiée par l'URI sur le serveur



# Méthode PUT

138

- Mise à jour de la ressource sur le système



# Représentation

139

Une représentation désigne les données échangées entre le client et le serveur pour une ressource:

- HTTP GET → Le serveur envoie au client l'état de la ressource
- PUT, POST → Le client envoie l'état d'une ressource au serveur

Peut être sous différent format :

- JSON
- XML
- XHTML
- CSV
- Text/plain
- .....

# JSON

140

JSON « **J**ava**S**cript **O**bject **N**otation » est un format d'échange de données, facile à lire par un humain et interpréter par une machine.

Basé sur JavaScript, il est complètement indépendant des langages de programmation mais utilise des conventions qui sont communes à tous les langages de programmation (C, C++, Perl, Python, Java, C#, VB, JavaScript,...)

Deux structures :

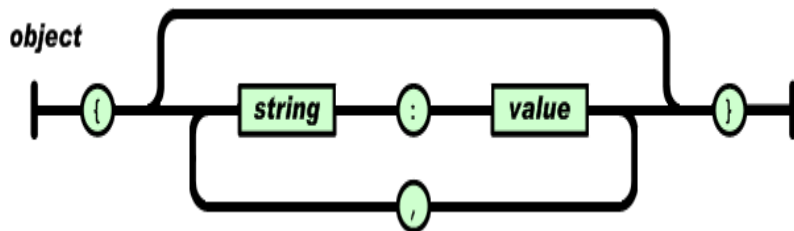
- Une collection de couple : clefs/valeurs → Object
- Une collection ordonnée d'objets → Array

# JSON

141

## Objet

Commence par un « { » et se termine par « } » et composé d'une liste non ordonnée de paire clefs/valeurs. Une clef est suivie de « : » et les paires clef/valeur sont séparés par « , »



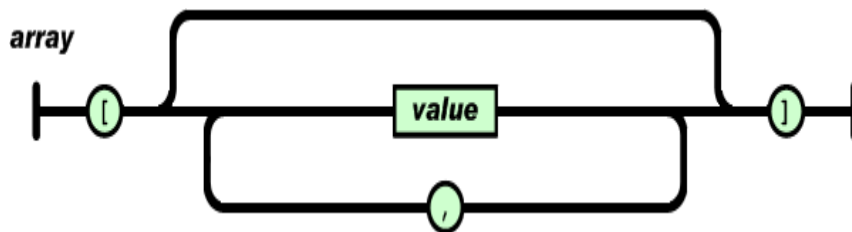
```
{ "id": 51,  
  "nom": "Mathematiques 1", "resume":  
  "Resume of math ", "isbn": "123654",  
  "categorie":  
    {  
      "id": 2, "nom": "Mathematiques",  
      "description": "Description of  
      mathematiques "  
    },  
  "quantite": 42,  
  "photo": ""  
}
```

# JSON

142

## ARRAY

Liste ordonnée d'objets commençant par « [ » et se terminant par « ] », les objets sont séparés l'un de l'autre par « , ».



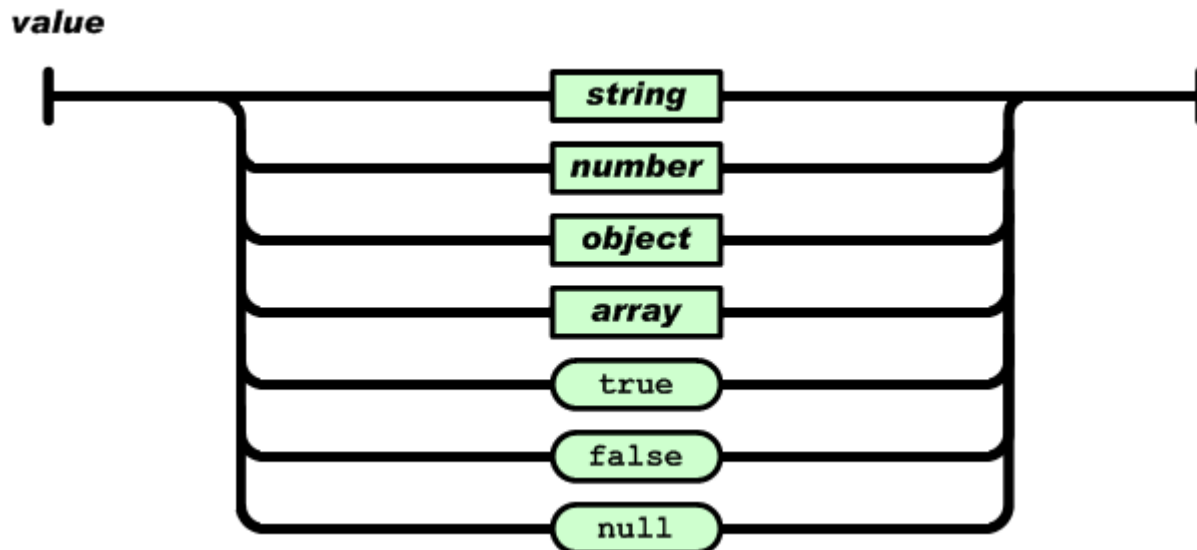
```
[  
  { "id": 51,  
    "nom": "Mathematiques 1",  
    "resume": "Resume of math ",  
    "isbn": "123654",  
    "quantite": 42,  
    "photo": ""  
  },  
  { "id": 102,  
    "nom": "Mathematiques 1",  
    "resume": "Resume of math ",  
    "isbn": "123654444455",  
    "quantite": 42,  
    "photo": ""  
  }  
]
```

# JSON

143

## Value

Un objet peut être soit un string entre « "" » ou un nombre (entier, décimal) ou un boolean (true, false) ou null ou un objet.





# Services Web étendus VS REST

144

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body> <ns2:hello
xmlns:ns2="http://services.bibliotheque.ntdp.miage.unice.fr/">
    <name>Miage NTDP</name>
  </ns2:hello>
</S:Body>
</S:Envelope>
```



Client

SOAP



Serveur



Client

<http://localhost:8080/Bibliotheque/webresources/category/Miage%20NTDP>

REST



Serveur

# Services Web étendus VS REST

145

## SOAP

### → Avantages

- Standardisé
- Interopérabilité
- Sécurité (WS-Security)

### → Inconvénients

- Performances (enveloppe SOAP supplémentaire)
- Complexité, lourdeur
- Cible l'appel de service

# Services Web étendus VS REST

146

## REST

### → Avantages

- Simplicité de mise en œuvre
- Lisibilité par un humain
- Evolutivité
- Repose sur les principes du web
- Représentations multiples (XML, JSON,...)

### → Inconvénients

- Sécurité restreinte par l'emploi des méthodes HTTP
- Cible l'appel de ressources

# WADL

147

- Web Application Definition Language est un langage de description des services REST au format XML. Il est une spécification de W3C initié par SUN ([www.w.org/Submission/wadl](http://www.w.org/Submission/wadl))
- Il décrit les éléments à partir de leur type (Ressources, Verbes, Paramètre, type de requête, Réponse)
- Il fournit les informations descriptives d'un service permettant de construire des applications clientes exploitant les services REST.

148

# Développer des Web Services REST avec JAVA

## JAX-RS

# JAX-RS

149

- ❑ Acronyme de Java API for RestFul Web Services
- ❑ Version courante 2.0 décrite par JSR 339
- ❑ Depuis la version, il fait partie intégrante de la spécification Java EE
- ❑ Décrit la mise en œuvre des services REST web coté client
- ❑ Son architecture se repose sur l'utilisation des classes et des annotations pour développer les services web

# JAX-RS → Implémentation

150

- JAX-RS est une spécification et autour de cette spécification sont développés plusieurs implémentations
- ★ □ JERSEY : implémentation de référence fournie par Oracle ( <http://jersey.java.net> )
- CXF : Fournie par Apache ( <http://cfx.apache.org> )
- RESTEasy : fournie par JBOSS
- RESTLET : L'un des premiers framework implémentant REST pour Java

# Implémentation REST : RESTEasy

151

- Fournie par JBOSS
- Implémentation de l'API JAX-RS
- Se base sur les annotations:
  - ▣ `@Path("/path")`
  - ▣ `@Consumes(MediaType.APPLICATION_XML)` ou `MediaType.APPLICATION_JSON`
  - ▣ `@Produces(MediaType.APPLICATION_XML )` ou `MediaType.APPLICATION_JSON`
  - ▣ `@POST`
  - ▣ `@DELETE`
  - ▣ `@PathParam`



# Application : Gestion des utilisateurs

152

URI	Methode HTTP	Description
/utilisateur/ajout	POST	Ajouter un utilisateur
/utilisateur/{id}/supp	Delete	Supprime l' utilisateur avec l' 'id' dans l'URI
/utilisateur/{id}/get	GET	Retourne l' utilisateur avec l' 'id' dans l'URI
/utilisateur/tous	GET	Retourne tous les utilisateurs

# Application : Gestion des employées

153

- Créer deux classes entité annotés `@XmlRootElement(name="*")` pour permettre à l'API JAXB de convertir les objects en xml et vice versa :
  - `Utilisateur(String firstName, String lastName, int id, String login, String password)`
  - `GenericResponse(boolean status, String message, String errorCode)`
- Créer la classe `Service` qui implémente l'interface `UtilisateurService`

# EmployeeService.java

154

```
package com.iit.glid3.service;

public interface UtilisateurService {
    public Response addUtilisateur(Utilisateur e);
    public Response deleteUtilisateur(int id);
    public Utilisateur getUtilisateur(int id);
    public Utilisateur[] getAllUtilisateurs();
}
```

# Configuration

155

- ❑ Configurer la Servlet RESTEasy dans le descripteur de déploiement comme contrôleur frontal.
- ❑ Configurer le paramètre d'initialisation :
  - `<init-param>`
    - `<param-name>javax.ws.rs.Application</param-name>`
    - `<param-value>`  
`com.iit.glid3.jaxrs.resteasy.app.UtilisateurApplication</param-value>`
  - `</init-param>`

# Classe application

156

```
package com.iit.glid3.app;

public class UtilisateurApplication extends Application {
    private Set<Object> singletons = new HashSet<Object>();
    public UtilisateurApplication() {
        singletons.add(new UtilisateurServiceImpl());
    }

    @Override
    public Set<Object> getSingletons() {
        return singletons;
    }
}
```

# Dépendance coté serveur

157

```
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-jaxrs</artifactId>
  <version>3.0.13.Final</version>
</dependency>
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-jaxb-provider</artifactId>
  <version>3.0.13.Final</version>
</dependency>
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-jackson-provider</artifactId>
  <version>3.6.2.Final</version>
</dependency>
```

# Dépendance coté client

158

```
<dependency>  
  <groupId>org.jboss.resteasy</groupId>  
  <artifactId>resteasy-client</artifactId>  
  <version>3.0.13.Final</version>  
</dependency>
```

159

# Développer des Web Services REST avec JAVA

## JERSEY



# Définition

160

- ❑ Framework implémenté en langage « Java » et fourni par la plateforme JEE
- ❑ Open-source
- ❑ Permettant le développement des WS
- ❑ Respectant l'architecture REST et les spécifications de « JAX-RS »



- Version actuelle 2.3.1 implémentant les spécifications de JAX-RS 2.0
- Intégré dans Glassfish et l'implémentation Java EE (6,7)
- Supportés dans Netbeans

# JAX-RS : Développement

162

- Basé sur POJO (Plain Old Java Object) en utilisant des annotations spécifiques JAX-RS
- Pas de modifications dans les fichiers de configuration
- Le service est déployé dans une application web
- Pas de possibilité de développer le service à partir d'un WADL contrairement à SOAP (application.wadl)
- Approche Bottom/Up
  - Développer et annoter les classes
  - Le WADL est automatiquement généré par l'API

# Annotation JAX-RS

163

La spécification JAX-RS dispose d'un ensemble d'annotation permettant d'exposer une classe java dans un services web :

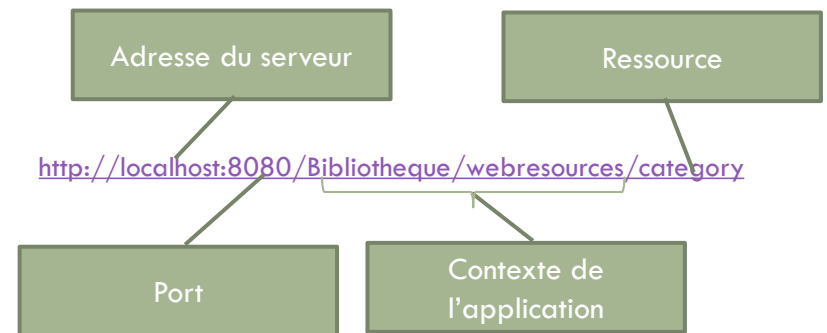
- ❑ `@Path`
- ❑ `@GET`, `@POST`, `@PUT`, `@DELETE`
- ❑ `@Produces`, `@Consumes`
- ❑ `@PathParam`

# JAX-RS : @PATH

164

- ❑ L'annotation permet de rendre une classe accessible par une requête HTTP
- ❑ Elle définit la racine des ressources (Root Racine Ressources)
- ❑ La valeur donnée correspond à l'uri relative de la ressource

```
@Path("category")
public class CategoryFacade {
    .....
}
```



# JAX-RS : @PATH

165

- ❑ L'annotation peut être utilisée pour annoter des méthodes d'une classe
- ❑ L'URI résultante est la concaténation entre la valeur de @pat de la classe et celle de la méthode

```
@Path("category")
public class CategoryFacade {
    @GET
    @Produces({MediaType.APPLICATION_XML,
        MediaType.APPLICATION_JSON})
    @Path("test")
    public String hello()
    {
        return "Hello World!";
    }
    ..
}
```

<http://localhost:8080/Bibliotheque/webresources/category/hello>

# JAX-RS : @PATH

166

- ❑ La valeur définie dans l'annotation `@Path` n'est forcément une constante, elle peut être variable.
- ❑ Possibilité de définir des expressions plus complexes, appelées Template Parameters
- ❑ Les contenus complexes sont délimités par « `{}` »
- ❑ Possibilité de mixer dans la valeur `@Path` des expressions

```
@GET
@Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
@Produces ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
@Path("/{nom}")
public String hello (@PathParam("nom") String nom){
    return "Hello " + nom;
}
```

<http://localhost:8080/Bibliotheque/webresources/category/hello/Miage>

# @GET, @POST, @PUT, @DELETE

167

- Permettent de mapper une méthode à un type de requête HTTP
- Ne sont utilisables que sur des méthodes
- Le nom de la méthode n'a pas d'importance, JAX détermine la méthode à exécuter en fonction de la requête

<http://localhost:8080/Bibliotheque/webresources/category/test>

```
@Path("category")
public class CategoryFacade {
    @GET
    @Produces({MediaType.APPLICATION_XML,
        MediaType.APPLICATION_JSON})
    @Path("test")
    public String hello()
    {
        return "Hello World!";
    }
    ..
}
```

```
@GET
@Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
@Produces ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
@Path("{nom}")
public String hello (@PathParam("nom") String nom){
    return "Hello " + nom;
}
```

<http://localhost:8080/Bibliotheque/webresources/category/Miage>



# @GET, @POST, @PUT, @DELETE

168

- Les opérations CRUD sur les ressources sont réalisées au travers des méthodes de la requête HTTP



GET, POST  
PUT, DELETE



/books

GET : Liste des livres

POST : Créer un nouveau livre

/books/{id}

GET : Livre identifié par l'id

PUT: Mis à jour du livre identifié par id

DELETE : Supprimer le livre identifié par id

# Implémentation du service

169

## □ Dépendance maven :

### □ <dependency>

- <groupId>org.glassfish.jersey.core</groupId>
- <artifactId>jersey-server</artifactId>
- <version>2.3.1</version>

### □ </dependency>

### □ <dependency>

- <groupId>org.glassfish.jersey.containers</groupId>
- <artifactId>jersey-container-servlet-core</artifactId>
- <version>2.3.1</version>

### □ </dependency>

### □ <dependency>

- <groupId>javax.ws.rs</groupId>
- <artifactId>javax.ws.rs-api</artifactId>
- <version>2.0</version>

### □ </dependency>

### □ <dependency>

- <groupId>com.fasterxml.jackson.jaxrs</groupId>
- <artifactId>jackson-jaxrs-json-provider</artifactId>
- <version>2.4.1</version>

### □ </dependency>

# Implémentation du service

170

## □ Dépendance maven : JDK>9

- `<dependency>`
- `<groupId>javax.xml.bind</groupId>`
- `<artifactId>jaxb-api</artifactId>`
- `<version>2.2.11</version>`
- `</dependency>`
- `<dependency>`
- `<groupId>com.sun.xml.bind</groupId>`
- `<artifactId>jaxb-core</artifactId>`
- `<version>2.2.11</version>`
- `</dependency>`
- `<dependency>`
- `<groupId>javax.activation</groupId>`
- `<artifactId>activation</artifactId>`
- `<version>1.1.1</version>`
- `</dependency>`

# Implémentation du service

171

## □ Modification du Web.xml

```
<servlet>
  <servlet-name>com.iit.glid3.app.UtilisateurApplication</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>com.iit.glid3.service</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>com.iit.glid3.app.UtilisateurApplication</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```

# Implémentation du service

172

```
@Path("/hello")
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_JSON)
public interface UtilisateurService {
    String bienvenu();
    Utilisateur getDummyEmployee(int id);
    Utilisateur getDummyEmployeeXML(int id);
}

public class UtilisateurServiceImpl implements UtilisateurService {
    @Override
    @GET
    @Path("/")
    @Produces(MediaType.TEXT_PLAIN)
    public String bienvenu() {
        return "Bienvu au service de gestion des Utilisateurs";
    }
    @Override
    @GET
    @Path("/{id}/getDummy")
    public Utilisateur getDummyEmployee(@PathParam("id") int id) {
        Utilisateur e = new Utilisateur();
        e.setSalary(8976.55);
        e.setName("Dummy");
        e.setId(id);
        return e;
    }
}
```

# Client du Service

173

```
public class App {  
    public static void main(String[] args) {  
  
        Client client = ClientBuilder.newClient();  
        WebTarget delireRestTarget = client.target("http://localhost:9999/Jersey-  
WS/hello/50/getDummy");  
        Invocation.Builder invocationBuilder =  
        delireRestTarget.request(MediaType.APPLICATION_JSON);  
        Response response = invocationBuilder.get();  
        System.out.println(response.getStatus());  
        System.out.println(response.readEntity(String.class));  
        System.out.println();  
    }  
}
```