Web Services

Mineure SOA

Cécile Hardebolle
cecile.hardebolle@supelec.fr

Programme

8 nov.	Introduction. Enjeux, rôle de l'architecte SI *Partie n°1 du cas d'étude*	
15 nov.	Architecture et cartographie	D1.13E
22 nov.	Modèle SOA	
29 nov.	Modélisation de processus *Partie n°2 du cas d'étude*	D1.13E
6 déc.	Web Services *Partie n°3 du cas d'étude*	D1.13E
13 déc.	Cloud *Partie n°4 du cas d'étude*	
20 déc.	Exécution de processus	D1.13E
10 jan.	Compléments et ouverture. Conclusion *Partie n°5 du cas d'étude*	

Deux intervenants :

- Olivier Besnard (Solucom)
- Cécile Hardebolle (Supélec)

27 jan. — Examen : présentation de vos travaux sur **l'étude de cas « Chaus'Star »**

Au programme ce matin…

- Les grands principes des Web Services
- Les technologies sur lesquelles reposent les Web Services
- Un peu de méthodologie
- De la pratique !

 à retenir

 avancé

 démo/exercice

Plan

① Qu'est-ce qu'un Web Service ?

② Les Web Services WS-*

③ Les Web Services RESTful

④ Synthèse et conclusion

Qu'est-ce qu'un Web Service ?

1. Définition et principe
2. Rappels sur le protocole HTTP
3. Rappels sur le langage XML
4. Implémentation des Web Services

Web Service = Service + Web ?

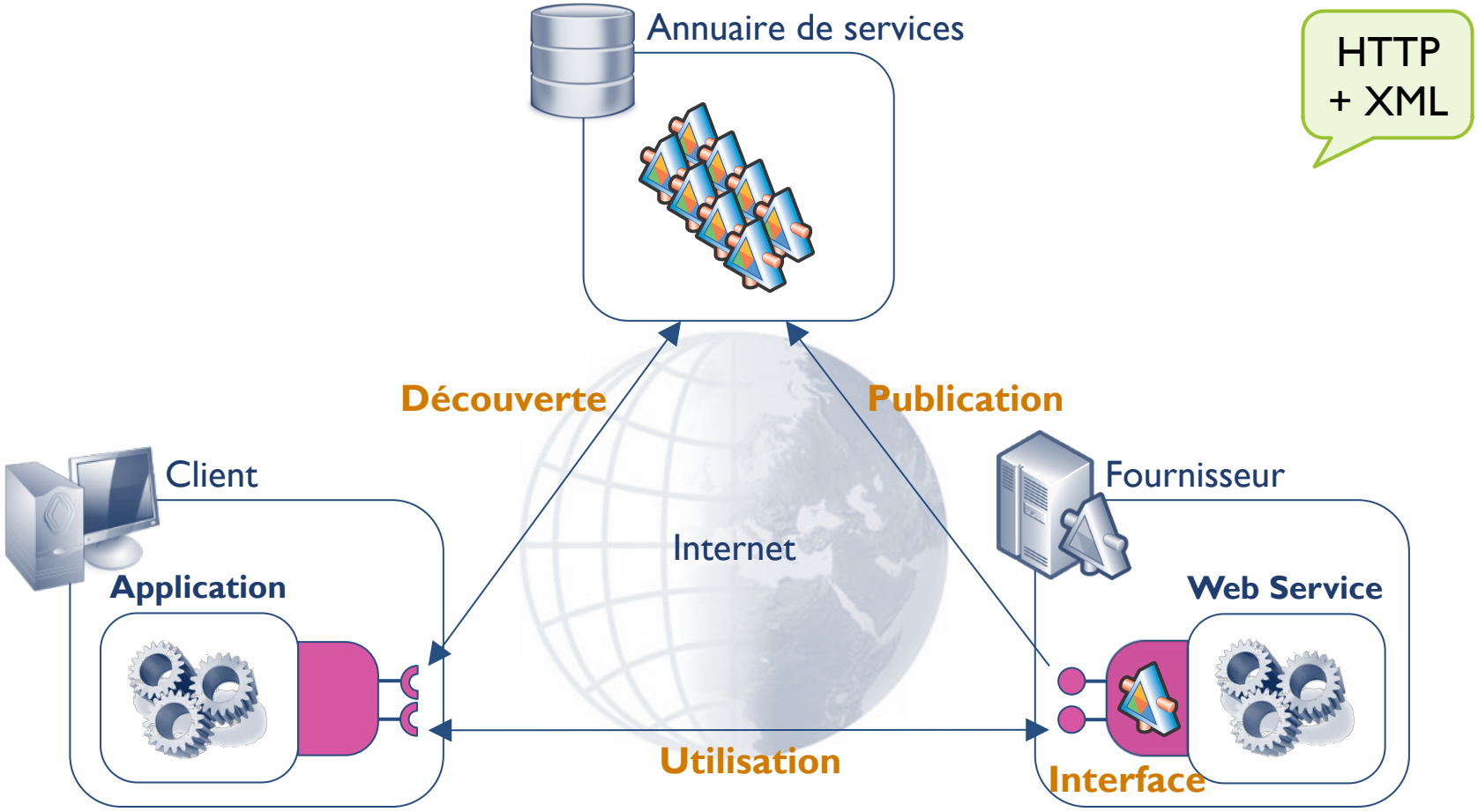
- Service = fonctionnalité mise à disposition et exécutée par un fournisseur lorsqu'elle est invoquée par un consommateur

  réutilisable + *composable* + *indépendant* + *granularité variable*

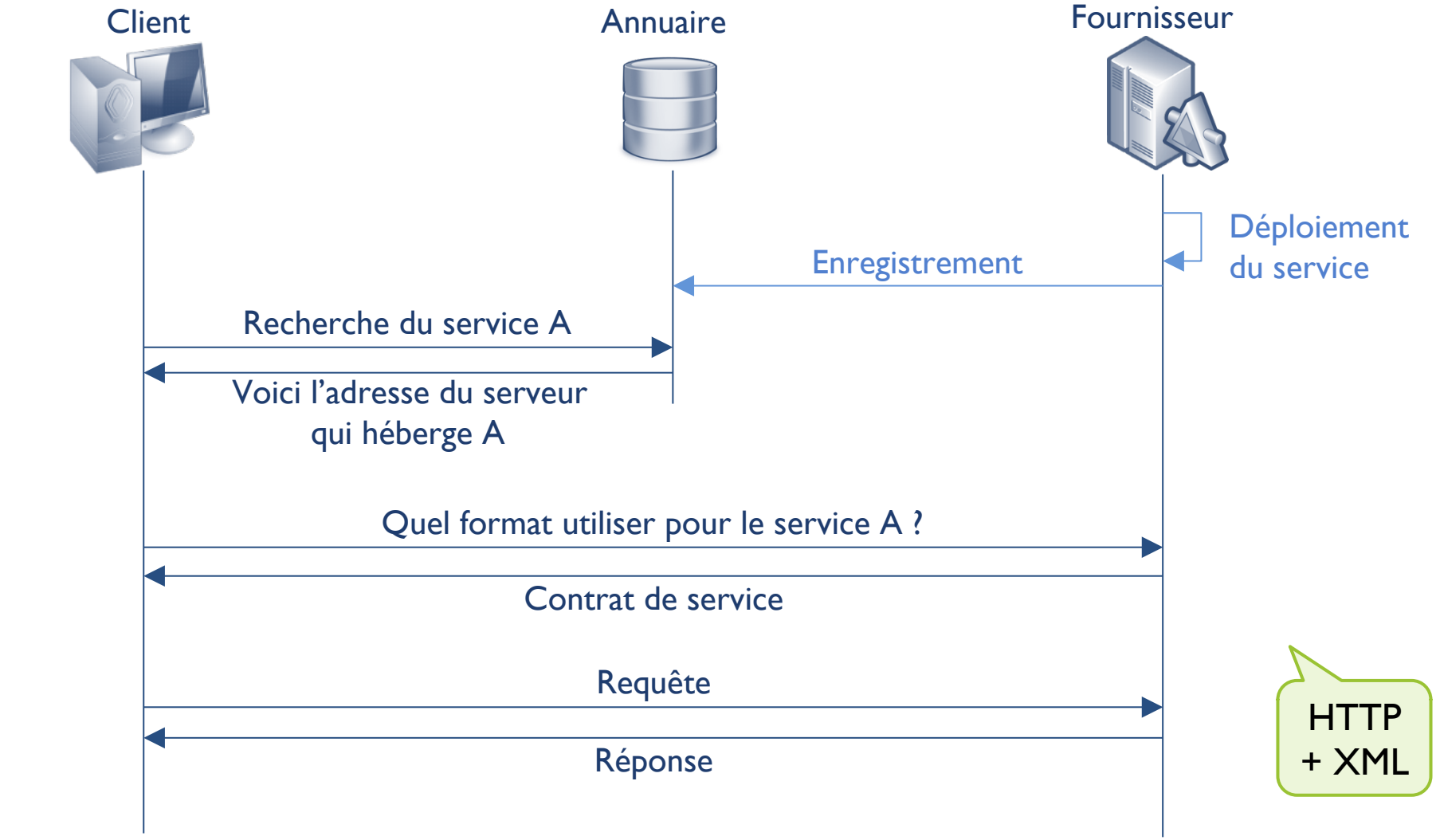
  - Interface :
    - Définit l'usage du service (syntaxe, sémantique, qualité) ☛ **contrat**
    - Masque l'implémentation du service pour un couplage consommateur/fournisseur faible
  - Format pivot : langage commun pour décrire et échanger les données

- Web Service = service mis à disposition sur Internet
  - Associé à une URL sur le web
  - Accessible via des protocoles internet standard — HTTP
  - Accessible indépendamment des technologies d'implémentation
  - Auto-descriptif — XML

Principe des Web Services



Utilisation d'un Web Service



Un exemple ?

Qu'est-ce qu'un Web Service ?

1. Définition et principe
2. Rappels sur le protocole HTTP
3. Rappels sur le langage XML
4. Implémentation des Web Services

HTTP (HyperText Transfer Protocol)

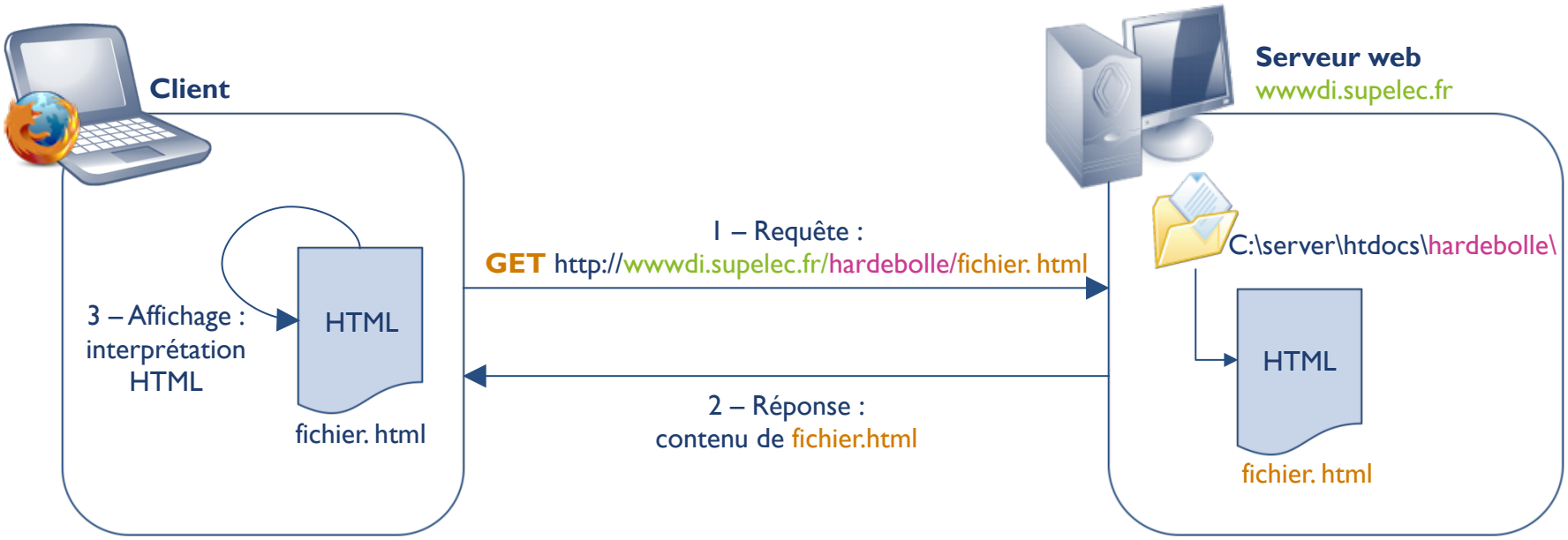
- Protocole de communication dédié au web
  - Chaque ressource du web est identifiée par une URL
- Mode de communication = requête / réponse
  - Requête
    - Méthode de requête + nom ressource
      - Lecture : GET, HEAD…
      - Modification : POST, PUT, DELETE…
    - En-tête : nom du serveur, …
  - Réponse
    - En-tête : code de statut, type de serveur, type de contenu…
    - Contenu de la ressource demandée
- Non conservation de l'état entre deux couples requête/réponse

```
GET /index.html HTTP/1.1
Host: www.example.com
```

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

HTTP
Exemple

- Accès en lecture à une page web pour affichage dans un navigateur



MIME (Multipurpose Internet Mail Extension)

- Standard définissant le type et le format de contenus échangés sur internet
  - Contenu textuel : langue, codage des caractères…
  - Contenu multimédia (images, sons, films…) : type de média…
    - Transfert sous forme binaire
  - Contenus multiples (pièces jointes…)

- Utilisé pour les emails avec SMTP
- Utilisé pour le web avec HTTP
  - En-tête :
    « Content-Type: type/sous-type »
  - Exemples :
    - text/xml
    - audio/mpeg
    - image/jpeg
    - application/pdf

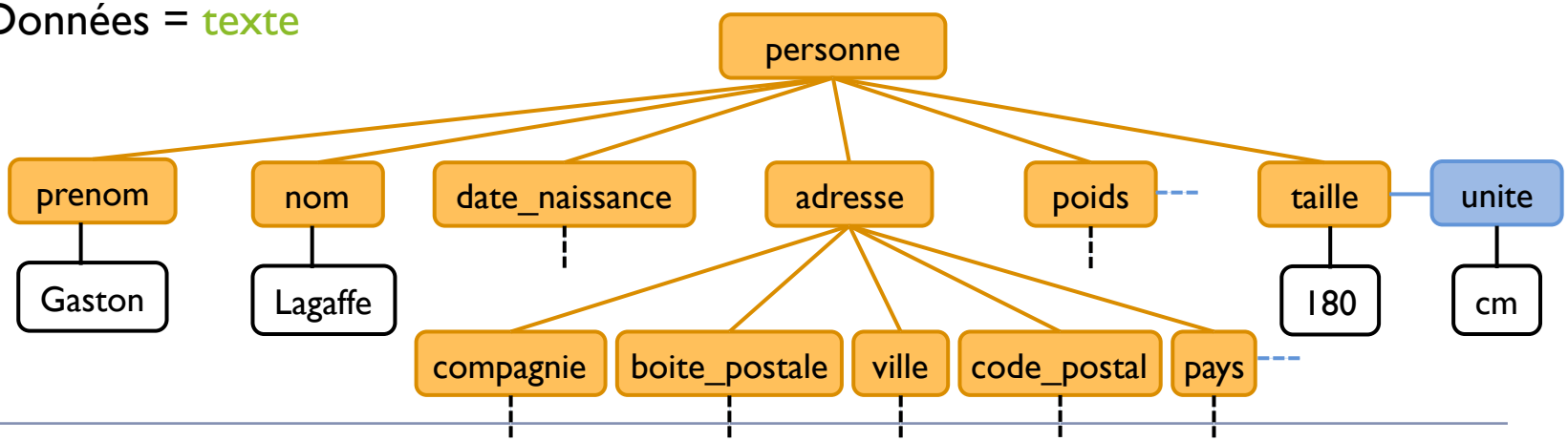
```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

Qu'est-ce qu'un Web Service ?

1. Définition et principe
2. Rappels sur le protocole HTTP
3. Rappels sur le langage XML
4. Implémentation des Web Services

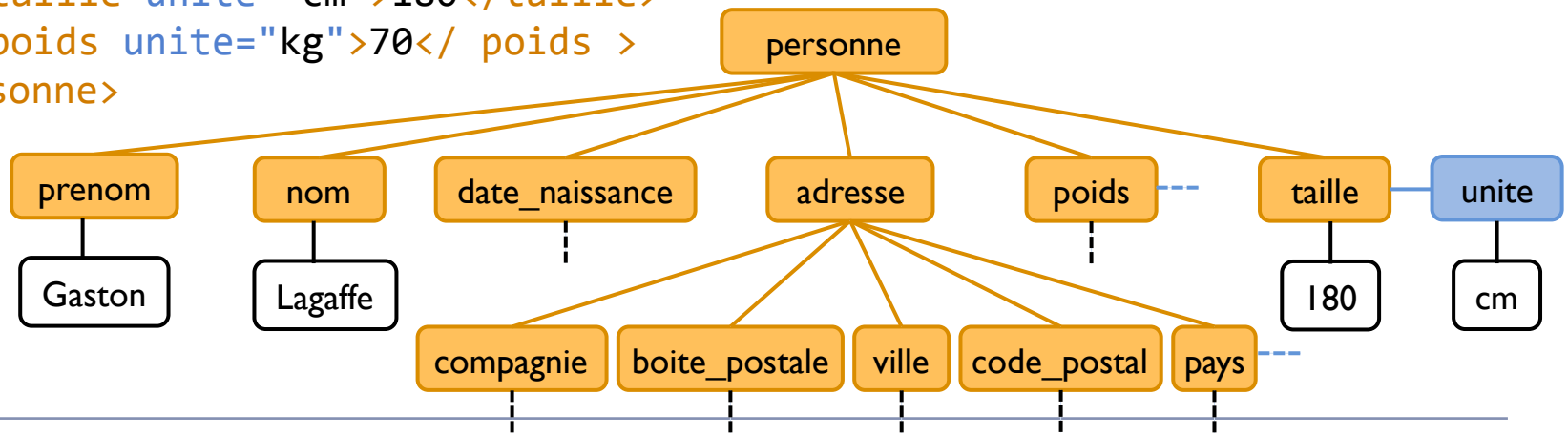
XML (eXtensible Markup Language)

- Standard du W3C depuis 1998
- XML = langage permettant de structurer des données de manière logique
  - Extensible
  - Indépendant des plates-formes et des systèmes d'exploitation
  - Concernant uniquement le contenu, pas la forme (apparence)
- Document XML = structure arborescente auto-descriptive
  - Structure des données = balises personnalisées (« tags »)
  - Données = texte



Exemple de document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<personne>
    <prenom>Gaston</prenom>
    <nom>Lagaffe</nom>
    <date_naissance>30/03/1976</date_naissance>
    <adresse>
            <compagnie>Journal Spirou</compagnie>
            <boite_postale>355</ boite_postale >
            <ville>Paris Cedex</ville>
            <code_postal>75116</ code_postal >
            <pays code="ISO-3166">FR</pays>
    </adresse>
    <taille unite="cm">180</taille>
    <poids unite="kg">70</ poids >
</personne>
```



Validité d'un document XML

- Grammaire = définition d'un vocabulaire valide et de règles de structure
- Pour XML, grammaire = schéma
  - Définit les balises et leurs attributs
  - Définit les contraintes de structure des documents
- XML Schema (XSD) = un des langages de description de schémas

```
<xsd:element name="personne">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="prenom" type="xsd:string"/>
         <xsd:element name="nom" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

```
<personne>
      <prenom>Gaston</prenom>
      <nom>Lagaffe</nom>
</personne>
```

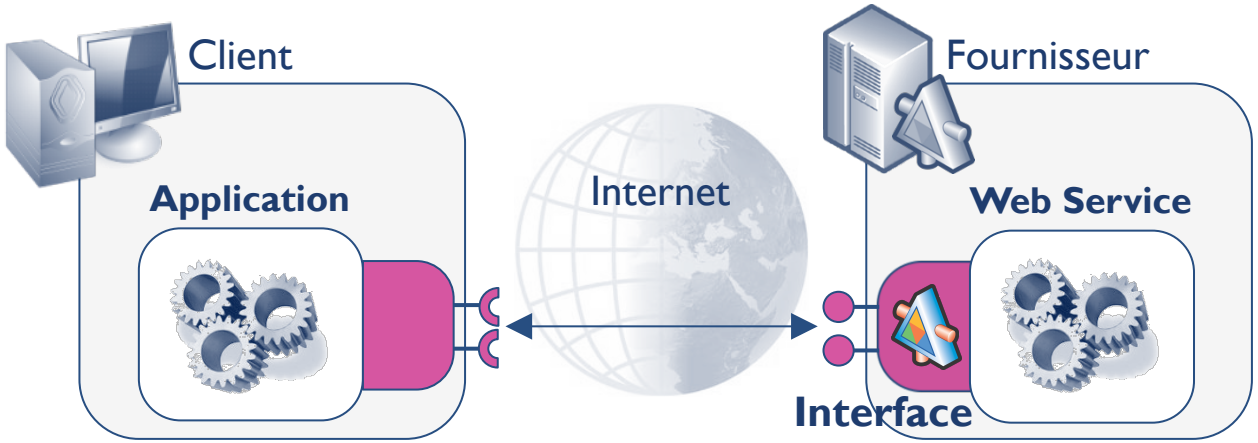
- Espace de noms = préfixe permettant d'éliminer les conflits lorsque plusieurs balises ont des noms identiques, *URL (fictive) utilisée comme identifiant*

```
<liv:auteur xmlns:liv="http://livres">…</liv:auteur>
```

Qu'est-ce qu'un Web Service ?

1. Définition et principe
2. Rappels sur le protocole HTTP
3. Rappels sur le langage XML
4. Implémentation des Web Services

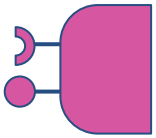
Implémentation (*hors annuaire*)



- Implémentation côté client et côté fournisseur :

Application « métier »

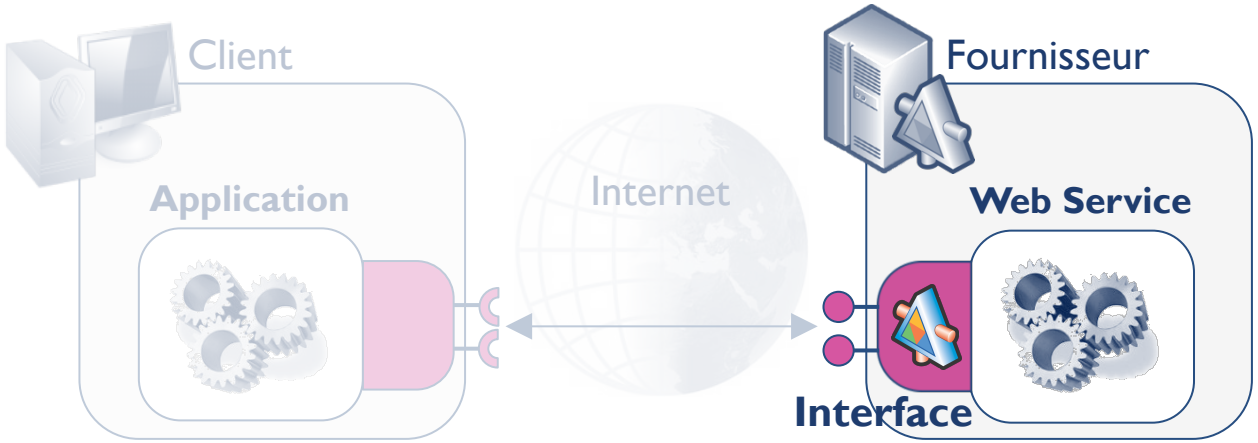
☛ toutes technologies possibles (Java, .NET, PHP…)



Traitements liés au protocole, basé sur HTTP/XML

☛ deux grandes familles : famille WS-* et famille RESTful

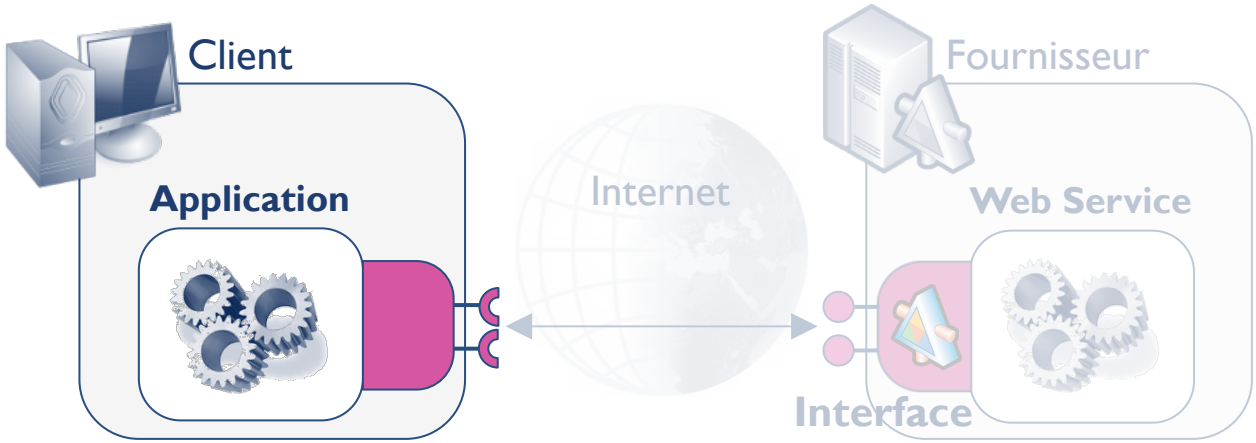
Côté fournisseur



- Pour créer un Web Service :
  1. Définir le contrat du service
  2. Développer le service
  3. Développer la couche de traitement XML
  4. Déployer sur le serveur
  5. Publier dans l'annuaire

Suivant les technologies, certaines tâches sont automatisées…

Côté client



- Pour créer une application cliente :
  1. Rechercher le service dans l'annuaire
  2. Récupérer le contrat du service
  3. Créer un stub/proxy
  4. Développer la couche de traitement XML
  5. Utiliser le service et présenter les résultats (rendu)

Suivant les technologies, certaines tâches sont automatisées…

Plan

① Qu'est-ce qu'un Web Service ?

② Les Web Services WS-*

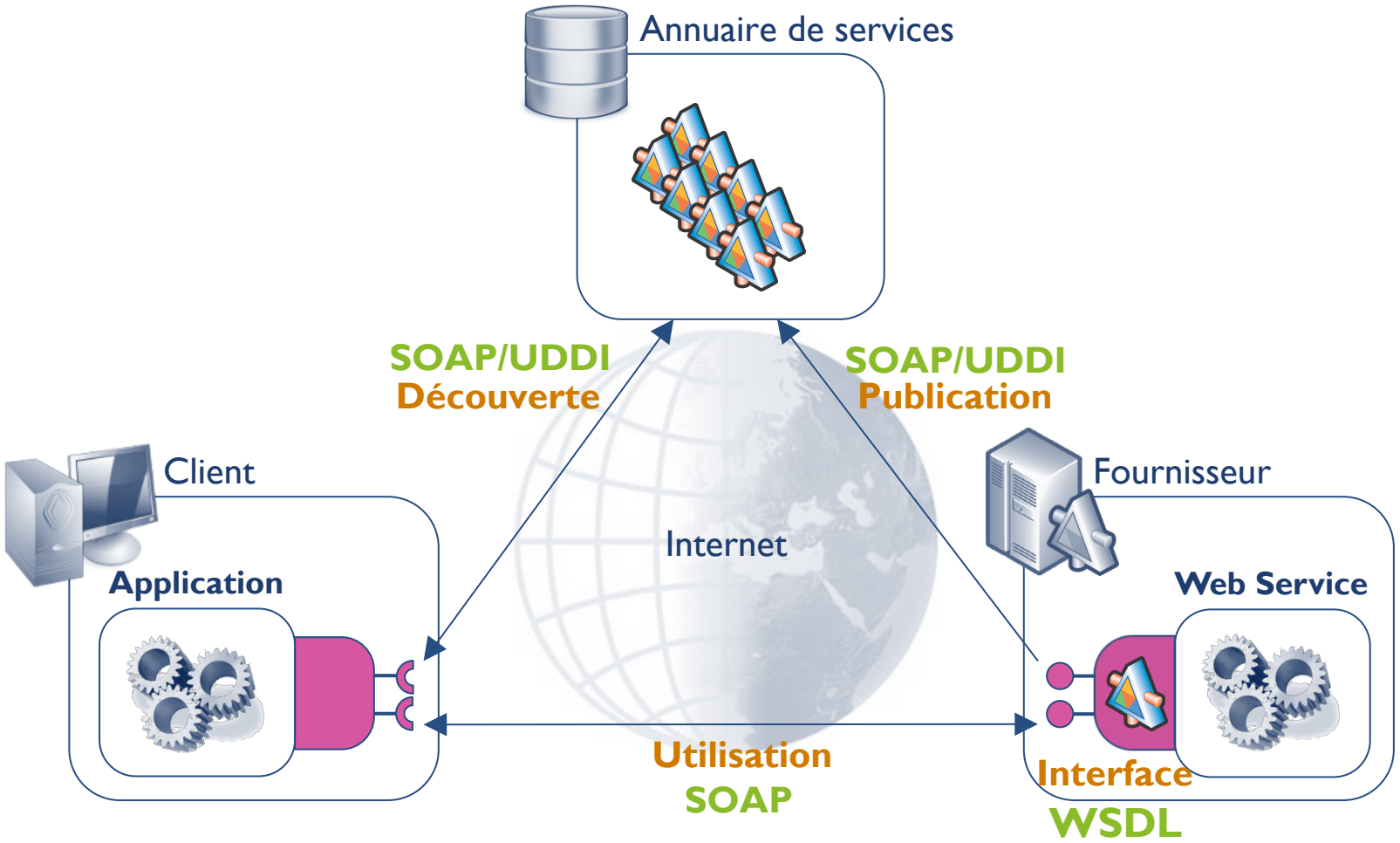
③ Les Web Services RESTful

④ Synthèse et conclusion

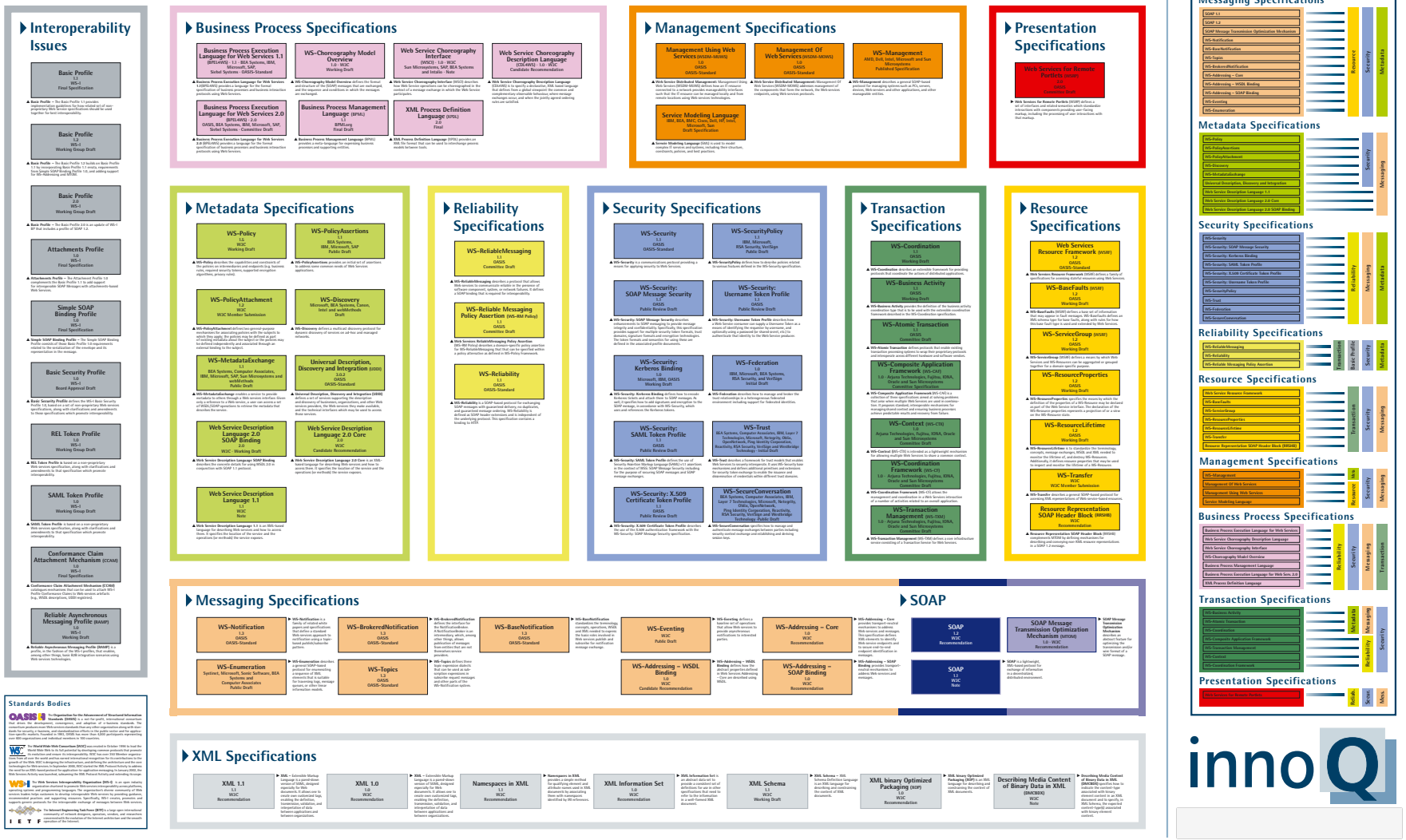
Les Web Services WS-*

1. Standards et acteurs
2. Principales technologies : WSDL, SOAP, UDDI
3. Exposer une application Java sous la forme d'un Web Service WS-*
4. Appeler un Web Service WS-* en Java
5. Et ça marche ?

Principales technologies



« Galaxie » des standards WS-*



Acteurs majeurs

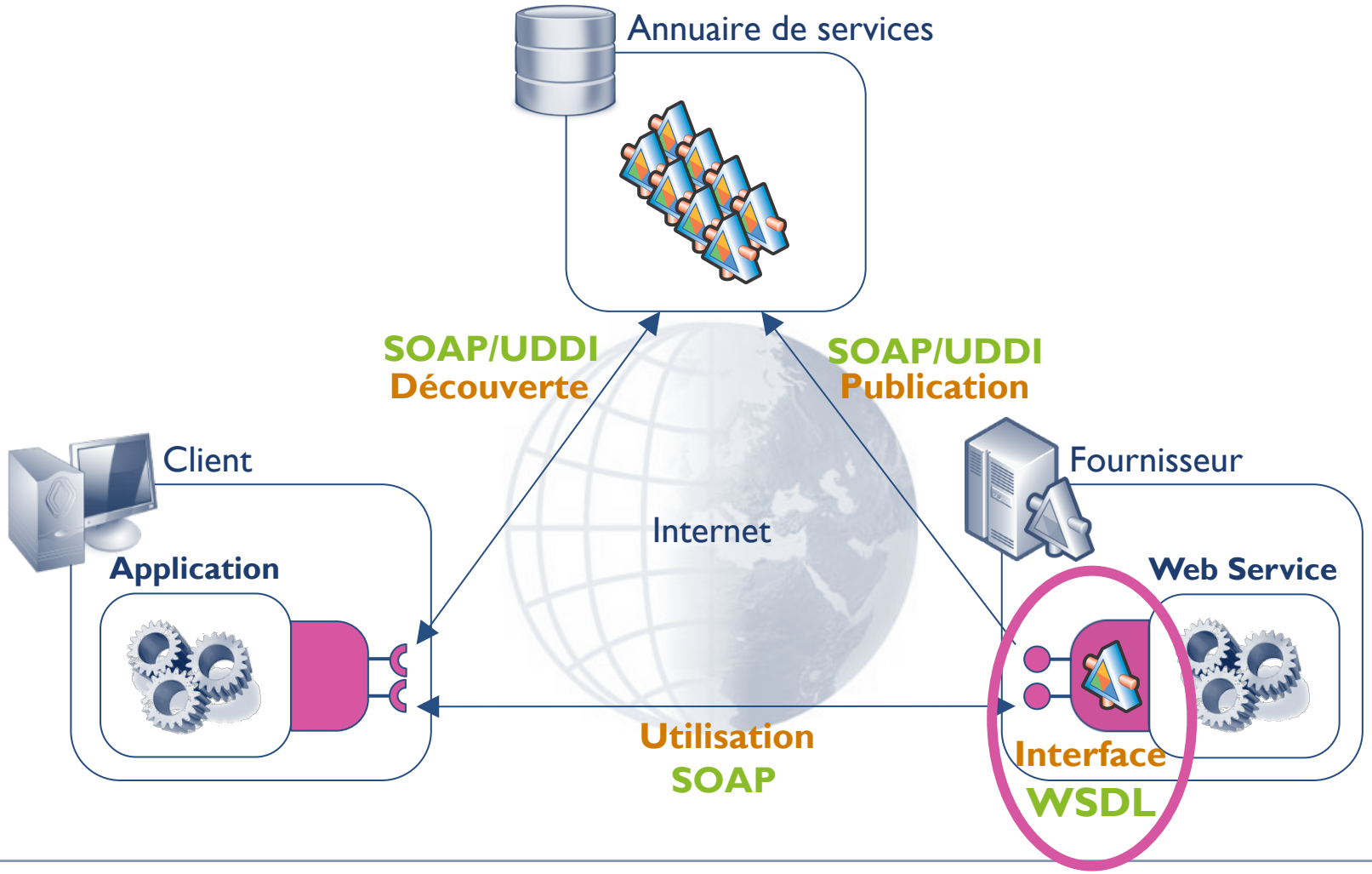
- W3C (World Wide Web Consortium)
  - Consortium académique international fondé en 1994
  - Principal organisme de standardisation concernant le web
    - HTTP, URI, HTML, XML…
  - A l'origine des technologies qui forment la base des Web Services
    - SOAP, WSDL
  - Mécanisme de recommandations

- OASIS
  (Organization for the Advancement of Structured Information Standards)
  - Consortium international d'éditeurs de logiciel
  - Objectif = développement, convergence et adoption de standards e-business
  - Organisme le plus productif dans le domaine des Web Services :
    - UDDI, BPEL, WSRP, WS-Security, SAML, WS-Transactions…

Les Web Services WS-*

1. Standards et acteurs
2. Principales technologies : WSDL, SOAP, UDDI
3. Exposer une application Java sous la forme d'un Web Service WS-*
4. Appeler un Web Service WS-* en Java
5. Et ça marche ?

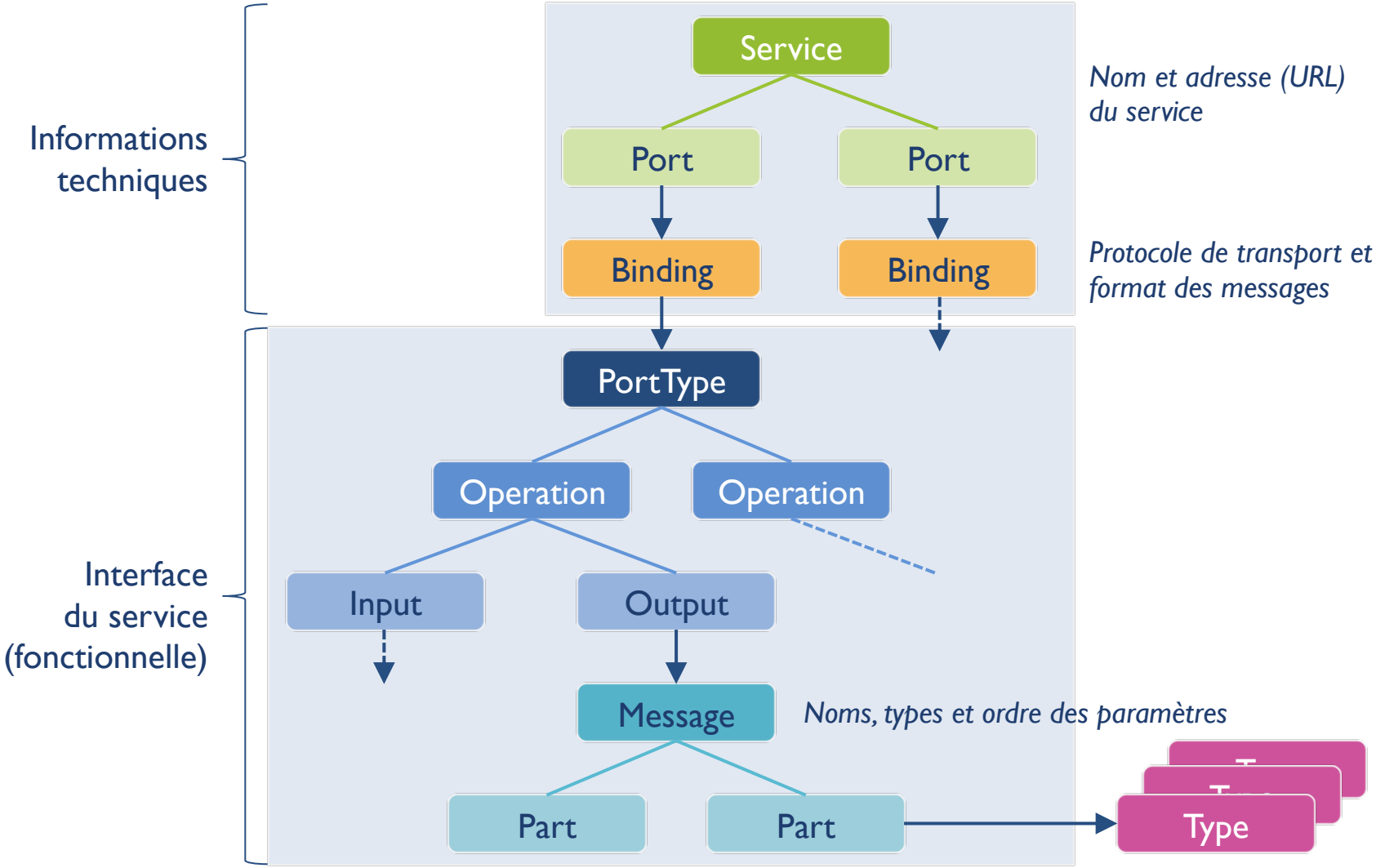
Principales technologies



WSDL
(Web Service Description Language)

- Standard du W3C
  - Version 1.1 en 2001
  - Version 2.0 en 2007, encore peu supporté par les outils
- Objectif = décrire l'interface publique d'un Web Service (contrat de service)
- Grammaire dérivée d'XML
- Interface d'un Web Service avec WSDL
  - Web Service = ensemble de ports de connexions mettant à disposition des opérations qui reçoivent et envoient des messages
  - Deux types d'informations
    - Fonctionnelles : interface du service (signature des méthodes…)
    - Techniques : URL, protocole…
- Fichier WSDL utilisable par des outils de génération de code

Structure d'un fichier WSDL 1.1



Structure d'un fichier WSDL 1.1

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" …>

    <types> […] </types>

    <message […]><part […]/></message>

    <portType […]>
        <operation […]>
            <input […] />
            <output […] />
        </operation>
    </portType>

    <binding […]>[…]</binding>

    <service […]>
        <port […]>[…]</port>
    </service>

</definitions>
```

Interface du service (fonctionnelle)

Informations techniques

WSDL
Exemple d'interface de service

- Avec des types simples

```
 <portType name="Hello">
    <operation name="sayHello">
        <input message="tns:sayHello" />
        <output message="tns:sayHelloResponse" />
    </operation>
</portType>

<message name="sayHello">
    <part name="n" type="xsd:string" />
</message>
<message name="sayHelloResponse">
    <part name="return" type="xsd:string" />
</message>
```

WSDL
Exemple d'interface de service

- Avec des types complexes
  - Déclarés dans un fichier XSD (XMLSchema) séparé
  - Ou déclarés dans le fichier WSDL

```
<part name="parameters"  element="sayHello" />
```

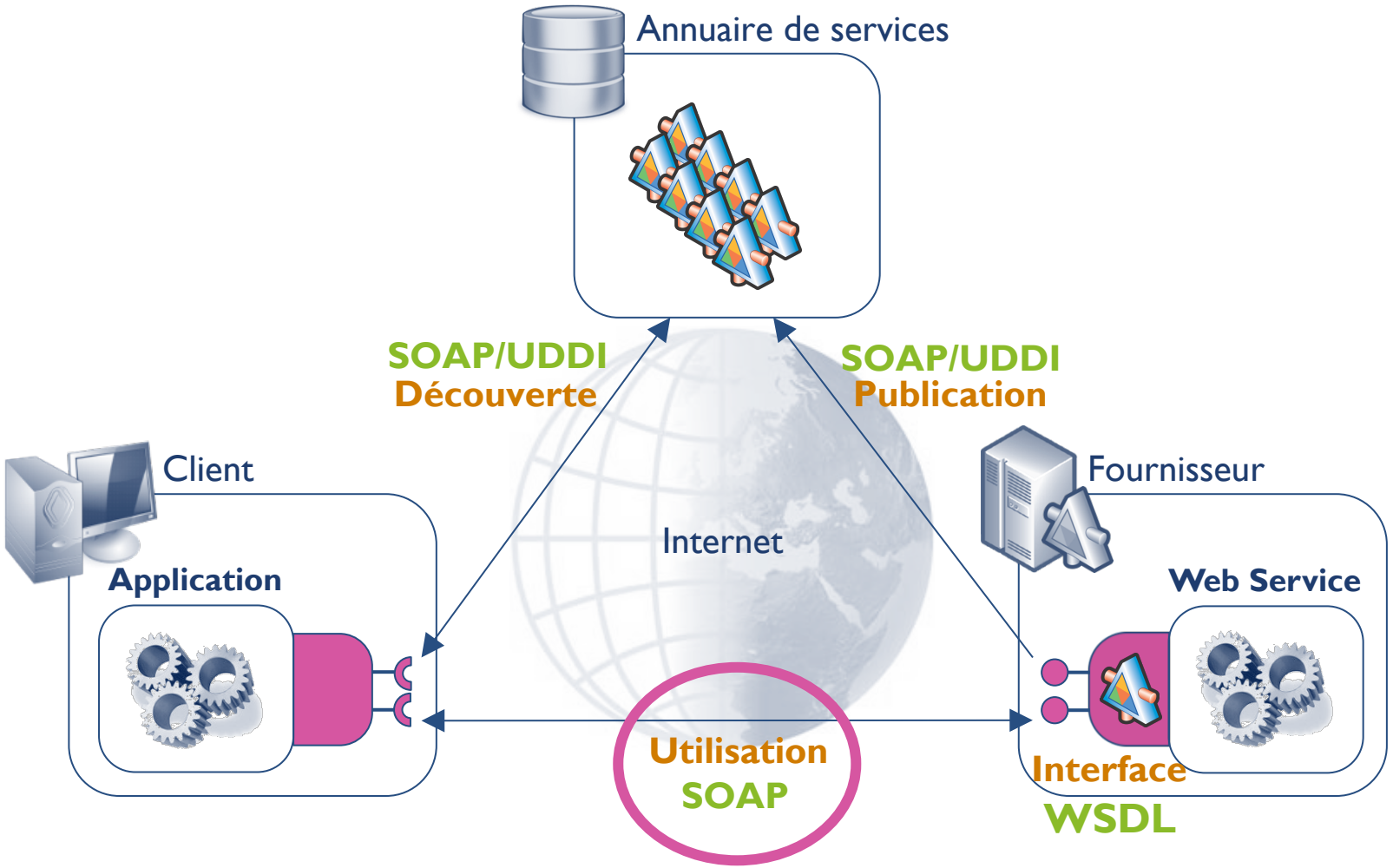
```
<xsd:element name="sayHello" >
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="n" type="xs:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Attention aux espaces de noms !

WSDL
Exemple d'informations techniques

```
<binding name="HelloPortBinding" type="tns:Hello">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                                                    style="document"/>
    <operation name="sayHello">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>

<service name="HelloService">
    <port name="HelloPort" binding="tns:HelloPortBinding">
        <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
</service>
```

Principales technologies



SOAP (Simple Object Access Protocol)

- Standard du W3C
  - Version1.2 en 2003
- Objectif = formater les requêtes et les réponses échangées entre client et Web Service pour le transport (notamment sur HTTP)
- Grammaire dérivée d'XML

- Définit principalement
  - Un modèle de structure pour les requêtes et les réponses (messages)
    - Envelope : obligatoire, définit un message SOAP
    - Header : optionnel, informations non applicatives (sécurité…) ou destinées aux intermédiaires
    - Body : décrit la requête ou la réponse
  - Un modèle de traitement des messages

```
<Envelope>
  <Header>
    <transId>1234</transId>
  </Header>
  <Body>
    <add>
      <varx>3</varx>
      <vary>4</varY>
    </add>
  </Body>
</Envelope>
```

SOAP
Exemple

- Requête : sayHello("Robert")

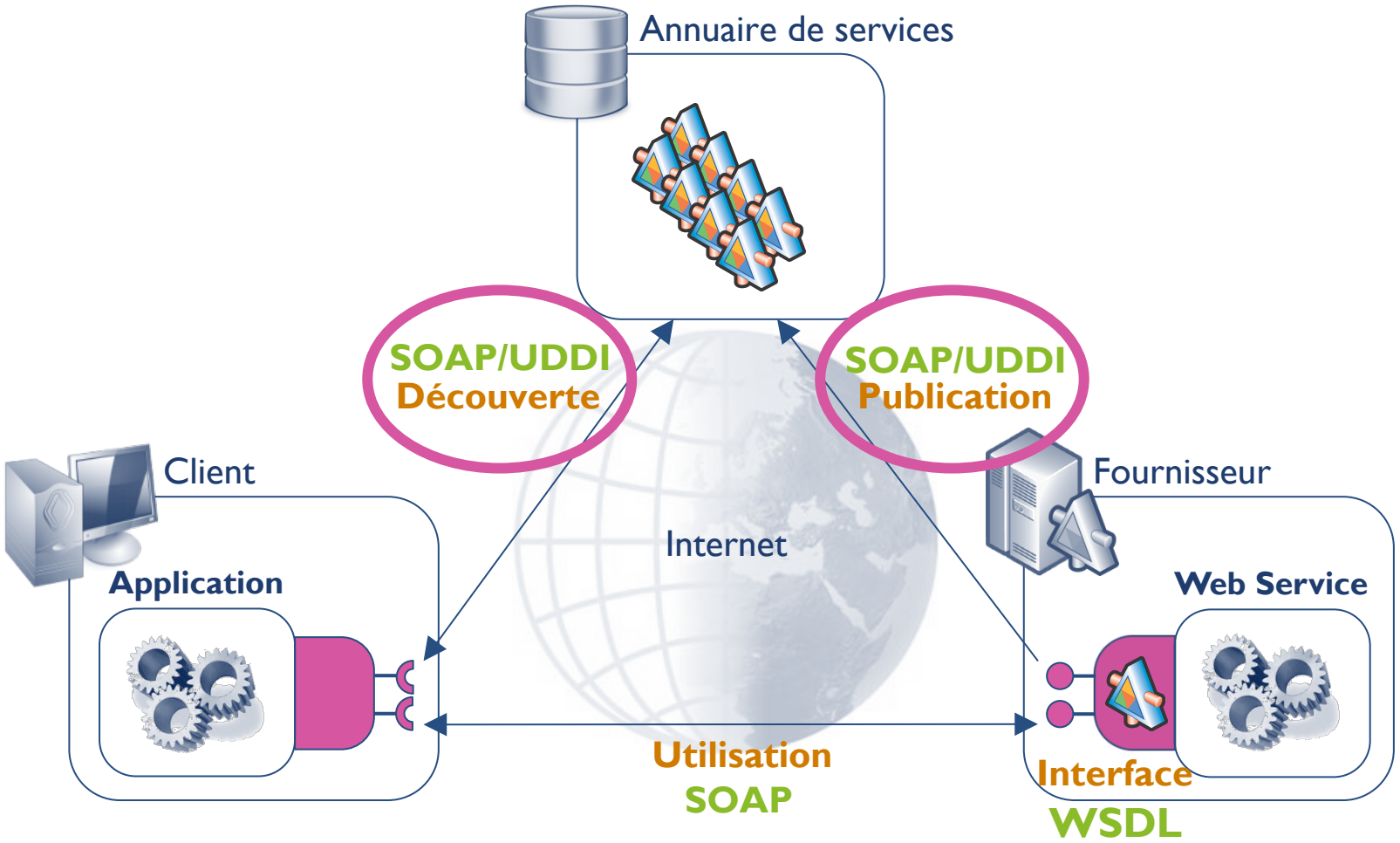
```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns:sayHello xmlns:ns="http://hello/">
            <n>Robert</n>
        </ns:sayHello>
    </soap:Body>
</soap:Envelope>
```

Précédés d'un message HTTP !

- Réponse : "Hello dear Robert !"

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns:sayHelloResponse xmlns:ns="http://hello/">
            <return>Hello dear Robert !</return>
        </ns:sayHelloResponse>
    </soap:Body>
</soap:Envelope>
```

Principales technologies



UDDI (Universal Discovery Description and Integration)

- Standard porté par un consortium d'industriels
  - Version 3 en 2005
- Objectif = publication et découverte de Web Services sur un réseau
- Définit :
  - UDDI Business Registry (UBR) = annuaire pour permettre d'automatiser les communications entre prestataires, clients, etc. (orienté « business »)
  - Méthodes de publications (basées sur SOAP)
  - Méthodes de consultation (basées sur SOAP)

Pages blanches	Pages jaunes	Pages vertes
▪ Nom de la société ▪ Information sur les contacts ▪ Description texte ▪ Identifications (DUNS, SIRET, etc.)	▪ Index services et produits ▪ Code d'industrie (APE, etc.) ▪ Index géographique ▪ Taxonomie	▪ Procédures e-business ▪ Descriptions technique des services ▪ Paramètres des services

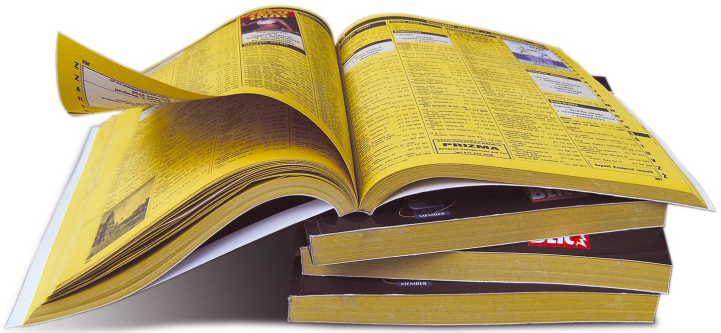
Publication d'un service

- Le référencement du service est important !!!

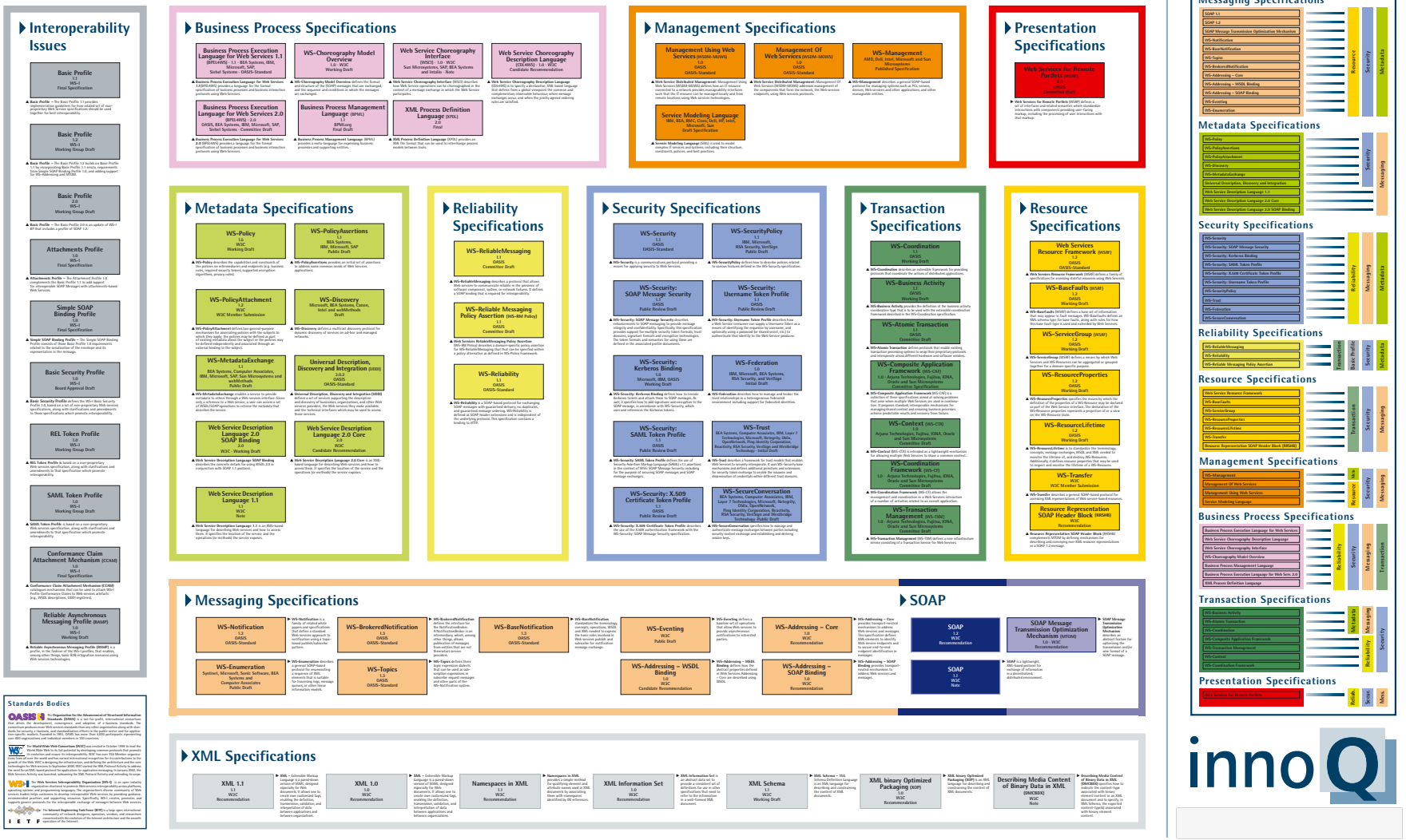
- Différents types de registres
  - Registre public (seekda.com, xmethods.net…)
  - Registre de branche
  - Registre privé

  - *A l'heure actuelle les registres sont majoritairement privés (internes aux entreprises)*

- Indexation par une ou plusieurs catégories dans la taxonomie du registre



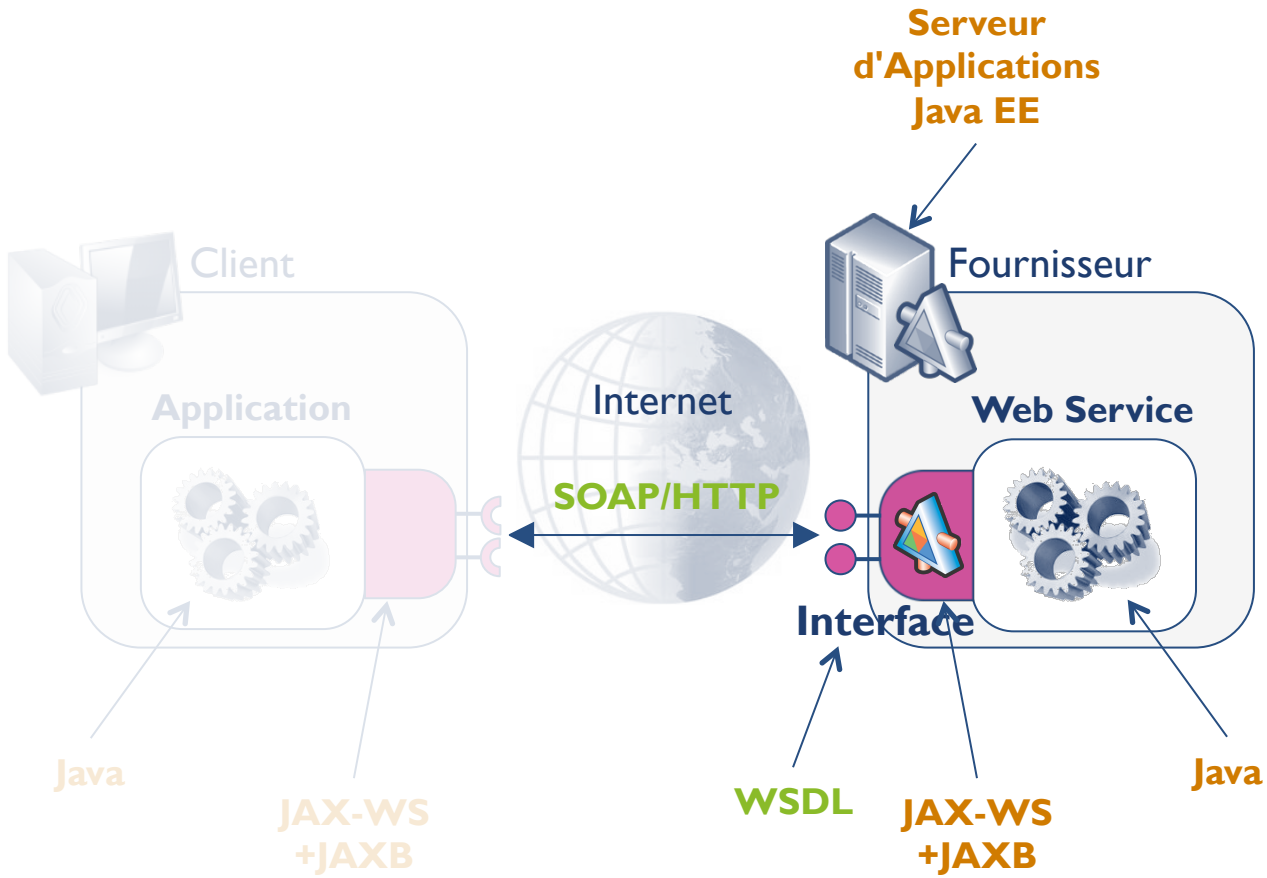
Et les autres WS-* ?



Les Web Services WS-*

1. Standards et acteurs
2. Principales technologies : WSDL, SOAP, UDDI
3. Exposer une application Java sous la forme d'un Web Service WS-*
4. Appeler un Web Service WS-* en Java
5. Et ça marche ?

Implémentation WS-* avec Java



Création d'un Web Service avec Java EE

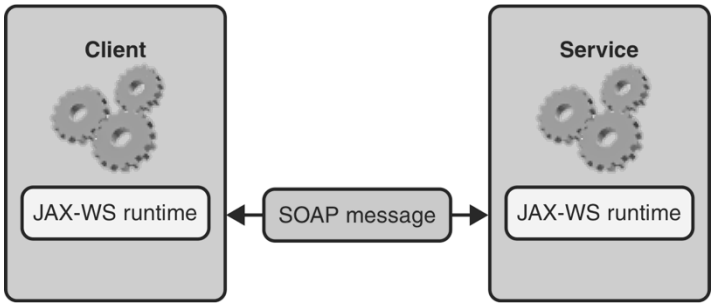
- Web Service = classe + annotations

```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService(serviceName="HelloService")
public class HelloService {

    @WebMethod(operationName="sayHello")
    public String sayHello(@WebParam(name="n") String n) {
        return "Hello dear "+n+" !";
    }
}
```

Correspondance annotation ↔ WSDL
= JAX-WS

JAX-WS
(Java API for XML Web Services)

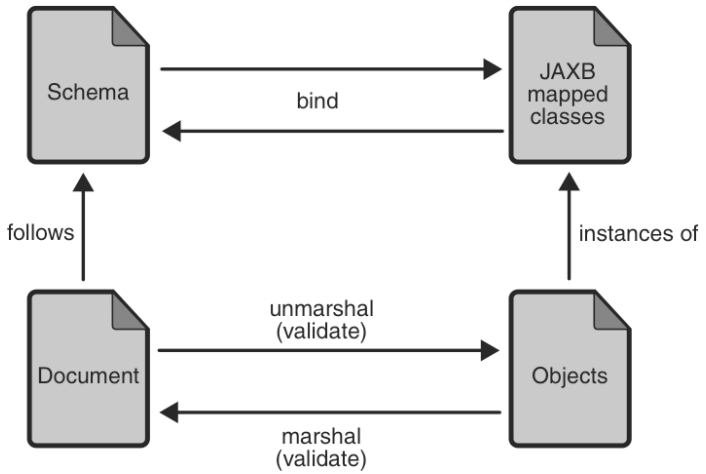
- Objectif = conversion WSDL ↔ Java et SOAP ↔ Java
  1. Correspondance automatique Classe (ou interface) Java ➜ WSDL
     - Génération de contrat (côté fournisseur ou client)
  2. Correspondance automatique WSDL ➜ Java
     - Génération d'un squelette de service à partir de son contrat
     - Génération d'un stub côté client
  3. Transformation automatiquement appel de méthode Java ↔ message SOAP



- Côté fournisseur, le serveur d'application exécute les opérations JAX-WS
- JAX-WS s'appuie sur JAX-B pour le traitement du XML

Objets passés en XML : JAXB (Java Architecture for XML Binding)

- Objectif = conversion XML ↔ Java
- Données nécessaires :
  - Schéma XML Schema
  - ou classes Java annotées
- Opérations supportées :
  - *Compilation* : XML Schéma ↔ classe Java annotée
  - *Exécution* : objet Java ↔ représentation XML = marshalling/unmarshalling
  - *Validation*



Source : The Java EE 5 Tutorial

Exemple avec JAXB

```java
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Product{

    @XmlElement
    private String name;
    @XmlElement
    private Double price;

    public Product(){…}

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
    …
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<product>
    <name>GPS TomTom Go Live 825M</name>
    <price>216.0</price>
</product>
```

Un peu de pratique !

Définition du contrat de service
« Code first » vs. « Contract first »

- Principe =
  1. Implémenter la logique métier
  2. Générer automatiquement le contrat WSDL pour le publier
- Avantages
  - Simple à réaliser
  - Utilité pour exposer du code legacy, ou pour faire des tests
- Inconvénients
  - Variations dans le contrat généré
  - Dépendance entre le code et le contrat
  - Développement de l'application cliente après développement du service

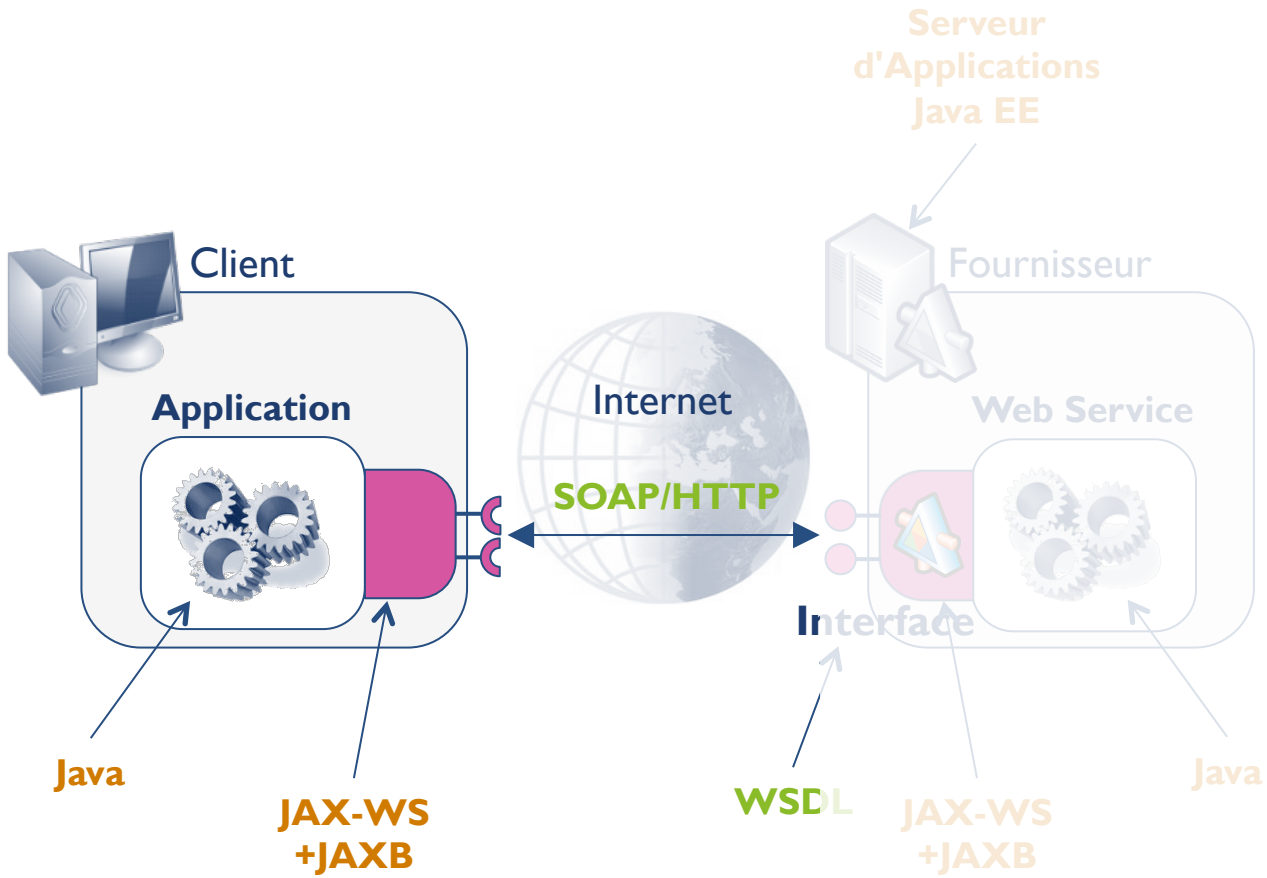
- Principe =
  1. Ecrire le contrat WSDL
  2. Implémenter la logique métier

  Possibilité de générer le squelette de code de la logique métier à partir du contrat
- Avantages
  - Meilleur découplage interface – implémentation, stabilité du WSDL
  - Meilleures performances généralement
- Inconvénients
  - Plus complexe à réaliser

Les Web Services WS-*

1. Standards et acteurs
2. Principales technologies : WSDL, SOAP, UDDI
3. Exposer une application Java sous la forme d'un Web Service WS-*
4. Appeler un Web Service WS-* en Java
5. Et ça marche ?

Implémentation WS-* avec Java



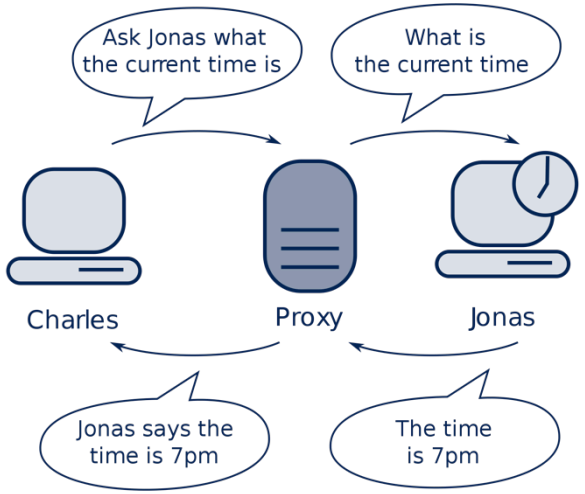
Client d'un Web Service

- Comme en RMI :
  ☛ stub/proxy = représentation du service dans l'espace du client,
  composant local qui délèguera les appels au composant distant

- Types de proxy :
  - Stub statique : classes générées à partir du WSDL
  - Proxy dynamique : classes générées *à l'exécution* à partir du WSDL
  - Dynamic Invocation Interface (DII) : *découverte* dynamique du service à l'exécution

JAX-WS

- Configuration : login / mot de passe, clé…



Création d'un client avec Java

- Créer/récupérer une interface Java représentant le service

```
@WebService
public interface HelloService {
    @WebMethod
    public String sayHello(@WebParam(name = "n") String n);
}
```

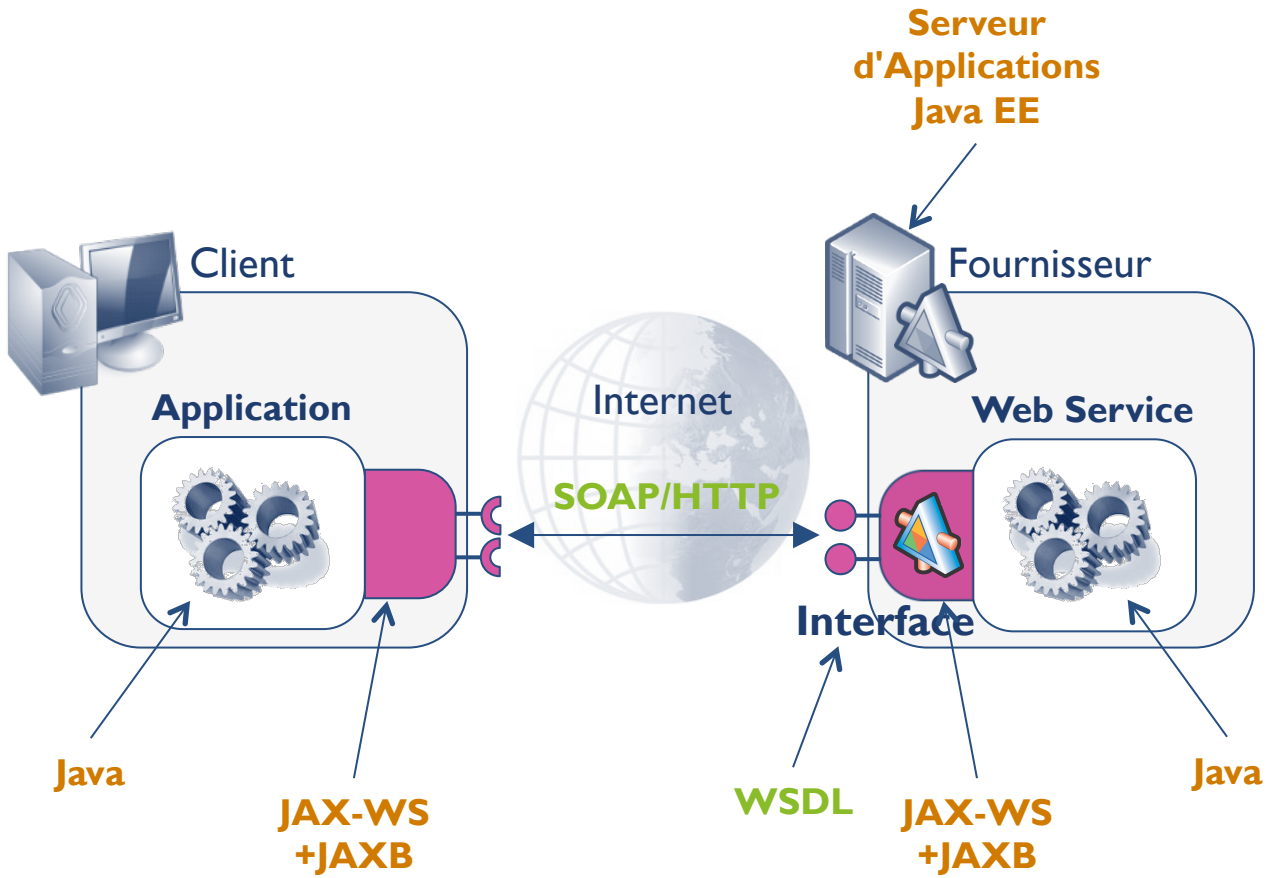
Il est possible de générer automatiquement un client : *voir les exercices…*

- A l'aide de JAX-WS, se connecter au service et appeler ses opérations :

```
// Creation du stub
URL wsdlURL = new URL("http://localhost:8080/HelloWebService/HelloService?WSDL");
QName serviceName = new QName("http://hello/", "HelloService");
Service serviceClient = Service.create(wsdlURL, serviceName);

QName portName = new QName("http://hello/", "HelloServicePort");
HelloService portStub = serviceClient.getPort(portName, HelloService.class);

System.out.println("portStub : "+portStub);


// Envoi d'une requête
System.out.println("réponse = " + portStub.sayHello("tutu tata"));
```

Un peu de pratique !

Implémentation WS-* avec Java



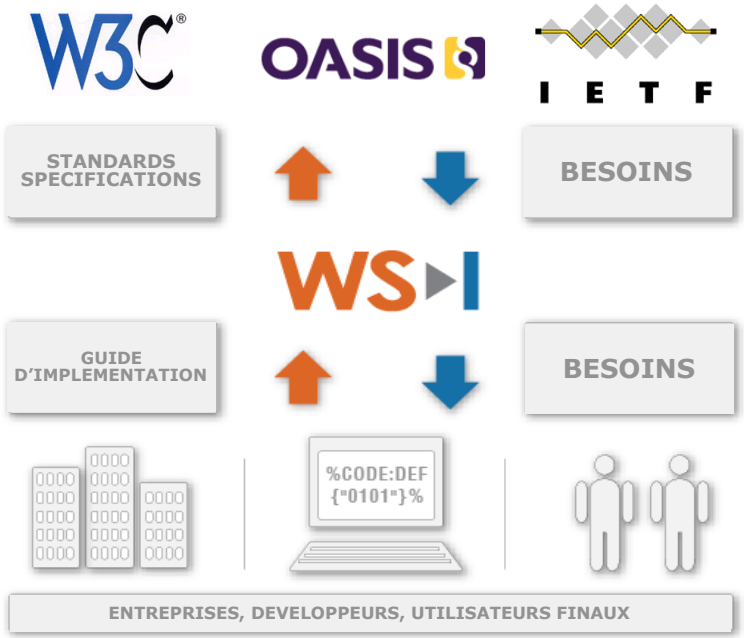
Les Web Services WS-*

1. Standards et acteurs
2. Principales technologies : WSDL, SOAP, UDDI
3. Exposer une application Java sous la forme d'un Web Service WS-*
4. Appeler un Web Service WS-* en Java
5. Et ça marche ?

Problématique de l'interopérabilité

- Problème = variations dans les implémentations des standards

- WS-I (Web Service Interoperability)
  - Consortium d'éditeurs de logiciels
  - Objectif = assurer l'interopérabilité entre les implémentations des normes liées aux Web Services
  - Produit
    - Des profils = ensembles de standards à implémenter + guides
    - Des exemples d'applications
    - Des outils de test



- WSIT (Web Services Interoperability Technologies)
  - Implémentation Java open source de certaines spécifications WS-* sélectionnées par WS-I et interopérables avec le WCF de .NET

Plan

① Qu'est-ce qu'un Web Service ?

② Les Web Services WS-*

③ Les Web Services RESTful

④ Synthèse et conclusion

Les Web Services RESTful

1. Principes
2. Appeler un Web Service RESTful en Java
3. Exposer une application Java sous la forme d'un Web Service RESTful

REST (Representational State Transfert)

- REST = style d'architecture orienté ressources semblable à celui du web
  - Ressource = information qui peut être identifiée de manière unique et référencée par un lien
    - Identifiant unique ➡ pour le web : URI
    - Plusieurs rendus possibles ➡ pour le web : HTML, XML…
  - Opérations CRUD sur les ressources = Create, Read, Update, Delete
    ➡ pour le web, opérations HTTP

- Objectif de l'architecture REST pour les Web Services : simplifier l'utilisation par rapport aux WS-*
  - Moins de standards à maîtriser et implémenter
  - Messages moins verbeux
  - Utilisation moins couteuse
  - …

RESTful Web Services

- Web Service = ressource avec une URI logique comme identifiant
  - URI obtenue par hiérarchie : http://supermarche.fr/produitsfrais/fruits/raisin
  - URI obtenue par construction : http://geographie.fr/altitude?lat=36&lon=10

- Opérations CRUD = opérations HTTP (requêtes/réponses)
  - PUT = Create = création de la ressource
  - GET = Read = lecture de la valeur de la ressource
  - POST = Update = modification de la valeur de la ressource
  - DELETE = Delete = destruction de la ressource

- Contraintes de conception :
  - Opérations idempotentes
  - Pas de session client-serveur (mais le client ou le serveur peut être stateful)

RESTful Web Services
Exemple de requête

- Requête HTTP GET

```
http://open.mapquestapi.com/nominatim/v1/reverse?
    format=xml&lat=48.7099500104522&lon=2.16758762635404
```

- Equivalent SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <serv:reverse xmlns:serv="http://open.mapquestapi.com/nominatim/v1/">
            <format>xml</format>
            <lat>48.7099500104522</lat>
            <lon>2.16758762635404</lon>
        </serv:reverse >
    </soap:Body>
</soap:Envelope>
```

- Les URLs peuvent être générées par des formulaires HTML !

RESTful Web Services
Exemple de réponse

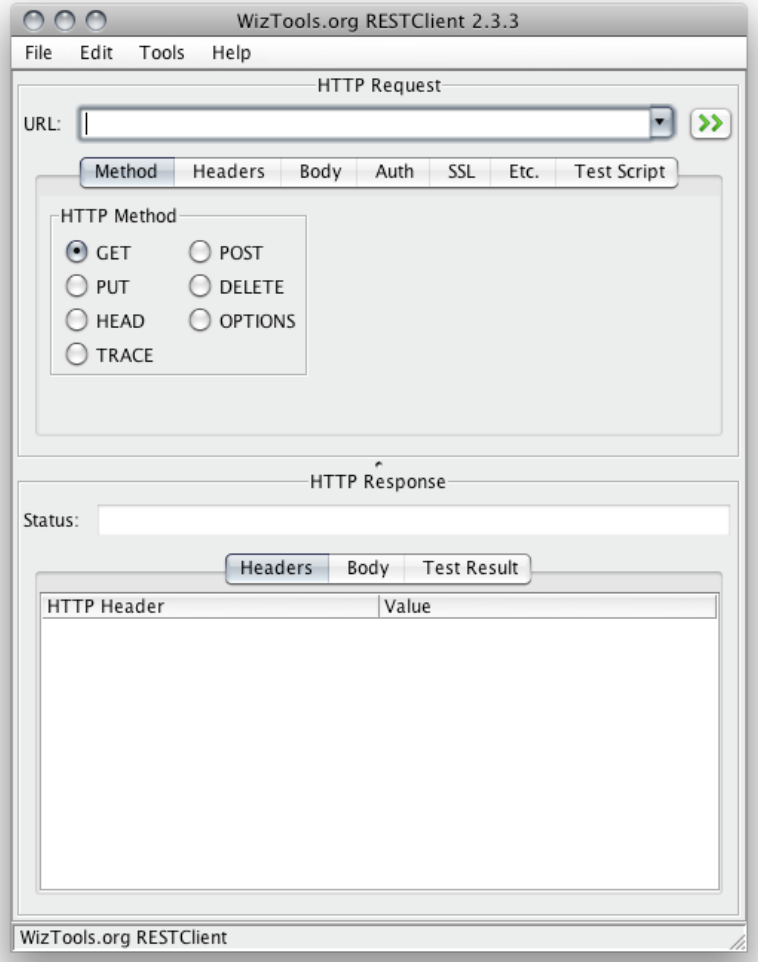
- Réponse HTTP standard
  - Chaine de caractères représentant le résultat de l'opération
  - Ou document XML représentant la ressource

```
<?xml version="1.0" encoding="UTF-8" ?>
…
    <addressparts>
        <bus_stop>Le Moulon</bus_stop>
        <road>Rue Joliot-Curie</road>
        <suburb>Montjay</suburb>
        <city>Gif-sur-Yvette</city>
        <administrative>Palaiseau</administrative>
        <county>Essonne</county>
        <state>Île-de-France</state>
        <postcode>91400</postcode>
        <country>France métropolitaine</country>
        <country_code>fr</country_code>
    </addressparts>
</reversegeocode>
```

Les Web Services RESTful

1. Principes
2. Appeler un Web Service RESTful
3. Exposer une application Java sous la forme d'un Web Service RESTful

Client d'un service REST



Client d'un service REST
Exemple avec Java

```
// Préparation de la connexion
Proxy proxy = new Proxy(Proxy.Type.HTTP,
                        new InetSocketAddress("proxy.supelec.fr", 8080));
URL url = new URL("http://localhost:8080/HelloREST/resources/helloREST");
HttpURLConnection connexion = (HttpURLConnection) url.openConnection(proxy);

// Envoi de la requête
connexion.setRequestMethod("GET");
connexion.connect(); // send GET request

// Récupération du contenu de la réponse
System.out.println("Réponse :");
InputStream stream = connexion.getInputStream();

BufferedReader r = new BufferedReader(new InputStreamReader(stream));
String line;
while((line = r.readLine()) != null){
     System.out.println(line);
}

// Déconnexion
connexion.disconnect();
```

Un peu de pratique !

Les Web Services RESTful

1. Principes
2. Appeler un Web Service RESTful
3. Exposer une application Java sous la forme d'un Web Service RESTful

Service REST
Exemple avec Java

```
@Path("/helloREST")
public class HelloResource {
    private String name;

    public HelloResource(){
        this.name="Robert";
    }

    @GET
    @Produces("text/plain")
    public String getHello() {
        return "Hello "+this.getName()+" !";
    }

    public String getName() {
        return name;
    }

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

Chemin d'accès de la ressource

Mapping des opérations HTTP sur des méthodes de la classe + type MIME du contenu produit/consommé (important)

Un peu de pratique !

Service REST
Autres méthodes HTTP

```
@Path("/helloREST")
@Singleton
public class HelloResource {
    private String name;

    public HelloResource(){
        this.name="Robert";
    }

    @GET
    @Produces("text/plain")
    public String getHello() {
        return "Hello "+this.getName()+" !";
    }

    @POST
    @Consumes("text/plain")
    @Produces("text/plain")
    public String putHello(String content) {
        this.setName(content);
        return "New name = "+this.getName();
    }
}
```

Composant EJB ayant une instance unique, dont le rôle est de "mémoriser" la chaîne de caractère "name"

Opération POST pour modifier la ressource + consomme et produit du texte

Plan

① Qu'est-ce qu'un Web Service ?

② Les Web Services WS-*

③ Les Web Services RESTful

④ Synthèse et conclusion

En résumé

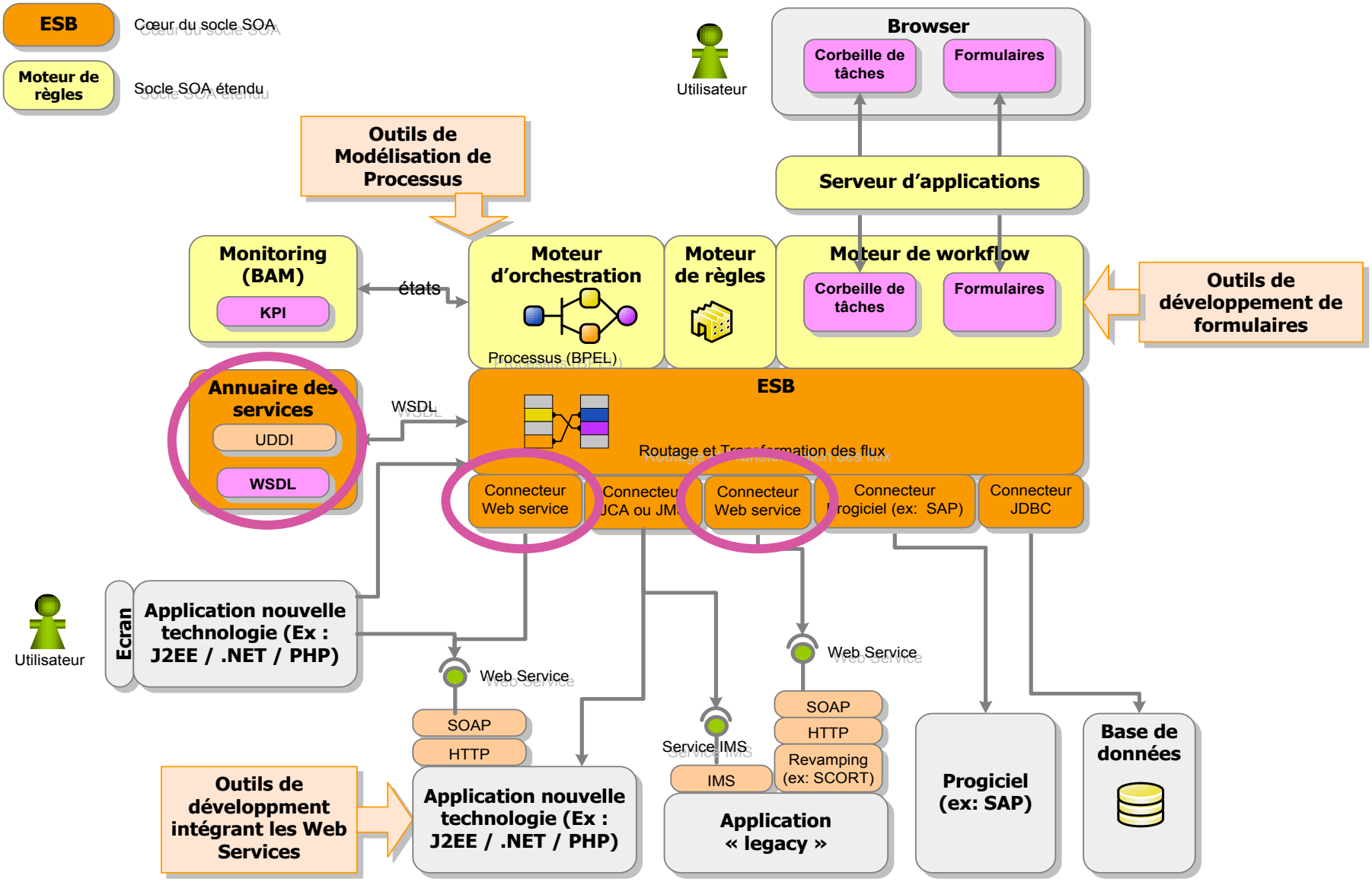
- Web Service = déclinaison technique des concepts SOA au niveau du web

- Technologies WS-* = SOAP + WSDL + UDDI
  - basées sur XML + HTTP
  - + d'autres technologies WS-* pour les aspects transverses

- Technologies REST = HTTP + XML

- Applications =
  - Intégration inter-applications point-à-point événementielle
    - Flux A2A : intégration de développement spécifiques, de COTS, de progiciels, de legacy, diversification des types de clients d'applications existantes…
    - Flux B2B : exposition de fonctionnalités à des partenaires
  - Intégration inter-applications type bus dans les solutions ESB

Rappel : ESB


ESB
Cœur du socle SOA
Moteur de règles
Socle SOA étendu
Utilisateur
Browser
Corbeille de tâches
Formulaires
Outils de Modélisation de Processus
Serveur d'applications
Monitoring (BAM)
KPI
états
Moteur d'orchestration
Processus (BPEL)
Moteur de règles
Moteur de workflow
Corbeille de tâches
Formulaires
Outils de développement de formulaires
Annuaire des services
UDDI
WSDL
WSDL
ESB
Routage et Transformation des flux
Connecteur Web service
Connecteur
Connecteur Web service
Connecteur
Connecteur JDBC
Utilisateur
Ecran
Application nouvelle technologie (Ex : J2EE / .NET / PHP)
Web Service
Web Service
SOAP
HTTP
SOAP
HTTP
Revamping (ex: SCORT)
Service IMS
IMS
Outils de développment intégrant les Web Services
Application nouvelle technologie (Ex : J2EE / .NET / PHP)
Application « legacy »
Progiciel (ex: SAP)
Base de données

Atouts & Challenges

- Interopérabilité
  - Environnement technologique hétérogène
  - Systèmes existants, composants sur étagère (COTS)
  - Diversité des clients
- Services riches
  - Agrégation dynamique de contenu
  - …
- Applications variées
  - A2A, B2B, ESB…

- Point-à-point inhérent
- Normes, produits et technologies en évolution
- Sécurité (authentification, autorisation, confidentialité)
- Robustesse
- Passage à l'échelle

Conclusion

- Les Web Services sont aujourd'hui incontournables dans le domaine de l'intégration inter-applicative
  - Ils ont révolutionné ce domaine par l'apport de standards
  - Ils ont révolutionné ce domaine en l'ouvrant à internet

- Mais
  - Malgré une « apparente » simplicité, les Web Services ne sont pas simples
    - Beaucoup de standards différents et concurrents
    - Beaucoup d'implémentations concurrentes
  - Le traitement de certains aspects transverses n'est pas suffisamment mature
    - Sécurité
    - Fiabilité et cohérence
    - Processus métier
  - La communication point-à-point n'est pas suffisante

Nécessité d'outils autour des Web Services (développement, exécution…)


MERCI!
THANK YOU!
FRAPAR.

Annexes

Définition du contrat de service Rapports d'erreurs

- Problématiques de la gestion des erreurs
  - Le client peut ne pas gérer les même exceptions ou pas de la même manière
  - La trace de la pile d'exécution n'est pas transmise au client

- Solution = convertir les exceptions applicatives
  en messages d'erreur spécifiques au service et significatifs pour le client

- Avec WSDL, élément « fault » permettant de décrire des rapports d'erreurs

Définition du contrat de service
Rapports d'erreur : exemple

```
<definitions>
…
    <message name="sayHello">
        <part name="parameter" type="xsd:string" />
    </message>
    <message name="sayHelloResponse">
        <part name="result" type="xsd:string" />
    </message>
    <message name="HelloException">
        <part name="HelloException" element="tns:HelloException" />
    </message>
    <portType name="HelloService">
        <operation name="sayHello">
            <input message="tns:sayHello" />
            <output message="tns:sayHelloResponse" />
            <fault name="HelloException" message="tns:HelloException"/>
        </operation>
    </portType>
…
</definitions>
```

Définition du contrat de service
Autres questions

- Granularité du service
  - Compromis flexibilité / interaction (grain fin) vs. performance (gros grain)
  - Imposée potentiellement par la logique métier
  - Possibilité de mise en cache de données sur le client ? (impact)

- Type des paramètres et type de retour
  - Compromis adaptation des types (types riches) vs. performance (types simples)
    - Coût des transformations entre formats locaux et XML
    - Non garantie de conservation de l'ordre des paramètres lors des transformations
  - Pas de correspondance standard pour tous les types
  - Attachements SOAP avec types MIMES

- Méthodes homonymes
  - Génération automatique de noms possible mais peu « ergonomique »
  - Désambiguïsation systématique et explicite des noms

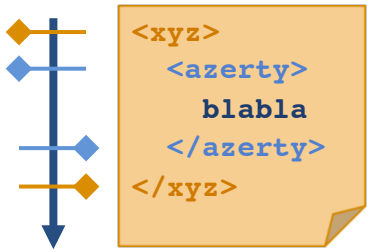
Développement de la couche traitement du XML

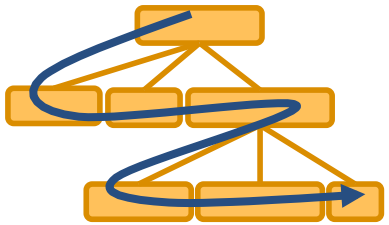
- Pré-traitement du XML entrant
  - Validation par rapport au schéma
  - Transformation vers un autre schéma si nécessaire
  - Correspondance avec les objets
- Passage des appels au cœur de l'application (logique métier)
  - Services synchrones : appel direct
  - Services asynchrones :
    - Message posté dans une file (JMS…)
    - Réponse par récupération dans la file ou par callback
- Formulation des réponses en XML
  - Gestion de cache éventuelle
  - Transformation des résultats de la couche métier (rendu)

Traitement du XML

- Manipulation de documents XML
  - Analyse lexicale et syntaxique
  - Vérification de la conformité du document avec la norme XML
  - Validation par rapport à un schéma (DTD, XML Schema...)
  - Éventuellement, modification du contenu

- Deux modèles de traitement
  - Modèle événementiel = vision « document balisé »
    - SAX (Simple API for XML), standard de fait
    - http://www.saxproject.org/
  - Modèle arborescent / objet = vision « arbre »
    - DOM (Document Object Model), standard W3C
    - http://www.w3.org/DOM/

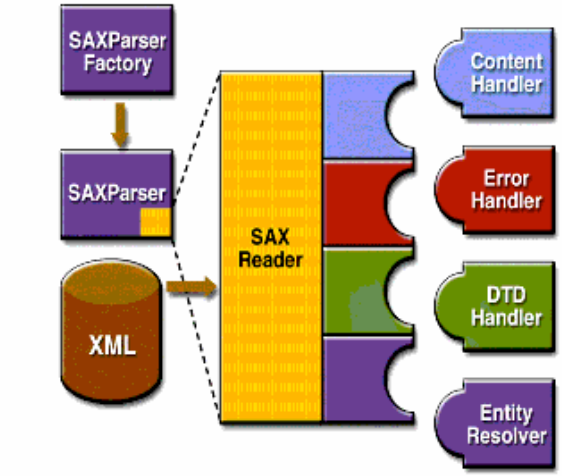


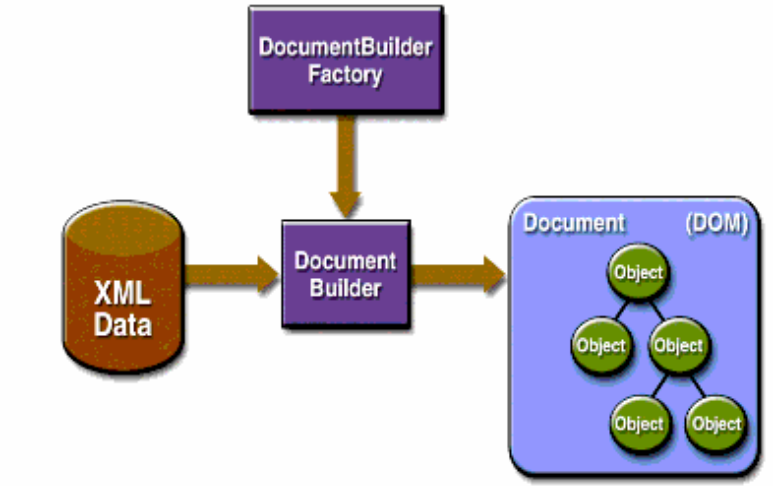


  - Des implémentations de SAX et DOM existent pour presque tous les langages (Java, Javascript, C, C++, Perl, Python, langages .Net…)

Exemple avec Java : JAXP (Java API for XML Processing)

- API de Java Standard Edition
- Objectif = traitement du XML
- Définit des APIs + des processeurs XML
  - Support du modèle événementiel SAX
  - Support du modèle arborescent DOM
  - Support du modèle événement StAX (Streaming API for XML)
    - ≈ SAX en mode « pull », streaming
  - Support des transformations XSLT (eXtensible Stylesheet Language Transformation)

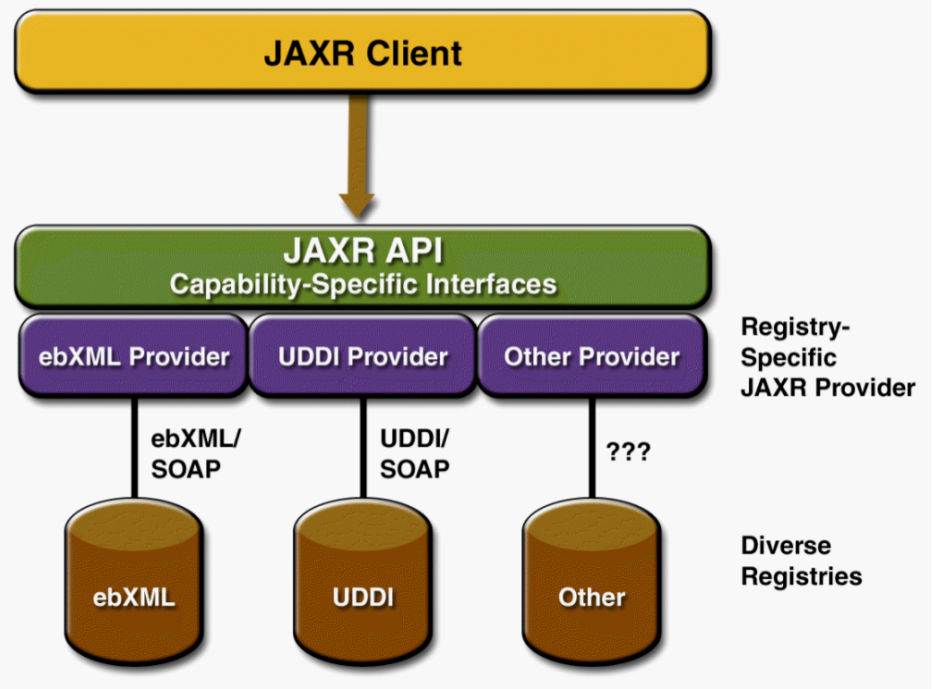




Interrogation d'un annuaire : JAXR (JavaAPI for XMLRegistries)

- API de Java Enterprise Edition

- Objectif = accès à un registre de Web Services
  - Support de plusieurs technologies d'implémentation de registre
    - UDDI
    - ebXML (OASIS)
  - Support de plusieurs opérations
    - Connexion au registre
    - Envoi de requêtes
    - Gestion des données du registre
    - Utilisation des taxonomies



Source : The Java EE 5 Tutorial