

Les Services Web

SOAP

Plan du cours

- ❖ La présentation des services web
- ❖ Les standards : SOAP, WSDL et UDDI
- ❖ Les différents formats de services web SOAP
- ❖ Des conseils pour la mise en œuvre
- ❖ Les étapes de mise en œuvre
- ❖ Les implémentations des services web : AXIS et AXIS 2
- ❖ Les API Java pour les services web : JAX-WS
- ❖ Inclure des pièces jointes dans SOAP
- ❖ WS-Interoperability (WS-I)

Mais, avant de commencer...un tour de table

Pré-requis:

- POO
- Java SE et Java EE
- XML

La présentation des services web(1/6)

- Les services web sont **des composants distribués** qui offrent des fonctionnalités aux applications au travers du réseau en utilisant **des standards ouverts**.
- Ils peuvent donc être utilisés par des applications écrites dans **différents langages** et exécutées dans différentes plate-forme sur différents systèmes.
- Un service web permet généralement de proposer une ou plusieurs **fonctionnalités métiers** qui seront invoquées par un ou plusieurs consommateurs.
- Il existe deux grandes familles de services web :
 - Les services web de type **SOAP (Sample Object Access Protocol)**
 - Les services web de type **REST (Representational State Transfer)**

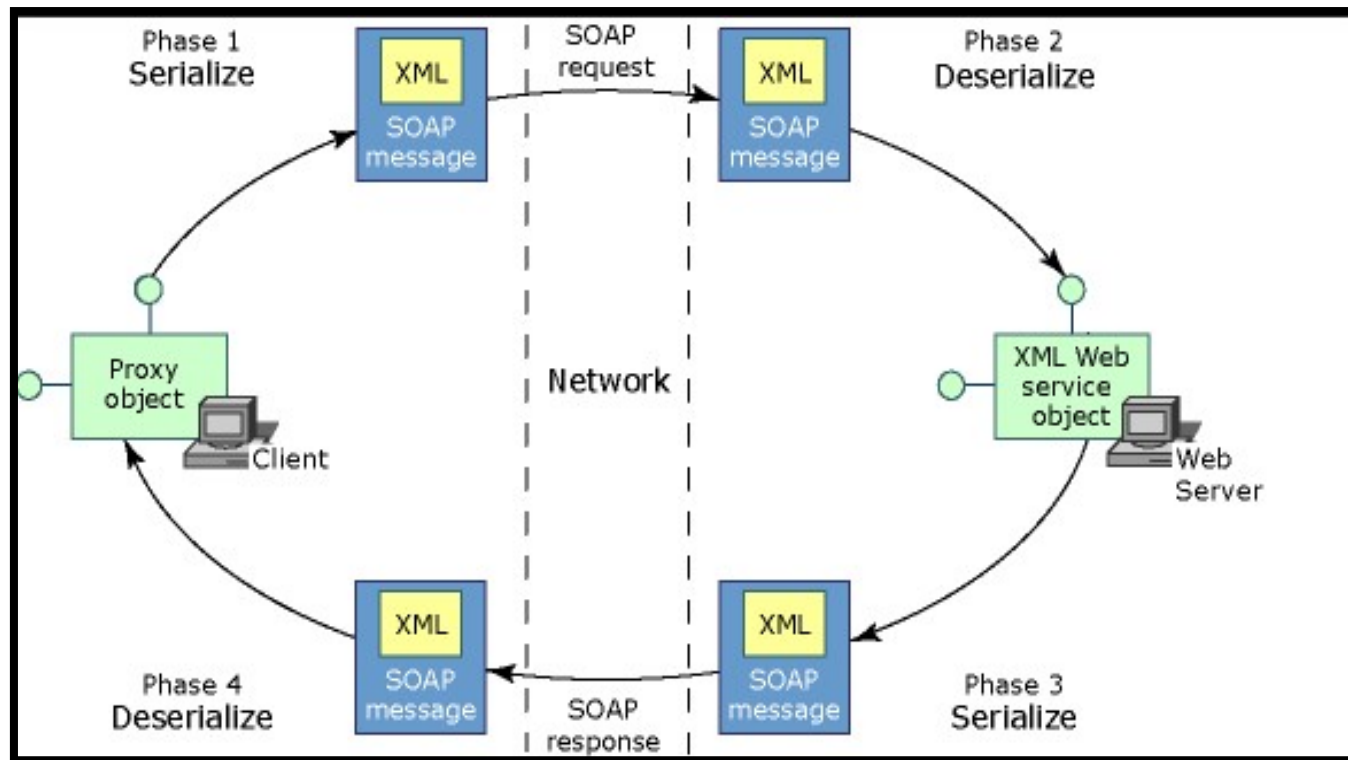
La présentation des services web(2/6)

- Il existe plusieurs définitions pour les services web mais la plus simple pourrait être "fonctionnalité utilisable à travers le réseau en mettant en œuvre **un format standard utilisant généralement XML**".
- Les services web de type SOAP font un usage intensif de XML, des namespaces XML et des schémas XML. Ces technologies font la force des services web pour permettre leur utilisation par des clients et des serveurs hétérogènes.
- XML est utilisé pour **stocker** et **organiser** les informations de la requête et de la réponse mais aussi pour **décrire** le service web. L'utilisation de XML pour le format des messages rend les échanges indépendants du système d'exploitation, de la plate-forme et du langage.

La présentation des services web(3/6)

- Il est possible de développer des services web avec une plate-forme (par exemple Java) et d'utiliser ces services web avec une autre plate-forme (par exemple .Net ou PHP).
- Un service web est donc une fonctionnalité accessible à travers le réseau grâce à des messages au format XML. Le format de ces messages est généralement SOAP.
- Les services web peuvent prendre plusieurs formes :
 - Métier
 - Technique
 - ...

La présentation des services web(4/6)



1. La présentation des services web(5/6)

L'appel à un service web de type SOAP suit plusieurs étapes :

1. Le client instancie une classe de type proxy encapsulant le service Web XML.
2. Le client invoque une méthode du proxy.
3. Le moteur SOAP sur le client crée le message à partir des paramètres utilisés pour invoquer la méthode
4. Le moteur SOAP envoie le message SOAP au serveur généralement en utilisant le protocole HTTP
5. Le moteur SOAP du serveur réceptionne et analyse le message SOAP
6. Le moteur fait appel à la méthode de l'objet correspondant à la requête SOAP
7. Le moteur SOAP sur le serveur crée le message réponse à partir de la valeur de retour
8. Le moteur SOAP envoie le message SOAP contenant la réponse au client généralement en utilisant le protocole http
9. Le moteur SOAP du client réceptionne et analyse le message SOAP
10. Le moteur SOAP du client instancie un objet à partir du message SOAP

La présentation des services web(6/6)

Les services web proposent un mécanisme facilitant :

- ✓ La communication entre applications hétérogènes : un service web développé dans une technologie peut être consommé par une application développée dans une autre technologie. Ceci est possible car les services web reposent sur des standards ouverts.
- ✓ L'exposition de fonctionnalités métiers aux applications internes mais aussi à des applications externes : dans ce dernier cas l'utilisation du protocole HTTP permet facilement de passer les pare-feu.
- ✓ La mise en œuvre d'une architecture SOA puisque les services web peuvent être une implémentation possible d'une telle architecture.

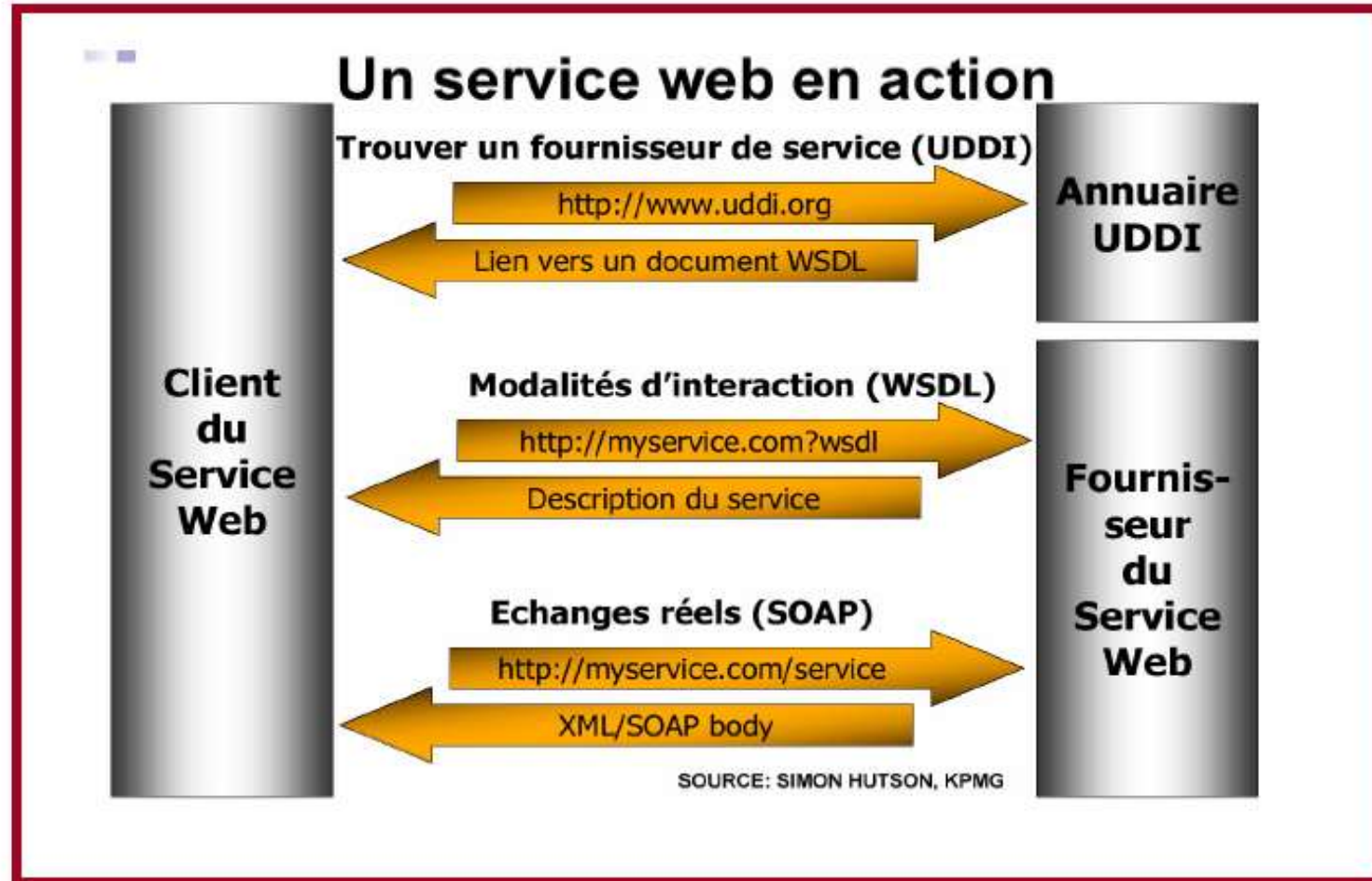
Les standards

L'architecture des services web est composée de quatre grandes couches utilisant plusieurs technologies :

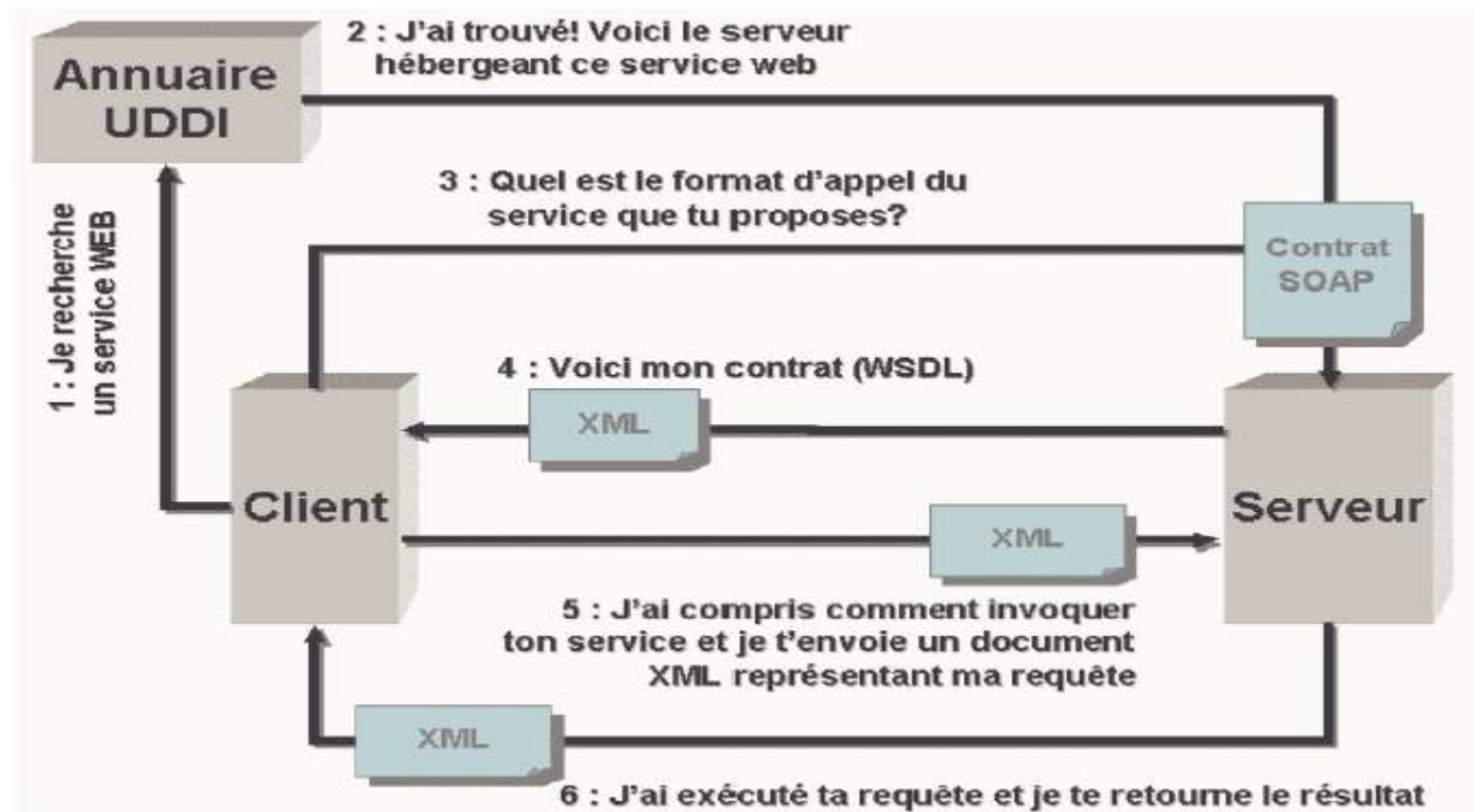
- **Découverte** : cette couche représente un annuaire dans lequel il est possible de publier des services et de les rechercher : **UDDI** (***Universal Description, Discovery, and Integration***) est le standard.
- **Description** : cette couche normalise la description de l'interface publique d'un service web en utilisant **WSDL** (***Web Service Description Language***).
- **Communication** : cette couche permet d'encoder les messages échangés (**SOAP** est le standard).
- **Transport** : cette couche assure le transport des messages : généralement **HTTP** est mis en œuvre mais d'autres protocoles peuvent être utilisés (SMTP, FTP, ...).

Les standards

Le scenario complet



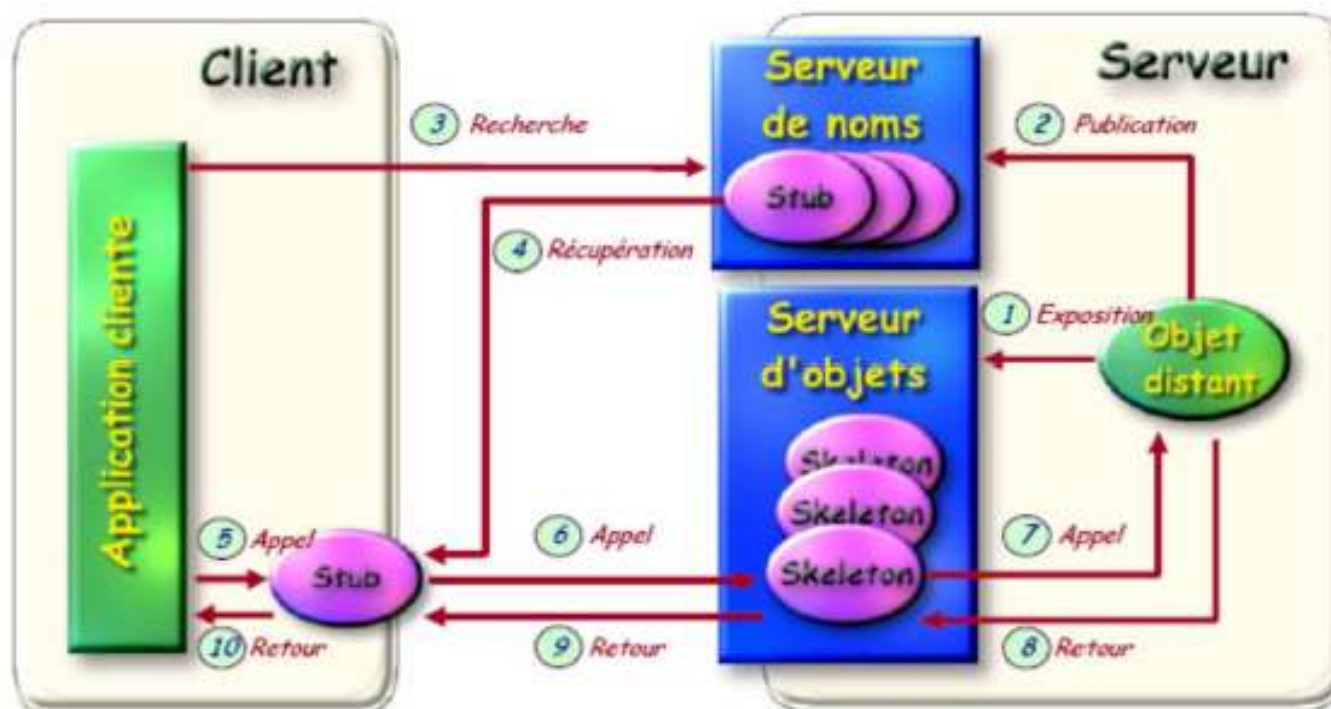
Les standards



Les standards

Comparaison avec les solutions middleware

- Les mêmes notions existent dans CORBA, EJB, RMI



Source: R. Voyer

Sample Object Access Protocol (SOAP)

Les standards

SOAP (1/12)

- SOAP (**Simple Object Access Protocol**) est un standard du W3C qui permet l'échange formaté d'informations entre un client et un serveur. SOAP peut être utilisé pour la requête et la réponse de cet échange.
- SOAP assure la partie messaging dans l'architecture des services web : il est utilisé pour normaliser le format des messages échangés entre le consommateur et le fournisseur de services web.
- SOAP est un protocole qui est principalement utilisé pour dialoguer avec des objets distribués.
- Son grand intérêt est d'utiliser XML ce qui le rend ouvert contrairement aux autres protocoles qui sont propriétaires : cela permet la communication entre un client et un serveur utilisant des technologies différentes. SOAP fait un usage intensif des espaces de nommages (namespaces).

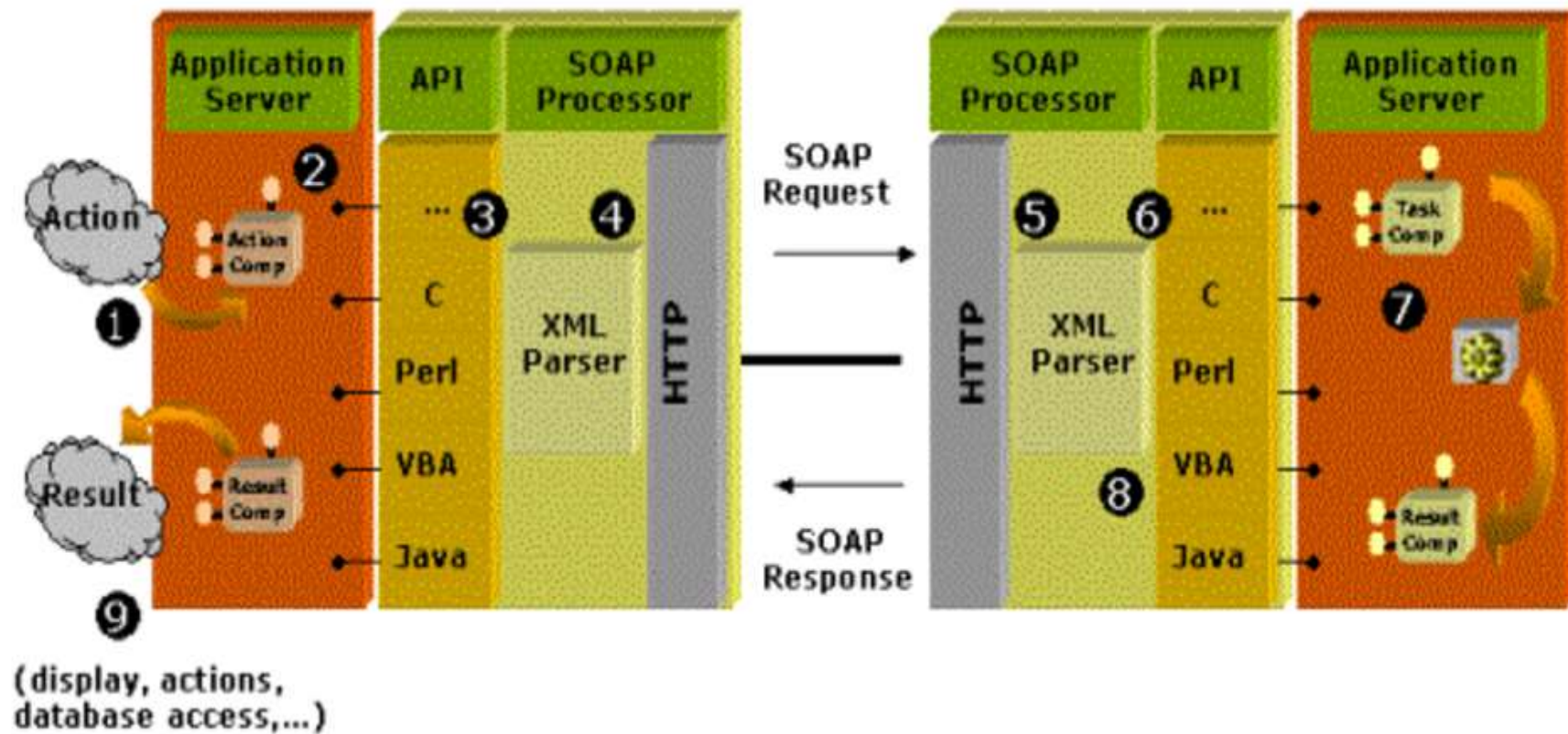
Les standards

SOAP (2/12)

- SOAP est défini pour être indépendant du protocole de transport utilisé pour véhiculer le message. Cependant, le protocole le plus utilisé avec SOAP est HTTP. Son utilisation avec SOAP permet de rendre les services web plus interopérables.
- D'autres protocoles peuvent être utilisés (par exemple SMTP ou FTP) mais leur configuration sera plus délicate car elle ne sera pas fournie en standard comme c'est le cas avec HTTP.
- SOAP est aussi indépendant de tout système d'exploitation et de tout langage de programmation car il utilise XML. Ceci permet une exposition et une consommation de services web avec des outils et des OS différents.
- SOAP peut être utilisé pour :
 - Appeler une méthode d'un service (SOAP RPC)
 - Echanger un message avec un service (SOAP Messaging)
 - Recevoir un message d'un service.

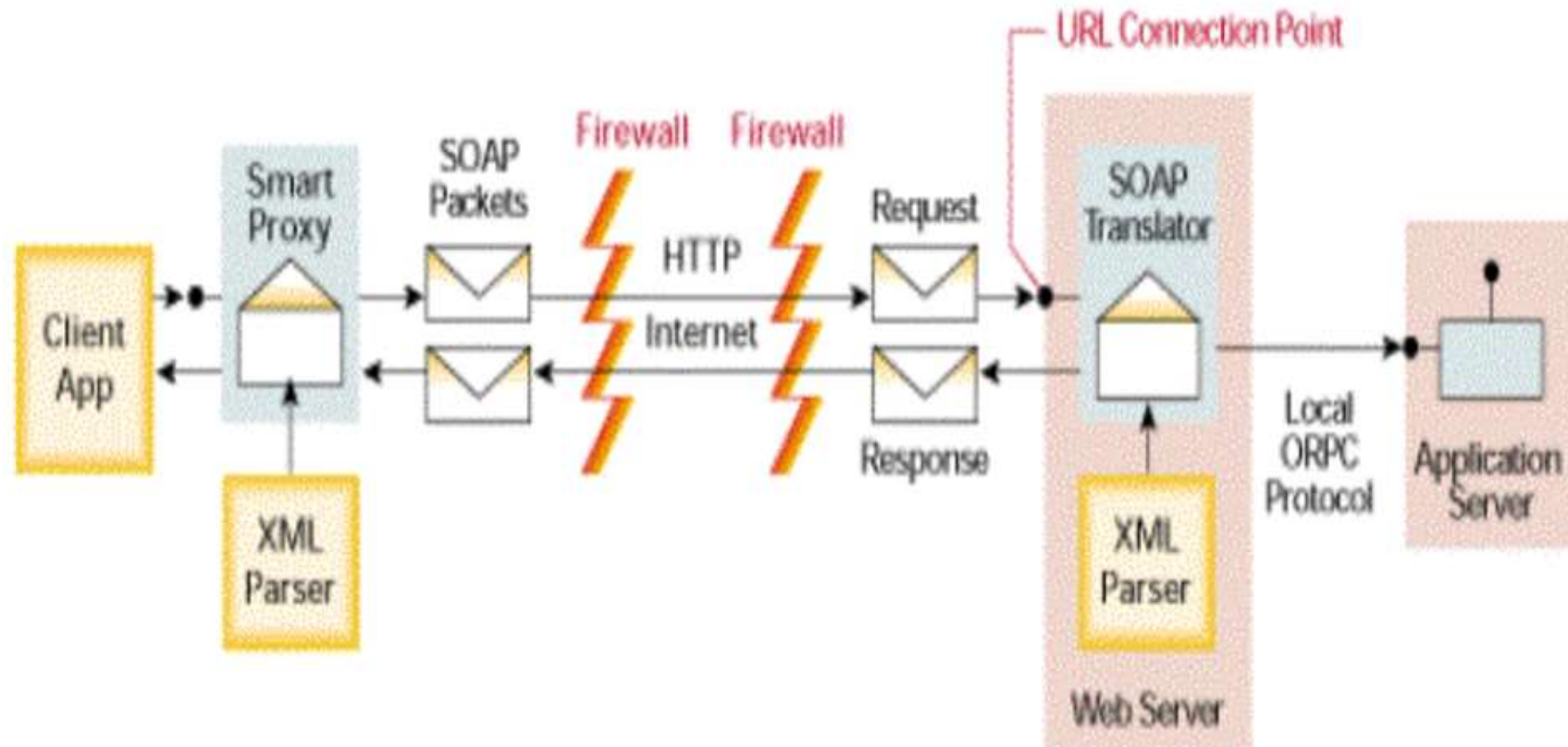
Les standards

SOAP (3/12)



Les standards

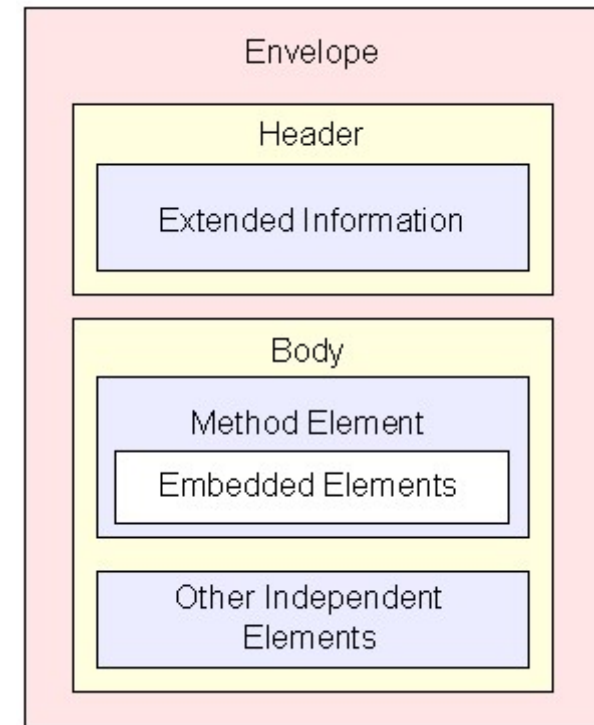
SOAP (4/12)



Les standards

SOAP (5/12)

- Un message SOAP est contenu dans une enveloppe, le tag racine d'un document SOAP est le tag **<Envelope>**.
- La structure d'une enveloppe SOAP se compose de plusieurs parties :
 - Une en-tête optionnelle (**Header**) composée d'un ou plusieurs headers : il contient des informations sur le traitement du message
 - Un corps (**Body**) : il contient les informations de la requête ou de la réponse
 - Une gestion d'erreurs optionnelle (Fault) contenue dans le corps
 - Des pièces jointes optionnelles (attachment) contenues dans le corps



Les standards

SOAP (6/12)

- L'en-tête contient des informations sur le traitement du message : ces informations sont contenues dans un tag <Header>. Pour des services web simples, cette partie peut être vide ou absente mais pour des services plus complexes elle peut contenir des informations concernant **les transactions, la sécurité, le routage**, etc ...
- Le corps du message SOAP est contenu dans un tag <**Body**> obligatoire. Il contient les données échangées entre le client et le service sous la forme d'un fragment de document XML.
- Tous ces éléments sont codés dans le message XML avec un tag particulier mettant en œuvre un espace de nommage particulier défini dans les spécifications de SOAP.
- Un message SOAP peut aussi contenir **des pièces jointes** placées chacune dans une partie optionnelle nommée **AttachmentPart**. Ces parties appartiennent à la partie SOAP Part.
- SOAP définit aussi l'encodage pour les différents types de données qui est basé sur la technologie schéma XML du W3C. Les données peuvent être de type simple (chaîne, entier, flottant, ...) ou de type composé.

Les standards

SOAP (7/12)

- Deux formats de messages SOAP sont définis (Ce qu'on appelle **le Style**):
 - **Remote Procedure Call (RPC)** : permet l'invocation d'opérations qui peuvent retourner un résultat.
 - **Message Oriented (Document)** : données au format XML définies dans un schéma XML
- Les règles d'encodage (Encoding rules) précisent les mécanismes de sérialisation des données dans un message. Il existe deux types :
 - **Encoded** : les paramètres d'entrée de la requête et les données de la réponse sont encodées en XML dans le corps du message selon un format particulier à SOAP.
 - **Literal** : les données n'ont pas besoin d'être encodées de façon particulière : elles sont directement encodées en XML selon un schéma défini dans le WSDL.

Les standards

SOAP (8/12)

- Le style (RPC ou Document) et le type d'encodage (Encoded ou Literal) permettent de définir comment les données seront sérialisées et désérialisées dans les requêtes et les réponses.
- La combinaison du style et du type d'encodage peut prendre plusieurs valeurs :
 - RPC/Encoded
 - RPC/Literal
 - Document /Encoded
 - Document/Literal
 - Wrapped Document/Literal : extension du Document/Literal proposé par Microsoft.
- Le style RPC/Encoded a largement été utilisé au début des services web : actuellement ce style est en cours d'abandon par l'industrie au profit du style **Document/Literal**.
- Le style et le type d'encodage sont précisés dans le WSDL. L'appel du service web doit obligatoirement se faire dans le style précisé dans le WSDL puisque celui-ci détermine le format des messages échangés.

Les standards

SOAP (9/12)

- Les versions de SOAP
 - 1.0 :
 - 1.1 :
 - 1.2 : permet l'utilisation de requête http de type GET
- La version 1.2 des spécifications de SOAP est plus précise et a permis de réduire les problèmes d'interopérabilité entre différentes implémentations.
- SOAP 1.2 propose un support pour des protocoles de transport différents de HTTP. La sérialisation de messages n'est pas obligatoirement en XML mais peut utiliser des formats binaires (XML Infoset par exemple).

Les standards

SOAP (10/12)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header> <!-- optional -->
    <!-- header blocks go here... -->
</soap:Header>
<soap:Body>
    <!-- payload or Fault element goes here... --> </soap:Body>
</soap:Envelope>
```

```
<soap:Header>
    <!-- security credentials -->
    <s:credentials xmlns:s="urn:examples-org:security">
        <username>dave</username>
        <password>evad</password>
    </s:credentials>
</soap:Header>
```

```
<soap:Body>
<x:TransferFunds xmlns:x="urn:examples-
org:banking">
    <from>22-342439</from>
    <to>98-283843</to>
    <amount>100.00</amount>
</x:TransferFunds>
</soap:Body>
```


Les standards

SOAP(11/12)

Message SOAP embarqué dans une requête HTTP :

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Les standards

SOAP(12/12)

Message SOAP embarqué dans une réponse HTTP :

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-
  ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web Service Description Language (WSDL)

Les standards

WSDL(1/9)

- WSDL (Web Service Description Language) est utilisé pour fournir une description d'un service web afin de permettre son utilisation. C'est une recommandation du W3C.
- Pour permettre à un client de consommer un service web, ce dernier a besoin d'une description détaillée du service avant de pouvoir interagir avec lui. Un WSDL fournit cette description dans un document XML. **WSDL joue un rôle important dans l'architecture des Services Web en assurant la partie description** : il contient toutes les informations nécessaires à l'invocation du service qu'il décrit.
- La description WSDL d'un service web comprend une définition du service, les types de données utilisés notamment dans le cas de types complexes, les opérations utilisables, le protocole utilisé pour le transport et l'adresse d'appel.
- C'est un document XML qui décrit un service web de manière indépendante de tout langage. Il permet l'appel de ses opérations et l'exploitation des réponses (les paramètres, le format des messages, le protocole utilisé, ...).

Les standards

WSDL(2/9)

- WSDL est conçu pour être indépendant de tout protocoles. Comme SOAP et HTTP sont les deux protocoles les plus couramment utilisés pour implémenter les services web, le standard WSDL intègre un support de ces deux protocoles.
- L'utilisation de XML permet à des outils de différents systèmes, plate-formes et langages d'utiliser le contenu d'un WSDL **pour générer du code permettant de consommer un service web.**
- Les moteurs SOAP proposent en général un outil qui va lire le WSDL et générer les classes requises pour utiliser un service web avec la technologie du moteur SOAP. Le code généré utilise un moteur SOAP qui masque toute la complexité du protocole utilisé et des messages échangés lors de la consommation de services web en agissant comme un proxy. Par exemple, **Axis** propose l'outil **WSDL2Java** pour **la génération de ces classes à partir du WSDL.**
- Les spécifications de WSDL sont consultables à l'url http://www.w3.org/standards/techs/wsdl#w3c_all

Les standards

WSDL(3/9)

- Un document WSDL définit plusieurs éléments :

Élément	Rôle
Type	La définition des types de données utilisés.
Message	La définition de la structure d'un message en lui attribuant un nom et en décrivant les éléments qui le composent avec un nom et un type.
PortType	La description de toutes les opérations proposées par le service web (interface du service) et identification de cet ensemble avec un nom.
Operation	La description d'une action proposée par le service web notamment en précisant les messages en entrée (input) et en sortie (output).
Binding	La description du protocole de transport et d'encodage utilisé par un PortType afin de pouvoir invoquer un service web.
Port	Référence un Binding (généralement cela correspond à l'url d'invocation du service web)
Service	C'est un ensemble de ports.

- Un WSDL est un document XML dont le tag racine est `<definitions>` et qui utilise l'espace de nommage "http://schemas.xmlsoap.org/wsdl/".

Les standards

WSDL(4/9)

- Un WSDL est composé de deux parties :
 - **des définitions abstraites** : celles-ci concernent l'interface du service (types, message, portType). Ces informations sont exploitées dans le code du client
 - **des définitions concrètes** : celles-ci concernent l'invocation du service (binding, service). Ces informations sont exploitées par le moteur SOAP.
- Le contenu du WSDL d'un service nommé MonService est de la forme :

```
<!--Structure d'un WSDL -->
<definitions name="MonService"
targetNamespace="http://com.test.ws.monservice/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <!-- Définitions abstraites -->
    <types> ... </types>
    <message> ... </message>
    <portType> ... </portType>
    <!-- Définitions concrètes -->
    <binding> ... </binding>
    <service> ... </service>
</definitions>
```

Les standards

WSDL(5/9)

exemple

Exemple :

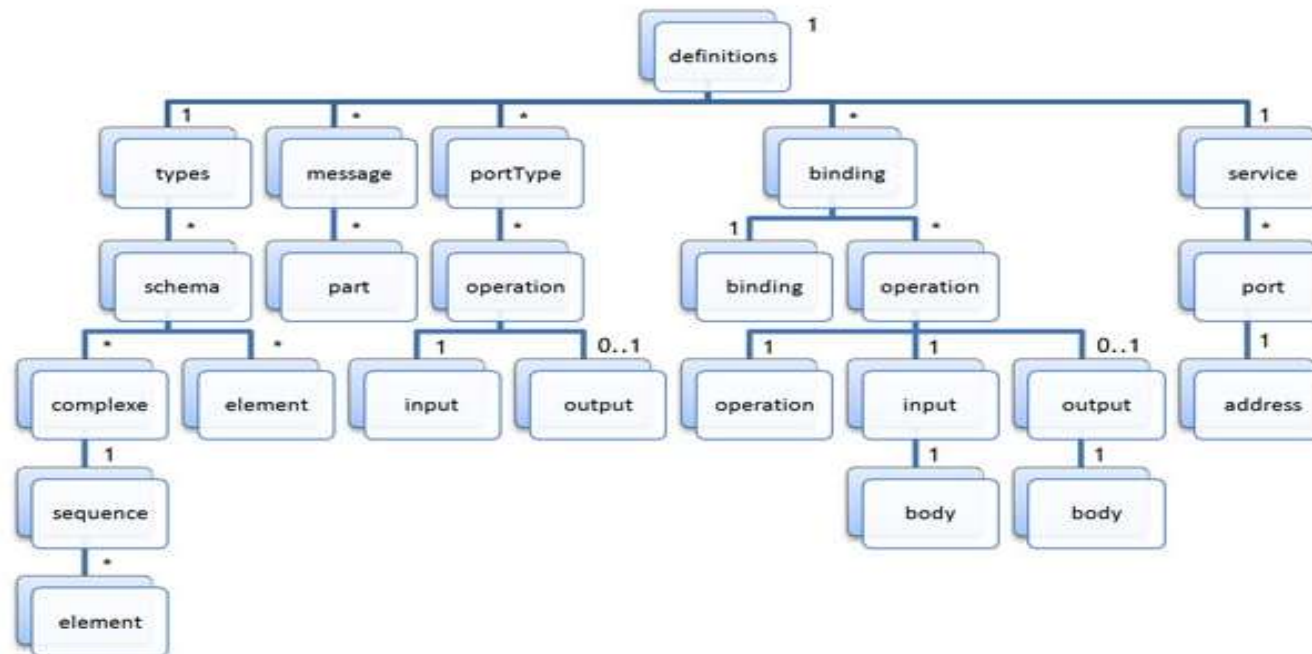
```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://axis.test.jmdoudoux.com"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://axis.test.jmdoudoux.com"
  xmlns:intf="http://axis.test.jmdoudoux.com"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!--
WSDL created by Apache Axis version: 1.3
Built on Oct 05, 2005 (05:23:37 EDT)
-->
  <wsdl:message name="additionnerRequest">
    <wsdl:part name="valeur1" type="xsd:int" />
    <wsdl:part name="valeur2" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="additionnerResponse">
    <wsdl:part name="additionnerReturn" type="xsd:long" />
  </wsdl:message>
  <wsdl:portType name="Calculer">
    <wsdl:operation name="additionner" parameterOrder="valeur1 valeur2">
      <wsdl:input message="impl:additionnerRequest" name="additionnerRequest" />
      <wsdl:output message="impl:additionnerResponse" name="additionnerResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CalculerSoapBinding" type="impl:Calculer">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="additionner">
      <wsdlsoap:operation soapAction="" />
      <wsdl:input name="additionnerRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://axis.test.jmdoudoux.com" use="encoded" />
      </wsdl:input>
      <wsdl:output name="additionnerResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://axis.test.jmdoudoux.com" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CalculerService">
    <wsdl:port binding="impl:CalculerSoapBinding" name="Calculer">
      <wsdlsoap:address location="http://localhost:8080/TestWS/services/Calculer" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```


Les standards

WSDL(6/9)

Le tag **<definitions>** peut contenir plusieurs tags fils :

- **<types>** : description des types de données utilisés
- **<message>** : description des messages qui peuvent être composés de plusieurs types
- **<portType>** : description des opérations du endpoint sous la forme d'échanges de messages. Ceci correspond à l'interface du service.
- **<binding>** : description du protocole et spécification du format des données pour un portType
- **<service>** : description des endpoints du service (binding et uri)



Les standards

WSDL(7/9)

Le tag <types> :

- L'élément <types> contient une définition des différents types de données qui seront utilisées.
- Cette définition peut être faite sous plusieurs formats mais l'utilisation des schémas XML est recommandée.
- Le **WS-I Basic Profile** impose que cette description soit faite avec des schémas XML.

Le tag <message> :

Le tag <message> décrit un message qui est utilisé en tant que requête ou réponse lors de l'invocation d'une opération : il contient une définition des paramètres pour un message échangé en entrée ou en sortie.

Le tag <part> possède un attribut "name" pour permettre d'y faire référence et utilise un attribut "element" (pour le style document qui représente l'élément XML inséré dans le body) ou un attribut "type" (pour le style RPC qui représente les paramètres de l'opération).

Le tag < portType > :

Le tag <portType> décrit l'interface d'un service web.

Le tag <operations> :

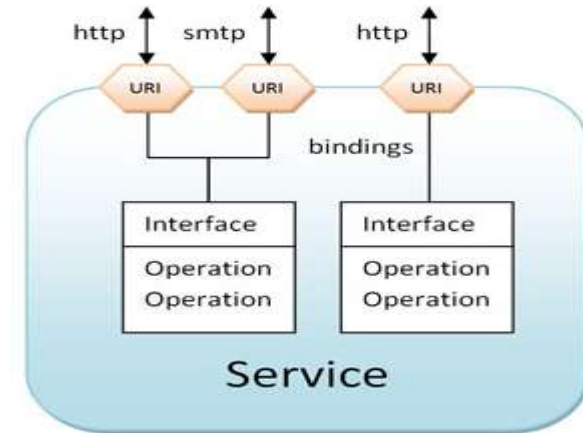
Ce tag peut avoir les tags fils <input>, <output> et <fault>. La présence et l'ordre des deux premiers tags définissent le mode d'invocation d'une opération, nommé MEP (Message Exchange Pattern)

Les standards

WSDL(8/9)

Le tag <binding> :

- La description du service doit aussi fournir des informations pour invoquer le service :
 - Le protocole utilisé pour le transport du message
 - L'encodage du message : style et mécanisme d'encodage
- Un binding permet de fournir des détails sur la façon dont les données sont transportées.
- Un service peut avoir plusieurs bindings mais chacun doit se faire sur une URI unique nommée **endpoint**.
- Le tag fils <soap:binding> est utilisé pour préciser que c'est la version 1.1 de SOAP qui sera utilisée. Son attribut style permet de préciser le style du message : les valeurs possibles sont **rpc** ou **document**. Son attribut **transport** permet de préciser le protocole de transport à utiliser, généralement "http://schemas.xmlsoap.org/soap/http" pour le protocole http.



Les standards

WSDL(9/9)

Le tag <service> :

- Le tag <definitions> ne peut avoir qu'un seul tag fils <service>.
- Un service possède un nom précisé dans la valeur de son attribut name.
- Un service est composé d'un ou plusieurs ports qui en SOAP correspondent à des endpoints. Chaque port est associé à un binding en utilisant l'attribut binding qui a comme valeur le nom d'un binding défini.
- Les tags fils du tag <port> sont spécifiques au binding utilisé : ce sont des extensions spécifiques qui précisent le endpoint selon le binding. Par exemple pour préciser l'url d'un service web utilisant http, il faut utiliser le tag <address> en fournissant l'url du endpoint comme valeur de l'attribut location. Le tag <endpoint> est utilisé à partir de la version 2.0 de WSDL.
- Un port permet de décrire la façon d'accéder au service, ce qui correspond généralement à l'url d'un endpoint et à un binding.

Travaux Pratiques

1. **TP n° 1** : Développement d'un Service Web, Publication et Invocation en utilisant Java 1.7, AXIS2 1.6.4, Tomcat 7.0 et Eclipse Mars.
2. **TP n° 2** : Utilisation du plugin "TCP/IP Monitor" pour visualiser les messages

Les standards

**Universal Description, Discovery and Integration
(UDDI)**

Les standards

UDDI (1/7)

- UDDI, acronyme de ***Universal Description, Discovery and Integration***, est utilisé pour publier et rechercher des services web. C'est un protocole et un ensemble de services pour utiliser un annuaire afin de stocker les informations concernant les services web et de permettre à un client de les retrouver. Les spécifications sont rédigées par l'Oasis.
- UDDI est une spécification pour permettre la publication et la recherche d'informations sur des entreprises et les services web qu'elles proposent.
- UDDI permet à une entreprise de s'inscrire dans l'annuaire, d'y enregistrer et de publier ses services web. Il est alors possible d'accéder à l'annuaire et de rechercher un service particulier. Le site web officiel d'UDDI est à l'url : ***<http://uddi.xml.org>***
- Un annuaire UDDI contient une description de services web mais aussi des entreprises qui les proposent. Ainsi, il est possible avec UDDI de faire une recherche par entreprise ou par activité. UDDI a été proposé par Microsoft, IBM, Ariba et SAP.
- Trois annuaires majeurs:
 - Le premier est hébergé par Microsoft.
 - Le second par IBM.
 - Le troisième (plus récent) par HP.

Les standards

UDDI (2/7)

- Les données incluses dans un annuaire UDDI sont classées dans trois catégories :
 - les pages blanches (white pages) : elles contiennent les informations générales sur une entreprise
 - les pages jaunes (yellow pages) : elles permettent une catégorisation des entreprises
 - les pages vertes (green pages) : elles contiennent les informations techniques sur les services proposés.
- UDDI n'est pas un élément indispensable à la mise en oeuvre des services web comme peut l'être XML, WSDL ou SOAP.
- UDDI est une spécification pour un annuaire dont l'accès aux fonctionnalités se fait sous la forme de services web de type SOAP pour les recherches et les mises à jour.

Les standards

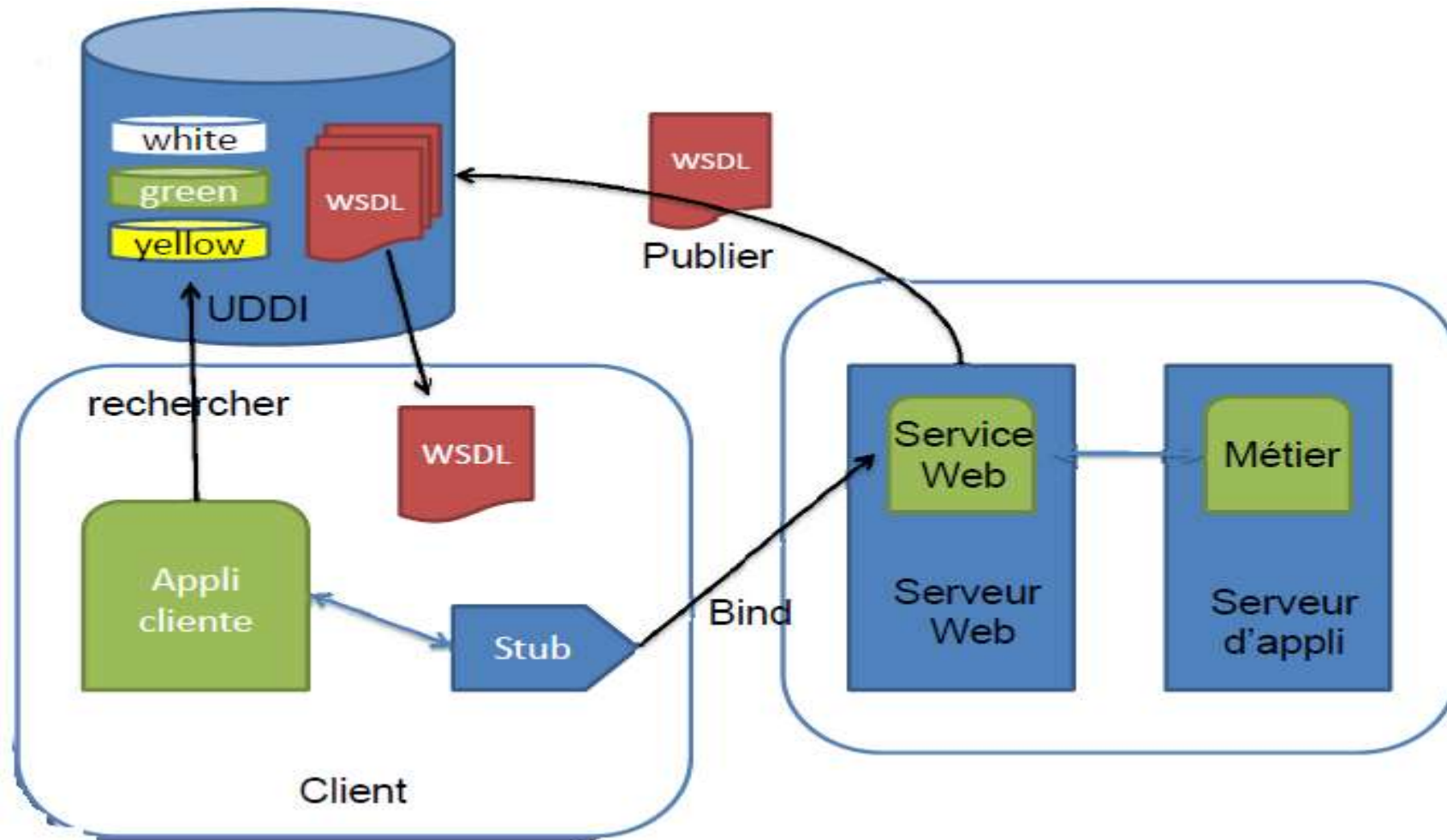
UDDI (3/7)

- Exemples de services web publics :

Lien WSDL	Fonctionnalité
http://www.webservicex.net/CreditCard.asmx?WSDL	Valider le numéro d'une carte de crédit
http://www.webservicex.net/ValidateEmail.asmx?WSDL	Valider une adresse email.
http://ws.cdyne.com/phoneverify/phoneverify.asmx?wsdl	Valider un numéro de téléphone.
http://www.ejse.com/WeatherService/Service.asmx?WSDL	Obtenir la météo en fonction du code postal / nom de la ville

Les standards

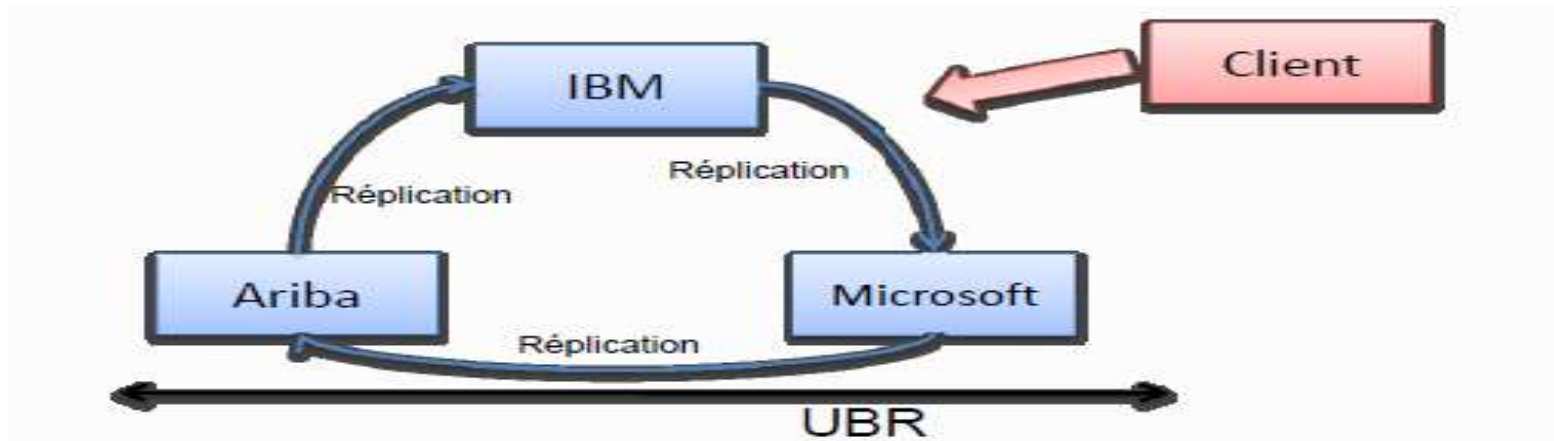
UDDI (4/7)



Les standards

UDDI (5/7)

- De nombreux nœuds dans un UBR (UDDI Business Registry). Donne l'impression d'accéder à un système unique mais en réalité composé d'un ensemble de nœuds. Les données sont synchronisées par réplication toutes les 24h :



- On peut mettre son propre annuaire privé.

Les standards

UDDI (6/7)

Le modèle UDDI comporte 5 structures de données :

- ✓ **BusinessEntity** : Ensemble d'informations sur l'entreprise qui expose le service ;
- ✓ **BusinessService** : Ensemble d'informations sur les services exposés par l'entreprise ;
- ✓ **BindingTemplate** : Ensemble d'informations sur le lieu d'hébergement du service ;
- ✓ **tModel** : Ensemble d'informations sur le mode d'accès au service (WSDL);
- ✓ **publisherAssertion** : Ensemble d'informations contractuelles pour accéder au service.

Les standards

UDDI (7/7)

- Les annuaires UDDI ont pour but de localiser des services Web hébergés dans le monde entier ;
- Différents types de recherches:
 - ✓ Pages blanches (BusinessEntity) : On cherche un service par contact, nom et adresse. Elles comprennent la liste des entreprises ainsi que des informations associées à ces dernières ;
 - ✓ Pages jaunes (BusinessService): On recherche un service par sujet, par domaine. Elle recensent les services Web de chacune des entreprises sous le standard WSDL,
 - ✓ Pages vertes (BindingTemplate): On recherche un service en fonction de ces caractéristiques techniques. Elles fournissent des informations techniques précises sur les services fournis. Ces informations concernent les descriptions de services et d'information de liaison ou encore les processus métiers associés.
- Les opérateurs UDDI vous garantissent la sécurité et l'intégrité des services Web contenus dans un annuaire.

Les différents formats de SW SOAP

(1/5)

- Un message SOAP peut être formaté de plusieurs façons en fonction de son style et de son type d'encodage.
- Il existe deux styles de services web reposant sur SOAP : RCP et Document.
- En plus du style, il existe deux types d'encodages : Encoded et Literal.
- Cela permet de définir quatre combinaisons. De plus, Microsoft est à l'origine d'un cinquième format qui bien que non standardisé est largement utilisé car il est mis en œuvre par défaut dans la plate-forme .Net et il offre un bon compromis entre performance et restrictions d'utilisation :

Style / Type d'encodage	Encoded	Literal
RPC	<u>RPC / Encoded</u>	RPC / Literal
Document	Document / Encoded	<u>Document / Literal</u> Document / Literal wrapped

- Généralement, les combinaisons utilisées sont **RPC/Encoded** et **Document/Literal**. La combinaison Document/Encoded n'est supportée par aucune implémentation.

Les différents formats de SW SOAP

(2/5)

- Le style RPC est parfaitement structuré alors que le type Document n'a pas de structure imposée mais son contenu peut être facilement validé grâce à un schéma XML ou traité puisque c'est un document XML.
- Avec le style document, il est donc possible de structurer librement le corps du message grâce au schéma XML.

Style	Description
RPC	Les messages contiennent le nom de l'opération Paramètres en entrée multiples et valeur de retour
Document	Les messages ne contiennent pas le nom de l'opération Un document XML en entrée et en retour

Les différents formats de WS SOAP

(3/5)

- Les deux désignations pour le style d'encodage (RPC et Document) peuvent être trompeuses car elles peuvent induire que le style RPC est utilisé pour l'invocation d'opérations distantes et que le style document est utilisé pour l'échange de messages. En fait, le style n'a rien à voir avec un modèle de programmation mais il permet de préciser comment le message SOAP est encodé.
- Dans le style RPC, le corps du message (tag <soap:body>) contient un élément qui est le nom de l'opération du service. Cet élément contient un élément fils pour chaque paramètre.
- Dans le style Document, le corps du message (tag <soap:body>) contient directement un document xml dont tous les composants doivent être décrits dans un ou plusieurs schémas XML. Le moteur Soap est alors responsable du mapping entre le contenu du message et les objets du serveur.

Encodage	Description
Encoded	Aussi appelé SOAP encoding car l'encodage est spécifique à SOAP sans utiliser de schéma XML
Literal	L'encodage du message repose sur les schémas XML
Literal wrapped	Idem Literal avec en plus l'encapsulation de chaque message dans un tag qui contient le nom de l'opération. Ce format est défini par Microsoft qui l'utilise dans sa plate-forme .Net

Les différents formats de WS SOAP

(4/5) Le choix du format à utiliser

L'utilisation de document/literal :

- Dans ce mode, le nom de l'opération n'est pas fourni dans le message : le mapping pour déterminer l'opération à invoquer repose donc sur les paramètres.
- Il n'est donc pas possible d'invoquer le service ci-dessous avec le mode document/literal :

Exemple :

```
public MonService {  
    public void maMethode(int x, int y);  
    public void maSecondeMethode(int x, int y);  
}
```

L'utilisation de Document/Literal Wrapped :

- Ce mode n'est pas supporté par tous les moteurs SOAP.
- Il n'est pas possible d'utiliser des opérations surchargées dans un service puisque le mapping de l'opération sur la méthode se fait sur le nom de la méthode. La classe ci-dessous ne peut pas être exposée sous la forme d'un service web invoqué par le mode Document/Literal Wrapped.

Exemple :

```
public MonService {  
    public void maMethode(int x, int y);  
    public void maMethode(int x);  
}
```

- WSDL 2.0 interdit l'utilisation des opérations surchargées.

Les différents formats de WS SOAP

(5/5) Le choix du format à utiliser

L'utilisation de RPC/Literal :

- Comme le mode Document/Literal ne contient pas le nom de l'opération à invoquer, il y a des cas où il faut utiliser le mode Document/Literal Wrapped ou un des deux modes RPC/Encoded ou RPC/Literal.
- L'exemple ci-dessous peut être invoqué avec **RPC/Literal** :

Exemple :

```
public MonService {  
    public void maMethode(int x, int y);  
    public void maMethode(int x);  
    public void maSecondeMethode(int x, int y);  
}
```

L'utilisation de RPC/Encoded :

- Le mode RPC/Encoded n'est pas WS-I Basic Profile compliant mais il est parfois nécessaire de l'utiliser. Ce mode est le seul qui puisse prendre en charge un graphe d'objets contenant plusieurs fois la même référence.

Exemple :

```
<complexType name="MonElement">  
    <sequence>  
        <element name="nom" type="xsd:string"/>  
        <element name="partie1" type="MonElement" xsd:nillable="true"/>  
        <element name="partie2" type="MonElement" xsd:nillable="true"/>  
    </sequence>  
</complexType>
```

Les conseils pour la mise en œuvre

- Avant de développer des services web, il faut valider la solution choisie avec un POC (Proof Of Concept) ou un prototype. Lors de ces tests, il est important de vérifier l'interopérabilité notamment si les services web sont consommés par différentes technologies.
- Le choix du moteur SOAP est aussi très important notamment vis-à-vis du support des spécifications, des performances, de la documentation, ...

Les étapes de mise en œuvre

Etape 1 : Définition des contrats des services métiers

Cette étape est une phase d'analyse qui va définir les fonctionnalités proposées par chaque service pour répondre aux besoins.

Etape 2 : Identification des services web

- Cette étape doit permettre de définir les contrats techniques des services web à partir des services métiers définis dans l'étape précédente.
- Un service métier peut être composé d'un ou plusieurs services web.
- La réalisation de cette étape doit tenir compte de plusieurs contraintes :
 - ✓ Penser forte granularité / faible couplage
 - ✓ Tenir compte de contraintes techniques
 - ✓ Préférer les services web indépendants du contexte client.
- L'invocation d'un service est coûteuse notamment à cause du mapping objet/xml et xml/objet réalisé à chaque appel. Cette règle est vraie pour toutes les invocations de fonctionnalités distantes mais encore plus avec les services web. Il est donc préférable de limiter les invocations de méthodes d'un service web en proposant des fonctionnalités à forte granularité.

Par exemple, il est préférable de définir une opération qui permet d'obtenir les données d'une entité plutôt que de proposer autant d'opérations que l'entité possède de champs. Ceci permet de réduire le nombre d'invocations du service web et réduit le couplage entre la partie front-end et back-end.

Les étapes de mise en œuvre

Etape 2 : identification des services web (suite)

- La définition des services web doit tenir compte de contraintes techniques liées aux performances ou à la consommation de ressources. Par exemple, si le temps de traitement d'un service web est long, il faudra prévoir son invocation de façon asynchrone ou si les données retournées sont des binaires de tailles importantes, il faudra envisager d'utiliser le mécanisme de pièces jointes (attachment).
- Il est préférable de définir des services web qui soient stateless (ne reposant pas par exemple sur une utilisation de la session http). Ceci permet de déployer les services web dans un cluster où la réplication de session sera inutile.

Etape 3 : écriture des services web

- Cette étape est celle du codage proprement dit des services web.
- Deux approches sont possibles :
 - ✓ écriture du WSDL en premier (**contract first**) : des outils du moteur Soap sont utilisés pour gérer le code des services web à partir du WSDL. Face à la complexité de la rédaction du WSDL, cette approche n'est pas toujours privilégiée.
 - ✓ écriture de la classe et génération du WSDL (**code first**) : chaque service est implémenté sous la forme d'une ou plusieurs classes et c'est le moteur Soap utilisé qui va générer le WSDL correspondant en se basant sur la description de la classe et des métadonnées.

Les étapes de mise en œuvre

Etape 4 : déploiement et tests

- Les services web doivent être packagés et déployés généralement dans un serveur d'applications ou un conteneur web.
- Pour tester les services web, il est possible d'utiliser des outils fournis par l'IDE ou d'utiliser des outils tiers comme SoapUI qui propose de très nombreuses fonctionnalités pour les tests des services web allant de la simple invocation à l'invocation de scénarios complexes et de tests de charges.

Etape 5 : consommation des services web par les applications clientes

- Il faut mettre en œuvre les outils du moteur Soap utilisé par l'application cliente pour générer les classes nécessaires à l'invocation des services web et utiliser ces classes dans l'application. C'est généralement le moment de faire quelques adaptations pour permettre une bonne communication entre le client et le serveur.

Les étapes de mise en œuvre

Quelques recommandations :

- Il ne faut pas se lier à un langage de programmation :
 - ✓ N'utiliser que des types communs : int, float, String, Date, ...
 - ✓ Ne pas utiliser de types spécifiques : Object, DataSet, ...
 - ✓ Eviter la composition d'objets
 - ✓ Utiliser un tableau ou des collections typées avec un generic plutôt qu'une collection non typée.
- Il faut éviter la surcharge des méthodes.
- Il faut éviter de transformer une classe en service web (notamment en utilisant des annotations) : il est recommandé de définir une interface qui va établir le contrat entre le service et son implémentation. Cette pratique venant de la POO doit aussi s'appliquer pour les services web.

Les étapes de mise en œuvre

Les problèmes liés à SOAP

- SOAP est assez complexe et sa mise en œuvre dépend de l'implémentation de la technologie utilisée côté consommateur et fournisseur de services web. Il en résulte des problèmes d'interopérabilités alors qu'un des buts de SOAP est pourtant de s'en affranchir.
- Il existe plusieurs types de problèmes :

Les problèmes liés aux versions de SOAP

- ✓ Les versions SOAP 1.1 et 1.2 étant incompatibles, cela peut entraîner des problèmes de compatibilité si les implémentations des moteurs SOAP utilisés supportent des versions différentes.
- ✓ Ceci est notamment le cas si l'implémentation du moteur SOAP est assez ancienne.

Les problèmes liés aux modèles de messages

- ✓ Un message Soap peut être encodé selon plusieurs modèles : le modèle le plus ancien (RPC) est abandonné au profit du modèle Document. Cela peut introduire des problèmes d'incompatibilité notamment entre des services web existants et des consommateurs plus récents ou vice-versa.

Les implémentations des WS

- Il existe de nombreuses implémentations possibles de moteurs SOAP permettant la mise en œuvre de services web avec Java, notamment plusieurs solutions open source :
 - ✓ Intégrées à la plate-forme Java EE 5.0 et Java SE 6.0
 - ✓ JWSDP de Sun
 - ✓ Axis et Axis 2 (Apache eXtensible Interaction System) du projet Apache
 - ✓ XFire
 - ✓ CXF du projet Apache
 - ✓ JBoss WS
 - ✓ Metro du projet GlassFish
 - ✓ ...
- A cause d'un effort de spécification tardif de JAX-WS, plusieurs implémentations utilisent une approche spécifique pour la mise en œuvre et le déploiement de services web, ce qui rend le choix d'une de ces solutions délicat. Heureusement, toutes tendent à proposer un support de JAX-WS.
- Même si les concepts sous-jacents sont équivalents, quelle que soit l'implémentation utilisée, sa mise en œuvre est très différente d'une implémentation à l'autre.
- De plus, la plupart des solutions historiques sont relativement complexes à mettre en œuvre car certains points techniques ne sont pas assez masqués par les outils (code à écrire, fichiers de configuration, descripteurs de déploiement, ...). Avec ces solutions, le développeur doit consacrer une part non négligeable de son temps à du code technique pour développer le service web.
- JAX-WS propose une solution pour simplifier grandement le développement des services grâce à l'utilisation d'annotations qui évitent d'avoir à écrire du code ou des fichiers pour la plomberie. JAX-WS, en tant que spécification, est implémentée dans plusieurs solutions.

Les implémentations des WS

AXIS 1.x (1/4)

- Axis (Apache eXtensible Interaction System) est un projet open-source du groupe Apache diffusé sous la licence Apache 2.0 qui propose une implémentation d'un moteur de service web implémentant le protocole SOAP : il permet de créer, déployer et consommer des services web.
- Son but est de proposer un ensemble d'outils pour faciliter le développement, le déploiement et l'utilisation des services web écrits en java. Axis propose de simplifier au maximum les tâches pour la création et l'utilisation des services web. Il permet notamment de générer automatiquement le fichier WSDL à partir d'une classe Java et le code nécessaire à l'appel du service web.
- Le site officiel est à l'url ***<http://ws.apache.org/axis/>***
- Pour son utilisation, Axis 1.0 nécessite un J.D.K. 1.3 minimum et un conteneur de servlets.
- C'est un projet open source d'implémentation du protocole SOAP. Il est historiquement issu du projet Apache SOAP.
- C'est un outil populaire qui de fait est la référence des moteurs de services web Open Source implémentant JAX-RPC en Java : son utilisation est répandue notamment dans des produits open source ou commerciaux.
- La version la plus récente est la 1.4, diffusée en avril 2006.
- Attention : Axis 1.x n'est plus supporté au profit de Axis 2 qui possède lui aussi des numéros de versions 1.x.

Les implémentations des WS

AXIS 1.x (2/4)

- Axis implémente plusieurs spécifications :
 - JSR 101 : Java API for XML-Based RPC (JAX-RPC) 1.1
 - JSR 67 : SOAP with Attachments API for Java Specification (SAAJ) 1.2
 - Java API for XML Registries Specification (JAXR) 1.0
- Axis permet donc la mise en œuvre de :
 - SOAP 1.1 et 1.2
 - WSDL 1.1
 - XML-RPC
 - WS-I Basic Profile 1.1
- Attention : Axis 1.0 n'est pas compatible avec
 - JSR 109 Web Services for EE (WS4EE) 1.0
 - JSR 224 Java API for XML-Based Web Services (JAX-WS) 2.0
 - JSR 181 Web Service Metadata for the Java Platform
 - JSR 222 Java Architecture for XML Binding (JAXB) 2.0

Les implémentations des WS

AXIS 1.x (3/4)

- Axis génère le document wsdl du service web : pour accéder à ce document il suffit d'ajouter **?wsdl** à l'url d'appel du service web.
- L'interopérabilité entre Axis et .Net 1.x est assurée tant que les types utilisés se limitent aux primitives, aux chaînes de caractères, aux tableaux des types précédents et aux Java Beans composés uniquement des types précédents ou d'autres Java Beans.
- L'interopérabilité entre Axis 1.4 et .Net 2.0 est bien meilleure. Par exemple, la gestion des objets Nullable dans .Net 2.0 est prise en compte (notamment pour les dates et types primitifs) : il n'est donc plus nécessaire d'utiliser une gestion particulière pour ces objets.

Les implémentations des WS

AXIS 2.0 (4/4)

- Axis 2 est le successeur du projet Axis : le projet a été complètement réécrit pour proposer une architecture plus modulaire.
- Il propose un modèle de déploiement spécifique : les services web peuvent être packagés dans un fichier ayant l'extension **.aar** (**Axis ARchive**) ou contenus dans un sous-répertoire du répertoire WEB-INF/services. La configuration se fait dans le fichier META-INF/services.xml
- Le runtime d'Axis 2 est une application web qui peut être utilisée dans n'importe quel serveur d'applications Java EE et même un conteneur web comme Apache Tomcat.
- Des modules complémentaires permettent d'enrichir le moteur en fonctionnalités notamment le support de certaines spécifications WS-*. Chaque module est packagé dans un fichier avec l'extension .mar
- Axis 2 permet de choisir le framework de binding XML/Objets.

Les implémentations des WS AXIS 2.0 (4/4)

Exemples complets :

Voir TP n°1, 2 et 3

Les API Java pour les services web

Java propose un ensemble d'API permettant la mise en œuvre des services web :

API	Rôle
JAXP	API pour le traitement de documents XML : analyse en utilisant SAX ou DOM et transformation en utilisant XSLT.
JAX-RPC	API pour le développement de services web utilisant SOAP avec le style RPC
JAXM	API pour le développement de services utilisant des messages XML orientés documents
JAXR	API pour permettre un accès aux annuaires de référencement de services web
JAXB	API et outils pour automatiser le mapping d'un document XML avec des objets Java
StAX	API pour le traitement de documents XML
SAAJ	API pour permettre la mise en œuvre des spécifications SOAP with Attachment
JAX-WS	API pour le développement grâce à des annotations de services web utilisant SOAP avec le style Document



JAX-WS 2.0

Java API for XML based Web Services



JAX-WS (1/30)

- JAX-WS (Java API for XML based Web Services) est une nouvelle API, mieux architecturée, qui remplace l'API JAX-RPC 1.1 mais n'est pas compatible avec elle. Il est fortement recommandé d'utiliser le modèle de programmation proposé par JAX-WS notamment pour les nouveaux développements.
- Elle propose un modèle de programmation pour produire (côté serveur) ou consommer (côté client) des services web qui communiquent via des messages XML de type SOAP.
- Elle a pour but de faciliter et simplifier le développement des services web notamment grâce à l'utilisation des annotations. JAX-WS fournit les spécifications pour le cœur du support des services web de la plateforme Java SE et Java EE.
- JAX-WS a été spécifié par la JSR 224 : Java API for XML-Based Web Services (JAX-WS) 2.0.
- JAX-WS permet la mise en œuvre de plusieurs spécifications :
 - ✓ JAX-WS respecte le standard WS-I Basic Profile version 1.1.
 - ✓ JAX-WS propose un support pour SOAP 1.1 et 1.2
 - ✓ JAX-WS permet le développement de services web orientés RPC (literal) ou orientés documents (literal/encoded/literal wrapped)
- JAX-WS repose sur plusieurs autres JSR :
 - ✓ JSR 181 (Web Services MetaData for the Java Platform) : propose un ensemble d'annotations qui permettent de définir les services web
 - ✓ JSR 109 et JSR 921 (Implementing Enterprise Web Services) : décrit comment déployer, gérer et accéder aux services web via un serveur d'applications
 - ✓ JSR 183 (Web Services Message Security APIs) : décrit la sécurisation des messages SOAP

JAX-WS (2/30)

- Le fournisseur de l'implémentation de JAX-WS utilise les spécifications de la JSR 921 pour générer les fichiers de configuration et de déploiement à partir des annotations et d'éventuelles métadonnées.
- JAX-WS utilise JAXB 2.0 et SAAJ 1.3.
 - JAXB propose une API et des outils pour automatiser le mapping d'un document XML et des objets Java. A partir d'une description du document XML (Schéma XML ou DTD), des classes sont générées pour effectuer automatiquement l'analyse du document XML et le mapping de ce dernier dans des objets Java.
- JAX-WS peut être combiné avec d'autres spécifications comme les EJB 3 par exemple.

JAX-WS (3/30)

La mise en œuvre de JAX-WS

- JAX-WS est une spécification : pour la mettre en œuvre, il faut utiliser une implémentation.
- L'implémentation de référence de JAX-WS est le projet Metro développé par la communauté du projet GlassFish. Il existe d'autres implémentations notamment Axis 2 qui propose son propre modèle de programmation mais aussi un support de JAX-WS.
- Le développement d'un service web en Java avec JAX-WS débute par la création d'une classe annotée avec **@WebService** du package javax.jws. La classe ainsi annotée définit le endpoint du service web.
- Le service endpoint interface (SEI) est une interface qui décrit les méthodes du service : celles-ci correspondent aux opérations invocables par un client.
- Il est possible de préciser explicitement le SEI en utilisant l'attribut endpointInterface de l'annotation **@WebService**

JAX-WS (4/30)

La production de service web avec JAX-WS

Le développement d'un service web avec JAX-WS requiert plusieurs étapes :

- Coder la classe qui encapsule le service
- Compiler la classe
- Utiliser la commande **wsgen** pour générer les fichiers requis pour le déploiement (schémas, WSDL, classes, ...)
- Packager le service dans un fichier.war
- Déployer le war dans un conteneur

Pour définir un endpoint avec JAX-WS, il a plusieurs contraintes :

- La classe qui encapsule le endpoint doit être public, non static, non final, non abstract et être annotée avec `@WebService`
- Elle doit avoir un constructeur par défaut (sans paramètre)
- Il est recommandé de définir explicitement l'interface du SEI
- Les méthodes exposées par le service web doivent être public, non static, non final et être annotées avec `@WebMethod`
- Les types des paramètres et de la valeur de retour de ces méthodes doivent être supportés par JAXB

Exemple :

```
import javax.xml.ws.WebService;  
import javax.xml.ws.WebMethod;  
@WebService  
public class MonService  
{  
    @WebMethod  
    public  
    String saluer()  
    {  
        return "Bonjour";  
    }  
}
```

JAX-WS (5/30)

Les annotations utiles

- JAX-WS repose **la JSR 181** (Web Services MetaData for the Java Platform) qui propose un ensemble d'annotations qui permettent de définir les services web.
- Les annotations sont utilisées dans la classe d'implémentation ou dans l'interface d'un service web.
- Toutes les annotations de la JSR 181 sont définies dans le package ***javax.jws***.
- Ces annotations sont exploitées au runtime.
- **Attention** : plusieurs implémentations fournissent, en plus des annotations de la JSR 181, des annotations qui leur sont propres. Même si elles sont pratiques, elles limitent la portabilité des services web à s'exécuter dans un autre moteur Soap (exemple : ***@EnableMTOM***, ***@ServiceProperty***, ***@ServicesProperties*** dans XFire).

JAX-WS (6/30)

@WebService

- L'annotation `javax.ws.WebService` permet de définir une classe ou une interface comme étant l'interface du endpoint d'un service web.
- L'annotation `WebService` est la seule annotation obligatoire pour développer un service web.
- Si l'annotation est utilisée sur l'interface du service web (SEI), il faut aussi l'utiliser sur la classe d'implémentation en précisant l'interface avec l'attribut `endpointInterface`.
- Cette annotation s'utilise sur une classe ou une interface uniquement.

Attribut	Rôle
String name	Le nom du service web utilisé dans l'attribut name de l'élément <code>wsdl:portType</code> du WSDL Par défaut, c'est le nom non qualifié de la classe
String targetNamespace	Espace de nommage utilisé dans le WSDL Par défaut c'est le nom du package
String serviceName	Le nom du service utilisé dans l'attribut name de l'élément <code>wsdl:service</code> du WSDL Par défaut, c'est le nom de la classe suffixée par "Service"
String wsdlLocation	URL relative ou absolue du WSDL prédéfini
String endpointInterface	Nom pleinement qualifié de l'interface du endpoint (SEI), ce qui permet de séparer l'interface de l'implémentation
String portName	Nom du port du service web utilisé dans l'attribut name de l'élément <code>wsdl:port</code> du WSDL

Exemple :

```
@WebService(name = "BonjourWS", targetNamespace = "http://jmdoudoux.developpez.com/ws/Bonjour")
public class BonjourServiceImpl {
    @WebMethod
    public String saluer() {
        return "Bonjour";
    }
}
```

JAX-WS (7/30)

@WebMethod

- L'annotation ***javax.ws.WebMethod*** permet de définir une méthode comme étant une opération d'un service web.
- Cette annotation s'utilise sur une méthode uniquement. La méthode sur laquelle cette annotation est appliquée doit être public.

Attribut	Rôle
String operationName	nom utilisé dans l'élément wsdl:operation du message Par défaut: le nom de la méthode
String action	action associée à l'opération : utilisé comme valeur du paramètre SOAPAction
boolean exclude	booléen qui précise si la méthode doit être exposée ou non dans le service web. Cette propriété n'est utilisable que dans une classe et doit être le seul attribut de l'annotation. Par défaut : false

Exemple :

```
@WebService(name = "BonjourWS", targetNamespace = "http://jmdoudoux.developpez.com/ws/Bonjour")
public class BonjourServiceImpl {
    @WebMethod
    public String saluer() {
        return "Bonjour";
    }
}
```

JAX-WS (8/30)

@OneWay

- L'annotation ***javax.ws.OneWay*** permet de définir une méthode comme étant une opération d'un service web **qui ne fournit pas de réponse** lors de son invocation. Elle permet une optimisation à l'exécution qui évite d'attendre une réponse qui ne sera pas fournie.
- Cette annotation s'utilise sur une méthode uniquement : celle-ci ne doit pas avoir de valeur de retour ou lever une exception puisque dans ce cas, il y a une réponse de type SoapFault.
- Cette annotation ne possède aucun attribut.

```
Exemple :  
@WebService  
public class MonService {  
  
    @WebMethod  
    @Oneway  
    public void MonOperation() {  
    }  
}
```


JAX-WS (9/30)

@WebParam

- L'annotation ***javax.ws.WebParam*** permet de configurer comment un paramètre d'une opération sera mappé dans le message SOAP.
- Cette annotation s'utilise uniquement sur un paramètre d'une méthode de l'implémentation du service.
- Cette annotation est pratique pour permettre d'utiliser le même paramètre dans plusieurs opérations d'un service web encodé en document literal.

Attribut	Rôle
String name	nom du paramètre utilisé dans le WSDL Par défaut: le nom du paramètre
Mode mode	mode d'utilisation du paramètre. Le type Mode est une énumération qui contient IN, OUT et INOUT Par défaut : IN
String targetNamespace	précise l'espace de nommage du paramètre dans les messages utilisant le mode document Par défaut : l'espace de nommage du service web
boolean header	booléen qui indique si la valeur du paramètre est contenue dans l'en-tête de la requête http plutôt que dans le corps Par défaut : false
String partName	Définit l'attribut name de l'élément <wsdl:part> des messages de type RPC et DOCUMENT/BARE

JAX-WS (10/30)

@WebResult

- L'annotation ***javax.ws.WebResult*** permet de configurer comment une valeur de retour d'une opération sera mappée dans l'élément ***wsdl:part*** message SOAP.
- Cette annotation s'utilise sur une méthode uniquement.
- Cette annotation est pratique pour permettre d'utiliser la même valeur de retour dans plusieurs opérations d'un service web encodé en Document Literal.

Attribut	Rôle
String name	nom de la valeur de retour utilisé dans le WSDL. Avec le style RPC, c'est l'attribut name de l'élément wsdl:part. Avec le style DOCUMENT, c'est le nom de l'élément dans la réponse Par défaut: return pour RPC et DOCUMENT/WAPPED et le nom de la méthode suffixé par "Response" pour DOCUMENT/BARE
String targetName space	espace de nommage de la valeur de retour dans les messages utilisant le mode document Par défaut : l'espace de nommage du service web
boolean header	booléen qui indique si la valeur de retour est stockée dans l'en-tête de la requête http plutôt que du corps Par défaut : false
String partName	attribut name de l'élément wsdl:part des messages de type RPC et DOCUMENT/BARE Par défaut : la valeur de l'attribut name

JAX-WS (11/30)

@SOAPBinding

- L'annotation ***javax.jws.soap.SOAPBinding*** permet de déterminer l'encodage du message et de la réponse SOAP.
- Cette annotation s'utilise sur une classe, une interface ou une méthode.

Attribut	Rôle
Style style	Définir le style d'encodage du message. Style est une énumération qui contient DOCUMENT et RPC. Par défaut: DOCUMENT
Use use	Définir le format du message. Use est une énumération qui contient ENCODED et LITERAL. Par défaut: LITERAL
ParameterStyle parameterStyle	Définir si les paramètres forment le contenu du message ou s'ils sont encapsulés par un tag du nom de l'opération à invoquer. ParameterStyle est une énumération qui contient BARE et WRAPPED. BARE ne peut être utilisé qu'avec le style DOCUMENT Par défaut: WRAPPED

Exemple :

```
@WebService
@SOAPBinding(style=Style.DOCUMENT, use=Use.LITERAL, parameterStyle=ParameterStyle.BARE)
public class MonService {

    @WebMethod

    public void MonOperation() {
    }
}
```

JAX-WS (12/30)

Les annotations propre à JAX-WS 2.0

- La JSR 224 définit les annotations spécifiques à JAX-WS 2.0.
- Toutes ces annotations sont dans le package javax.xml.ws.

@javax.xml.ws.BindingType

- L'annotation **@BindingType** permet de préciser le binding qui sera utilisé pour invoquer le service.
- Elle s'utilise sur une classe

Attribut	Rôle
value	Identifiant du binding à utiliser. Les valeurs possibles sont : <ul style="list-style-type: none">- SOAPBinding.SOAP11HTTP_BINDING,- SOAPBinding.SOAP12HTTP_BINDING ou- HTTPBinding.http_BINDING La valeur par défaut est SOAP11_HTTP_BINDING

JAX-WS (13/30)

Les annotations propre à JAX-WS 2.0

@javax.xml.ws.RequestWrapper

- L'annotation @RequestWrapper permet de préciser la classe JAXB de binding qui sera utilisée dans la requête à l'invocation du service. Elle s'utilise sur une méthode

Attribut	Rôle
String className	Préciser le nom pleinement qualifié de la classe qui encapsule la requête (Obligatoire)
String localName	Définir le nom de l'élément dans le schéma qui encapsule la requête. Par défaut, c'est la valeur de l'attribut operationName de l'annotation WebMethod
String targetNamespace	l'espace de nommage. Par défaut, c'est le targetNamespace du SEI

JAX-WS (14/30)

Les annotations propre à JAX-WS 2.0

@javax.xml.ws.ResponseWrapper

- L'annotation @L'annotation @ResponseWrapper permet de préciser la classe JAXB de binding qui sera utilisée dans la réponse à l'invocation du service.
- Elle s'utilise sur une méthode

Attribut	Rôle
String localName	Définir le nom de l'élément dans le schéma qui encapsule la réponse. Par défaut c'est le nom de l'opération défini par l'annotation @WebMethod concaténé à Response
String targetNamespace	Définir l'espace de nommage. Par défaut, c'est le targetNamespace du SEI
String ClassName	Préciser le nom pleinement qualifié de la classe qui encapsule la réponse (Obligatoire)

JAX-WS (15/30)

Les annotations propre à JAX-WS 2.0

@javax.xml.ws.ServiceMode

- Cette annotation permet de préciser si le provider va avoir accès uniquement au payload du message (PAYLOAD) ou à l'intégralité du message (MESSAGE).
- Elle s'utilise sur une classe qui doit obligatoirement implémenter un Provider.

Attribut	Rôle
Service.Mode value	Indiquer si le provider va avoir accès uniquement au payload du message (PAYLOAD) ou à l'intégralité du message (MESSAGE). La valeur par défaut est PAYLOAD

Exemple :

```
@ServiceMode(value=Service.Mode.PAYLOAD)
public class MonOperationProvider implements Provider<Source> {
    public Source invoke(Source source)
        throws WebServiceException {
        Source source = null;
        try {

            // code du traitement de la requete et generation de la reponse

        } catch (Exception e) {
            throw new WebServiceException("Erreur durant les traitements du Provider", e);
        }
        return source;
    }
}
```

JAX-WS (16/30)

Les annotations propre à JAX-WS 2.0

@javax.xml.ws.WebFault

- Cette annotation s'utilise sur une classe qui encapsule une exception afin de personnaliser certains éléments de la partie Fault du message Soap.
- Elle s'utilise sur une exception levée par une opération.

Attribut	Rôle
String name	Le nom de l'élément fault Cet attribut est obligatoire
String targetNamespace	Définir l'espace de nommage pour l'élément fault.
String faultBean	Nom pleinement qualifié de la classe qui encapsule l'exception

JAX-WS (17/30)

Les annotations propre à JAX-WS 2.0

@javax.xml.ws.WebEndPoint

- Cette annotation permet de préciser le portName d'une méthode du SEI.
- Elle s'utilise sur une méthode.

Attribut	Rôle
String name	Définir le nom qui va identifier de façon unique l'élément <wsdl:port> du tag <wsdl:service>

JAX-WS (18/30)

Les annotations propre à JAX-WS 2.0

@javax.xml.ws.WebServiceProvider

- Cette annotation est à utiliser sur une implémentation d'un Provider
- Elle s'utilise sur des classes qui héritent de la classe Provider.

Attribut	Rôle
String portName	nom du port du service (élément <wsdl:portName>)
String serviceName	nom du service (élément <wsdl:service>)
String targetNamespace	espace de nommage
String wsdlLocation	chemin du WSDL du service

Exemple :

```
@WebServiceProvider
public class MonOperationProvider implements Provider<Source> {
    public Source invoke(Source source)
        throws WebServiceException {
        Source source = null;
        try {

            // code du traitement de la requete et generation de la reponse

        } catch (Exception e) {
            throw new WebServiceException("Erreur durant les traitements du Provider", e);
        }
        return source;
    }
}
```

JAX-WS (19/30)

Les annotations propre à JAX-WS 2.0

@javax.xml.ws.WebServiceRef

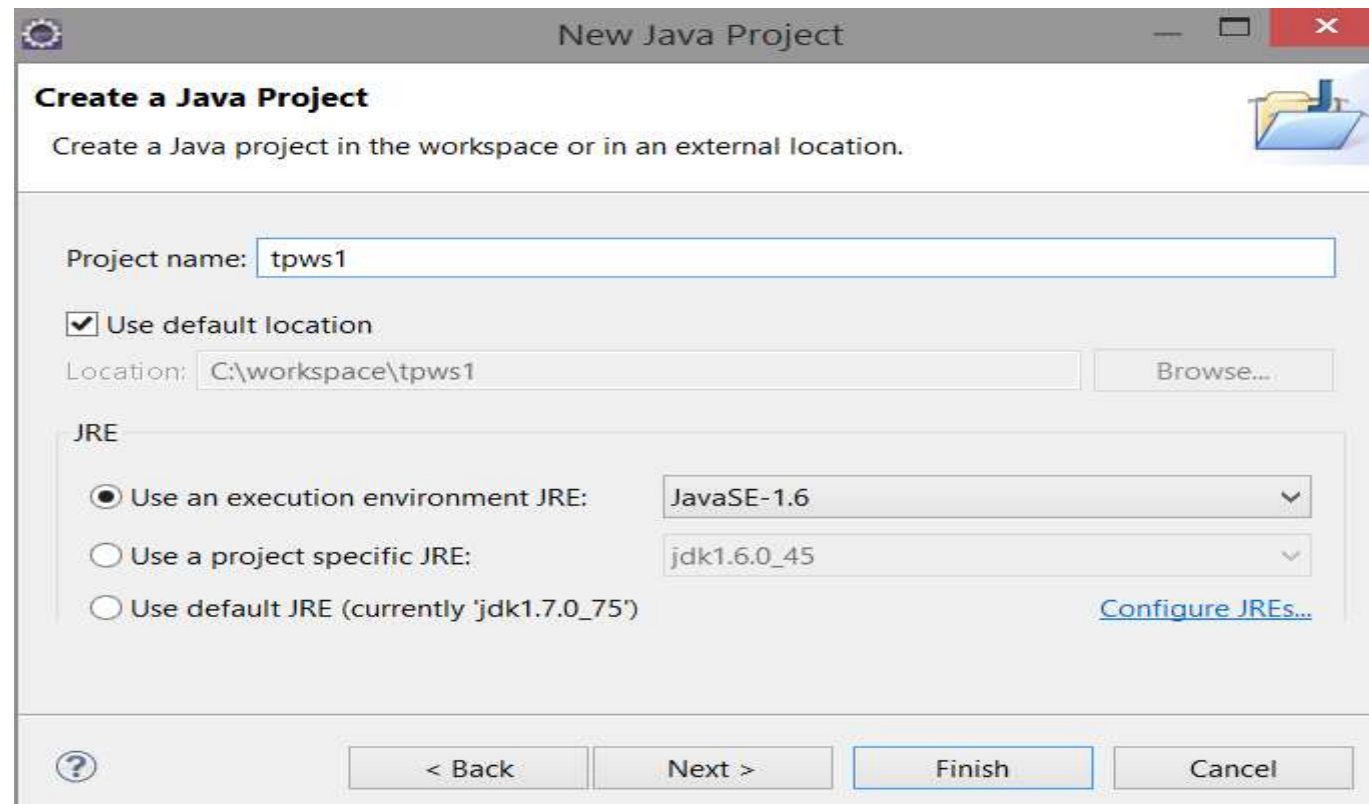
- L'annotation WebServiceRef permet de définir une référence sur un service web et éventuellement autorise son injection.
- Cette annotation est à utiliser dans un contexte Java EE.
- Elle s'utilise sur une classe, une méthode (getter ou setter) ou un champ.

Attribut	Rôle
String name	nom JNDI de la ressource. Par défaut sur un champ c'est le nom du champ. Par défaut sur un getter ou un setter, c'est le nom de la propriété
Class type	type de la ressource. Par défaut sur un champ, c'est le type du champ. Par défaut sur un getter ou un setter, c'est le type de la propriété
String mappedName	nom spécifique au conteneur sur lequel le service est mappé (non portable)
Class value	classe du service qui doit étendre javax.xml.ws.Service
String wsdlLocation	chemin du WSDL du service

JAX-WS (20/30)

Exemple complet

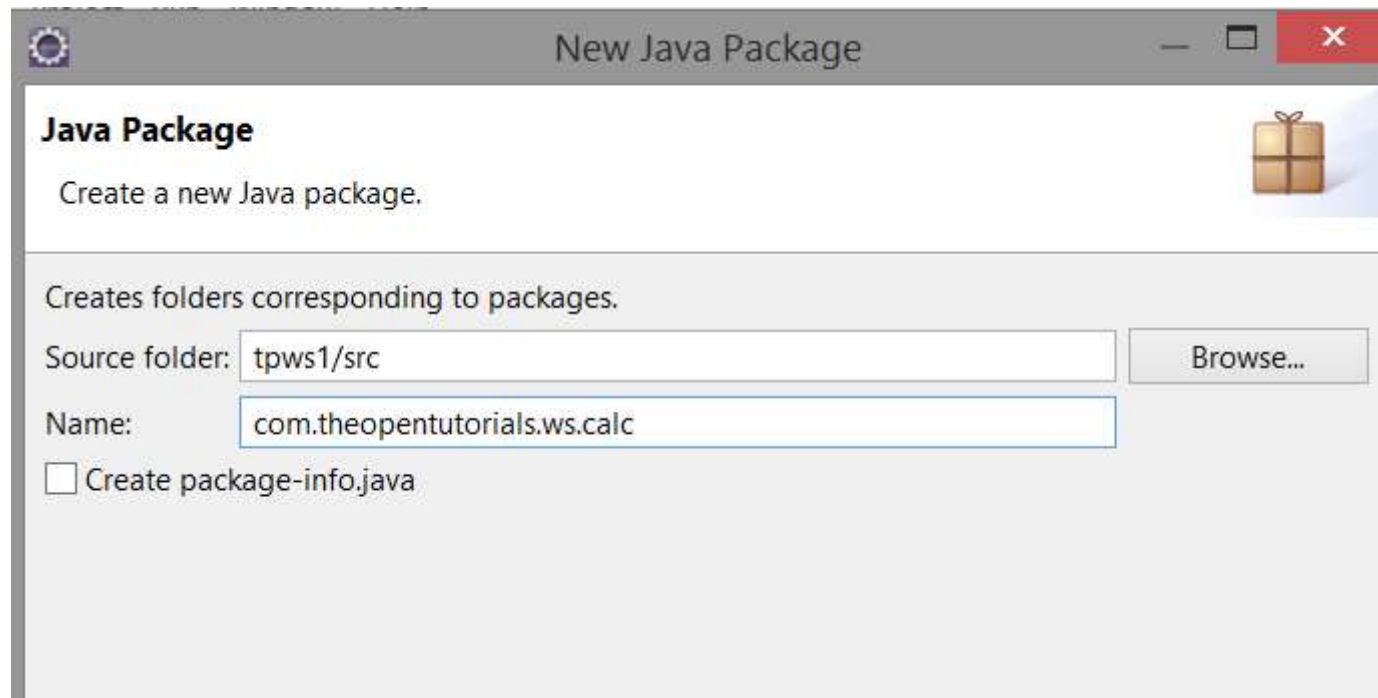
1°) Création d'un projet java avec Eclipse :



JAX-WS (21/30)

Exemple complet

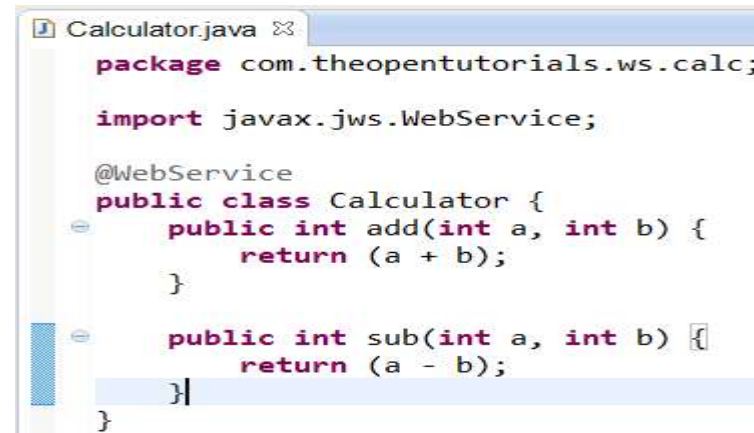
2°) Création du package *com.theopentutorials.ws.calc* :



JAX-WS (22/30)

Exemple complet

2°) Création de la classe *Calculator* :



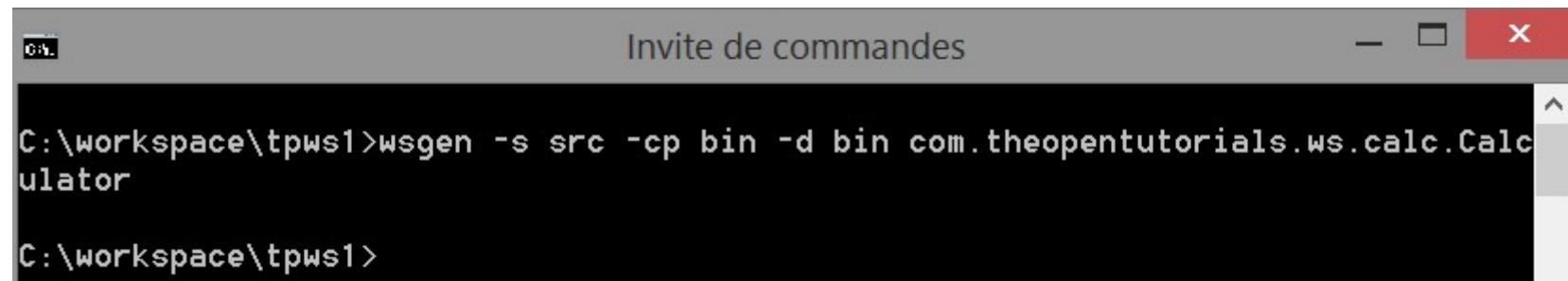
```
Calculator.java
package com.theopentutorials.ws.calc;

import javax.jws.WebService;

@WebService
public class Calculator {
    public int add(int a, int b) {
        return (a + b);
    }

    public int sub(int a, int b) {
        return (a - b);
    }
}
```

3°) Lancer la commande *wsgen -s src -cp bin -d bin com.theopentutorials.ws.calc.Calculator* :



```
Invite de commandes

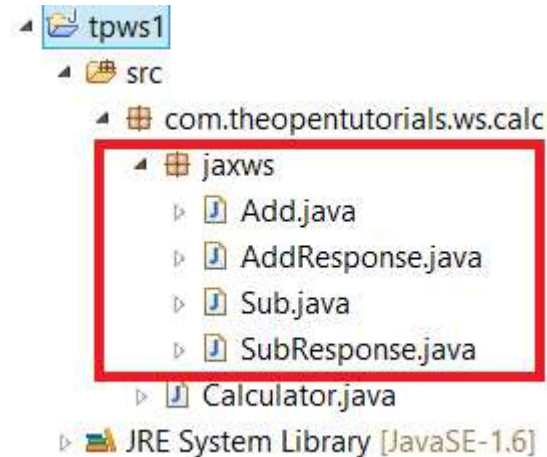
C:\workspace\tpws1>wsgen -s src -cp bin -d bin com.theopentutorials.ws.calc.Calculator

C:\workspace\tpws1>
```

JAX-WS (23/30)

Exemple complet

- La commande **wsgen** exécutée permet de générer les classes nécessaires pour le lancement du WS :

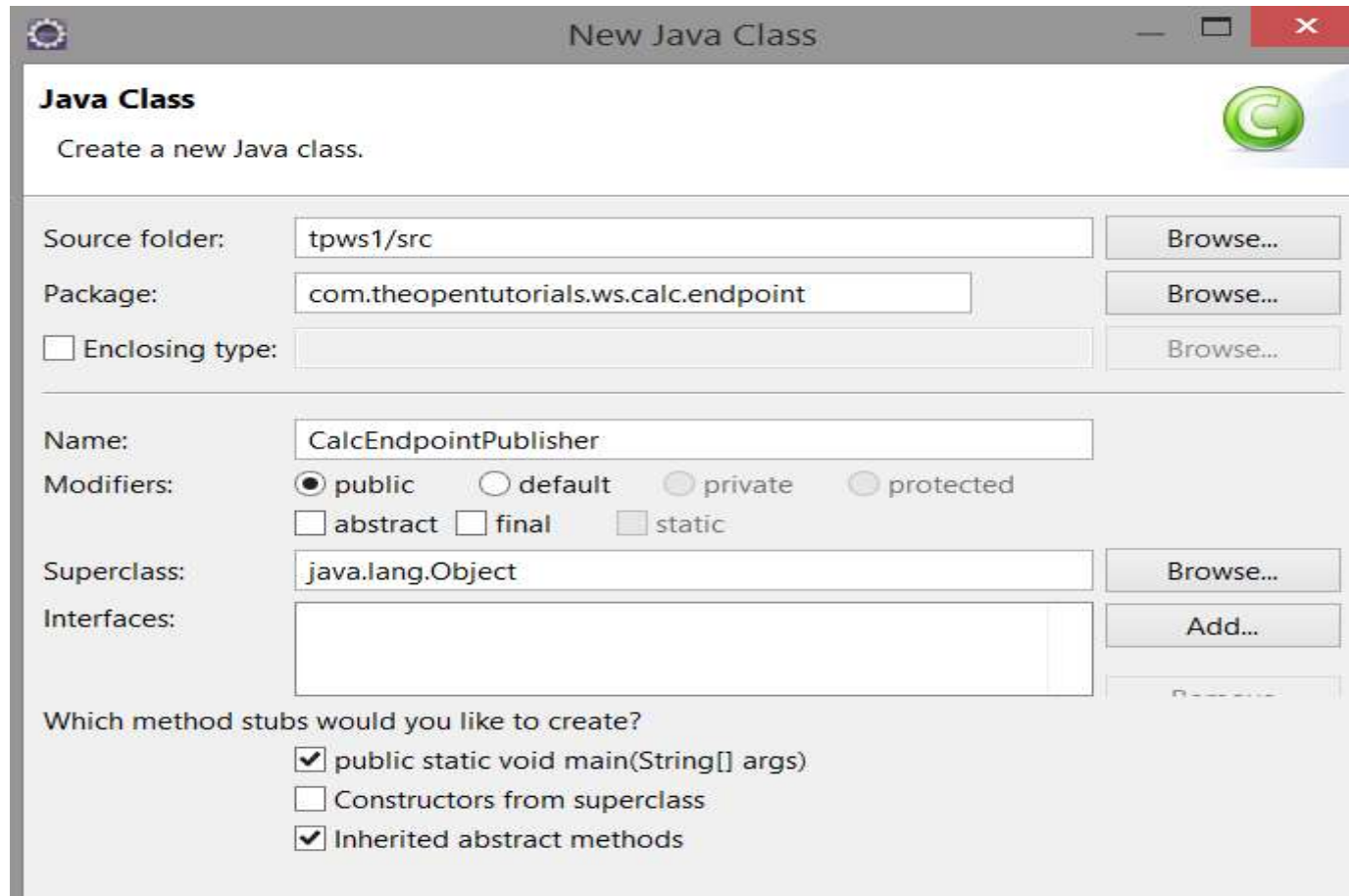


- Maintenant nous avons besoin de publier notre classe comme un **endpoint** de service Web. Pour cela, utiliser la méthode statique **publish()** de la classe **javax.xml.ws.Endpoint** de publier notre classe «Calculator» en tant que service web.

JAX-WS (24/30)

Exemple complet

- Créer la classe ***CalcEndpointPublisher*** :



New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☒ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

JAX-WS (25/30)

Exemple complet

- Le code de la classe **CalcEndpointPublisher** :

```
CalcEndpointPublisher.java
package com.theopentutorials.ws.calc.endpoint;

import javax.xml.ws.Endpoint;

import com.theopentutorials.ws.calc.Calculator;

public class CalcEndpointPublisher {

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/CalcWS/Calculator", new Calculator());
    }
}
```

- Exécuter la méthode **main**. Pour vérifier si le WS a été bien publié, vérifier le lien suivant : ***http://localhost:8080/CalcWS/Calculator?wsdl***

JAX-WS (26/30)

Exemple complet

← → ↺ 📄 localhost:8080/CalcWS/Calculator?wsdl

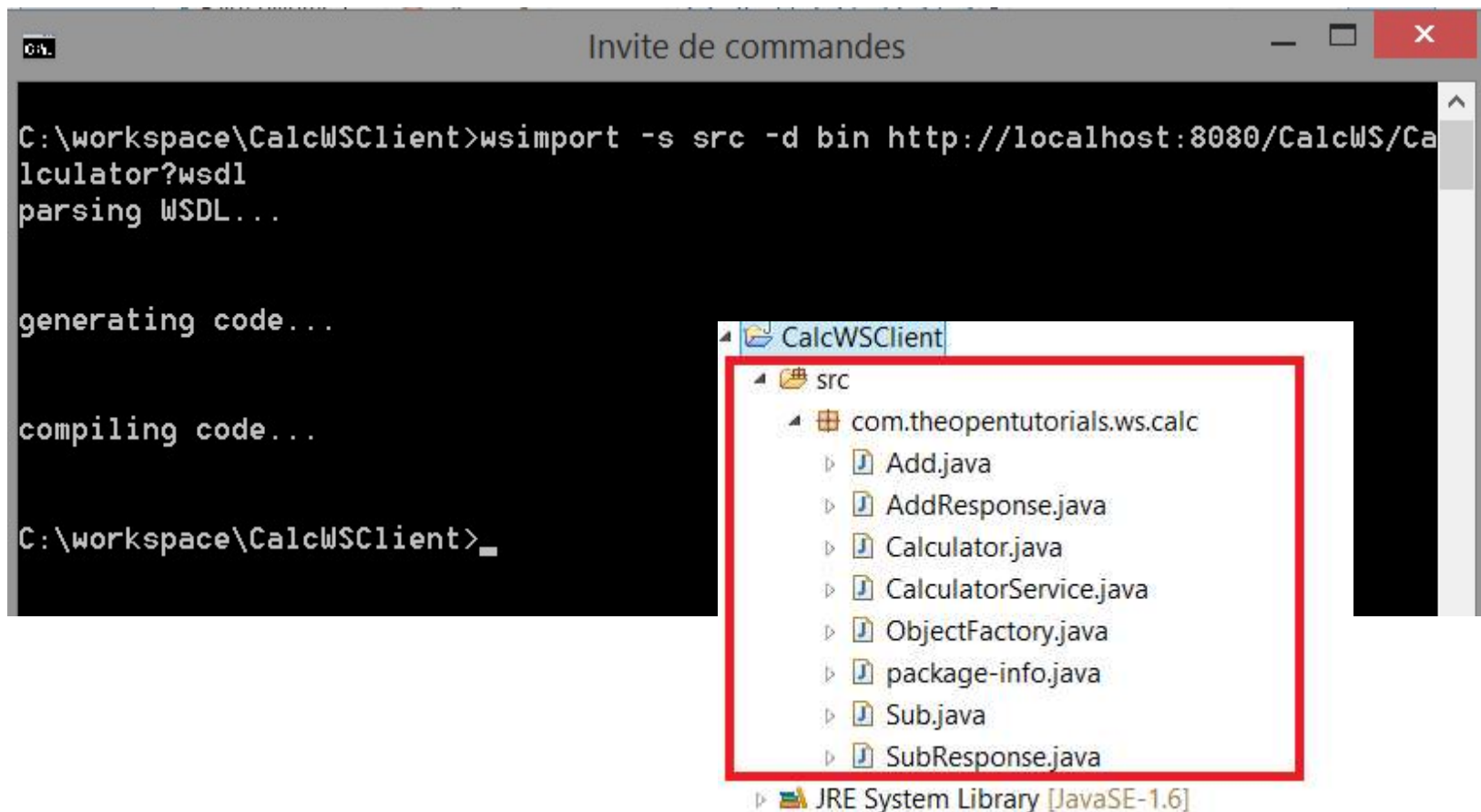
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.6 in JDK 6.
-->
▼<!--
  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.6 in JDK 6.
-->
▼<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://calc.ws.theopentutorials.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://calc.ws.theopentutorials.com/" name="CalculatorService">
  ▼<types>
    ▶<xsd:schema>...</xsd:schema>
  </types>
  ▼<message name="add">
    <part name="parameters" element="tns:add"/>
  </message>
  ▼<message name="addResponse">
    <part name="parameters" element="tns:addResponse"/>
  </message>
  ▼<message name="sub">
    <part name="parameters" element="tns:sub"/>
  </message>
  ▼<message name="subResponse">
    <part name="parameters" element="tns:subResponse"/>
  </message>
  ▼<portType name="Calculator">
    ▼<operation name="add">
      <input message="tns:add"/>
      <output message="tns:addResponse"/>
    </operation>
    ▼<operation name="sub">
      <input message="tns:sub"/>
      <output message="tns:subResponse"/>
    </operation>
  </portType>
</definitions>
```

JAX-WS (27/30)

Exemple complet

- Créer un projet java client pour consommer le service web Calculator.
- Exécuter la commande **wsimport -s src -d bin http://localhost:8080/CalcWS/Calculator?wsdl**



The screenshot shows a Windows command prompt window titled "Invite de commandes" with the following text:

```
C:\workspace\CalcWSClient>wsimport -s src -d bin http://localhost:8080/CalcWS/Calculator?wsdl
parsing WSDL...

generating code...

compiling code...

C:\workspace\CalcWSClient>_
```

Overlaid on the bottom right of the command prompt is an IDE window titled "CalcWSClient" showing a project structure. A red rectangle highlights the "src" folder, which contains the following files:

- com.theopentutorials.ws.calc
 - Add.java
 - AddResponse.java
 - Calculator.java
 - CalculatorService.java
 - ObjectFactory.java
 - package-info.java
 - Sub.java
 - SubResponse.java

Below the "src" folder, the "JRE System Library [JavaSE-1.6]" is also visible.

JAX-WS (28/30)

Exemple complet

- Créer la classe de test suivante :

```
CalcClient.java
package com.theopentutorials.ws.calc.client;

import com.theopentutorials.ws.calc.Calculator;
import com.theopentutorials.ws.calc.CalculatorService;

public class CalcClient {

    public static void main(String[] args) {
        int a = 10;
        int b = 12;
        CalculatorService calcService = new CalculatorService();
        Calculator calc = calcService.getCalculatorPort();
        System.out.println(a + " + " + b + " = " + calc.add(a, b));
        System.out.println(a + " - " + b + " = " + calc.sub(a, b));
    }
}
```

```
Console
<terminated> CalcClient [Java]
10 + 12 = 22
10 - 12 = -2
|
```

JAX-WS (29/30)

Les annotations utiles

- JAX-WS utilise JAXB-2.0 pour le mapping entre les objets et XML : les objets échangés par les services web peuvent utiliser les annotations de JAXB pour paramétrer finement certains éléments du message SOAP.
- Le WSDL généré définit l'élément avec un nom dont la première lettre est en minuscule.

Exemple :

```
package com.jmdoudoux.test.ws;

import javax.xml.ws.WebMethod;
import javax.xml.ws.WebParam;
import javax.xml.ws.WebService;

@WebService()
public class PersonneWS {

    @WebMethod(operationName = "Saluer")
    public String Saluer(@WebParam(name = "personne") final Personne personne)
        return "Bonjour " + personne.getNom() + " " + personne.getPrenom();
    }
}
```

Exemple :

```
package com.jmdoudoux.test.ws;

import java.util.Date;

public class Personne {

    private String nom;
    private String prenom;
    private Date dateNaiss;

    public Personne() {
        super();
    }
}
```

Exemple :

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
JAX-WS RI 2.1.7-hudson-48-. -->
<xs:schema xmlns:tns="http://ws.test.jmdoudoux.com/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="1.0" targetNamespace="http://ws.test.jmdoudoux.com/">

    <xs:element name="Saluer" type="tns:Saluer" />

    <xs:element name="SaluerResponse" type="tns:SaluerResponse" />

    <xs:complexType name="Saluer">
        <xs:sequence>
            <xs:element name="personne" type="tns:personne" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="personne">
        <xs:sequence>
            <xs:element name="dateNaiss" type="xs:dateTime" minOccurs="0" />
            <xs:element name="nom" type="xs:string" minOccurs="0" />
            <xs:element name="prenom" type="xs:string" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="SaluerResponse">
        <xs:sequence>
            <xs:element name="return" type="xs:string" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```


JAX-WS (30/30)

Les annotations utiles

En utilisant l'annotation `@XmlType`, il est possible de forcer le nom de l'élément généré dans le schéma :

Exemple :

```
package com.jmdoudoux.test.ws;

import java.util.Date;

import javax.xml.bind.annotation.XmlType;

@XmlType(name = "Personne")
public class Personne {

    private String nom;
    private String prenom;
    private Date dateNaiss;

    ...

}
```

Exemple :

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
JAX-WS RI 2.1.7-hudson-48-. -->
<xs:schema xmlns:tns="http://ws.test.jmdoudoux.com/"
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            version="1.0" targetNamespace="http://ws.test.jmdoudoux.com/">

    <xs:element name="Saluer" type="tns:Saluer" />

    <xs:element name="SaluerResponse" type="tns:SaluerResponse" />

    <xs:complexType name="Saluer">
        <xs:sequence>
            <xs:element name="personne" type="tns:Personne" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="Personne">
        <xs:sequence>
            <xs:element name="dateNaiss" type="xs:dateTime" minOccurs="0" />
            <xs:element name="nom" type="xs:string" minOccurs="0" />
            <xs:element name="prenom" type="xs:string" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="SaluerResponse">
        <xs:sequence>
            <xs:element name="return" type="xs:string" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Inclure des pièces jointes dans SOAP

Pour inclure des données binaires importantes dans un message SOAP, il faut utiliser le mécanisme des pièces jointes (attachment).

Ce mécanisme est implémenté par plusieurs standards :

- **SOAP With Attachments** : définis par le W3C dans la version 1.1 de SOAP
- **XOP/MTOM** : définis par le W3C dans la version 1.2 de SOAP

MTOM (Message Transmission Optimisation Mechanism) devient le standard utilisé par Java (JAX-WS) et .Net (WSE 3.0)

WS-I (Web Service Interoperability)

- Les nombreuses spécifications concernant les services web sont fréquemment incomplètes ou peu claires : il en résulte plusieurs incompatibilités lors de leur mise en œuvre.
- Le consortium **WS-I** (Web Service Interoperability) <http://www.ws-i.org/> a été créé pour définir des profiles qui sont des recommandations dont le but est de faciliter l'interopérabilité des services web entre plateformes, systèmes d'exploitation et langages pour promouvoir ces normes.
- Le WS-I a défini plusieurs spécifications :
 - ✓ WS-I Basic Profile
 - ✓ WS-I Basic Security Profile
 - ✓ Simple Soap Binding Profile
 - ✓ ...