

Web Services

SOAP / WSDL

JAX-WS / JAXB / SAAJ

Pierre-Yves Gibello – Janvier 2009

(avec quelques emprunts à Didier Donsez,
en particulier les slides de présentation SOAP / WSDL)

Licence : Creative Commons Attribution-ShareAlike

Définition

- Un WS est un composant logiciel accessible via les technologies internet
 - Fonction ou service “métier” accessible par d’autres applications (client, serveur, autre WS)
 - Utilisation des protocoles disponibles (ex. SOAP sur HTTP)
- Un WS est une entité qui fournit des services à la demande, via une interface XML bien définie sous forme de messages
 - XML fait référence pour l’automatisation des flux métier

Positionnement

- Appels de procédures distantes en Client/Serveur
 - CORBA
 - Multilangage, multi-plateforme, MultiVendeurs, OMG
 - Java RMI
 - mono-langage : Java, multi-plateforme (JVM), SUN
 - DCOM
 - multi-langages, plateforme Win32, Propriétaire Microsoft
- Web Services (Protocole SOAP)
 - multi-langages, multi-plateforme
 - Réponses et requêtes en XML
 - Transport sur RPC, HTTP ou autre (SMTP, MOM)
 - Spécification indépendante W3C

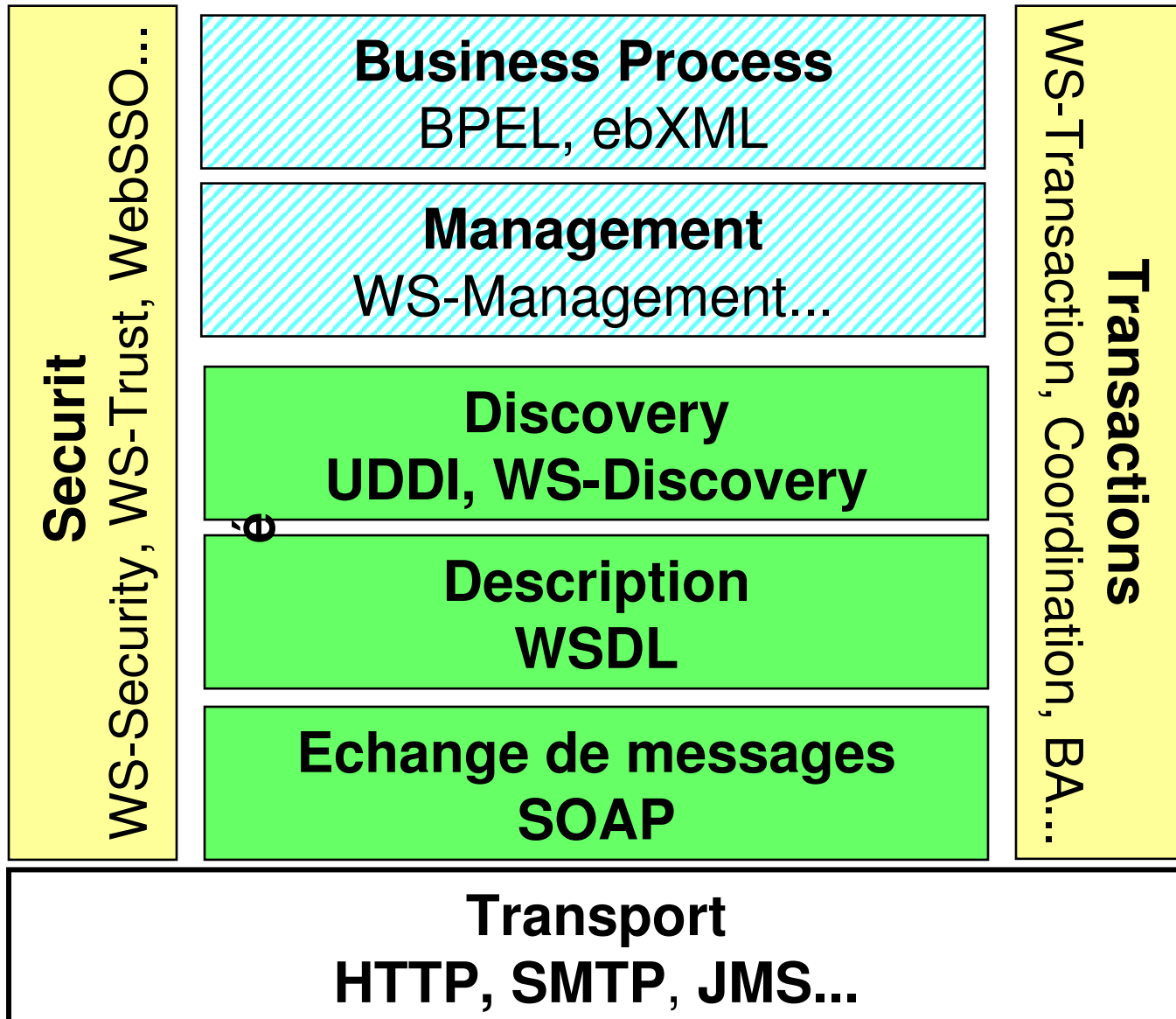
Caractéristiques d'un WS

- accessible via le Web,
- exporte une interface XML
- échange des messages XML via les protocoles du Web
- adapté aux systèmes interconnectés de manière flexible (liens transitoires, adaptation dynamique...)
- [localisable via un annuaire]

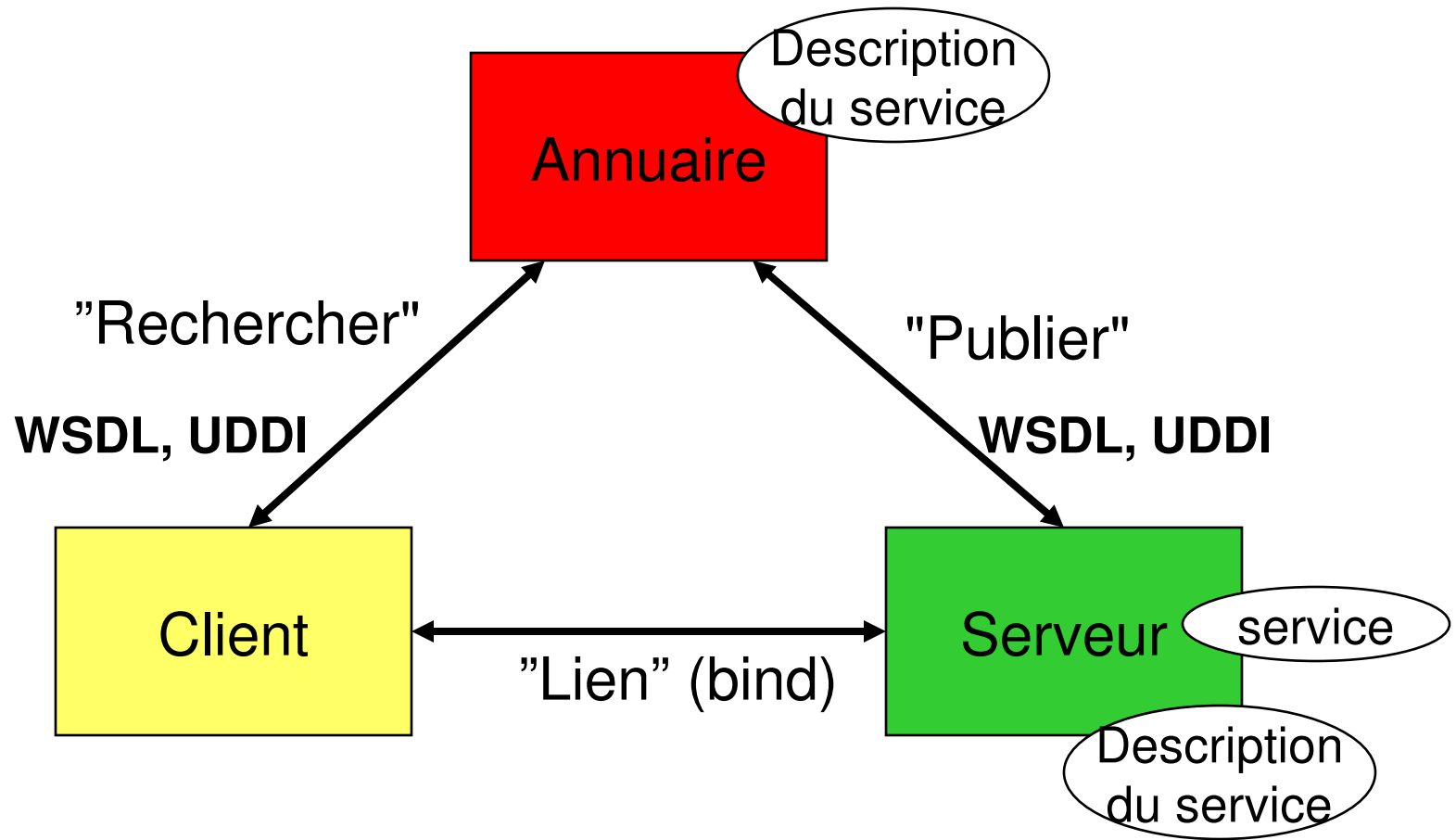
Web services et standards

- Enjeu : l'intéropérabilité
- Organismes de normalisation
 - W3C, Oasis, WS-I ...
- Initiatives privées
 - Microsoft, IBM, BEA ...
- Open-source
 - Sun Metro (RI), Apache Axis, Apache CXF

Architecture



Les différents rôles



Rappels XML

- Markup (balises avec attributs, et contenu)
 - `<elem [attribut=« valeur »]*>[valeur]</elem>`
 - `<element/>`
 - `<!-- commentaire -->`
- Extensible
 - Liberté de définir ses propres « dialectes »
 - Validation : DTD ou XMLSchema

Le Document

- [Prologue (en-tête)]
- Arbre d 'éléments (racine unique)
<meteo>
 <ville nom=« Grenoble »>
 <temp>14</temp>
 </ville>
</meteo>
- [Commentaires et instructions de traitement]

Buts

- Séparer la sémantique de la présentation
- Echange de données
 - Définir une syntaxe commune
 - Proposer des « dialectes » métier standards
ex. MathML, SMIL... dialectes métier
industriels

Parsing/Validation

- Document « bien formé »
 - Obéit aux règles syntaxiques de XML
 - Passe dans un parser non validant
- Document « valide »
 - Bien formé
 - Obéit à une structure type (DTD ou XMLSchema)
 - Passe dans un parser validant ou non

DTD

- Document Type Declaration
- Structure du document
- Description des éléments
 - Typage faible (String ou format externe)
- Ok pour traitement de document
 - Validation syntaxique et formelle
 - Edition, affichage, impression...

Exemple

DTD : bulletin.dtd

```
<!ELEMENT METEO (VILLE)+>
```

```
<!ELEMENT VILLE (TEMP)>
```

```
<!ATTLIST VILLE NOM CDATA #REQUIRED>
```

```
<!ELEMENT TEMP (#PCDATA)>
```

Note : PCDATA (« Parsed » CDATA) analysé par le parser XML, contrairement à CDATA (« character data »)

XML :

```
<!DOCTYPE meteo SYSTEM « bulletin.dtd »>
```

```
<meteo>
```

```
  <ville nom=« Grenoble »>
```

```
    <temp>14</temp>
```

```
  </ville>
```

```
</meteo>
```

XmlSchema

- Plus puissant (et complexe) que la DTD
- Typage fort, types complexes
- Héritage de type « objet »
- Contrôles sémantiques
 - Domaines de définition, énumérations, unicité de valeurs...

Domaines Nominiaux

- Ou « Espaces de noms » : « NameSpaces »
- Associer un nom à une DTD ou un Schéma
 - Utilisation de DTD/Schémas multiples
- Défini dans un élément
<meteo xmlns:mto=« <http://mameteo.com/bulletin> »>
- Préfixer les entités par le nom défini
<mto:ville mto:nom=« Grenoble »>...</mto:ville>

Exemple

```
<?xml version=« 1.0 » encoding=« ISO-8859-1 » ?>  
<meteo xmlns=«http://www.mameteo.com/bulletin »  
  xmlns:mto=« http://www.mameteo.com/mesures »>  
  <ville nom=« Grenoble »>  
    <mto:station>SMH</mto:station>  
    <temp mto:unite=« Celsius »>14</temp>  
  </ville>  
</meteo>
```


CSS

- Cascading Style Sheet
 - Feuilles de style
- Syntaxe non XML

```
selecteur { [propriété: valeur;]+ }  
titre { font-weight: bold; font-size: 15pt; }
```
- Mise en forme pure
 - Pas de modification de structure
 - Pas de modification de contenu

Les parsers XML

- Interchangeables
 - API standard (SAX, StAX ...)
 - Format de sortie standard (DOM)
- Validants ou non
 - DTD ou XmlSchema

SAX

- Simple API for XML
 - Extension standard de J2SE (javax.xml.parsers, org.xml.sax ...) – au départ, www.saxproject.org
 - Les parsers SAX sont interchangeables
- Envoi d'événements
 - Début/fin de document
 - Début/fin d'entité (ex. `startElement()`)
 - Caractères (contenu)
- Validation possible

SAX : ContentHandler

- Etendre `org.xml.sax.helpers.DefaultHandler`
- S'enregistrer auprès du `XMLReader`
`reader.setContentHandler(handler);`
- Méthodes de gestion d'événement :
`startDocument()`, `endDocument()`
`startElement()`, `endElement()`
`characters()`, etc...
- Implémentation par défaut (ne fait rien)

DOM

- Document Object Model
 - API standard (implémentations interchangeables)
- Arbre qui reflète la structure du document
- Format de sortie standard pour parser XML

Plateforme à Web Services

- Intégrée au serveur d'applications
- Protocoles et transport
 - SOAP sur HTTP, SMTP, JMS...
 - Synchrone / asynchrone
- Localisation et lien (« binding »)
- Déploiement / Cycle de vie
 - Déploiement d'objets java, EJBs...
 - Metadonnées (WSDL...)
 - Administration, Sécurité
- Outils de développement
 - APIs de niveau message (SAAJ, JAX-WS dispatch) ou XML (JAXB)
 - Génération de stubs depuis le WSDL ou de WS depuis un POJO (JAX-WS)

Quelques plateformes...

- Apache (open-source)
 - Axis : <http://ws.apache.org/axis2>
 - CXF : <http://cxf.apache.org>
- Sun Metro / Glassfish (open-source)
 - <https://metro.dev.java.net/>
- Microsoft *.NET*
- IBM *WebSphere*
- Oracle *Application Server*
- NuSOAP (PhP)
- ...

SOAP

- Simple Object Access Protocol
 - Accès à distance à un Web Service
 - Analogue à un protocole d 'objets distribués
 - Standard W3C (interopérabilité)
- Protocole d 'échange de messages en XML
 - Format du message
 - Encodage
 - Règles d 'échange

Portée de SOAP

- SOAP est « simple » et extensible
- et donc il ne couvre pas les fonctions suivantes :
 - Distributed garbage collection
 - Regroupement de messages
 - Passage d 'objets par référence
 - Activation (nécessite le passage par référence)

Message SOAP

- Message unidirectionnel
 - d'un expéditeur vers un récepteur
- Structure
 - Envelope
 - Élément racine
 - Namespace : SOAP-ENV
<http://schemas.xmlsoap.org/soap/envelope/>
 - Header
 - Élément optionnel
 - Contient des entrées non applicatives
 - Sessions (SessionId de servlet/jsp/asp), Transactions (BTP), ...
 - Body
 - Contient les entrées du message
 - Nom d'une procédure, valeurs des paramètres, valeur de retour
 - Peut contenir les éléments « fault » (erreurs)

En-tête message (Header)

- Contient des entrées non applicatives
 - Transactions, sessions, ...
- L'attribut mustUnderstand
 - Rien ou =0 : l'élément est optionnel pour l'application réceptrice
 - =1 : l'élément doit être compris de l'application réceptrice (sinon échec du traitement du message)
 - Exemple

```
<SOAP-ENV:Header>  
  <t:Transaction xmlns:t="some-URI" SOAP-  
ENV:mustUnderstand="1">  
    5  
  </t:Transaction>  
</SOAP-ENV:Header>
```

Corps du message (Body)

- Contient des entrées applicatives
 - Encodage des entrées
- Namespace pour l'encodage
 - SOAP-ENC
<http://schemas.xmlsoap.org/soap/encoding/>
 - xsd : XML Schema

Encodage

■ Types primitifs

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

```
<price>15.57</price>
```

```
<element name="greeting" type="xsd:string"/>
```

```
<greeting id="id1">Bonjour</greeting>
```

■ Structures

```
<element name="Book"><complexType>
```

```
  <element name="author" type="xsd:string"/>
```

```
  <element name="title" type="xsd:string"/>
```

```
</complexType></element>
```

```
<e:Book>
```

```
<author>J.R.R Tolkien</author>
```

```
<title>Bilbo le Hobbit</title>
```

```
</e:Book>
```

■ Références

```
<element name="salutation" type="xsd:string"/>
```

```
<salutation href="#id1"/>
```

■ Tableaux

```
<SOAP-ENC:Array id="id3" SOAP-ENC:arrayType=xsd:string[2,2]>
```

```
  <item>r1c1</item> <item>r1c2</item> <item>r2c1</item> <item>r2c2</item>
```

```
</SOAP-ENC:Array>
```

Exemple de requête (HTTP)

Demande de cotation à un serveur

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
```

```
<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:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>IBM</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemple de réponse (HTTP)

HTTP/1.1 200 OK

Content-Type: text/xml; charset="utf-8"

Content-Length: nnnn

```
<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>
```

Retour d 'erreur (Fault)

- 4 éléments
 - Faultcode (obligatoire)
 - Code d'erreur utilisé par le logiciel (switch(faultcode) { case ...)
 - Faultstring (obligatoire)
 - Explication lisible par un humain
 - faultactor (optionnel)
 - Erreur en cours de cheminement du message (firewall, proxy, MOM)
 - Detail
 - Détail de l'erreur non lié au Body du message
 - Autres
 - D'autres éléments qualifiés par un namespace peuvent être ajoutés
- Faultcode
 - 4 groupes de code d'erreur
 - Client, Server, MustUnderstand, VersionMismatch
 - Ex: `Client.Authentication`

WSDL

- Web Services Description Language
- Objectifs
 - Décrire les services comme un ensemble d'opérations et de messages abstraits relié (bind) à des protocoles et des serveurs réseaux
- Grammaire XML (schema XML)
 - Modulaire (import d'autres documents WSDL et XSD)

WSDL : structure

- `<types>`
 - Contient les définitions de types utilisant un système de typage (comme XmlSchema).
- `<message>`
 - Décrit les noms et types d'un ensemble de champs à transmettre
 - Paramètres d'une invocation, valeur du retour, ...
- `<porttype>`
 - Décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou de fautes
- `<binding>`
 - Spécifie une liaison d'un `<porttype>` à un protocole concret (SOAP1.1, HTTP1.1, MIME, ...). Un porttype peut avoir plusieurs liaisons.
- `<port>`
 - Spécifie un point d'entrée (endpoint) comme la combinaison d'un `<binding>` et d'une adresse réseau.
- `<service>`
 - Une collection de points d'entrée (endpoint) relatifs.

Messages : exemple

```
<!-- message declarations -->
```

```
<message name="AddEntryRequest">
```

```
<part name="name" type="xsd:string"/>
```

```
<part name="address" type="typens:address"/>
```

```
</message>
```

```
<message
```

```
name="GetAddressFromNameRequest">
```

```
<part name="name" type="xsd:string"/>
```

```
</message>
```

```
<message
```

```
name="GetAddressFromNameResponse">
```

```
<part name="address" type="typens:address"/>
```

```
</message>
```

PortType : exemple

```
<!-- port type declarations (un porttype est un ensemble
d 'opérations)-->
<portType name="AddressBook">

  <!-- One way operation -->
  <operation name="addEntry">
    <input message="AddEntryRequest"/>
  </operation>

  <!-- Request-Response operation -->
  <operation name="getAddressFromName">
    <input message="GetAddressFromNameRequest"/>
    <output message="GetAddressFromNameResponse"/>
  </operation>

</portType>
```

UDDI

- Universal Description, Discovery, and Integration
 - Annuaire de web services
 - Pour chaque WS : identifiant unique, et description détaillée (en WSDL)
 - APIs : JAXR (Java API for XML registries)
- UDDI définit un modèle de données (en XML) et des interfaces SOAP pour l'enregistrement de WS, et la recherche d'information.
- Alternative : LDAP ?

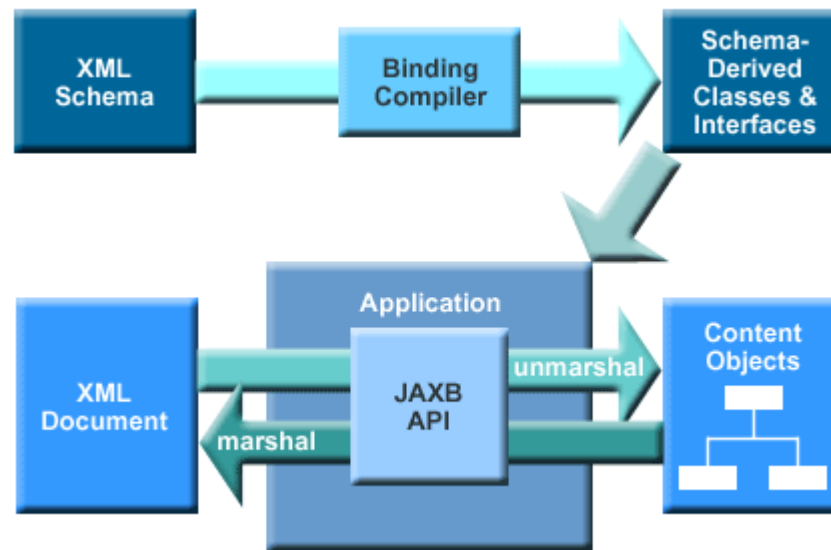
Les API java / WS

- JAX-WS (Java API for XML WS)
 - Web services (serveur et client)
- APIs XML
 - JAXB (Java Architecture for XML binding) :
mapping paramétrable XML / Java Beans
 - Parsing : StAX (Streaming API = pull parsing),
SAX (Simple API for Xml = push parsing)

JAXB

Java Architecture for XML Binding

<https://jaxb.dev.java.net>



Source : Sun Developer Network

Liaison (« binding ») XSD / java

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- meteo.xsd : XML Schema pour relevés de température -->
<xs:schema xmlns:tns="mameteo"
xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="mameteo"
version="1.0">
  <xs:element name="meteo" type="tns:meteo"></xs:element>
  <xs:complexType name="meteo">
    <xs:sequence>
      <xs:element name="ville" type="xs:string"></xs:element>
      <xs:element name="temperature" type="xs:float"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Liaison au schéma (« binding ») : génération de code java pour unmarshalling / marshalling (XML->java / java->XML)

```
xjc -p mameteo meteo.xsd
```

Package : mameteo

Code java généré

- Une classe par type complexe
 - Meteo.java (java bean avec accesseurs get/set pour Ville et Temperature)
- Une classe ObjectFactory
 - Méthodes pour créer chaque type complexe et chaque élément du schéma

Marshalling (java->XML)

```
JAXBContext ctx = JAXBContext.newInstance("mameteo");
```

```
// Utilisation du mapping pour créer l'arbre XML
```

```
ObjectFactory factory = new ObjectFactory( );
```

```
Meteo data = factory.createMeteo( );
```

```
data.setVille("Grenoble");
```

```
data.setTemperature(12);
```

```
Marshaller marshaller = ctx.createMarshaller( );
```

```
marshaller.setProperty(
```

```
    Marshaller.JAXB_FORMATTED_OUTPUT, true);
```

```
// Ecriture des données XML dans un fichier
```

```
marshaller.marshal(factory.createMeteo(data),
```

```
    new FileOutputStream("grenoble.xml"));
```

Unmarshalling (XML->java)

```
JAXBContext ctx = JAXBContext.newInstance("mameteo");
```

```
Unmarshaller dec = ctx.createUnmarshaller();
```

```
// Lecture du flux XML, construction de l'arbre
```

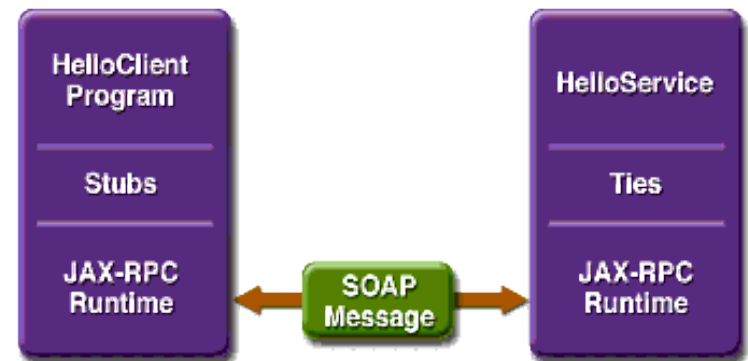
```
JAXBElement<Meteo> document = dec.unmarshal(  
    new javax.xml.transform.stream.StreamSource(new File("grenoble.xml")),  
    Meteo.class);
```

```
// Utilisation du mapping JAXB
```

```
Meteo m = document.getValue();  
System.out.println(m.getVille() + " : " + m.getTemperature());
```

JAX-WS

- Java API for XML WS (javax.xml.ws)
 - Intégré à J2SE 6+
- Cache la complexité de SOAP
 - Génération de proxy (« stub ») client
 - Déploiement serveur par annotations
 - Spécification d'interface WSDL ou java
- Synchrones / asynchrones



Outils J2SE 1.6+

- `wsimport`
 - Generation de stubs à partir d'une description WSDL
- `wsgen / apt`
 - Annotations Web Services
- `xjc`
 - JAXB : Générateur XSD->JAVA
- `schemagen`
 - JAXB : Générateur java -> XSD

JAX-WS EndPoint

- Implémentation côté serveur
- Service Endpoint Interface
 - Interface java (explicite ou implicite : classe)
 - Annotations : `@WebService` / `@WebMethod` ...
 - Méthodes métier : paramètres / retours compatibles JAXB
 - Constructeur public par défaut, pas de `finalize()`
- Cycle de vie
 - Annotations : `@PostConstruct` / `@PreDestroy`

Endpoint : Exemple (1)

```
package calculette;  
import javax.jws.*;
```

```
// Compilation : javac -d . puis wsgen -cp . (ou apt -d .)
```

```
@WebService
```

```
public class Calculette {
```

```
    @WebMethod
```

```
    public int addition (int n1, int n2) { return n1 + n2; }
```

```
    @WebMethod
```

```
    public int soustraction (int n1, int n2) { return n1 - n2; }
```

```
}
```

EndPoint : Exemple (2)

```
package calculette;
import javax.xml.ws.Endpoint;

// Serveur web exportant le WS (lancer avec java)
public class CalcServer {

    public static void main (String args[]) throws Exception {
        String port = "8080";
        if(args.length > 0) port = args[0];
        Endpoint endpoint = Endpoint.publish(
            "http://localhost:" + port + "/calculette", new Calculette());

        System.out.println("WSDL : http://localhost:8080/calculette?wsdl");
        System.out.println("XSD : http://localhost:8080/calculette?xsd=1");
    }
}
```


Client : exemple

- Génération du stub :
 - `wsimport -p stub. -s . http://localhost:8080/calculatrice?wsdl`
 - `-d` = répertoire cible, `-s` = cible pour les sources java
 - Stub généré (classes utilisables dans le client) :
 - Une interface (« Service Endpoint Interface » ou SEI) par « port » du WSDL (ex. Calculatrice)
 - Une classe par « service » du WSDL (ex. CalculatriceService)
 - Mapping JAXB : 1 classe par « message » du WSDL (ex. Addition, AdditionResponse), et 1 classe ObjectFactory
- Ecriture du client

```
Calculatrice port = new CalculatriceService().getCalculatricePort();  
System.out.println("1+2=" + port.addition(1, 2));
```

En partant du WSDL

- Génération des stubs (et du mapping JAXB) avec `wsimport`
- Ecriture d'une classe qui implémente la « Service Endpoint Interface » générée
 - Annotation `@WebService`
(`endpointInterface= « interfaceGeneree »`)
 - Annotations `@Webmethod` pour l'implémentation des méthodes de l'interface

Debug : tcpmon

<https://tcpmon.dev.java.net>

Lancement tcpmon

```
java -jar tcpmon.jar
```

Configuration tcpmon

Local port : 8080

Server name : 127.0.0.1

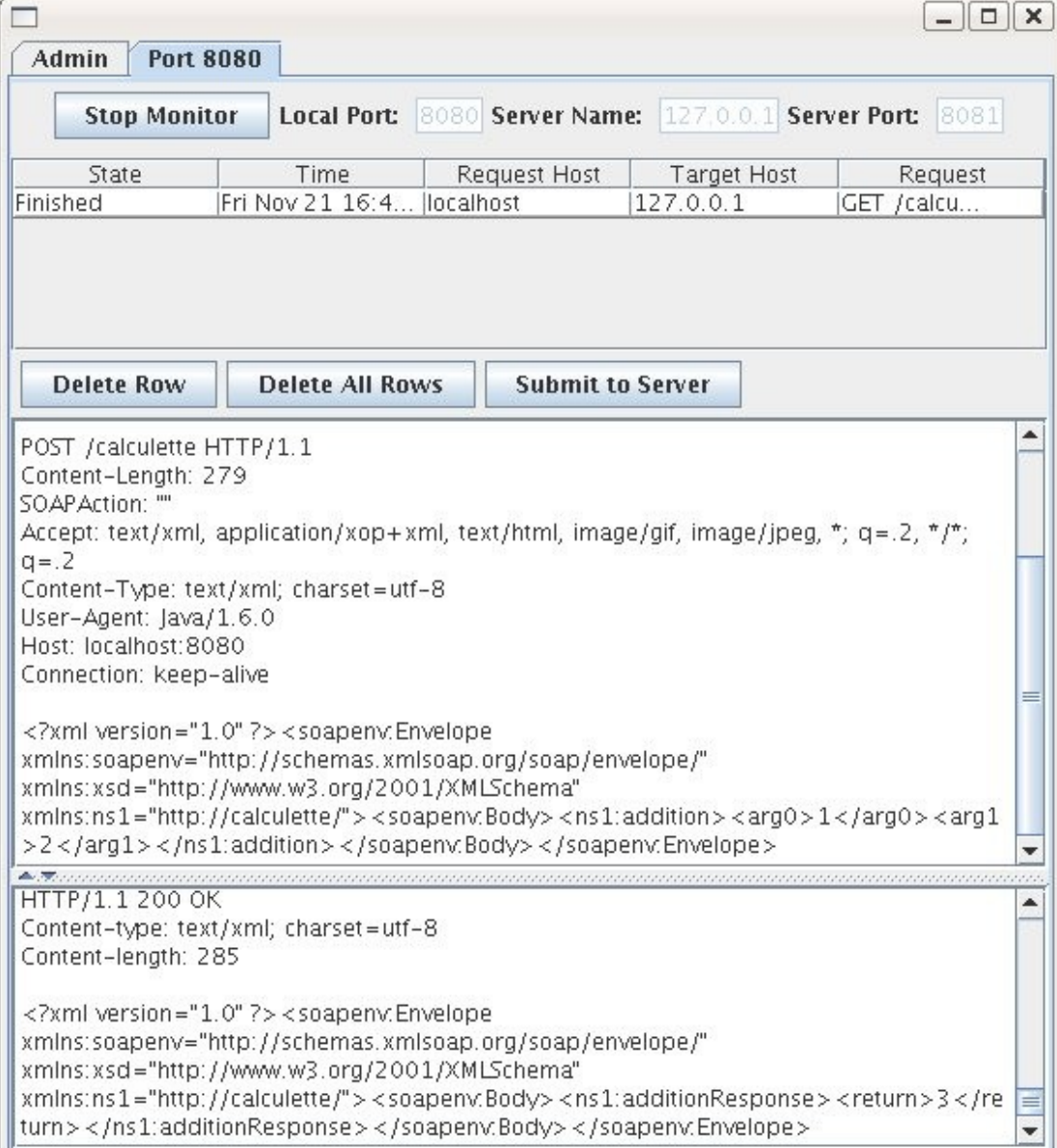
Server port : 8180

SEI et client

Web Service sur 8180

Client se connecte à 8080

Les requêtes sont visibles



The screenshot shows the tcpmon application window. At the top, there are tabs for 'Admin' and 'Port 8080'. Below the tabs, there is a 'Stop Monitor' button and configuration fields for 'Local Port: 8080', 'Server Name: 127.0.0.1', and 'Server Port: 8081'. A table displays the request details:

State	Time	Request Host	Target Host	Request
Finished	Fri Nov 21 16:4...	localhost	127.0.0.1	GET /calcu...

Below the table are buttons for 'Delete Row', 'Delete All Rows', and 'Submit to Server'. The main area shows the raw HTTP request and response:

```
POST /calcullette HTTP/1.1
Content-Length: 279
SOAPAction: ""
Accept: text/xml, application/xop+xml, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Type: text/xml; charset=utf-8
User-Agent: Java/1.6.0
Host: localhost:8080
Connection: Keep-alive

<?xml version="1.0" ?> <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://calcullette/"> <soapenv.Body> <ns1:addition> <arg0>1 </arg0> <arg1
>2 </arg1> </ns1:addition> </soapenv.Body> </soapenv.Envelope>

HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Content-length: 285

<?xml version="1.0" ?> <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://calcullette/"> <soapenv.Body> <ns1:additionResponse> <return>3 </re
turn> </ns1:additionResponse> </soapenv.Body> </soapenv.Envelope>
```

Handlers

- Intercepteurs
 - Message entrant, sortant, fault
 - Implémente LogicalHandler ou SOAPHandler
 - Package javax.xml.ws.handler [.soap]
 - Possibilité de définir une chaîne de Handlers
- Déclaration
 - Annotation @HandlerChain(file=«handlers.xml»)
 - Appliquée à la classe d'implémentation du Web Service
 - Méthode setHandlerChain() de javax.xml.ws.Binding
 - endPoint.getBinding();
- Usage : logging, sécurité, modification de contenu

Handlers : exemple (déclaration)

```
@WebService
@HandlerChain(file="handlers.xml")
public class Calculette {
    // ...
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- handlers.xml -->
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <handler>
      <handler-class>calculette.LogHandler</handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

Handlers : exemple (code)

```
public class LogHandler implements SOAPHandler<SOAPMessageContext> {  
  
    public boolean handleMessage(SOAPMessageContext smc) {  
        try {  
            smc.getMessage().writeTo(System.out);  
        } catch(Exception ignore) { }  
        return true;  
    }  
  
    public boolean handleFault(SOAPMessageContext smc) {  
        return handleMessage(smc);  
    }  
  
    public Set<QName> getHeaders() { return null; }  
    public void close(MessageContext messageContext) { }  
}
```

Le problème de la réentrance

- Un WS n'est pas réentrant
 - Collisions si mises à jour concurrentes
- Contournements possibles
 - Pattern « facade » avec maintien de session par le client
 - SessionId affecté par le WS, ou utilisation de mécanismes intégrés au transport (session HTTP)
 - Méthodes de mise à jour « synchronized »

Web Service Context

- Accès au contexte du message
 - HTTP : principaux en-têtes et contexte servlet
 - Sécurité : `getUserPrincipal()`, `isUserInRole(role)`
- Initialisé par injection de dépendance
 - Annotation `@Resource` (`javax.annotation.Resource`)

`@WebService`

```
public class Hello {
```

```
    @Resource
```

```
    private javax.xml.ws.WebServiceContext wsContext;
```

```
    // ...
```

```
}
```


Exemple : accès à la session HTTP

- Méthode de maintien de session
 - Sur transport HTTP
- Web Service avec scope « session »
 - Moyen de contourner les problèmes de réentrance

```
MessageContext mc = wsContext.getMessageContext();
```

```
HttpServletRequest req =
```

```
(HttpServletRequest)mc.get(MessageContext.SERVLET_REQUEST);
```

```
HttpSession session = req.getSession(true);
```

Invocation asynchrone

- A préciser lors de l'appel à `wsimport`
 - « binding » `wsimport : option -b`
 - `EnableAsyncMapping = true`
 - Génération de méthodes `xxxAsync` pour chaque opération `xxx`
- 2 modèles de programmation
 - Polling
 - Callback

Asynchrone : binding wsimport

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<!-- fichier a fournir en option -b de wsimport -->  
<bindings  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"  
  wsdlLocation="http://localhost:8080/calculatrice?wsdl"  
  xmlns="http://java.sun.com/xml/ns/jaxws">  
  <bindings node="wSDL:definitions">  
    <enableAsyncMapping>true</enableAsyncMapping>  
  </bindings>  
</bindings>
```

Asynchrone : polling

```
import javax.xml.ws.Response;
```

```
// ...
```

```
// Methode additionAsync générée par wsimport
```

```
// (enableAsyncMapping=true)
```

```
Response<AdditionResponse> rsp = port.additionAsync(8, 2000);
```

```
while(! rsp.isDone()) {
```

```
    System.out.println("Attente reponse en polling");
```

```
    Thread.sleep(1000);
```

```
}
```

```
AdditionResponse resultat = rsp.get();
```

```
System.out.println("8+2000=" + resultat.getReturn());
```

Asynchrone : callback (invocation)

```
import java.util.concurrent.Future;
```

```
// ...
```

```
AdditionCallback callbackHandler = new AdditionCallback();  
System.out.println("Attente reponse en callback");
```

```
// Enregistrement du listener (classe gestionnaire)
```

```
// Methode additionAsync générée par wsimport (enableAsyncMapping)
```

```
Future<?> response = port.additionAsync(8, 2000, callbackHandler);
```

```
Thread.sleep(7000);
```

```
if(response.isDone()) {
```

```
    AdditionResponse result = callbackHandler.getResultat();
```

```
    System.out.println("8+2000=" + result.getReturn());
```

```
}
```

Asynchrone : callback (gestionnaire)

// Classe gestionnaire (listener) à enregistrer

```
public class AdditionCallback  
implements AsyncHandler<AdditionResponse> {
```

```
    AdditionResponse resultat_;
```

// Méthode callback appelée à l'arrivée d'une réponse

```
public void handleResponse(Response<AdditionResponse> response) {  
    System.out.println("Reception reponse");  
    try {  
        resultat_ = response.get();  
    } catch (Exception e) { e.printStackTrace(); }  
}
```

```
public AdditionResponse getResultat() { return resultat_; }  
}
```

SAAJ

- SOAP with Attachments for Java
 - javax.xml.soap
 - Avantages : attachements MIME
 - Inconvénients : API bas niveau (SOAP)
- API de niveau message
 - SOAPConnectionFactory / SOAPConnection
 - SOAPMessage
 - (SOAPPart (SOAPEnvelope (SOAPBody (SOAPElement)*))
 - SOAPMessage reply = connection.call(message, destination);

SAAJ : Exemple

```
SOAPConnectionFactory soapCF = SOAPConnectionFactory.newInstance();  
SOAPConnection connection = soapCF.createConnection();
```

```
MessageFactory messageFactory = MessageFactory.newInstance();  
SOAPMessage message = messageFactory.createMessage();
```

```
SOAPPart soapPart = message.getSOAPPart();  
SOAPEnvelope envelope = soapPart.getEnvelope();  
SOAPBody body = envelope.getBody();
```

```
SOAPElement bodyElement = body.addChildElement(  
    envelope.createName("addition", "ns1", "http://calcullette/"));  
bodyElement.addChildElement("arg0").addTextNode("1");  
bodyElement.addChildElement("arg1").addTextNode("3");  
message.saveChanges();
```

```
SOAPMessage reply = connection.call(message, "http://localhost:8080/calcullette");
```


SAAJ : Attachement

```
import javax.xml.soap.AttachmentPart;
```

```
MessageFactory messageFactory = MessageFactory.newInstance();  
SOAPMessage m = messageFactory.createMessage();
```

```
AttachmentPart ap1 = m.createAttachmentPart();  
ap1.setContent("Ceci est un cours SAAJ", "text/plain");  
m.addAttachmentPart(ap1);
```

```
AttachmentPart ap2 = m.createAttachmentPart();  
ap2.setRawContent(new FileInputStream("logo.jpg"), "image/jpeg");  
m.addAttachmentPart(ap2);
```

Annexes

- Déploiement : format WAR (Web Application aRchive)

Déploiement : Applications Web

- Ensemble des composants d'une application
 - Servlets et JSP, bibliothèques de classes, fichiers de configuration...
- Format standard de packaging
 - Web Application Archive (fichier .war)
 - Format jar, avec organisation standard
- Déploiement par le conteneur de servlets
 - Automatique
 - Paramétrable par configuration
- Exemple : déploiement Tomcat
 - Copier le fichier .war dans webapps/
 - Redémarrer le serveur

Web Application aRchive

```
monappliweb.war  
http://serveur/monappliweb
```

monappliweb

WEB-INF

web.xml - configuration de l'appl

Ressources locales

HTML, iso...

classes

épertoire du classpath

lib

zip et .jar du classpath