EPSI I4 Web Services / **David Gayerie** - 2013-2014

WEB SERVICES SOAP

LA QUÊTE DE L'INTEROPÉRABILITÉ

Comment échanger des informations entre deux processus alors que :

- les programmes sont écrits dans des langages différents
- les processus s'exécutent sur des machines différentes
- les machines appartiennent à des réseaux différents

Avant l'an 2000, on trouve des solutions telles que Sun RPC, CORBA, DCOM, RMI.

À partir des années 2000, le succés des réseaux IP, de HTTP et de XML apporte une nouvelle perspective : définir un protocole d'échange de messages XML *via* HTTP entre deux processus.

Ce protocole originellement défini par Microsoft puis maintenu par le W3 s'appelle **SOAP**.

Depuis SOAP a été enrichi de nombreuses spécifications et l'ensemble de ces technologies sont désignées sous le terme **WS-***.

WS-* est un ensemble de technologies, il ne s'agit ni d'un modèle de conception ni d'un modèle d'architecture.

Exemple de message SOAP

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:m="http://www.mymeteo.com/webservices/samples">
   <soapenv:Header>
   </soapenv:Header>
   <soapenv:Body>
      <m:meteo>
         <m:temperature>12</m:temperature>
         <m:unite>celsius</m:unite>
         <m:ville>Bordeaux</m:ville>
      </m:meteo>
   </soapenv:Body>
</soapenv:Envelope>
```

UN PREMIER SERVICE WEB SOAP AVEC JAX-WS

JAX-WS est l'API Java EE pour l'implémentation de services Web SOAP.

JAX-WS utilise JAXB pour le binding Java/XML.

Un premier service Web SOAP

```
import javax.ejb.Stateless;
import javax.jws.WebParam;
import javax.jws.WebService;

//L'annotation Stateless est nécessaire pour TomEE
@Stateless
@WebService
public class HelloService {

  public String sayHello(@WebParam(name="who") String name) {
    return "Hello " + name;
  }

}
```

Cette classe est appelée la **SEI** (Service Endpoint Implementation).

Une requête au Web service

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:i4="http://i4.epsi.fr/">
   <soapenv:Header/>
   <soapenv:Body>
      <i4:sayHello>
         <who>david</who>
      </i4:sayHello>
   </soapenv:Body>
</soapenv:Envelope>
```

Le message doit être envoyé avec la méthode HTTP POST à l'URL du service (appelée aussi **endpoint**)

PARLEZ-VOUS XML ?

XML = Extensible Markup Language

XML est un langage de balisage générique.

Un document XML est **bien formé** (well-formed) s'il respecte la **recommandation W3 (http://www.w3.org/TR/xml/)** .

Ce document XML est-il bien formé ?

```
<cours>
  <intitulé>Web services</intitulé>
  <enseignant>D. Gayerie</enseignant>
</cours>
```

Réponse : oui

Ce document XML est-il bien formé ?

```
<?xml version="1.0" encoding="utf-8"?>
<cours>
  <code id="I4WS">
  <intitulé>Web services</intitulé>
  <enseignant>D. Gayerie</enseignant>
</cours>
```

Réponse : non

L'élément code doit être fermé :

```
<?xml version="1.0" encoding="utf-8"?>
<cours>
  <code id="I4WS"/>
  <intitulé>Web services</intitulé>
  <enseignant>D. Gayerie</enseignant>
</cours>
```

Ce document XML est-il bien formé ?

```xml
<cours>
  <intitulé>Génie logiciel</intitulé>
</cours>
<cours>
  <intitulé>Web services</intitulé>
</cours>
```

Réponse : non

Le document doit contenir un seul élément racine :

```xml
<programme>
  <cours>
    <intitulé>Génie logiciel</intitulé>
  </cours>
  <cours>
    <intitulé>Web services</intitulé>
  </cours>
</programme>
```

Ce document XML est-il bien formé ?

```xml
<programme>
  <cours code=I4GL>
    <intitulé>Génie logiciel</intitulé>
  </cours>
  <cours code=I4WS>
    <intitulé>Web services</intitulé>
  </cours>
</programme>
```

Réponse : non

Les attributs d'un élément doivent être entre guillemets :

```xml
<programme>
  <cours code="I4GL">
    <intitulé>Génie logiciel</intitulé>
  </cours>
  <cours code="I4WS">
    <intitulé>Web services</intitulé>
  </cours>
</programme>
```

Ce document XML est-il bien formé ?

```xml
<test>
  <question>Ce document est-il valide ?</question>
  <exemple>
  <![CDATA[
  <document></DOCUMENT>
  ]]>
  </exemple>
</test>
```

Réponse : oui

La balise spéciale CDATA permet de délimiter une zone traitée comme du texte.

Ce document XML est-il bien formé ?

```xml
<test>
  <question>Cette expression est-elle vraie ?</question>
  <expression>2 < 3</expression>
</test>
```

Réponse : non

Les caractères & ' " < > doivent être échappés respectivement par & ' " < >

```xml
<test>
  <question>Cette expression est-elle vraie ?</question>
  <expression>2 &lt; 3</expression>
</test>
```

Un élément d'un document XML peut être associé à un espace de nom. Un espace de nom XML (**XML namespace**) est identifié par une URI. Un espace de nom est déclaré grâce à l'attribut **xmlns**.

Déclaration d'un espace de nom XML

```xml
<programme xmlns="http://epsi.fr/syllabus">
  <cours code="I4GL">
    <intitulé>Génie logiciel</intitulé>
  </cours>
  <cours code="I4WS">
    <intitulé>Web services</intitulé>
  </cours>
</programme>
```

Un espace de nom XML s'applique à l'élément portant l'attribut xmlns **et** à tous les éléments fils.

Il est possible de mélanger dans un même document XML des éléments appartenant à des espaces de noms différents.

Document XML contenant plusieurs espaces de nom

```xml
<html xmlns="http://www.w3.org/1999/xhtml">
  <body>
    <h1>Exemple MathML</h1>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mrow>
        <mroot>
          <mrow><mi>b</mi></mrow>
          <mrow><mn>3</mn></mrow>
        </mroot>
      </mrow>
    </math>
  </body>
</html>
```

Afin de résoudre les cas ambigus, un espace de nom peut être identifié par un préfixe.

Document XML contenant plusieurs espaces de nom avec un préfixe

```xml
<h:html xmlns:h="http://www.w3.org/1999/xhtml"
        xmlns:ma="http://www.w3.org/1998/Math/MathML">
  <h:body>
    <h:h1>Exemple MathML</h:h1>
    <ma:math>
      <ma:mrow>
        <ma:mroot>
          <ma:mrow><ma:mi>b</ma:mi></ma:mrow>
          <ma:mrow><ma:mn>3</ma:mn></ma:mrow>
        </ma:mroot>
      </ma:mrow>
    </ma:math>
  </h:body>
</h:html>
```

L'espace de nom sans préfixe est appelé l'espace de nom par défaut (**default XML namespace**).

Un élément XML est défini par :

Son nom

Le nom apparaît dans la balise du document.

Exemple : body

Son espace de nom

L'espace de nom est donné par l'attribut xmlns de l'élément, un préfixe ou est hérité de l'élément parent.

Exemple : http://www.w3.org/1999/xhtml

Son nom qualifié (qualified name ou QName)

L'association de l'espace de nom et du nom de l'élement. Il s'agit en fait du nom complet de l'élement.

Exemple : {http://www.w3.org/1999/xhtml}:body

Un document XML est **valide** (valid) si le QName et l'ordre des éléments sont conformes à un document de validation.

Un **schéma XML** (XSD) permet d'écrire un document de validation au format XML.

Exemple de schéma XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="http://epsi.fr/syllabus"
        targetNamespace="http://epsi.fr/syllabus"
        elementFormDefault="qualified">
  <element name="programme">
    <complexType>
      <sequence>
        <element name="cours" type="tns:typeCours"
         maxOccurs="unbounded" minOccurs="0"/>
      </sequence>
    </complexType>
  </element>
  <complexType name="typeCours">
    <sequence>
      <element name="intitulé" type="string"
       maxOccurs="1" minOccurs="1"/>
    </sequence>
    <attribute name="code" type="string"/>
  </complexType>
</schema>
```

POUR ALLER PLUS LOIN...

XML

http://www.w3schools.com/xml/

XML namespace

http://www.w3schools.com/xml/xml_namespaces.asp

XML schema

http://www.w3schools.com/schema/

SOAP : LE LANGAGE D'ÉCHANGE

SOAP est un simple format de message définit par un schéma XML.

Ce schéma XML est disponible à l'adresse de son namespace XML :
http://schemas.xmlsoap.org/soap/envelope/

Structure d'un message SOAP

```xml
<!-- l'enveloppe du message -->
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

  <!-- l'en-tête du message -->
  <soapenv:Header>
    <!-- les éventuelles méta-informations -->
  </soapenv:Header>

  <!-- le corps de la requête -->
  <soapenv:Body>
    <!-- le message (payload) -->
  </soapenv:Body>

</soapenv:Envelope>
```

Le format d'un message est le même pour la requête et pour la réponse.

SOAP définit un format pour un type de réponse particulière : une SOAP fault.

Une SOAP fault signale un problème lors du traitement de la requête.

Une réponse contenant une SOAP fault

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>Internal server error</faultstring>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Avec JAX-WS, la SEI peut récupérer les informations provenant du corps de la requête mais aussi des en-têtes.

L'annotation **@WebParam (http://docs.oracle.com/javaee/6/api/javax/jws/WebParam.html)** possède les attributs suivants :

name
: le nom de l'élement XML

targetNamespace
: l'espace de nom de l'élement XML

header
: un booléan indiquant si l'élement XML doit être extrait de l'en-tête ou s'il s'agit du corps du message.

Avec JAX-WS, la SEI peut déclarer des méthodes ayant plusieurs valeurs de retour : le corps de la réponse mais aussi les en-têtes de la réponse.

L'annotation **@WebParam** possède l'attribut suivant :

mode

Il s'agit d'une énumération pouvant prendre les valeurs IN, OUT ou INOUT. Pour les valeurs OUT et INOUT, le type du paramètre est obligatoirement de type **javax.xml.ws.Holder<T> (http://docs.oracle.com/javaee/6/api/javax/xml/ws/Holder.html)** .

```
public Comments sayHello(@WebParam Article article,
     @WebParam(header=true) EditorCredentials editorCredentials,
     @WebParam(header=true, mode=Mode.OUT) Holder<ServerInfo> serverInfo) {
     // ...
}
```

Avec JAX-WS, les exceptions lancées par une méthode d'une SEI sont automatiquement transformées en SOAP fault.

JAX-WS définit une annotation particulière : **@Oneway (http://docs.oracle.com/javaee/6/api/javax/jws/Oneway.html)** . Cette annotation ne peut être utilisée que sur des méthodes n'ayant aucune valeur de retour (la méthode retourne void, elle ne déclare aucun paramètre OUT ou INOUT et elle ne lance aucune exception).

Une méthode @Oneway permet de réaliser un traitement asynchrone : une réponse SOAP est retournée au client pendant que la méthode est exécutée.

WSDL

La condition pour utiliser un service Web SOAP est de connaître les opérations disponibles et le format des requêtes et des réponses.

WSDL (Web Service Description Language) est un langage XML permettant la description complète d'un service Web SOAP.

La plupart des *stacks* SOAP donne accès au WSDL (en le générant dynamiquement si nécessaire).

Pour les serveurs d'application Java EE, le WSDL est disponible à la même URL que le service en ajoutant **wsdl** comme paramètre.

Exemple : **http://localhost:8080/hello-service/webservices/HelloService?wsdl**

Structure d'un WSDL

```
<wsdl:definitions
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://epsi/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
  name="HelloServiceService" targetNamespace="http://epsi/">
```

Les types de données au format XML schema

```
  <wsdl:types>
    <!-- on trouve ici le schéma XML décrivant le contenu des messages -->
  </wsdl:types>
```

Description des messages

```
  <wsdl:message name="sayHelloResponse">
    <wsdl:part element="tns:sayHelloResponse" name="result"/>
    <wsdl:part element="tns:serverTime" name="serverTime"/>
  </wsdl:message>
  <wsdl:message name="sayHello">
    <wsdl:part element="tns:sayHello" name="parameters"/>
  </wsdl:message>
```

Description des interfaces (indépendantes de SOAP)

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

Liaison des interfaces avec le protocole SOAP

```
  <wsdl:binding name="HelloServiceServiceSoapBinding"
                type="tns:HelloService">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="sayHello">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="sayHello">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="sayHelloResponse">
        <soap:header message="tns:sayHelloResponse"
                     part="serverTime" use="literal"/>
        <soap:body parts="result" use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
```

Localisation du service

```
  <wsdl:service name="HelloServiceService">
    <wsdl:port binding="tns:HelloServiceServiceSoapBinding"
               name="HelloServicePort">
      <soap:address location="http://localhost:8080/hello-service/webservices/HelloService"/>
    </wsdl:port>
  </wsdl:service>
```

```
</wsdl:definitions>
```

POUR ALLER PLUS LOIN...

WSDL

http://www.w3schools.com/webservices/ws_wsdl_intro.asp

DÉVELOPPER UN SERVICE WEB SOAP

On distingue deux approches pour développer un service Web WS-* :

Contract first
: On définit le contrat du service en spécifiant le WSDL et on utilise des outils pour générer le code.

Contract last
: On développe le service et on utilise des outils pour générer le WSDL.

Les outils Java

Java → WSDL (Contract last)
: On peut demander au serveur de générer le WSDL après le déploiement du service en passant *wsdl* comme paramètre de l'URL.

L'utilitaire **wsgen (http://docs.oracle.com/javase/7/docs/technotes/tools/share/wsgen.html)** livré avec le JDK peut générer un fichier WSDL à partir d'une classe compilée.

WSDL → Java (Contract first)
: L'utilitaire **wsimport (http://docs.oracle.com/javase/7/docs/technotes/tools/share/wsimport.html)** livré avec le JDK.

Pour utiliser **wsimport** :

Générer les classes compilées

```
wsimport http://localhost:8080/hello-service/webservices/HelloService?wsdl
```

Générer uniquement les sources

```
wsimport -Xnocompile http://localhost:8080/hello-service/webservices/HelloService?wsdl
```

Une fois la SEI (Service Endpoint Interface) générée par wsimport, il suffit de fournir une implémentation de cette interface.

Exemple de l'implémentation d'une SEI

```
// Nécessaire uniquement avec TomEE.
// L'attribut name spécifie la terminaison de l'URL d'accès au service.
@Stateless(name="HelloService")
// L'attribut endpointInterface désigne l'interface annotée avec JAX-WS.
@WebService(endpointInterface="epsi.HelloService")
public class HelloServiceImpl implements epsi.HelloService {

  @Override
  public SayHelloResponse sayHello(SayHello parameters) {
    // ...
  }

}
```

CONSOMMER DES SERVICES WEB SOAP AVEC JAX-WS

Une application cliente d'un service Web SOAP utilise un **stub**. Un stub est une classe générée qui masque la complexité des échanges de messages avec le serveur.

Il est possible de créer un client à partir du code généré par **wsimport**.

Exemple d'accès à la SEI pour un client

```
// L'URL du WSDL (peut-être un fichier sur disque)
URL wsdlDocumentLocation = new URL("file:HelloService.wsdl");

// Le qualified name du service spécifié par les attributs
// targetNamespace et name de la balise racine dans le WSDL
QName serviceName = new QName("http://epsi/","HelloServiceService");

// HelloServiceService est une classe générée par wsimport
Service service = HelloServiceService.create(wsdlDocumentLocation,
                                             serviceName);

// La SEI offrant les méthodes du service Web
HelloService helloService = service.getPort(HelloService.class);
```

Le code précédent pose un problème : le client utilise l'URL du service spécifié dans la section *wsdl:service* du WSDL. Il est souvent utile de pouvoir changer dynamiquement l'URL du endpoint.

Modification de l'URL du endpoint du service

```
URL wsdlDocumentLocation = new URL("file:HelloService.wsdl");
QName serviceName = new QName("http://epsi/","HelloServiceService");

Service service = HelloServiceService.create(wsdlDocumentLocation,
                                             serviceName);

HelloService helloService = service.getPort(HelloService.class);

// Un service implémente l'interface javax.xml.ws.BindingProvider
BindingProvider bp = (BindingProvider) helloService;
// endpointUrl est une variable donnant l'URL du endpoint
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                           endpointUrl);
```

Pour une application Java EE, le stub client est une ressource que l'on peut injecter avec l'annotation **@WebServiceRef (http://docs.oracle.com/javaee/6/api/javax/xml/ws/WebServiceRef.html)** .

```
import javax.ejb.Stateless;
import javax.xml.ws.WebServiceRef;

import epsi.HelloService

@Stateless
public class MyWebServiceClient {

  @WebServiceRef
  private HelloService helloService;

  // ...
}
```

Attention, le stub client n'est pas **thread-safe** !