Services Web – WSDL

Fabrice Rossi

http://apiacoa.org/contact.html.

Université Paris-IX Dauphine

Plan du cours WSDL

1. Présentation
2. Structure générale
3. Les types
4. Les messages
5. Les types de port
6. Le *binding*
7. Les services

WSDL

Web Services Description Language :

- dialecte XML (ensemble de schémas W3)
- permet de décrire “complètement” un service web :
  - types des données
  - messages
  - opérations
  - *binding* (incarnation)
- évolution :
  - version 1.1 : IBM et MS, note W3 du 15/03/01 `http://www.w3.org/TR/wsdl`, destinée à SOAP 1.1
  - version 1.2 : en cours de définition par le W3, destinée à SOAP 1.2 (encore du travail !)
- dans ce cours : version 1.1

Rôle pratique

- Assez semblable à l'Interface Definition Language (IDL) de CORBA
- Fournit une description d'un service web indépendante du langage et de la plate-forme
- Sert d'espéranto entre le serveur et le client. Exemple :
  - service programmé en VB.NET
  - description WSDL engendrée automatiquement par les outils .NET
  - interface `Remote` engendrée automatiquement à partir du WSDL par les outils `Java`
  - *stubs* engendrés automatiquement à partir du WSDL par les outils `Java`
  - client programmé en `Java`

Verbeux

WSDL est assez verbeux :

- api amazon :
  - 1150 lignes de WSDL
  - pour 23 opérations (34 lignes de `Java`)
  - engendre 12000 lignes (!) de code `Java` (avec axis)
  - essentiel de la taille : définition des types (et mapping `Java`)
- api google :
  - 200 lignes de WSDL
  - pour 3 opérations (14 lignes de `Java`)
  - engendre 1075 lignes de code `Java`
  - beaucoup moins de types dans ce service

Concepts

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

Exemple

HelloWorld.wsdl

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions name="HelloWorld"
3          targetNamespace="http://hello.jaxrpc.samples/"
4          xmlns:tns="http://hello.jaxrpc.samples/"
5          xmlns="http://schemas.xmlsoap.org/wsdl/"
6          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
8    <types />
9    <message name="sayHello">
10     <part name="String_1" type="xsd:string" />
11   </message>
12   <message name="sayHelloResponse">
13     <part name="result" type="xsd:string" />
14   </message>
15   <portType name="Hello">
16     <operation name="sayHello" parameterOrder="String_1">
17       <input message="tns:sayHello" />
18       <output message="tns:sayHelloResponse" />
19     </operation>
20   </portType>
```

Exemple (2)

HelloWorld.wsdl

```
21    <binding name="HelloBinding" type="tns:Hello">
22      <operation name="sayHello">
23        <input>
24          <soap:body
25          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
26          use="encoded" namespace="http://hello.jaxrpc.samples/" />
27        </input>
28        <output>
29          <soap:body
30          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
31          use="encoded" namespace="http://hello.jaxrpc.samples/" />
32        </output>
33        <soap:operation soapAction="" />
34      </operation>
35      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
36      style="rpc" />
37    </binding>
38    <service name="HelloWorld">
39      <port name="HelloPort" binding="tns:HelloBinding">
40        <soap:address
41        location="http://localhost:8080/axis/Hello" />
42      </port>
43    </service>
44  </definitions>
```

Exemple (3)

- le service `HelloWorld`
- propose l'opération `sayHello` (qui prend comme paramètre une chaîne de caractères et renvoie une chaîne de caractères)
- et est accessible sur `http://localhost:8080/axis/Hello`
- par des messages SOAP
- sous forme RPC au dessus de HTTP

Structure générale

Un document WSDL est organisé de la façon suivante :

- l'espace de noms principal est `http://schemas.xmlsoap.org/wsdl/` (préfixe classique `wsdl`)
- la racine du document est `wsdl:definitions`
- les fils directs de la racine sont (dans l'ordre)
  - `wsdl:types` (facultatif) : définition des types des données
  - `wsdl:message` (nombre quelconque) : définition des messages échangeables
  - `wsdl:portType` (nombre quelconque) : définition des ensembles d'opérations
  - `wsdl:binding` (nombre quelconque) : définition des *bindings*
  - `wsdl:service` (nombre quelconque) : définition des services web

Structure générale (2)

- partie abstraite :
  - les types
  - les messages
  - les types de port
- partie concrète :
  - les *bindings*
  - les services
- la partie concrète propose une ou plusieurs réalisations de la partie abstraite (par exemple, un type de port peut être réalisé par SOAP+RPC+HTTP et/ou par SOAP+RPC+STMP)

Constructions générales

- les “objets” définis en WSDL peuvent être placés dans un NS, grâce à l’attribut `targetNamespace` de `wsdl:definitions`
- les références internes sont gérées grâce à l’introduction d’un préfixe pour ce NS (en général `tns`)
- la construction `wsdl:import` permet de découper le WSDL en plusieurs parties :
  - `namespace` : espace de noms dans lequel sont placées les “objets” définis par le fichier inclus
  - `location` : emplacement du fichier inclus (URI)
  - une application classique : séparer les types (schéma W3) du reste
- `wsdl:document` : permet d’ajouter de la documentation à un fichier WSDL (le format est du XML libre)

Exemple de séparation

HelloWorldAbstract.wsdl

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions name="HelloWorld"
3          targetNamespace="http://hello.jaxrpc.samples/"
4          xmlns:tns="http://hello.jaxrpc.samples/"
5          xmlns="http://schemas.xmlsoap.org/wsdl/"
6          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
8    <types />
9    <message name="sayHello">
10     <part name="String_1" type="xsd:string" />
11   </message>
12   <message name="sayHelloResponse">
13     <part name="result" type="xsd:string" />
14   </message>
15   <portType name="Hello">
16     <operation name="sayHello" parameterOrder="String_1">
17       <input message="tns:sayHello" />
18       <output message="tns:sayHelloResponse" />
19     </operation>
20   </portType>
21 </definitions>
```

Exemple de séparation (2)

HelloWorldBind.wsdl

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions name="HelloWorld"
3          targetNamespace="http://hello.jaxrpc.samples/"
4          xmlns:tns="http://hello.jaxrpc.samples/"
5          xmlns="http://schemas.xmlsoap.org/wsdl/"
6          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
8    <import namespace="http://hello.jaxrpc.samples/"
9                location="HelloWorldAbstract.wsdl"/>
10   <binding name="HelloBinding" type="tns:Hello">
11     <operation name="sayHello">
12       <input>
13         <soap:body
14         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
15         use="encoded" namespace="http://hello.jaxrpc.samples/" />
16       </input>
17       <output>
18         <soap:body
19         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
20         use="encoded" namespace="http://hello.jaxrpc.samples/" />
21       </output>
22       <soap:operation soapAction="" />
23     </operation>
24     <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
25     style="rpc" />
26   </binding>
```

Exemple de séparation (3)

HelloWorldBind.wsdl

```
27    <service name="HelloWorld">
28      <port name="HelloPort" binding="tns:HelloBinding">
29        <soap:address
30        location="http://localhost:8080/axis/Hello" />
31      </port>
32    </service>
33  </definitions>
```

Définition de types

- inutile si on se contente de types de base
- donnée par un élément `wsdl:types`
- le contenu de cet élément peut être un schéma du W3 :
  - espace de noms `http://www.w3.org/2001/XMLSchema` (préfixe classique `xsd`)
  - pas d'attribut (les données sont représentées sans utiliser d'attribut), sauf pour les tableaux
  - types tableaux obtenus par dérivation à partir de `Array` (définit sans le NS d'*encoding* de SOAP)
  - type des éléments d'un tableau spécifiés par l'attribut `wdsl:arrayType`
- en théorie, on peut utiliser un autre langage pour définir les types (le *draft* de WSDL 1.2 donne comme exemple RELAX NG)

Exemple étudié

On va étudier un modèle de service web simple :

- service PhoneBookWS
- propose une opération :
  - `getPhoneNumber`
  - prend comme paramètre une personne (prénom et nom)
  - renvoie comme résultat un tableau d'entiers (numéro séparé en composantes)

Exemple

PersonType.wsdl

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions
3     targetNamespace="urn:PhoneBook"
4     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
5     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
6   <wsdl:types>
7    <xsd:schema targetNamespace="urn:PhoneBook">
8     <xsd:complexType name="Person">
9      <xsd:sequence>
10      <xsd:element name="first" type="xsd:string"/>
11      <xsd:element name="last" type="xsd:string"/>
12     </xsd:sequence>
13    </xsd:complexType>
14   </xsd:schema>
15  </wsdl:types>
16 </wsdl:definitions>
```

Person.xml

```
1  <param xsi:type="Person">
2   <first xsi:type="xsd:string">John</first>
3   <last xsi:type="xsd:string">Doe</last>
4  </param>
```

Autre exemple

IntArrayType.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   targetNamespace="urn:PhoneBook"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <xsd:schema targetNamespace="urn:PhoneBook">
   <xsd:complexType name="ArrayOf_xsd_int">
    <xsd:complexContent>
     <xsd:restriction base="soapenc:Array">
      <xsd:attribute ref="soapenc:arrayType"
                     wsdl:arrayType="xsd:int[]"/>
     </xsd:restriction>
    </xsd:complexContent>
   </xsd:complexType>
  </xsd:schema>
 </wsdl:types>
</wsdl:definitions>
```

IntArray.xml

```
<return xsi:type="ArrayOf_xsd_int" enc:arrayType="xsd:int[3]">
 <item>33</item>
 <item>0</item>
 <item>144054405</item>
</return>
```

Définir les Messages

- les messages échangés par les services sont décrits par des éléments `wsdl:message`
- chaque message possède un nom (attribut `name`) et est constitué de parties
- chaque partie est décrite par un fils `wsdl:part` précisé par :
  - son nom (attribut `name`)
  - soit son type (attribut `type`)
  - soit directement le nom de l'élément qui la constitue (attribut `element`)
- la définition des parties utilise le mécanisme de référence classique des documents XML :
  - les types et éléments définis sont placés dans NS (`targetNamespace`)
  - on associe un préfixe à ce NS

Exemple

PhoneMessages.wsdl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <wsdl:definitions ...>
3    <!-- définition des types -->
4    <wsdl:message name="getPhoneNumberResponse"
5                  xmlns:tns="urn:PhoneBook">
6    <!-- l'association tns au targetNamespace devrait plutôt se faire
7         dans wsdl:definitions -->
8      <wsdl:part name="getPhoneNumberReturn" type="tns:ArrayOf_xsd_int"/>
9    </wsdl:message>
10   <wsdl:message name="getPhoneNumberRequest"
11                 xmlns:tns="urn:PhoneBook">
12     <wsdl:part name="in0" type="tns:Person"/>
13   </wsdl:message>
14 </wsdl:definitions>
```

- deux messages (appel de l'opération et résultat de celle-ci)
- une seule partie pour chaque message

Définir les types de port

- un type de port ressemble à une interface `Java`
- les types de port sont décrits par des éléments `wsdl:portType`
- chaque type de port est identifié par un nom (attribut `name`)
- un type de port décrit un ensemble d'opérations, chacune précisée par un élément `wsdl:operation` :
  - identifiée par un nom (attribut `name`)
  - contenant une spécification des messages échangés pour réaliser l'opération
- quatre modèles pour les opérations (basés sur le contenu de l'élément `wsdl:operation`) :
  - envoi de message (*One way*)
  - question – réponse (*Request–response*)
  - sollicitation – réponse (*Solicit–response*)
  - alerte (*Notification*)

Une opération

- trois sous-éléments possibles pour une opération :
  - `wsdl:input` : message reçu par le service
  - `wsdl:output` : message produit par le service
  - `wsdl:fault` : message d'erreur (produit par le service)
- chaque sous-élément est précisé par des attributs :
  - `name` : donne un nom (au niveau de l'opération) au message (facultatif, noms par défaut obtenus automatiquement)
  - `message` : type du message, référence à un message défini par un élément `wsdl:message`
- surcharge :
  - plusieurs opérations peuvent avoir le même nom
  - à condition qu'elles diffèrent par les noms des messages d'entrée et/ou de sortie
  - **Attention** : cette possibilité est actuellement supprimée du brouillon pour WSDL 1.2

Modèles d'opérations

Les deux modèles les plus classiques :

- envoi de message (approche message) :
  - le client envoie un message et n'attend pas de réponse du service
  - un seul message `wsdl:input`
- question – réponse (approche RPC ou échange de documents) :
  - le client envoie un message auquel le service répond
  - un `wsdl:input`, suivi d'un `wsdl:output` et d'éventuels `wsdl:faut`
- attention : pas de message d'erreur dans le modèle d'envoi de messages

Modèles d'opérations (2)

Modèles avec dialogue inversé, nécessitant vraisemblablement un abonnement préalable :

- sollicitation – réponse (approche RPC inversée) :
  - le client reçoit un message du service et répond au service
  - un `wsdl:output`, suivi d'un `wsdl:input` et d'éventuels `wsdl:faut`
- alerte :
  - le client reçoit un message du service mais ne doit pas répondre
  - un `wsdl:output`
- pas de message d'erreur dans le modèle alerte

Beaucoup moins utilisés dans la pratique.

Exemple

PhoneBookPortType.wsdl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <wsdl:definitions ...>
3    <!-- définition des types et des messages -->
4    <wsdl:portType name="PhoneBook" xmlns:tns="urn:PhoneBook">
5      <wsdl:operation name="getPhoneNumber" parameterOrder="in0">
6        <wsdl:input message="tns:getPhoneNumberRequest"
7          name="getPhoneNumberRequest"/>
8        <wsdl:output message="tns:getPhoneNumberResponse"
9          name="getPhoneNumberResponse"/>
10     </wsdl:operation>
11   </wsdl:portType>
12 </wsdl:definitions>
```

Attribut `parameterOrder` (facultatif)

- utilisé comme aide pour le modèle RPC (s'applique aux modèles question – réponse et sollicitation – réponse)
- donne l'ordre souhaité pour les paramètres de l'opération sous forme des noms des parties des messages échangés (en `input` comme en `output`)

Définir les *bindings*

La partie pénible de WSDL :

- un *binding* propose une réalisation concrète d'un type de port
- élément racine `wsdl:binding`
- précisé par un attribut `name` : le nom du *binding*
- et par un attribut `type` : le type de port concerné par le *binding*
- contient un élément `wsdl:operation` pour chaque opération du type de port (attribut `name` pour indique l'opération concernée)
- chaque élément `wsdl:operation` contient des éléments définissant le *binding* des messages associés (à chaque fois précisé par un attribut `name`) :
  - `wsdl:input`
  - `wsdl:output`
  - `wsdl:fault`

Modèle général

BindingModele.wsdl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <wsdl:definitions ...>
3    <wsdl:binding name="unPortBinding" type="tns:unPort">
4      <!-- extension -->
5      <wsdl:operation name="uneOpération">
6        <!-- extension -->
7        <wsdl:input name="uneOpérationRequest">
8          <!-- extension -->
9        </wsdl:input>
10       <wsdl:output name="uneOpérationResponse">
11         <!-- extension -->
12       </wsdl:output>
13       <wsdl:fault name="uneOpérationFault">
14         <!-- extension -->
15       </wsdl:fault>
16     </wsdl:operation>
17   </wsdl:binding>
18 </wsdl:definitions>
```

Le *binding* SOAP

- WSDL définit d'abord une coquille vide pour le *binding*
- un *binding* se fait grâce à des éléments additionnels (qui **ne sont pas** dans le NS WSDL)
- la norme spécifie les éléments permettant de réaliser un *binding* vers SOAP :
  - espace de noms `http://schemas.xmlsoap.org/wsdl/soap/` (préfixe classique `soap` ou `wsdlsoap`)
  - définit quelques éléments et attributs :
    - `soap:binding` (fils direct de `wsdl:binding`)
    - `soap:operation` (fils direct de `wsdl:operation`)
    - `soap:body`, `soap:header` et `soap:headerfault` (fils direct de `wsdl:input` et `wsdl:output`)
    - `soap:fault` (fils direct de `wsdl:fault`)

Exemple

PhoneBookBinding.wsdl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<wsdl:definitions ...>
  <!-- définition des types, des messages et des types de port-->
  <wsdl:binding name="PhoneBookWSSoapBinding"
    xmlns:tns="urn:PhoneBook" type="tns:PhoneBook">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getPhoneNumber">
      <soap:operation soapAction=""/>
      <wsdl:input name="getPhoneNumberRequest">
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   namespace="http://localhost:8080/axis/services/PhoneBookWS"
                   use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getPhoneNumberResponse">
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   namespace="http://localhost:8080/axis/services/PhoneBookWS"
                   use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Le *binding* SOAP (2)

- `soap:binding` :
  - précise qu'on utilise SOAP
  - l'attribut `transport` indique le protocole utilisé pour l'échange des messages SOAP, (en général HTTP précisé par l'URI `http ://schemas.xmlsoap.org/soap/http`)
  - l'attribut `style` précise si SOAP doit fonctionner en mode `rpc` ou `document` (cf la description de `soap:body`)
- `soap:operation` :
  - l'attribut `soapAction` donne la valeur du *header* HTTP correspondant
  - l'attribut `style` a le même rôle que dans `soap:binding`

Le *Body* des messages SOAP

- la partie subtile (pénible ?)
- `soap:body` :
  - précise le format des messages échangés par une opération sous forme de contraintes sur le `Body` du format SOAP
  - l'attribut `parts` permet de préciser à quelles parties du message les contraintes s'appliquent
  - l'attribut `use` précise comment le format des messages (défini par les éléments `wsdl:message`) doit être compris
- si le `style` englobant est `rpc`, le `Body` contient :
  - un élément englobant portant le nom de l'opération
  - l'élément englobant contient un élément par partie dans le message considéré (portant le nom de la partie), dans l'ordre rpc éventuellement précisé par `parameterOrder`
  - les paramètres de l'opération sont contenus dans les éléments correspondant aux parties

Le *Body* des messages SOAP (2)

- si le `style` englobant est `document`, le `Body` contient **directement** les parties du messages : SOAP comme protocole d'échange de documents XML (très utilisé par `.NET`)
- si `use` vaut `encoded` (le cas le plus classique) :
  - l'attribut `encodingStyle` précise le mécanisme de représentation XML (valeur classique `http://schemas.xmlsoap.org/soap/encoding/`, le format de SOAP)
  - chaque partie de message est typée par un attribut `type`
  - l'attribut `namespace` précise le NS des types des parties de messages
- si `use` vaut `literal` (plutôt utilisé pour le style document) :
  - il n'y a pas de transformation des parties de messages
  - elles apparaissent directement

Message SOAP encoded

PhoneSoap-encoded.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <soapenv:Envelope
3    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
4    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6   <soapenv:Body>
7    <ns1:getPhoneNumber
8      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
9      xmlns:ns1="http://localhost:8080/axis/services/PhoneBookWS">
10    <in0 href="#id0"/>
11   </ns1:getPhoneNumber>
12   <multiRef id="id0"
13     soapenc:root="0"
14     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
15     xsi:type="ns2:Person"
16     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
17     xmlns:ns2="urn:PhoneBookWS">
18    <first xsi:type="xsd:string">John</first>
19    <last xsi:type="xsd:string">Doe</last>
20   </multiRef>
21  </soapenv:Body>
22 </soapenv:Envelope>
```

Message SOAP literal

PhoneSoap-literal.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <soapenv:Envelope
3    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
4    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6   <soapenv:Body>
7    <getPhoneNumber
8       xmlns="http://localhost:8080/axis/services/PhoneBookWS">
9     <in0 xmlns="">
10     <first>John</first>
11     <last>Doe</last>
12    </in0>
13   </getPhoneNumber>
14  </soapenv:Body>
15 </soapenv:Envelope>
```

Le *Body* des messages SOAP (3)

- Pourquoi ?! :
  - pour le style `rpc`, l'utilisation de `encoded` et du mécanisme de traduction de SOAP est naturel
  - pour le style `document`, la traduction SOAP est gênante (par exemple, il n'y a pas d'attribut)
- même mécanisme que `soap:body` pour :
  - `soap:fault`
  - `soap:header`
  - `soap:headerfault`

Définir un service

- un service est une collection de ports
- un élément `wsdl:service` portant un nom (attribut `name`)
- contenant un élément `wsdl:port` par port, précisé par
  - un attribut `name` donnant le nom du port
  - un attribut `binding` donnant le nom du *binding* associé
- ce n'est qu'un cadre général : des éléments d'extension (sous-éléments de `wsdl:port`) précisent la définition
- pour SOAP, un élément `soap:address` précise l'URI du port grâce à son attribut `location`

Exemple

PhoneBookService.wsdl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <wsdl:definitions ...>
3   <!-- types, messages, types de port et binding -->
4    <wsdl:service name="PhoneBookService" xmlns:tns="urn:PhoneBook">
5      <wsdl:port binding="tns:PhoneBookWSSoapBinding"
6          name="PhoneBookWS">
7        <soap:address
8           location="http://localhost:8080/axis/services/PhoneBookWS"/>
9      </wsdl:port>
10   </wsdl:service>
11 </wsdl:definitions>
```

Autres extensions

- `http` :
  - permet de définir un *binding* basé sur HTTP sans pour autant utiliser des messages SOAP
  - permet d'utiliser des requêtes GET
- `mime` :
  - permet d'utiliser MIME pour préciser le contenu des messages
  - peut s'utiliser en conjonction avec HTTP
  - le type `multipart/related` permet de fournir des messages avec pièces jointes

Conclusion

- WSDL est une généralisation de la notion de définition d'objets proposée par IDL (mais ce n'est pas un modèle objet)
- WSDL offre beaucoup plus de souplesse qu'IDL :
  - choix de la représentation des messages
  - choix du protocole
  - plusieurs implémentations concrètes (*binding*) pour un même service
  - etc.
- ne résout pas tous les problèmes (!) :
  - publication et découverte avec une description de haut niveau (UDDI)
  - faire le lien entre l'aspect technique (WSDL) et l'aspect haut niveau (UDII) : Web Services Inspection Language (WSIL)
  - déploiement, appel, etc.