



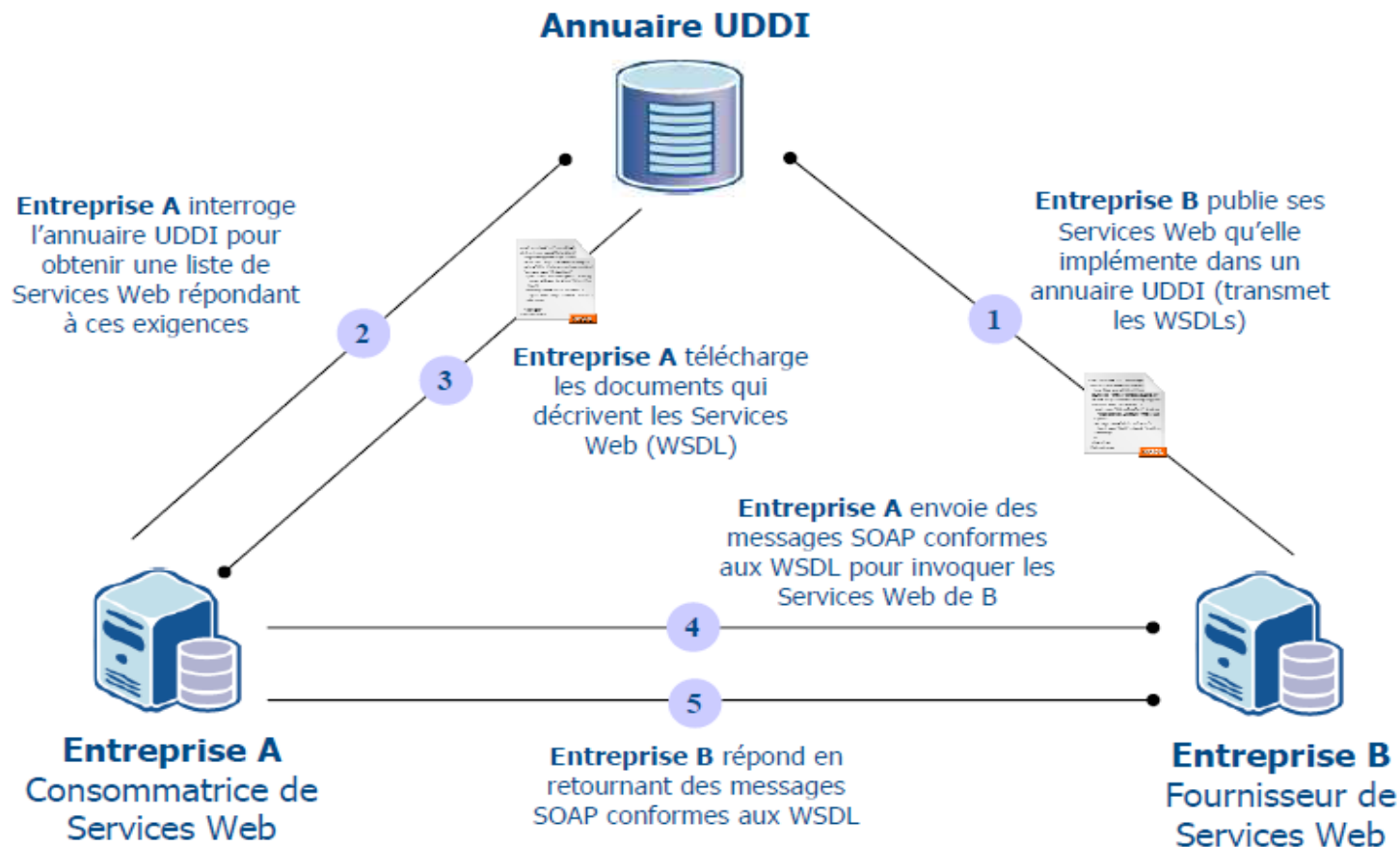
Architecture orientée services (SOA)

Services Web étendus

I. Mouakher

Services Web étendus

Chercher, Publier et Consommer



Standards de l'architecture

Les standards sont un élément clé d'une SOA, ils assurent l'interopérabilité



SOAP

W3C

Simple Object
Access Protocol

Transporte



WSDL

W3C

Web Services
Description Language

Décrit le contrat

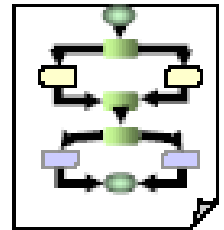


UDDI

Microsoft, IBM, HP

Universal Description
Discovery and Integration

**Spec pour
Repository/Registry**



BPEL

Oasis

Business Process
Execution Language

**Décrit les
processus métier**

Les trois piliers des Services Web

Services Web étendus

◆ Pile des standards pour les Services Web étendus

Langages de Processus Métier (BPL)
BPEL

Orchestration

Sécurité
WS-Security

Fiabilité
WS-RM

Transaction
WS-Transactions

Qualité de Service

WSDL, UDDI

Découverte &
Description

SOAP 1.1 et 1.2

Message

HTTP, SMTP, FTP, BEEP

Transport



Web Service Description Language (WSDL)

Plan

- ◆ Introduction
- ◆ Document WSDL
- ◆ Description abstraite
- ◆ Description concrète

Généralités WSDL

- ◆ Basé sur le langage XML et permet de décrire un service Web
- ◆ Fournit une description indépendante du langage et de la plate-forme
- ◆ Par comparaison WSDL est assez semblable au langage IDL défini par CORBA
- ◆ Spécification du W3C
 - WSDL 1.1 : <http://www.w3.org/TR/wsdl>
 - WSDL 2.0 : <http://www.w3.org/TR/wsdl20/>
- ◆ A partir d'un document WSDL il est possible
 - Générer un client pour appeler un Service Web
 - Générer le code pour implémenter un Service Web

Document WSDL

- ◆ Défini en XML
- ◆ Structuré comme un ensemble de définitions
 - élément racine <definitions>
- ◆ Modulaire
 - Fragmentation des définitions en plusieurs fichiers
 - ◆ Séparation des descriptions abstraites et concrètes
- ◆ Réutilisation de définitions de services
 - import d'autres documents WSDL et XSD

Document WSDL - En-tête

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions name="HelloWorld"
  targetNamespace="http://helloworldwebservice.lisi.ensma.fr/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://helloworldwebservice.lisi.ensma.fr/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
```

.....

```
</definitions>
```

Concepts d'un document WSDL

- ◆ **Une donnée** : information typée
- ◆ **Un message** : regroupe un ensemble de données
- ◆ **Une opération** : action fournie par le Service Web (~ méthode au sens Java)
- ◆ **Un type de port** : ensemble d'action (~ interface au sens Java)
- ◆ **Un binding** : définit pour un type de port le protocole utilisé pour transmettre les informations et le format des données
- ◆ **Un port** : définit où est localisé le Service Web et le binding à utiliser
- ◆ **Un service** : un ensemble de ports

Description abstraite

- ◆ Définition de l'interface du service : décrit les messages et les opérations disponibles
- ◆ <types>
 - Définition des types de données utilisés dans les messages
 - Référence à un système de typage tel que Schéma XML
- ◆ <message>
 - Définition typée abstraite des données échangées
 - Ex : appel de méthode, retour de l'appel
- ◆ <operation>
 - Définition abstraite d'un ensemble cohérent de messages (entrées/sorties) correspondant à l'interaction avec le Web Service
 - Ex: appel de méthode + retour de l'appel
- ◆ <portType> (*type de port*)
 - Définition abstraite d'un ensemble d'opérations

Élément Types

- ◆ L'élément <types> contient la définition des types utilisés pour décrire la structure des messages échangés par le Web Service
- ◆ Le système de typage est généralement un Schema XSD mais d'autres systèmes sont autorisés (RELAX NG par exemple)
- ◆ Cet élément peut être facultatif si les types utilisés par les messages sont des types de bases (Integer, Boolean, ...)
- ◆ Dans le cas de structures complexes (Person par exemple) un Schema XML est alors employé

Elément Types – Exemple 1

```
<types>
  <xsd:schema targetNamespace="http://notebookwebservice.lisi.ensma.fr/">
    <xsd:complexType name="person">
      <xsd:sequence>
        <xsd:element name="address" type="xs:string" minOccurs="0"/>
        <xsd:element name="birthyear" type="xs:string" minOccurs="0"/>
        <xsd:element name="name" type="xs:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="personArray" final="#all">
      <xsd:sequence>
        <xsd:element name="item" type="tns:person" minOccurs="0"
          maxOccurs="unbounded" nillable="true"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

Élément Types

- ◆ La définition des types peut également être importée à partir d'un fichier Schema XML
- ◆ Le fichier XML est accessible au même titre que le document WSDL
- ◆ L'adresse de l'hôte du Schema XML n'est pas forcément la même que celle du document WSDL
- ◆ Cette séparation permet
 - de réutiliser des types dans plusieurs WSDL différents
 - d'éviter d'alourdir le document WSDL
- ◆ Par la suite nous privilégierons la séparation des types du document WSDL

Élément Types – Exemple 2

```
<types>
<xsd:schema>
<xsd:import namespace="http://notebookwebservice.lisi.ensma.fr/"
schemaLocation="Notebook_schema1.xsd"/>
</xsd:schema>
</types>
```

↳ Import le fichier XSD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
targetNamespace="http://notebookwebservice.lisi.ensma.fr/"
...>
<xs:complexType name="person">
<xs:sequence>
<xs:element name="address" type="xs:string" minOccurs="0"/>
<xs:element name="birthyear" type="xs:string" minOccurs="0"/>
<xs:element name="name" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="personArray" final="#all">
<xs:sequence>
<xs:element name="item" type="tns:person" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

Élément Message

- ◆ L'élément <message> permet de décrire les messages échangés par les services
 - Paramètres d'entrées des opérations
 - Paramètres de sorties
 - Exception
- ◆ Chaque <message> est identifié par un nom (attribut name) et est constitué d'un ensemble d'éléments <part>
- ◆ En quelque sorte un élément <part> correspond à un paramètre d'une opération
- ◆ Si une opération est décrite par plusieurs paramètres, plusieurs éléments <part> seront à définir
- ◆ L'élément <part> est défini par
 - un nom (attribut name)
 - un type (attribut type)

Élément Message - Exemple

Message utilisé pour l'appel d'une opération avec une seule partie



```
<message name="addPersonWithComplexType">  
  <part name="newPerson" type="tns:person"/>  
</message>
```

```
<message name="addPersonWithComplexTypeResponse">  
  <part name="addPersonWithComplexTypeResult" type="xsd:boolean"/>  
</message>
```

```
<message name="addPersonWithSimpleType">  
  <part name="name" type="xsd:string"/>  
  <part name="address" type="xsd:string"/>  
  <part name="birthyear" type="xsd:string"/>  
</message>
```

Message utilisé pour l'appel d'une opération avec trois parties

Élément portType

- ◆ Un élément <portType> est un regroupement d'opérations et peut comparé à une interface Java
- ◆ Caractéristique d'un élément <portType>
 - Identifiable par un nom (attribut name)
 - Composé de sous élément <operation>
- ◆ Une opération est comparable une méthode Java
 - Identifiable par un nom (attribut name)
 - La description des paramètres est obtenue par une liste de messages
- ◆ Un document WSDL peut décrire plusieurs « port Types »

Élément opération


- ◆ Une opération exploite les messages via les sous éléments
 - `<input>` : message transmis au service
 - `<output>` : message produit par le service
 - `<fault>` : message d'erreur (très proche des exceptions)
- ◆ Chaque sous élément possède les attributs suivants
 - `name` : nom explicite donné au message (optionnel)
 - `message` : référence à un message (défini précédemment)
- ◆ La surcharge d'opération est autorisée sous condition Messages `<input>` et/ou `<output>` soient différents

Élément opération - Exemple

◆ L'opération addPerson est surchargée

```
<portType name="Notebook">
  <operation name="addPerson">
    <input message="tns:addPersonWithComplexType"/>
    <output message="tns:addPersonWithComplexTypeResponse"/>
  </operation>

  <operation name="addPerson" parameterOrder="name address birthyear">
    <input message="tns:addPersonWithSimpleType"/>
  </operation>
  ....
</portType>
```



Possibilité de fixer l'ordre des paramètres définis par cette opération

Élément opération

- ◆ Quatre types d'opérations possibles

- Combinaison de messages
- Utilisation des balises <input>, <output>

- ◆ Opération one-way Web

- Le service reçoit un message <input>.

- ◆ Opération request-response

- Le service reçoit un message requête <input> puis renvoie au client un message réponse <output> ou un message erreur <fault>.

- ◆ Opération solicit-response

- Le service envoie un message sollicitation <output> puis reçoit du client un message réponse <input> ou un message d'erreur <fault>.

- ◆ Opération notification

- Le service envoie un message notification <output>.



Exemple - Service HelloWorld

- ◆ Le service HelloWorld fournit deux opérations
 - Une opération makeHello qui prend en paramètre une chaîne de caractères et retourne une chaîne caractères
 - Une opération simpleHello sans paramètre en entrée et retourne une chaîne de caractères
- ◆ L'accès au service est réalisé par l'intermédiaire de messages SOAP (étudié en détail dans le prochain cours)
- ◆ Le protocole utilisé pour l'échange des messages SOAP est HTTP
- ◆ Le style utilisé est du RPC

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions name="HelloWorld"
  targetNamespace="http://helloworldwebservice.lisi.ensma.fr/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://helloworldwebservice.lisi.ensma.fr/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types/>
  <message name="makeHelloWorld">
    <part name="value" type="xsd:string"/>
  </message>
  <message name="makeHelloWorldResponse">
    <part name="helloWorldResult" type="xsd:string"/>
  </message>
  <message name="simpleHelloWorld"/>
  <message name="simpleHelloWorldResponse">
    <part name="helloWorldResult" type="xsd:string"/>
  </message>
  <portType name="HelloWorld">
    <operation name="makeHelloWorld">
      <input message="tns:makeHelloWorld"/>
      <output message="tns:makeHelloWorldResponse"/>
    </operation>
    <operation name="simpleHelloWorld">
      <input message="tns:simpleHelloWorld"/>
      <output message="tns:simpleHelloWorldResponse"/>
    </operation>
  </portType>

```

```
<binding name="HelloWorldPortBinding" type="tns:HelloWorld">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  <operation name="makeHelloWorld">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="http://helloworldwebservice.lisi.ensma.fr"/>
    </input>
    <output>
      <soap:body use="literal" namespace="http://helloworldwebservice.lisi.ensma.fr"/>
    </output>
  </operation>
  <operation name="simpleHelloWorld">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="http://helloworldwebservice.lisi.ensma.fr"/>
    </input>
    <output>
      <soap:body use="literal" namespace="http://helloworldwebservice.lisi.ensma.fr"/>
    </output>
  </operation>
</binding>
<service name="HelloWorld">
  <port name="HelloWorldPort" binding="tns:HelloWorldPortBinding">
    <soap:address location="TODO"/>
  </port>
</service>
</definitions>
```


Description concrète

- ◆ Définition de l'implantation d'un service donné
- ◆ <binding> (*liaison*)
 - Description des protocoles (SOAP, HTTP...) et des formats de messages concrets pour chaque <porttype>
- ◆ <port>
 - Définition d'un point d'entrée en associant un <binding> à une adresse réseau d'un serveur (URL).
- ◆ <service>
 - Définition d'un Web Service comme un ensemble de <port>.

Élément Binding (1/2)

- ◆ Un élément <binding> permet de réaliser la partie concrète d'un élément <portType>
 - un nom (attribut name)
 - un portType (attribut type)
- ◆ Il décrit précisément le protocole à utiliser pour manipuler un élément <portType>
 - SOAP 1.1 et 1.2
 - HTTP GET & Post (pour le transfert d'images par exemple)
 - MIME
- ◆ Plusieurs éléments <binding> peuvent être définis de sorte qu'un élément portType peut être appelé de différentes manières
- ◆ La structure de l'élément <binding> dépend du protocole utilisé

Élément Binding (2/2)

◆ Structure générale de l'élément <binding>

```
<definitions>
...
<binding name="NamePortBinding" type="tns:portType">
  <!-- Décrit le protocole à utiliser -->
  <operation name="operation1">
    <!-- Action du protocole sur l'opération -->
    <input>
      <!-- Action du protocole sur les messages d'entrés (input) -->
    </input>
    <output>
      <!-- Action du protocole sur les messages de sorties (ouput) -->
    </output>
    <fault>
      <!-- Action du protocole sur les messages d'erreurs (fault) -->
    </fault>
  </operation>
</binding>
...
</definitions>
```

Le binding SOAP

- ◆ Le schema XML de WSDL ne décrit pas les sous éléments de binding, operation, input, output et fault. Ces éléments sont spécifiques aux protocoles utilisés
- ◆ Le binding SOAP est défini par l'espace de noms suivant
 - <http://schemas.xmlsoap.org/wsdl/soap/>
 - Préfixe utilisé est généralement soap (de la forme <soap:binding>)
- ◆ Différentes versions peuvent être utilisées : 1.1 et 1.2
- ◆ Les principales balises à exploiter dans le binding sont
 - <soap:binding>
 - <soap:operation>
 - <soap:body>, <soap:header>, <soap:headerfault>
 - <soap:fault>
- ◆ A noter que nous détaillerons certains aspects dans le cours consacré au protocole SOAP

Le binding SOAP - Exemple

```
<binding name="HelloWorldPortBinding" type="tns:HelloWorld">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  <operation name="makeHelloWorld">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="http://helloworldwebservice.lisi.ensma.fr"/>
    </input>
    <output>
      <soap:body use="literal" namespace="http://helloworldwebservice.lisi.ensma.fr"/>
    </output>
  </operation>
  <operation name="simpleHelloWorld">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="http://helloworldwebservice.lisi.ensma.fr"/>
    </input>
    <output>
      <soap:body use="literal" namespace="http://helloworldwebservice.lisi.ensma.fr"/>
    </output>
  </operation>
</binding>
<service name="HelloWorld">
  <port name="HelloWorldPort" binding="tns:HelloWorldPortBinding">
    <soap:address location="TODO"/>
  </port>
</service>
</definitions>
```

Le binding SOAP - Élément `<soap:binding>`

- ◆ L'élément `<soap:binding>` doit être présent lors de la définition d'un binding à base de messages SOAP

```
<definitions ...>
<!-- Définition de la partie Abstraite du WSDL -->
  <binding ...>
    <soap:binding transport="uri"? style="rpc|document"/>
  </binding>
</definitions>
```

- ◆ L'attribut `style` permet d'indiquer la façon dont sont créés les messages SOAP pour l'ensemble des opérations
 - `rpc` : encodage défini par SOAP RPC
 - `document` : encodage sous forme d'élément XML
- ◆ L'attribut `transport` permet de préciser le protocole à utiliser pour le transport des messages SOAP
 - HTTP (<http://schemas.xmlsoap.org/soap/http>), SMTP, FTP, ...
 - Le protocole HTTP est massivement utilisé pour le transport des messages SOAP

Le binding SOAP -

Élément <soap:operation>

- ◆ L'élément <soap:operation> doit être présent pour chaque opération définie dans la partie abstraite du document

```
<definitions ...>
  <!-- Définition de la partie Abstraite du WSDL -->
  <binding ...>
    <operation ...>
      <soap:operation soapAction="uri"? style="rpc|document"?>
    </operation>
  </binding>
</definitions>
```

- ◆ L'attribut soapAction permet de préciser la valeur de l'en-tête HTTP (dans notre cas la valeur sera vide)
- ◆ L'attribut style permet de préciser la façon dont sont créés les messages SOAP de l'opération en question (RPC ou document)
- ◆ Le choix entre RPC et Document n'a pas d'importance puisque le résultat peut être identique

Le binding SOAP -

Élément <soap:body>

- ◆ L'élément <soap:body> précise le format des messages échangés par une opération
- ◆ Il y a autant d'élément <soap:body> qu'il y a de messages définis par une opération (<input>, <output> et <fault>)

```
<operation ...>
  <input>
    <soap:body parts="nmtokens"? use="literal|encoded"? encodingStyle="uri-list" ? namespace="uri"?>
  </input>
  <output>
    <soap:body parts="nmtokens"? use="literal|encoded"? encodingStyle="uri-list"? namespace="uri"?>
  </output>
</operation>
```

- ◆ L'attribut use caractérise la forme des parties des messages
 - encoded : transformation suivant un mécanisme défini par l'attribut encodingStyle (n'est pratiquement plus supportée par les boîtes à outils Web)
 - literal : pas de transformation des parties des messages, elles apparaissent directement

Éléments Service et Port (1/2)

- ◆ Un élément service définit l'ensemble des points d'entrée du Service Web, en regroupant des éléments <port>
- ◆ L'élément <port> permet de spécifier une adresse pour un binding donné
- ◆ Un port est défini par deux attributs
 - name : nom du port
 - binding : nom du binding (défini précédemment)
- ◆ Le corps de l'élément <port> est spécifique au protocole utilisé pour définir le binding
- ◆ Dans le cas d'un binding de type SOAP, un élément <soap:address> précise l'URI du port
- ◆ Il est par conséquent possible d'appeler un service à des endroits différents (plusieurs éléments port)

Élément Service et Port (2/2)

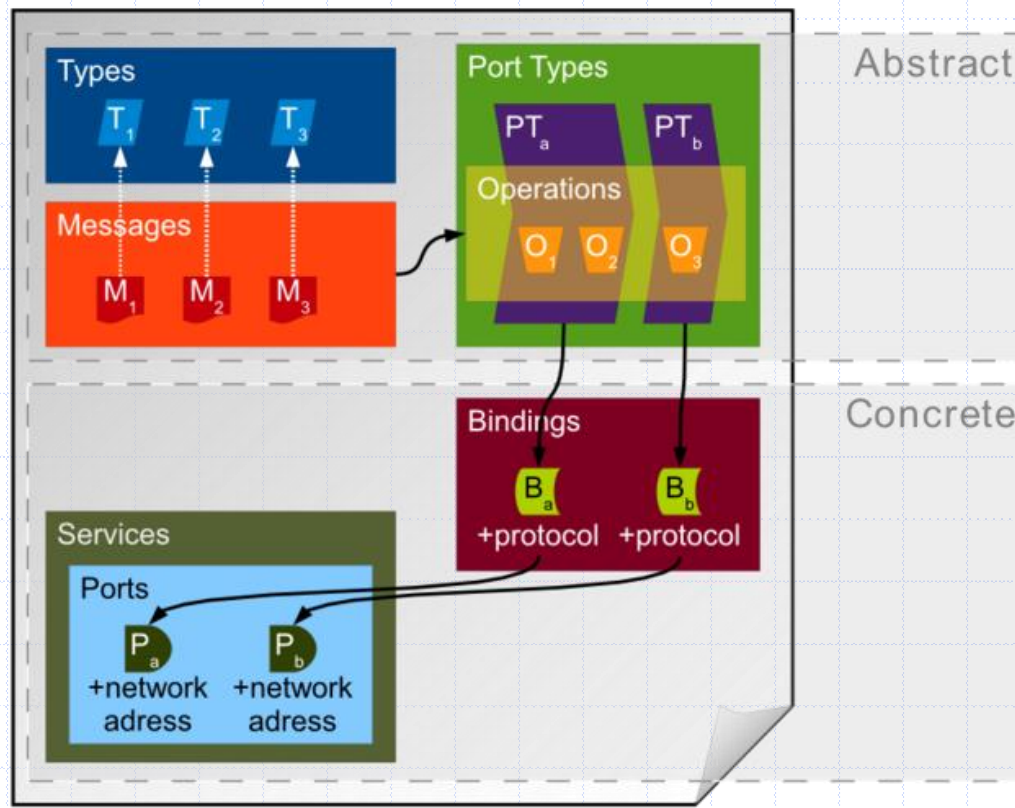
◆ Exemple : Définition d'un service

```
<definitions ...>

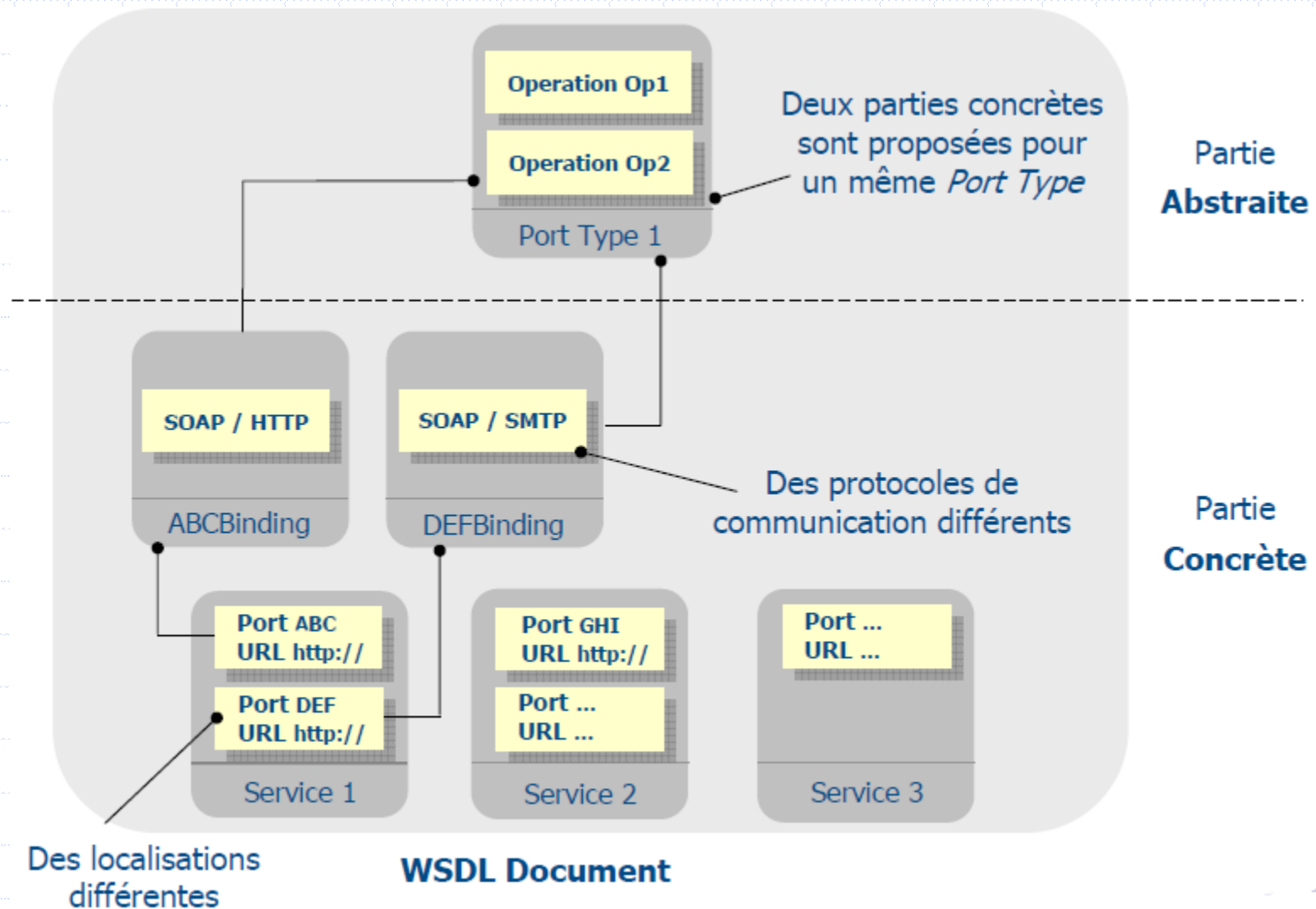
.....

<service name="Notebook">
  <port name="NoteBookPort" binding="tns:NoteBookPortBinding">
    <soap:address location="http://localhost:8080/NotebookWebService/notebook"/>
  </port>
</service>
</definitions>
```

Une représentation des concepts définis par un document WSDL 1.1



Organisation d'un document WSDL



Exemple : Carnet d'adresse (1/6)

- ◆ Le service Notebook fournit trois opérations
 - Une opération addPerson qui prend en paramètre un objet Person et retourne un booléen pour indiquer l'état de création
 - Une opération addPerson qui prend en paramètre trois chaînes de caractères (name, address et birthyear) sans retour
 - Une opération getPersonByName qui prend en paramètre une chaîne de caractère et retourne un objet Person
 - Une opération getPersons sans paramètre en entrée et qui retourne un tableau d'objets Person
- ◆ L'accès au service est réalisé par l'intermédiaire de messages SOAP (étudié en détail dans le prochain cours)
- ◆ Le protocole utilisé pour l'échange des messages SOAP est HTTP et le style utilisé est du RPC

Exemple : Carnet d'adresse (2/6)

```
<definitions
  name="Notebook"
  targetNamespace="http://notebookwebservice.lisi.ensma.fr/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://notebookwebservice.lisi.ensma.fr/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <xsd:schema targetNamespace="http://notebookwebservice.lisi.ensma.fr/">
      <xsd:complexType name="person">
        <xsd:sequence>
          <xsd:element name="address" type="xs:string" minOccurs="0"/>
          <xsd:element name="birthyear" type="xs:string" minOccurs="0"/>
          <xsd:element name="name" type="xs:string" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="personArray" final="#all">
        <xsd:sequence>
          <xsd:element name="item" type="tns:person" minOccurs="0" maxOccurs="unbounded"
            nillable="true"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>
```

Exemple : Carnet d'adresse (3/6)

```
<message name="addPersonWithComplexType">
  <part name="newPerson" type="tns:person"/>
</message>
<message name="addPersonWithComplexTypeResponse">
  <part name="addPersonWithComplexTypeResult" type="xsd:boolean"/>
</message>
<message name="addPersonWithSimpleType">
  <part name="name" type="xsd:string"/>
  <part name="address" type="xsd:string"/>
  <part name="birthyear" type="xsd:string"/>
</message>
<message name="getPerson">
  <part name="personName" type="xsd:string"/>
</message>
<message name="getPersonResponse">
  <part name="getPersonResult" type="tns:person"/>
</message>
<message name="getPersons"/>
<message name="getPersonsResponse">
  <part name="getPersonsResult" type="tns:personArray"/>
</message>
```

Exemple : Carnet d'adresse (4/6)

```
<portType name="Notebook">
  <operation name="addPerson">
    <input message="tns:addPersonWithComplexType"/>
    <output message="tns:addPersonWithComplexTypeResponse"/>
  </operation>
  <operation name="addPerson" parameterOrder="name address birthyear">
    <input message="tns:addPersonWithSimpleType"/>
  </operation>
  <operation name="getPerson">
    <input message="tns:getPerson"/>
    <output message="tns:getPersonResponse"/>
  </operation>
  <operation name="getPersons">
    <input message="tns:getPersons"/>
    <output message="tns:getPersonsResponse"/>
  </operation>
</portType>
```

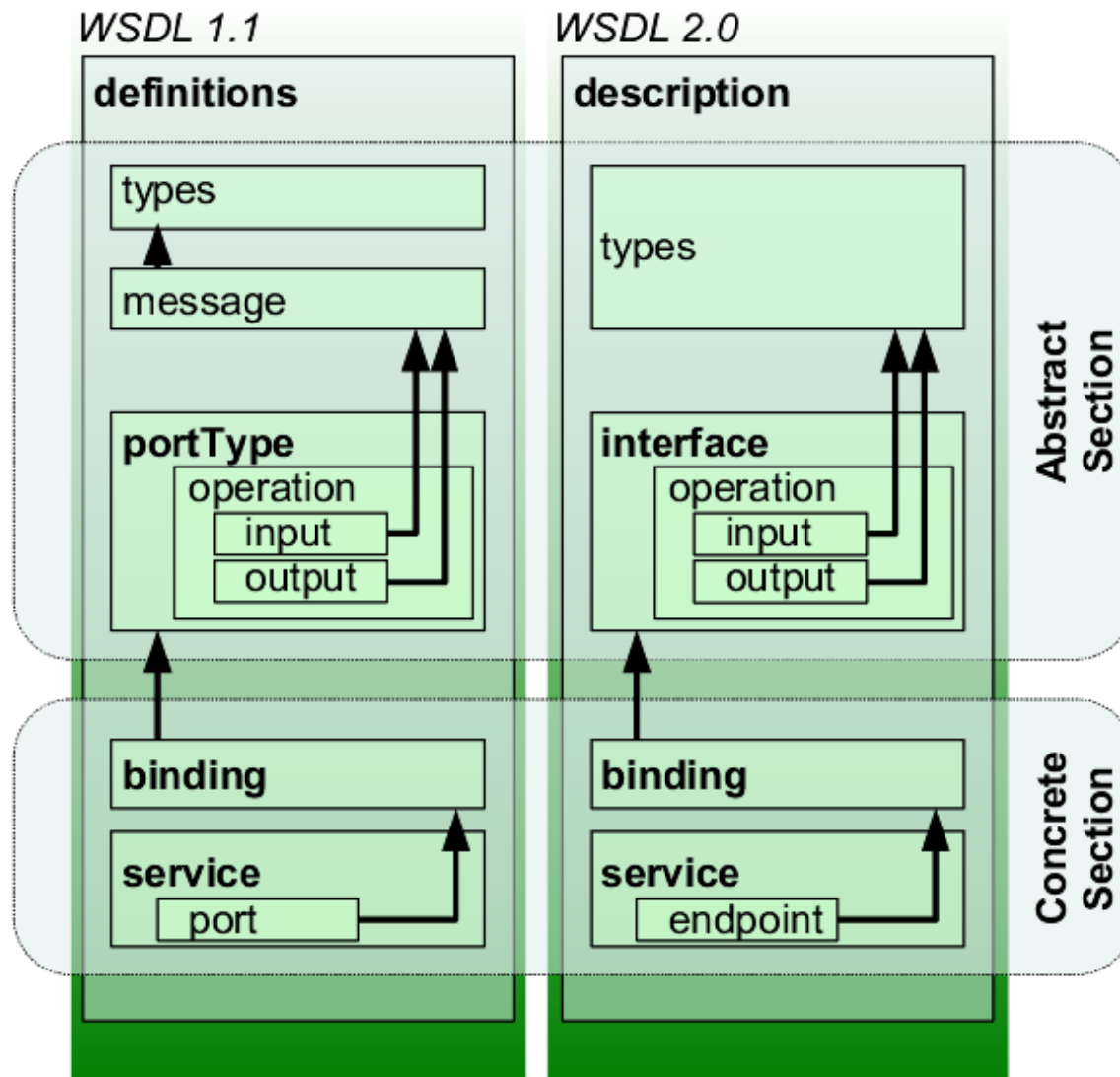


```
<binding name="NoteBookPortBinding" type="tns:Notebook">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  <operation name="addPersonWithComplexType">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr"/>
    </input>
    <output>
      <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr"/>
    </output>
  </operation>
  <operation name="addPersonWithSimpleType">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr"/>
    </input>
  </operation>
  <operation name="getPerson">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr"/>
    </input>
    <output>
      <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr"/>
    </output>
  </operation>
  ...
</binding>
```

Exemple : Carnet d'adresse (6/6)

```
<service name="Notebook">  
  <port name="NoteBookPort" binding="tns:NoteBookPortBinding">  
    <soap:address location="http://localhost:8080/NotebookWebService/notebook"/>  
  </port>  
</service>  
</definitions>
```

Representation of concepts defined by WSDL 1.1 and WSDL 2.0 documents





Simple Object Access Protocol (SOAP)

Introduction

- ◆ SOAP est un protocole de RPC orienté objet bâti sur XML (successeur du protocole XML-RPC).
- ◆ SOAP permet d'échanger des structures de données complexes en XML ;
- ◆ Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur.
- ◆ Le transfert se fait le plus souvent à l'aide du protocole HTTP, mais peut également se faire par un autre protocole, comme SMTP.
- ◆ SOAP est indépendant des langages de programmation ou des systèmes d'exploitation sur lesquels il est implémenté

Concepts d'un message SOAP

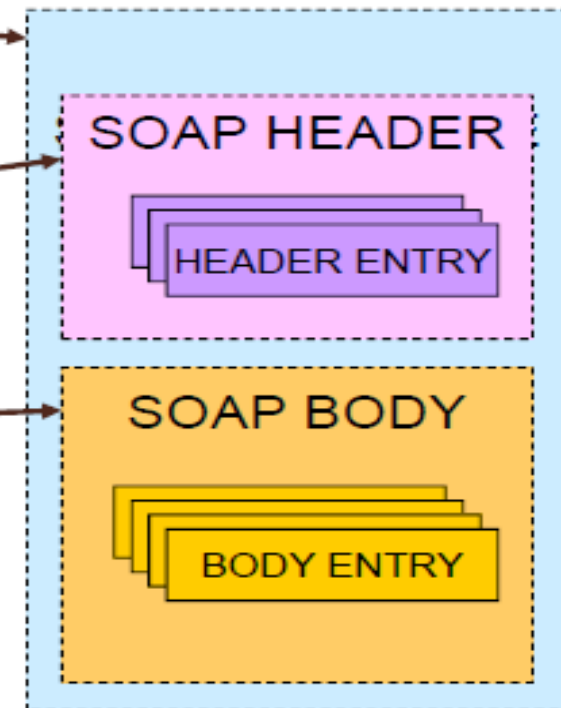
- ◆ Les messages SOAP sont utilisés pour envoyer (requête) et recevoir (réponse) des informations d'un récepteur
- ◆ Un message SOAP peut être transmis à plusieurs récepteurs intermédiaires avant d'être reçu par le récepteur final (~ chaîne de responsabilité)
- ◆ Le format SOAP peut contenir des messages spécifiques correspondant à des erreurs identifiées par le récepteur
- ◆ Un message SOAP est véhiculé vers le récepteur en utilisant un protocole de transport (HTTP, SMTP, ...)

Structure d'un message (1/2)

- ◆ Le protocole SOAP est composé de deux parties :
 - une enveloppe, contenant des informations sur le message lui-même afin de permettre son acheminement et son traitement,
 - Un modèle de données, définissant le format du message, c'est-à-dire les informations à transmettre.

Structure d'un message (2/2)

- **Envelope**
 - Contenant d'un message,
 - Élément racine XML,
 - Schéma XML
 - <http://www.w3.org/2002/06/soap-envelope/>
- **Header (optionnel)**
 - Entrées non applicatives,
 - Ex : Numéros de session.
- **Body (obligatoire)**
 - Entrées applicatives,
 - Ex : nom des procédures, nom des paramètres, valeurs de paramètres,
 - Retour d'erreurs.



L'enveloppe SOAP

- ◆ L'enveloppe est la racine d'un message SOAP identifiée par la balise <soapenv:Enveloppe>
- ◆ La spécification impose que la balise et les sous balises soient explicitement associées à un namespace
- ◆ La spécification SOAP définit deux namespaces
 - SOAP-ENV ou soapenv :
<http://schemas.xmlsoap.org/soap/envelope/>
 - SOAP-ENC : <http://schemas.xmlsoap.org/soap/encoding/>
- ◆ La requête et la réponse ont la même structure

L'enveloppe SOAP – Exemple (requête)

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:hel="http://helloworldwebservice.lisi.ensma.fr/">
  <soapenv:Header/>
  <soapenv:Body>
    <hel:makeHelloWorld>
      <value>Mickael BARON</value>
    </hel:makeHelloWorld>
  </soapenv:Body>
</soapenv:Envelope>
```

Message SOAP pour appeler l'opération makeHelloWorld contenant un paramètre value

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:hel="http://helloworldwebservice.lisi.ensma.fr/">
  <soapenv:Header/>
  <soapenv:Body>
    <hel:simpleHelloWorld/>
  </soapenv:Body>
</soapenv:Envelope>
```

Message SOAP pour appeler l'opération simpleHelloWorld ne contenant pas de paramètre

L'enveloppe SOAP – Exemple (réponse)

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns2:makeHelloWorldResponse xmlns:ns2="http://helloworldwebservice.lisi.ensma.fr/">
      <helloWorldResult>Hello World to Mickael BARON</helloWorldResult>
    </ns2:makeHelloWorldResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

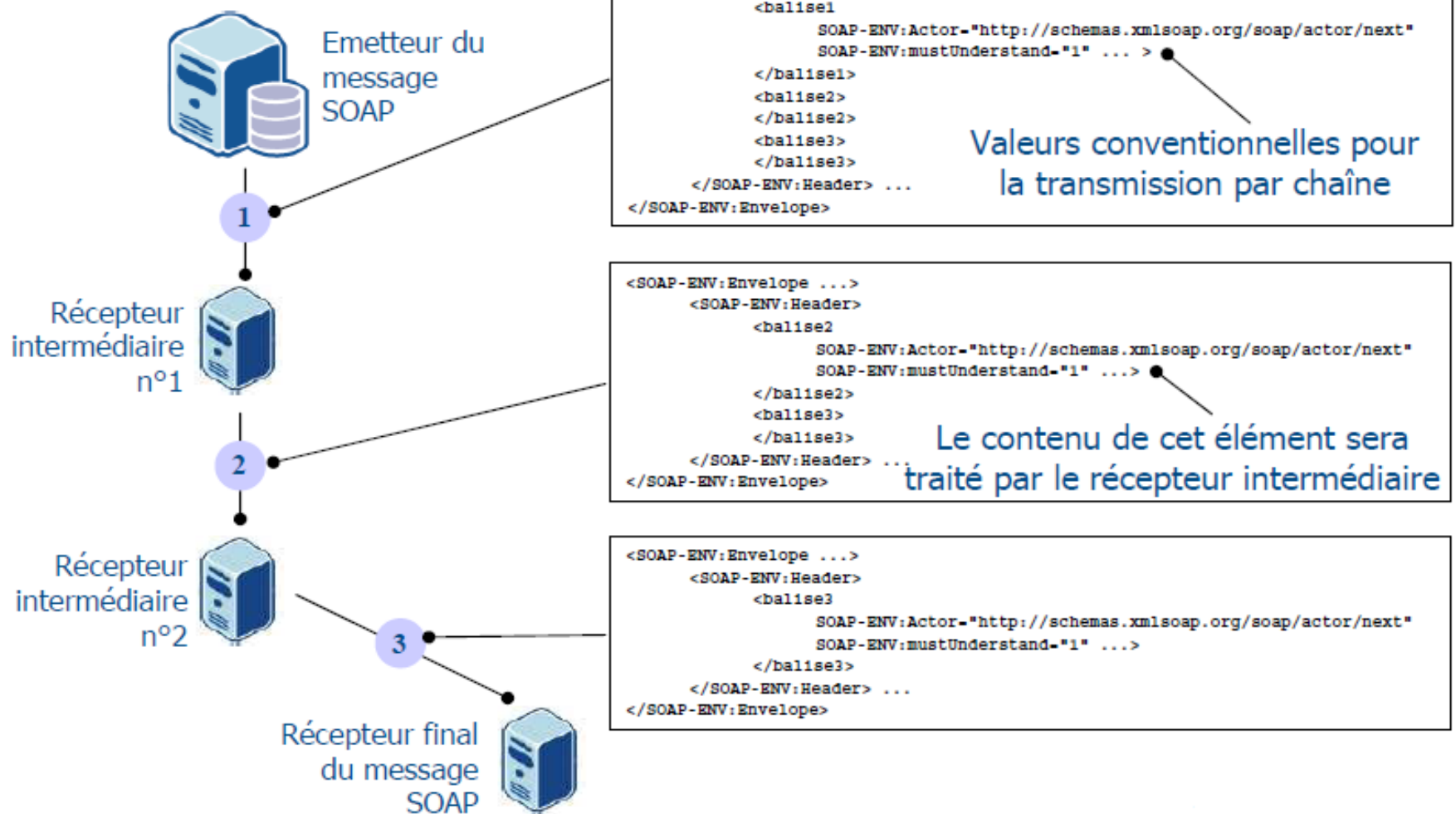
Les réponses sont
sensiblement identiques

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns2:simpleHelloWorldResponse
      xmlns:ns2="http://helloworldwebservice.lisi.ensma.fr/">
      <helloWorldResult>Hello World to everybody</helloWorldResult>
    </ns2:simpleHelloWorldResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

En-tête SOAP (1/2)

- ◆ L'en-tête d'un message SOAP est utilisé pour transmettre des informations supplémentaires sur ce même message
- ◆ L'en-tête est défini par la balise <SOAP-ENV:Header>
 - L'élément peut être facultatif
 - Doit être placé avant le corps
- ◆ Différents usages de l'en-tête ?
 - Informations authentifiant l'émetteur
 - Contexte d'une transaction
 - Pour certains protocoles de transport (FTP par exemple), l'en-tête peut être utilisé pour identifier l'émetteur du message
- ◆ Un message SOAP peut transiter par plusieurs intermédiaires avant le traitement par le récepteur final
 - Pattern « Chaîne de responsabilité »
 - Zone lecture / écrite par les intermédiaires

En-tête SOAP (2/2)



Corps SOAP

- ◆ Le corps d'un message SOAP est constitué par un élément `<SOAP-ENV:Body>`
- ◆ L'élément `<SOAP-ENV:Body>` peut contenir soit
 - Une erreur en réponse à une requête (élément `<SOAP-ENV:Fault>`)
 - Des informations adressées au destinataire du message SOAP respectant un encodage déterminé
- ◆ L'encodage des informations est précisé par les bindings du document WSDL
 - Attribut style (Document et RPC)
 - Attribut use (encoded et literal)
- ◆ Pour faire simple nous utiliserons les services Web dans le cadre de l'appel à une procédure distante

Corps SOAP - ENC des entrées

- ◆ Les informations adressées au destinataire de messages SOAP doivent respecter un certain nombre de convention
- ◆ Appel d'une opération représentée par une structure
 - Le nom de la structure est celui de l'opération à appeler
 - Chaque paramètre de l'opération est défini comme un sous élément de la structure
 - Si un paramètre est un type complexe (Person par exemple) une nouvelle structure est définie contenant à son tour des sous éléments...
- ◆ Le résultat est également représenté par une structure
 - Le nom de la structure est celui de l'opération suivi de **Response**
 - Les paramètres sont également structurés

Corps SOAP – Exemples (1/2)

- ◆ Message SOAP pour appeler l'opération addPersonWithComplexType
Paramètre de type complexe défini dans une structure (newPerson)

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:not="http://notebookwebservice.lisi.ensma.fr/">
  <soapenv:Header/>
  <soapenv:Body>
    <not:addPersonWithComplexType>
      <newPerson>
        <address>Poitiers</address>
        <birthyear>17081976</birthyear>
        <name>BARON Mickael</name>
      </newPerson>
    </not:addPersonWithComplexType>
  </soapenv:Body>
</soapenv:Envelope>
```


Corps SOAP - Exemples (2/2)

- ◆ Message SOAP pour appeler l'opération `addPersonWithSimpleType` avec trois paramètres

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:not="http://notebookwebservice.lisi.ensma.fr/">
  <soapenv:Header/>
  <soapenv:Body>
    <not:addPersonWithSimpleType>
      <name>BARON Mickael</name>
      <address>Poitiers</address>
      <birthyear>17081976</birthyear>
    </not:addPersonWithSimpleType>
  </soapenv:Body>
</soapenv:Envelope>
```

Corps SOAP - Le retour d'erreurs

- ◆ L'élément <Fault> composé de 4 sous élément
 - <faultCode> (obligatoire) : code d'erreur
 - <faultString> (obligatoire) : Explication lisible d'un humain
 - <faultActor>(optionel) : Erreur en cours de cheminement du message (firewall, proxy..)
 - <detail> Détail de l' erreur non lié au Body du message

Corps SOAP - Exemple

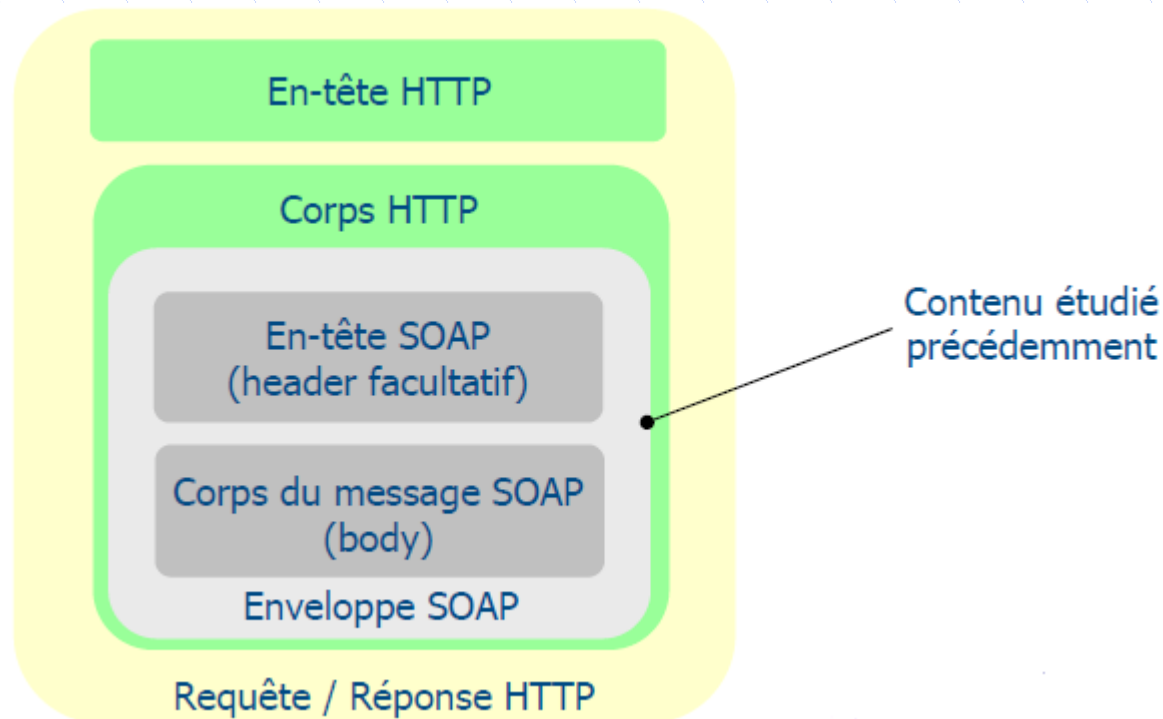
- ◆ Faultcode : 4 groupes(Client, Server, MustUnderstand, VersionMismatch)

HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>Server Error</faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="Some-URI">
          <message> My application didn't work </message>
          <errorcode>1001</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP transporté par HTTP (1/3)

- ♦ SOAP utilise un protocole de transport pour véhiculer les messages SOAP de l'émetteur au récepteur HTTP, SMTP, FTP, POP3 et NNTP
- ♦ Le modèle requête/réponse de SOAP convient parfaitement au modèle requête/réponse HTTP



SOAP transporté par HTTP (2/3)

- ◆ Requête SOAP HTTP
 - Méthode de type POST
 - Nécessite un attribut SOAPAction (URI)

```
POST http://localhost:8080/NotebookWebService/notebook
HTTP/1.1
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
User-Agent: Jakarta Commons-HttpClient/3.1
Host: localhost:8080
Content-Length: 459
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:not="http://notebookwebservice.lisi.ensma.fr/">
  <soapenv:Header/>
  <soapenv:Body>
    <not:addPersonWithComplexType>
      <newPerson>
        <address>Poitiers</address>
        <birthyear>17081976</birthyear>
        <name>BARON Mickael</name>
      </newPerson>
    </not:addPersonWithComplexType>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP transporté par HTTP (3/3)

◆ Réponse SOAP HTTP

- Exploite les codes retours HTTP Si code de type 2xx, message SOAP reçu
- Si code 500, message en erreur, le corps SOAP doit contenir fault

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

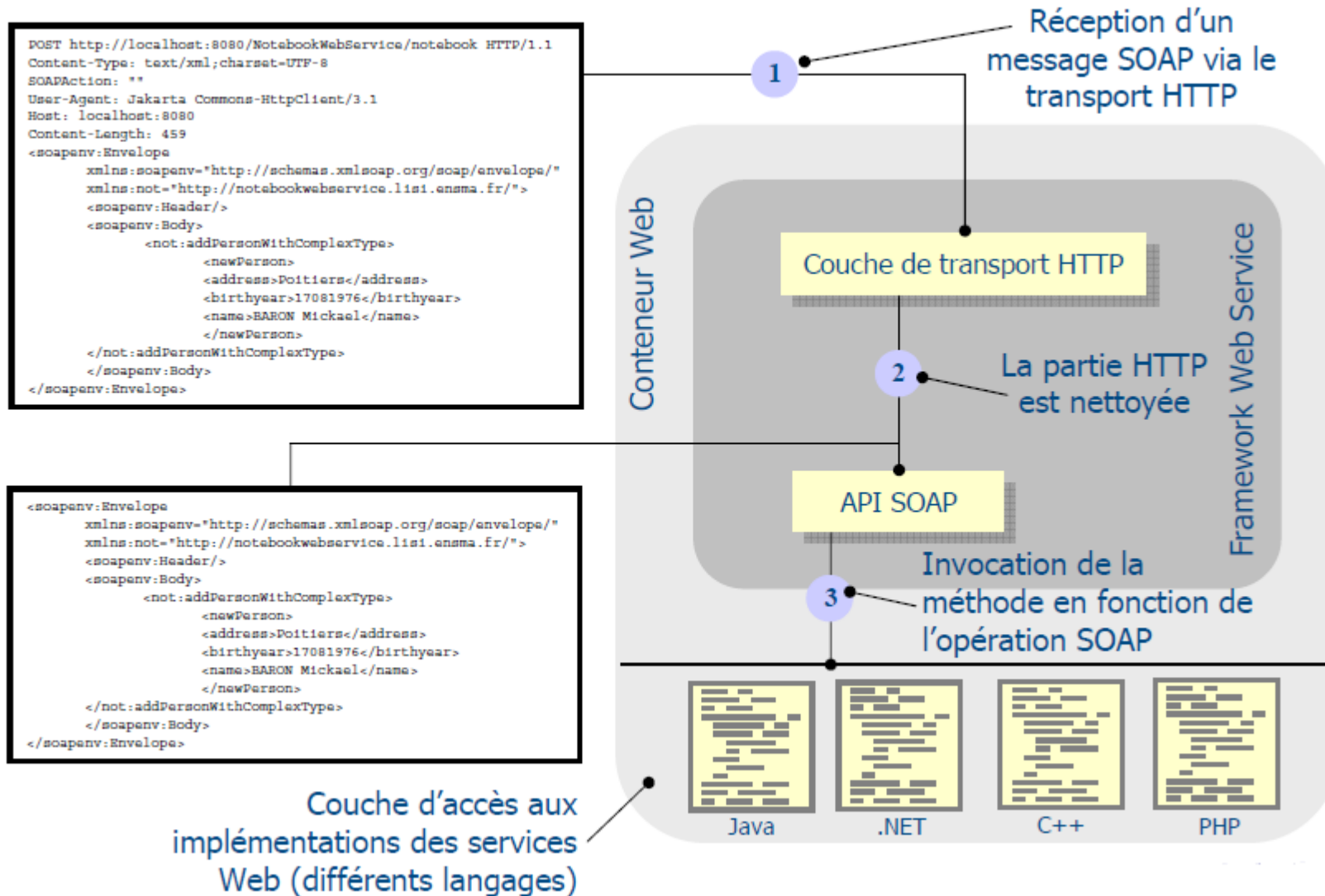
Content-Type: text/xml; charset=utf-8

Transfer-Encoding: chunked

Date: Sun, 13 Dec 2009 12:00:33 GMT

```
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns2:addPersonWithComplexTypeResponse
      xmlns:ns2="http://notebookwebservice.lisi.ensma.fr/">
      <addPersonWithComplexTypeResult>
        true
      </addPersonWithComplexTypeResult>
    </ns2:addPersonWithComplexTypeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

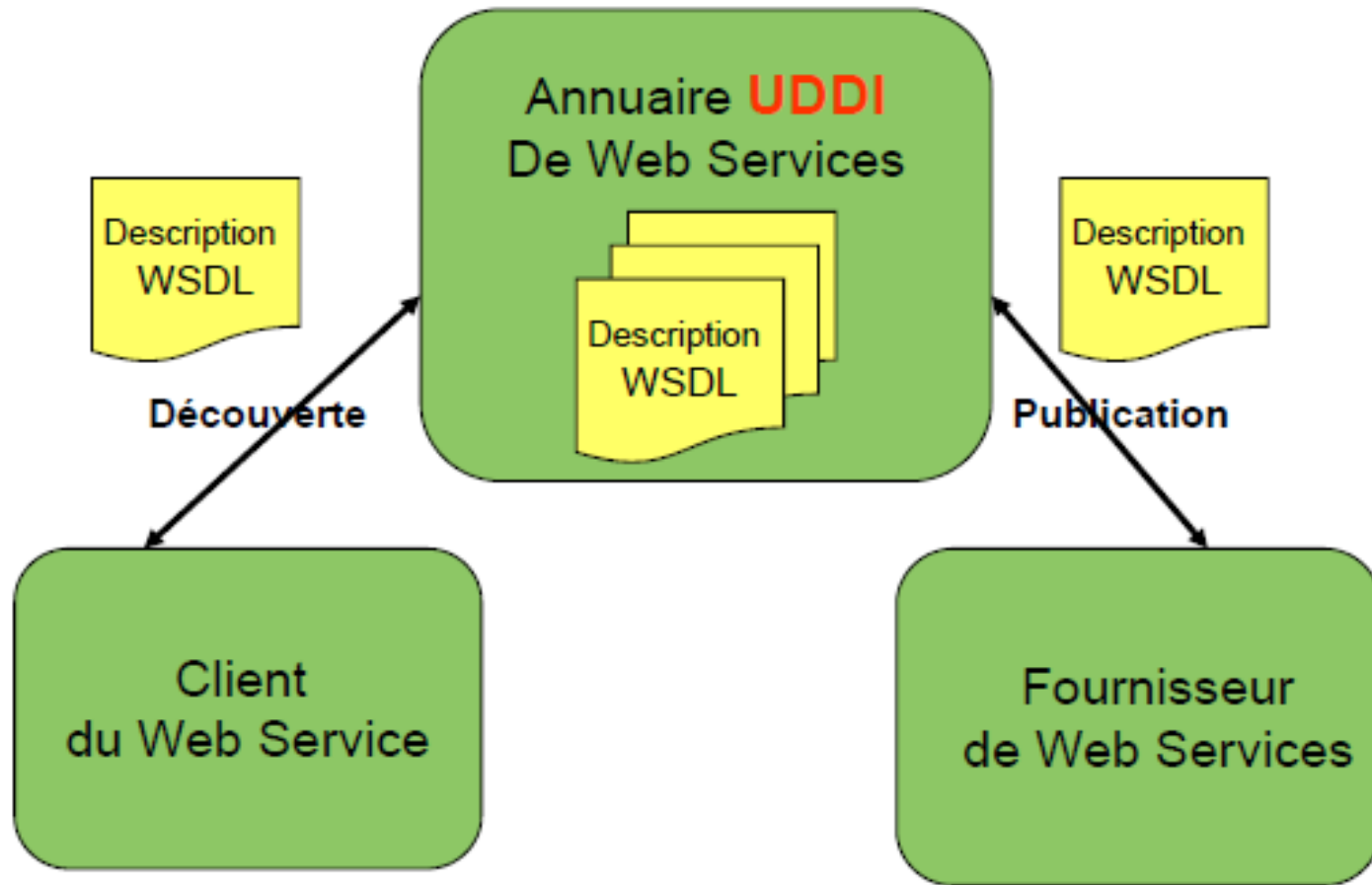
Traitement des messages SOAP





Universal Description, Discovery and Integration (UDDI)

Découverte et publication d'un Web Service



Qu'est ce que UDDI ?

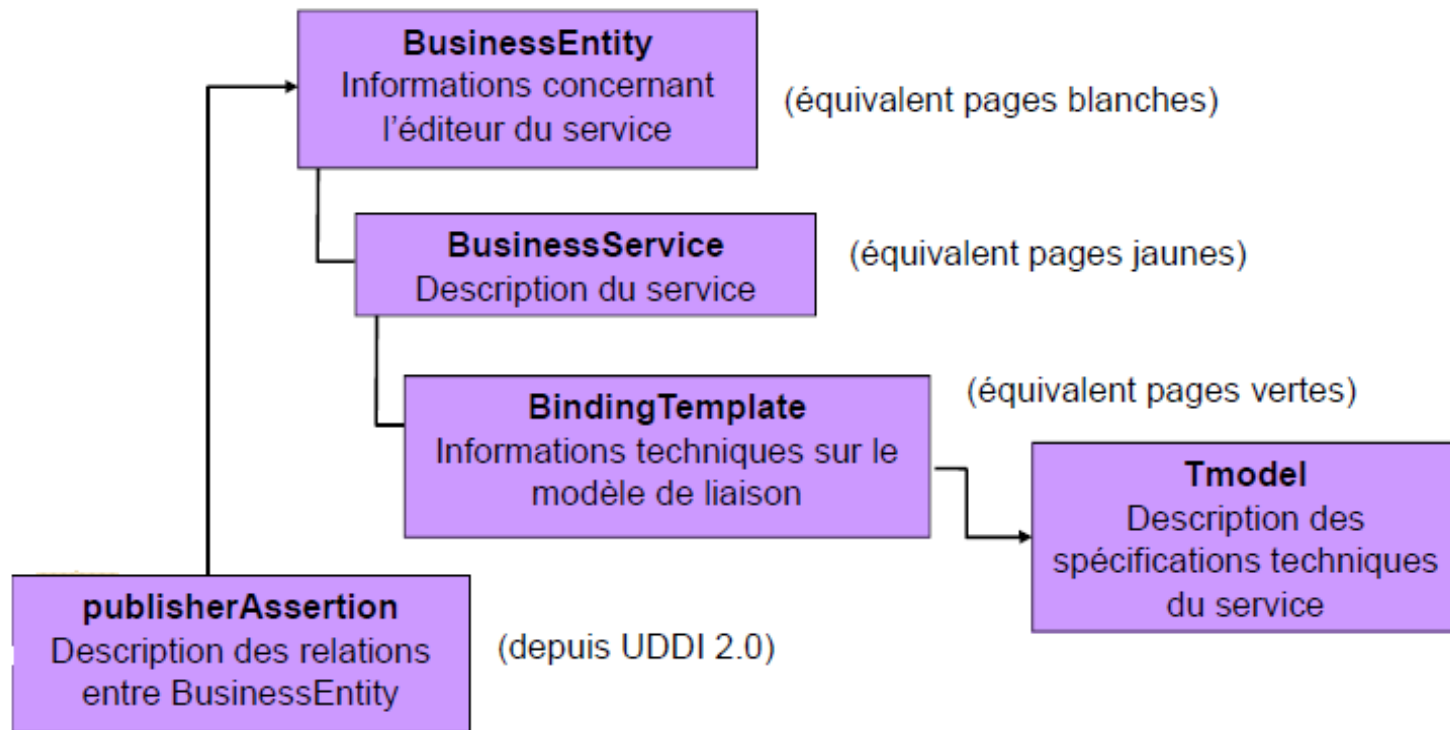
- ◆ UDDI (Universal Description, Discovery and Integration) est un standard ayant pour but la création d'un annuaire distribué de services web.
- ◆ Spécification définie par un consortium industriel
 - UDDI 3.0, UDDI Working Group, OASIS
- ◆ Accès par une API vue comme un Web Service
- ◆ L'API UDDI propose deux fonctionnalités principales :
 - la recherche de services (envoi de requêtes).
 - la publication de services dans un annuaire UDDI.
- ◆ Structure de données définie par une grammaire XML
- ◆ Les clients UDDI interrogent les serveurs (les sites opérateurs) UDDI en envoyant des requêtes formatées en SOAP (sur HTTP avec la méthode POST).

Annuaire(s) UDDI

- ◆ Pages blanches
 - Description des entreprises,
 - ◆ Nom, adresses, contacts, identifiants
- ◆ Pages jaunes
 - Catégories industrielles fondées sur des taxonomies standards (NAICS, UN/SPSC)
 - ◆ Types d'industries,
 - ◆ Types de produits de services,
 - ◆ Information sur la localité géographique.
- ◆ Pages vertes
 - Références techniques sur les services fournis,
 - ◆ Références à la spécification d'un Web Service

Structure de données

- ◆ Format défini par XML Schéma
 - http://uddi.org/schema/uddi_v3.xsd

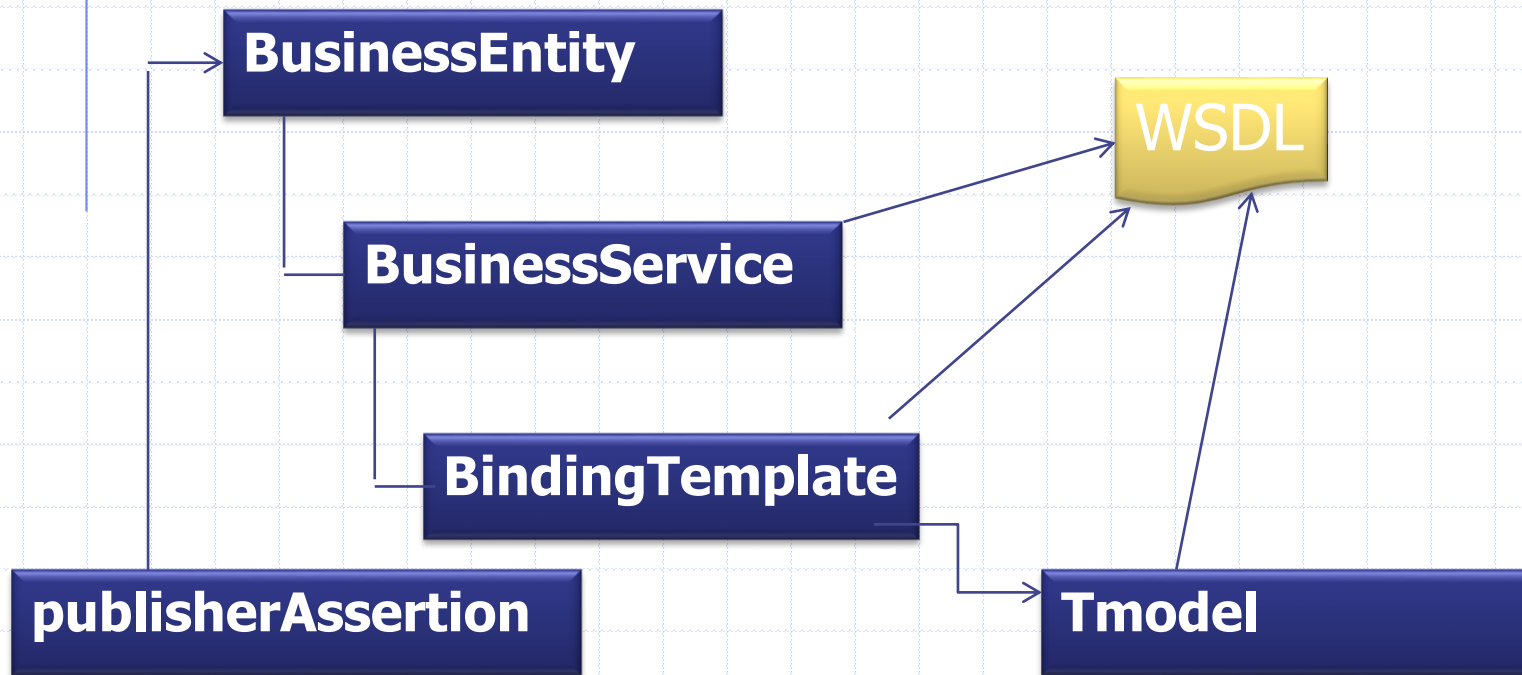


Accès à l'annuaire

- ◆ Accès direct par un navigateur Web
- ◆ Accès par un programme en utilisant une API
 - API vue comme un Web Service
 - ◆ Décrite en WSDL et Appelée avec SOAP
 - Fonctions de recherche (Inquiry API),
 - ◆ Recherche et consultation de données
 - ◆ Ex: find_business, get_serviceDetail
 - Fonctions de publications (Publishing API)
 - ◆ Création, Modification, Suppression de données
 - ◆ Ex: save_binding, delete_tmodel
 - Fonctions annexes
 - ◆ Réplication entre annuaires, Sécurisation des données

WSDL et UDDI

- ◆ Mapping WSDL avec UDDI
 - Fragmentation de la description WSDL complet





Orchestration vs. Chorégraphie

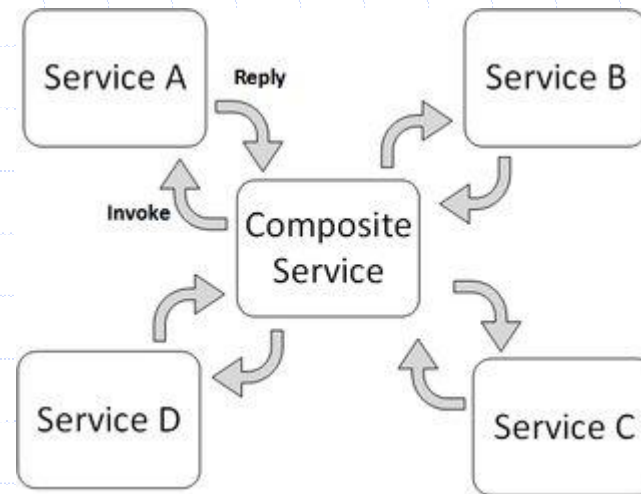
Composition de services

- ◆ Les technologies de base telles que (XML, SOAP, WSDL) fournissent des moyens de décrire, localiser et invoquer des services en tant qu'entité à part entière.
- ◆ Cependant, ces technologies ne donnent pas de détails comportementaux riches sur le rôle du service dans une collaboration plus complexe.
- ◆ Cette collaboration comprend une séquence d'activités et de relations entre les activités, qui construisent le processus métier. Il existe deux façons de créer ce processus: l'orchestration des services et la chorégraphie des services.

Orchestration des services

- ◆ L'orchestration de services représente un seul processus métier exécutable centralisé (l'orchestrateur) qui coordonne l'interaction entre différents services. L'orchestrateur est responsable de l'appel et de la combinaison des services.
- ◆ La relation entre tous les services participants est décrite par un seul point d'extrémité (c'est-à-dire le service composite). L'orchestration comprend la gestion des transactions entre les services individuels. L'orchestration utilise une approche centralisée pour la composition des services.

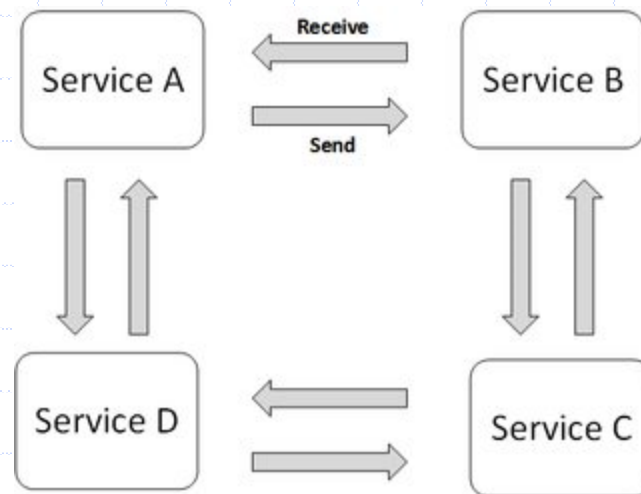
Orchestration des services



Chorégraphie de service

- ♦ La chorégraphie des services est une description globale des services participants, qui est définie par l'échange de messages, des règles d'interaction et des accords entre deux ou plusieurs points d'extrémité. La chorégraphie utilise une approche décentralisée pour la composition des services.
- ♦ La chorégraphie décrit les interactions entre plusieurs services, où l'orchestration représente le contrôle du point de vue d'une partie. Cela signifie qu'une **chorégraphie** *diffère* d'une **orchestration en ce** qui concerne l'emplacement de la logique qui contrôle les interactions entre les services impliqués.

Chorégraphie de service



Orchestration avec WS-BPEL

- ◆ Un processus métier est un enchaînement d'activités entre acteurs d'une entreprise pour aboutir à un service donné.
 - Acteurs : humains, applications/services, processus métier
- ◆ Description XML d'un processus métier
- ◆ Référence les descriptions WSDL
- ◆ Activité de base
 - Invoquer un service Web <invoke>
 - Recevoir une invocation <receive>
 - Répondre à une invocation <reply>
- ◆ Activité structurée
 - Séquence ordonnée <sequence>
 - Branchement conditionnel <switch>, <if>
 - Boucle <while>
 - Exécution en parallèle <flow>
- ◆ Spécification du consortium industriel OASIS

Exemple : WS-BPEL

