Contents

1 Les services web SOAP

1.1 Définitions et généralités

1.1.1 URI

1.1.1.1 Définition

Une URI décrit :

1. le **mécanisme** pour accéder à une **ressource** (e.g., http, ftp, mailto, ...).
2. la **machine** où se trouve la ressource (e.g., www.w3.org, www.google.com, localhost, ...)
3. le **nom** de la ressource sur la machine (e.g., index.html, /path/to/resource.json, ...)

1.1.1.2 Exemples

1. **URL** : https://www.w3.org/index.html
2. **mail** : mailto:toto@domain.com
3. **FTP** : ftp://ftp.computer.org

1.1.2 Association

1. **définition** : une **association** (*binding*) entre une **interface**, un **protocole concret** et un **format de données**.

2. **rôle** : spécification du protocole et du format des données utilisés pour l'échange des messages afin d'utiliser une interface.

1.1.3 Service web (Web Service (WS))

1.1.3.1 Définition générale

1. un service web est un :
   - moyen permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.
   - un ensemble de fonctionnalités exposées sur un réseau (Internet/Intranet), par et pour des applications/machines, sans intervention humaine, de manière (a)synchrone, essentiellement en utilisant le protocole de transport HTTP.
2. **technologies principales** :
   - **services web REST** (**REpresentational State Transfer**) : exposent des fonctionnalités comme un ensemble de ressources identifiables par une URI et accessibles par la syntaxe et la sémantique du protocole HTTP.
   - **services web WS-*** : exposent des fonctionnalités sous forme de services exécutables à distance, implémentant des spécifications WS-* et reposant sur les standards SOAP et WSDL.

1.1.3.2 Définition du W3C

un service web est un système logiciel :

1. identifié par une **URI** (**Uniform Resource Identifier**)
2. dont les **interfaces** publiques et les **associations** sont définies et décrites en XML.
3. proposant des fonctionnalités que d'autres programmes peuvent :
   - **découvrir** ;
   - **utiliser** selon les modalités indiquées dans sa définition grâce à des messages XML transmis via des protocoles internet.

1.1.3.3 Définition opérationnelle

1. **ressource** : offre une/plusieurs interfaces.
2. **interface** : collection d'opérations.
3. **opération** : un échange de messages entre le fournisseur d'un service et le demandeur du service.
4. **endpoint** : association d'une interface à un protocole, identifiée par une URI.
5. **service** : une collection d'endpoints d'une même interface et, par conséquent, d'une même ressource.

1.1.3.4 Objectifs

1. remplacer les protocoles de programmation distribuée actuels (*e.g. RPC, DCOM, RMI, ...*)
2. faire interagir des composants hétérogènes, distants et indépendants avec un protocole standard (*i.e., SOAP*)
3. s'utiliser par les applications **P2P**, **B2B**, **EAI** (**Enterprise Application Integration**)

1.1.3.5 Exemple

Une agence de voyage web, combinant plusieurs services :

1. **réservation de billets d'avion, de train** → *e.g. utilisation de services proposés par SNCF.*
2. **réservation de logement** → *e.g. utilisation de services proposés par AirBnB.*
3. **réservation de véhicules de location** → *e.g. utilisation de services proposés par RentACar.*
4. ...

1.2 Services web et applications distribuées

1.2.1 Les applications distribuées

1. **Object RPC (ORPC)**:
   - **rôle** : coupler l'approche objet aux protocoles réseau.
   - **ORPC request** : un identifiant ou nom symbolique utilisé par un serveur pour localiser un objet cible.
   - **protocoles lourds**.
2. évolution en termes de :
   - **protocoles d'échange** : *RMI, CORBA, DCOM, ...*
   - **langages de programmation** ;
   - **interfaces offertes**.
3. **couplage fort** aux technologies utilisées : le client et le serveur doivent être implémentée en utilisant le même modèle de l'objet distribué.
4. **difficultés d'usage** avec les firewalls et les serveurs proxy : e.g., la majorité des firewalls sont configurés pour l'autorisation du HTTP, mais pas du IIOP.

1.2.2 Le Web

1. protocoles de communication légers avec services plus faibles : *HTTP, FTP, ...*
2. un client léger (*le navigateur*).

3. présentation des ressources en **HTML**.
4. **couplage faible** aux technologies utilisées.

1.2.3 Les services web

1.2.3.1 Principes

1. combinent à la fois les caractéristiques des applications distribuées et conforment aux contraintes du Web.
2. **modèle de communication** : client/serveur.
3. **protocole de transport** : HTTP sur TCP/IP.
4. **format d'échange de données** : XML (*HTTP transporte du texte*).
5. client web $\rightarrow$ machine web.

1.2.3.2 Caractéristiques

1. assurer l'interopérabilité dans des environnements applicatifs distribués.
2. accéder à des applications à travers des pare-feux (*firewalls*)
3. utiliser différentes plateformes et différents langages de programmation.

1.2.3.3 Contextes d'utilisation

1. applications bien délimitées et sans forte interactivité : *service météo, cotations boursières, ...*
2. assemblage de composants faiblement couplés : *une agence de voyage*
3. applications orientées message.

1.2.3.4 Architecture

1.2.3.4.1 Acteurs impliqués

1. **le client** : utilise/invoque un WS.
2. **le fournisseur** :
   - fournit le WS.
   - représenté par une serveur d'applications.
3. **l'annuaire** : publier un WS (*i.e. le rendre accessible aux clients*).
4. **un proxy** : éventuellement entre le client et le fournisseur.

1.2.3.4.2 Technologies

1. **WSDL (Web Service Description Language)** :
   - description des WS.
   - dialecte XML.
   - standard W3C.

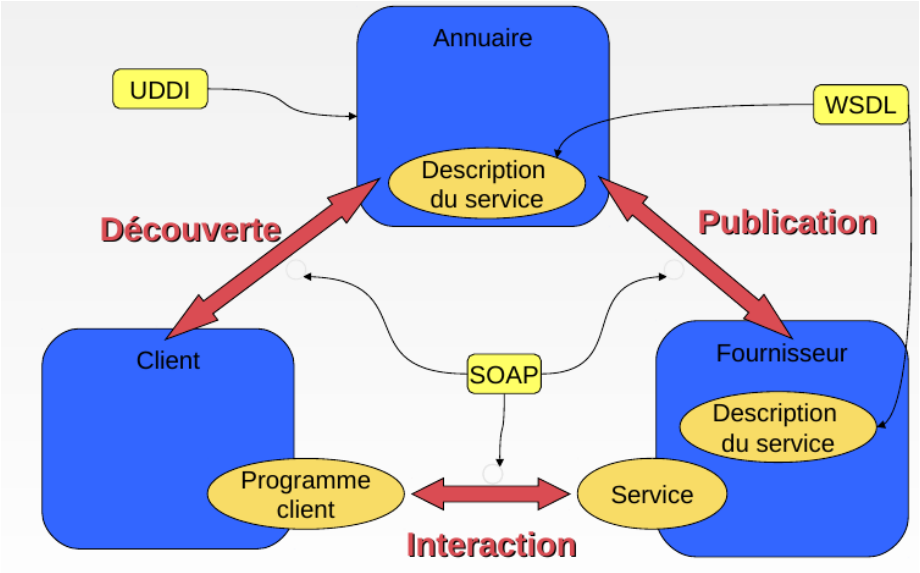

Figure 1: Architecture d'un service web

2. **SOAP** :
   - protocole de communication des WS.
   - dialecte XML.
   - standard W3C.
   - **remarque** : SOAP n'est plus un acronyme depuis la version 1.2.
3. **UDDI (Universal Description, Discovery and Integration)** :
   - annuaire de WS.
   - publication de descriptions de WS par des fournisseurs.
   - découverte de descriptions de WS par des clients.
4. **outils** : les outils se chargent d'automatiser :
   - la transcription de WSDL en proxy.
   - la génération des requêtes SOAP nécessaires.
   - la décortiquation des messages SOAP.

1.3 SOAP

1.3.1 Aperçu

1. protocole simple, assez léger et extensible pour l'échange d'informations dans un environnement distribué.
2. moyen permettant de réaliser des appels de méthodes sur le Web et fonctionnant avec l'infrastructure Internet existante.

3. usage de XML pour le format d'échange des messages et HTTP pour leur transport.
4. interopérabilité au sein d'un environnement distribué en utilisant des pratiques et protocoles standardisées.
5. portabilité d'une application distribuée sur différentes plateformes et technologies.
6. ne désigne pas vraiment un système distribué à objets :
   - pas de passage d'objets par référence ;
   - pas de contrôle de types ;
   - pas de ramasse-miettes.

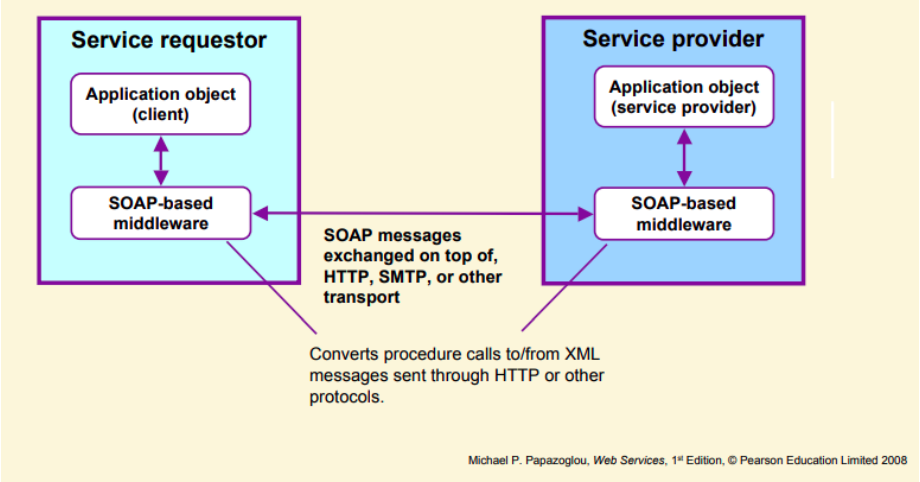

Figure 2: Aperçu de SOAP

1.3.2 Introduction

1.3.2.1 Origines

1. extension de HTTP pour l'échange de messages XML.
2. dialecte XML.

1.3.2.2 Définition

SOAP est un protocole d'échange de messages minimal basé sur XML et permettant :

1. la **connexion d'applications clients** à des **WS** et l'invocation de leurs opérations.
2. la spécification du **format de représentation** et des **régles d'encodage** des messages à échanger : documents XML ou données RPC.

3. la spécification du **protocole de transport** des messages à échanger en utilisant HTTP (pour les interactions Web) ou SMTP (pour les interactions email), ...:
4. l'indépendence :
   - d'un langage dédié ;
   - d'un modèle de programmation ;
   - d'utiliser :
     1. un **ORB** (**Object Request Broker**) : *e.g., CORBA, DCOM, ...*
     2. un serveur web dédié : *e.g., Apache, IIS, ...*

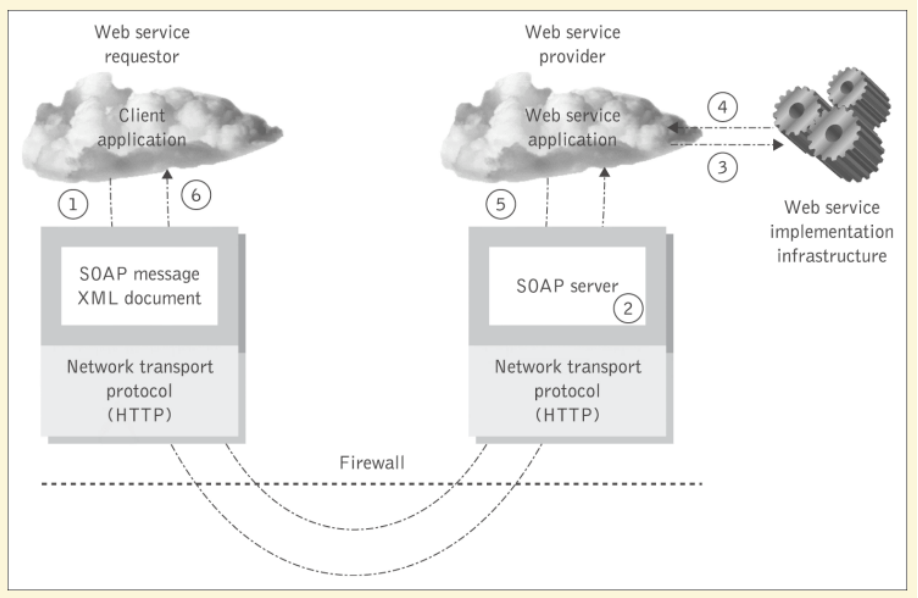

Figure 3: Échange distribué de messages avec SOAP

1.3.2.3 Objectif

1. assurer l'interopérabilité au sein d'un environnement distribué en utilisant des pratiques et protocoles standardisées.
2. assurer la portabilité d'une application distribuée sur différentes plateformes et technologies.

1.3.2.4 Facettes d'une requête/réponse SOAP

1.3.2.4.1 Un protocole d'échange de messages

1. une requête envoyée par un client contient un seul message désignant l'appel sérialisé d'une opération du WS.
2. une réponse retournée par un serveur contient un seul message désignant le retour sérialisé d'un appel d'une opération du WS.

1.3.2.4.2 Un format de représentation de messages

1. une requête envoyée par un client contient un document XML.
2. une réponse retournée par un serveur contient une version transformée du document XML envoyé au sein de la requête traitée.

1.3.2.5 Workflow

1. encapsulation d'un WS au sein d'une classe implémentée dans un langage de programmation (e.g., Java).
2. encapsulation des opérations du WS au sein de méthodes correspondantes déclarées par la classe du WS.
3. démarrage d'un thread écoutant les requêtes adressées au WS, tel que les requêtes sont au format SOAP et contiennent le nom du WS et les paramètres requis par l'opération à invoquer.
4. décodage de la requête par le thread et sa transformation en une invocation de l'opération désirée du WS.
5. invocation de l'opération du WS et récupération du résultat.
6. envoi du résultat au demandeur.

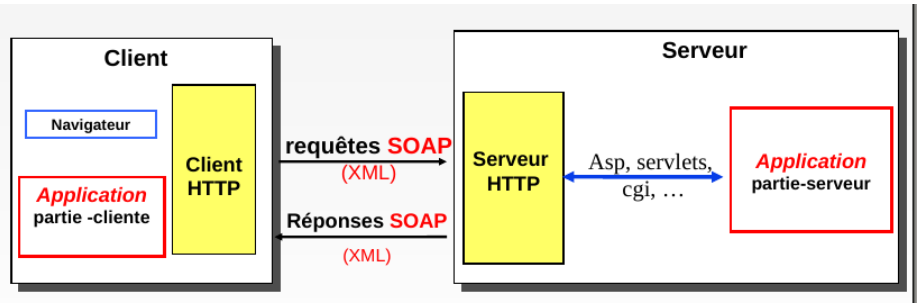

Figure 4: Workflow de SOAP

1.3.2.6 Sécurité

1. basée sur HTTPS et le certificat X.509.
2. les firewalls peuvent facilement filtrer les messages SOAP.
3. chaque développeur choisit de publier telle ou telle opération d'un WS.
4. les paramètres des opérations à invoquer sont typés lors du transport.
5. pas de transfert de code applicatif, mais uniquement de données.

1.3.3 Les messages SOAP

1.3.3.1 Définition

1. un message SOAP est un document XML contenant une **enveloppe SOAP** obligatoire délimitant son début et sa fin et structurant son contenu.
2. la structure de l'enveloppe change en changeant la version de SOAP utilisée.
3. selon le protocole de transport utilisé, un message SOAP peut être encapsulé au sein d'une structure englobante (*e.g., une requête/réponse HTTP*).

1.3.3.2 Enveloppe SOAP

1. **définition** : élément racine d'un message SOAP servant comme un mécanisme de packaging en délimitant son début et sa fin et structurant son contenu.
2. **balise** : `<SOAP-ENV:Envelope>`.
3. **namespaces pour `SOAP-ENV`** :
   - **SOAP 1.1** : http://schemas.xmlsoap.org/soap/envelope
   - **SOAP 1.2** : http://www.w3.org/2001/12/soap-envelope
4. **composition** (*en ordre*) : **SOAP Header** (*optionnel*) et **SOAP Body** (*obligatoire*).

```
<!--../source/soap/xml/snippets/soap_generic_structure.xml-->

<!--Generic structure of a SOAP Envelope-->
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
    SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <SOAP-ENV:Header>
        ...
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        ...
        <SOAP-ENV:Fault>
            ...
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.3.3.3 SOAP Header

1. **définition** : une partie facultative d'une enveloppe SOAP contenant un/plusieurs éléments *header* déclarés les uns à la suite des autres avant

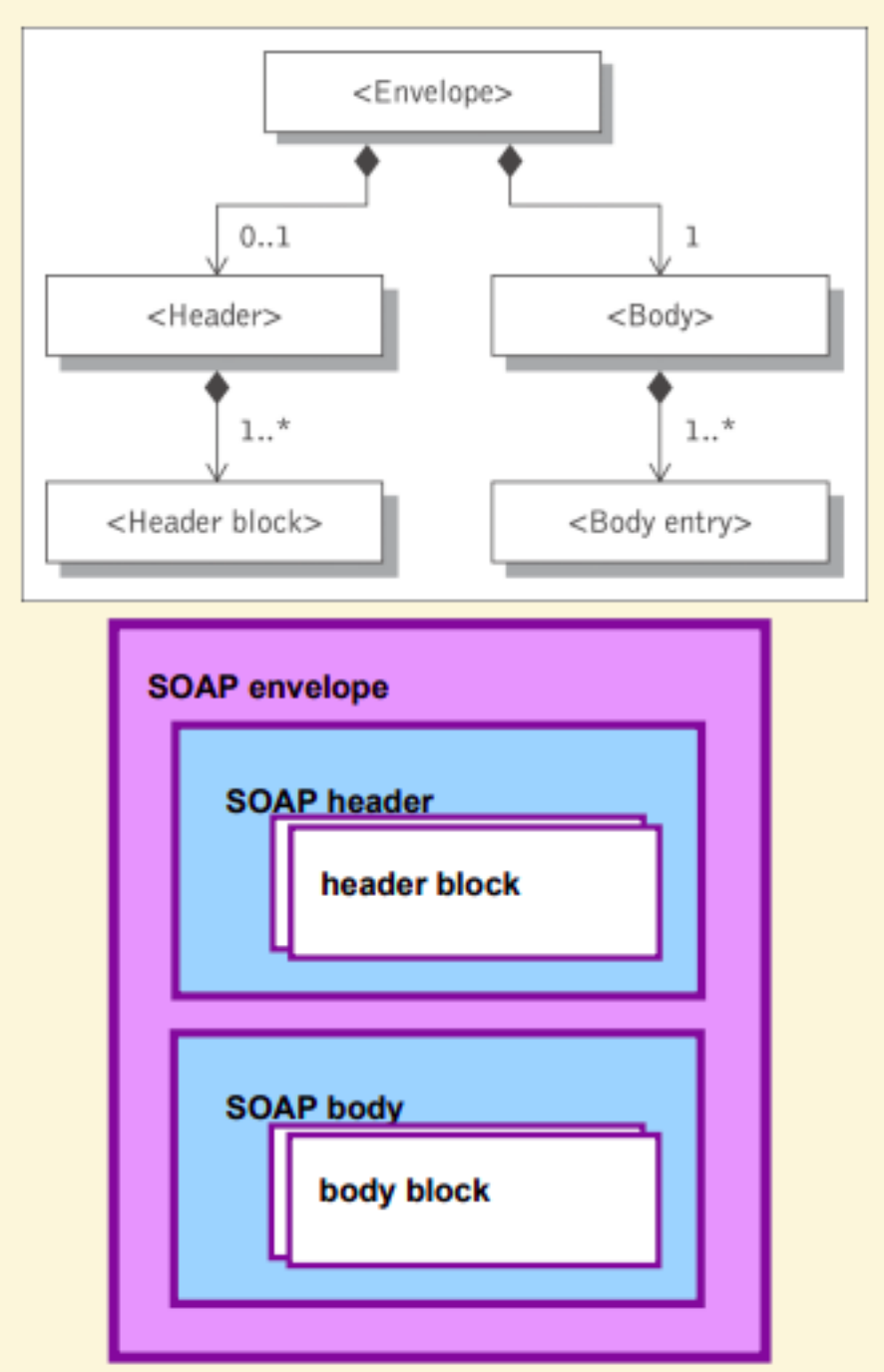

Figure 5: Enveloppe SOAP

le SOAP Body et régissant l'utilisation d'un WS.

2. **balise** : `<SOAP-ENV:Header>`.
3. **élément header**:
   - permet de personnaliser et contrôler l'usage d'un WS en interagissant avec le moteur SOAP : *e.g., signature digitale pour un WS nécessitant un mot de passe, numéro de compte pour des WS payants, ...*
   - défini au sein d'un namespace utilisateur : *e.g.,* `<t:Transaction>` *où* `<Transaction>` *est un élément header défini au sein du namespace utilisateur* `t`.
   - peut avoir les attributs `actor` et `mustUnderstand`.
4. **l'attribut** `actor` :
   - **motivation** : le protocole SOAP définit un chemin de messages en tant qu'une liste de noeuds de services SOAP, tels que chaque noeud peut appliquer des traitements, puis transférer le message vers le prochain noeud dans le chemin.
   - **définition** : spécifie le récipient d'un header SOAP.
5. **l'attribut** `mustUnderstand` :
   - indique si un élément du SOAP header est facultative ou obligatoire.
   - si la valeur `true` est affectée, alors le recevant du message SOAP doit comprendre et traiter l'élément du SOAP header ou bien retourner un élément `Fault`.

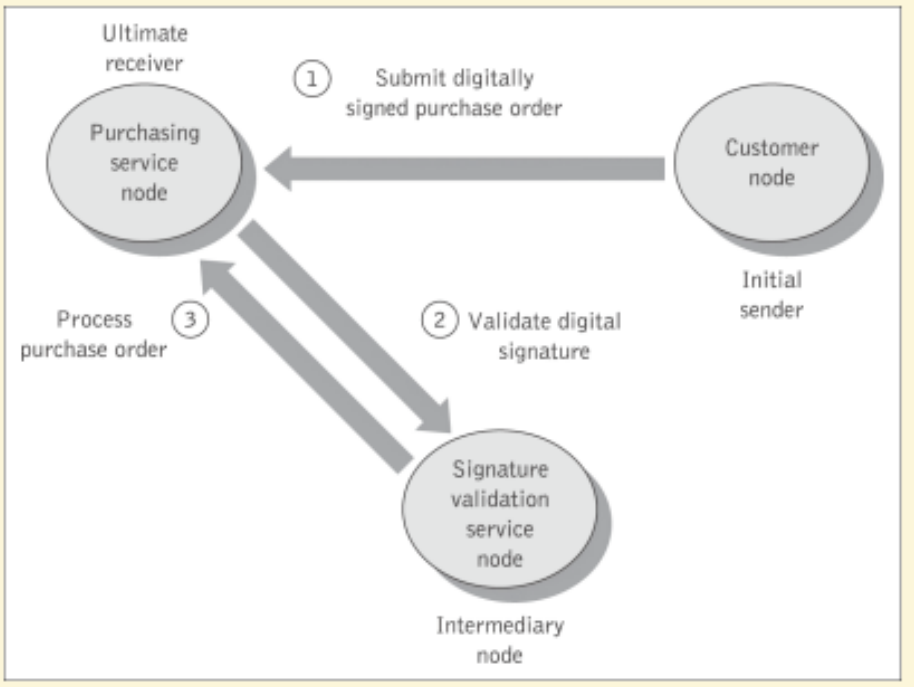

Figure 6: Intermédiaires SOAP

```
<!--../source/soap/xml/snippets/soap_header_example.xml-->

<!--Example of a SOAP envelope with a SOAP header-->
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
    SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <SOAP-ENV:Header>
        <t:Transaction
            xmlns:t = "http://www.tutorialspoint.com/transaction/"
            SOAP-ENV:mustUnderstand = "true">5
        </t:Transaction>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        ...
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<!--../source/soap/xml/snippets/soap_header_message_routing_example.xml-->

<!--Example of SOAP envelope with a SOAP header with message routing-->
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <m:order
        xmlns:m="http://www.plastics_supply.com/purchase-order"
        env:role="http://www.w3.org/2003/05/soapenvelope/role/next"
        env:mustUnderstand="true">
            <m:order-no >uuid:0411a2daa</m:order-no>
            <m:date>2004-11-8</m:date>
        </m:order>
        <n:customer
        xmlns:n="http://www.supply.com/customers"
        env:role="http://www.w3.org/2003/05/soapenvelope/role/next"
        env:mustUnderstand="true">
            <n:name> Marvin Sanders </n:name>
        </n:customer >
    </env:Header>
    <env:Body>
        <!-- Payload element goes here -->
    </env:Body>
</env:Envelope>
```

1.3.3.4 SOAP Body

1. **définition** : une partie obligatoire d'une enveloppe SOAP contenant les données XML à échanger (*pouvant être éventuellement vide*), incluant :
   - des informations sur l'appel d'une méthode d'un service web et éventuellement de ses arguments encodés.
   - des informations sur le retour d'un appel d'une méthode.
   - des informations sur les erreurs survenant lors du traitement d'une requête SOAP.
2. **balise** : `<SOAP-ENV:Body>`.
3. **contenu** :
   - **entrées du message échangé en XML** (*obligatoire*) :
     1. *noms d'une procédure à appeler, valeurs paramètres, valeur de retour.*
     2. les entrées des messages sont définis dans un schéma SOAP associé (*e.g., en utilisant WSDL*)
   - **éléments** `Fault` (*optionnels*) : des erreurs lors du traitement des messages.
   - **contrainte** : soit des entrées du message échangé, soit des entrées `Fault`, pas les deux.

```
<!--../source/soap/xml/snippets/soap_request_example.xml-->

<!--SOAP Request-->
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
    SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <SOAP-ENV:Body>
        <m:GetQuotation xmlns:m="http://www.tp.com/Quotation">
            <m:Item>Computers</m:Item>
        </m:GetQuotation>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<!--../source/soap/xml/snippets/soap_response_example.xml-->

<!--SOAP Response-->
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
    SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <SOAP-ENV:Body>
        <m:GetQuotationResponse xmlns:m="http://www.tp.com/Quotation">
            <m:Quotation>This is Qutation</m:Quotation>
        </m:GetQuotationResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<!--../source/soap/xml/snippets/soap_version_1.2_example.xml-->

<!--Example using another SOAP version-->
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
            <n:priority>1</n:priority>
            <n:expires>2001-06-22T14:00:00-05:00</n:expires>
        </n:alertcontrol>
    </env:Header>
    <env:Body>
        <m:alert xmlns:m="http://example.org/alert">
            <m:msg>Pick up Mary at school at 2pm</m:msg>
        </m:alert>
    </env:Body>
</env:Envelope>
```

1.3.3.5 Les éléments `Fault`

1. **rôle** : signaler des cas d'erreur lors du traitement des messages SOAP.
2. **contrainte** : un message SOAP peut contenir au plus un élément `Fault`.
3. **sous-éléments** :
   - `<faultCode>` : code identifiant le **type d'erreur** :
     1. `SOAP-ENV:Client` : le contenu du message est mal formatté ou contient des informations incorrectes.
     2. `SOAP-ENV:Server` : erreur côté serveur empêchant le traitement du message.
     3. `SOAP-ENV:VersionMismatch` : l'espace de noms utilisé pour l'enveloppe SOAP est invalide.
     4. `SOAP-ENV:MustUnderstand` : un élément du SOAP header ayant une valeur `true` pour son attribut `mustUnderstand` n'a pas été reconnu.
   - `<faultString>` : explication de l'erreur en langage naturel.
   - `<faultActor>` :
     1. **motivation** : étant donné qu'un message SOAP peut être traité par un chemin composé de plusieurs noeuds de services SOAP, il est essentiel de pouvoir identifier le noeud qui pourrait causer une erreur lors du traitement du message.
     2. **définition** : information identifiant l'initiateur de l'erreur.
     3. **conséquence** : un noeud non final dans un chemin de traitement d'un message SOAP doit inclure un élément `<faultActor>` dans l'éléments `Fault` éventuellement renvoyé.
   - `<detail>` : définition précise de l'erreur

```xml
<!--../source/soap/xml/snippets/fault_element_example.xml-->

<!--Example d'un élément Fault-->
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
            <faultstring xsi:type="xsd:string">
            Failed to locate method (ValidateCreditCard) in class (examplesCreditCard) at
                /usr/local/ActivePerl-5.6/lib/site_perl/5.6.0/SOAP/Lite.pm line 1555.
            </faultstring>
      </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.3.4 Encodage des messages SOAP

1.3.4.1 Introduction

1. **besoin** : un message SOAP contient des données typées → nécessité de définir un moyen pour les encoder.
2. **vocabulaire** :
   - **valeurs** : données échangées.
   - **types** : type des données échangées, définis via des schémas de définition XML ou importés depuis un espace de noms.
3. **encodage** :
   - **définition** : représentation XML des données échangées.
   - **mécanisme** : valeur de l'attribut `SOAP-ENV:encodingStyle`.
   - **namespaces** :
     1. **SOAP1.1** : "http://schemas.xmlsoap.org/soap/encoding"
     2. **SOAP1.2** : "http://www.w3.org/2001/12/soap-encoding"
4. **décodage** : construction des données à échanger à partir de leurs représentations XML.

1.3.4.2 Types

1.3.4.2.1 Types simples (*Scalar Types*)

1. **définition** : types à valeurs simples (*idem que ceux définis par la spécification XML Schema*).

2. `<element xsi:type = "xsd:type_simple">...</element>`.
3. **exemples** : *int, string, boolean, binary, nonPositiveInteger, negativeInteger, ...*
4. **namespaces** :
   - http://www.w3.org/2001/XMLSchema-instance
   - http://www.w3.org/2001/XMLSchema

```
<!--../source/soap/xml/snippets/soap_encoding_example.xml-->

<!--Example d'encodage des données d'un message SOAP-->
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:getPriceResponse
            xmlns:ns1="urn:examples:priceservice"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
            <return xsi:type="xsd:double">54.99</return>
        </ns1:getPriceResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.3.4.2.2 Types composés (*Compound Types*)

1. **définition** : types à valeurs composées.
2. **exemples** : *structures, arrays, ...*
3. **structures** :
   - un type composé dans lequel les membres sont accessibles uniquement grâce à des noms différents.
   - une imbrication de balises de types différents dans une balise racine.
   - le type de la balise racine est défini par l'utilisateur dans un espace de noms utilisateur.
   - les balises imbriquées sont des éléments ayant des types simples/composés.
   - `<element xsi:type = "nsUtilisateur:structure">...</element>`.
4. **arrays** :
   - un type composé dans lequel les membres sont accessibles uniquement grâce à leurs positions.
   - une imbrication de balises du même type dans une balise racine.
   - `<element xsi:type="Array" arrayType="element_type[size]">...</element>` (***`Array`*** *et* ***`arrayType`*** *sont définis dans http://www.w3.org/2001/09/soap-encoding*).

```
<!--../source/soap/xml/snippets/soap_structure_response_example.xml-->
```

```xml
<!--Example of SOAP response with a structure return type-->
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:getProductResponse
            xmlns:ns1="urn:examples:pricelistservice"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
            <return
                xmlns:ns2="urn:examples"
                xsi:type="ns2:product">
                <name xsi:type="xsd:string">Red Hat Linux</name>
                <price xsi:type="xsd:double">54.99</price>
                <description xsi:type="xsd:string">Red Hat Linux Operating System</descripti
                <SKU xsi:type="xsd:string">A358185</SKU>
            </return>
        </ns1:getProductResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```xml
<!--../source/soap/xml/snippets/soap_array_response_example.xml-->

<!--Example of SOAP response with an array return type-->
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:getPriceListResponse
            xmlns:ns1="urn:examples:pricelistservice"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
            <return
                xmlns:ns2="http://www.w3.org/2001/09/soap-encoding"
                xsi:type="ns2:Array" ns2:arrayType="xsd:double[2]">
                <item xsi:type="xsd:double">54.99</item>
                <item xsi:type="xsd:double">19.99</item>
            </return>
        </ns1:getPriceListResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.3.5 Modèles de communication

SOAP supporte deux modèles de communication :

1. **Remote Procedure Call** (**RPC**);
2. XML documents.

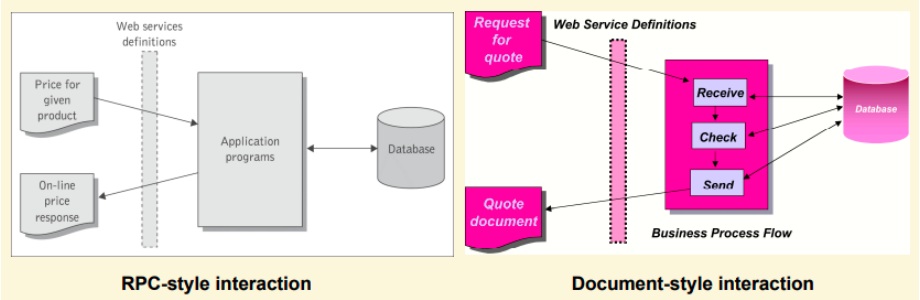

Figure 7: Modèles de communication de SOAP

1.3.5.1 SOAP en mode RPC

1. **description** :
   - un service web SOAP utilisant le modèle de communication RPC apparaît en tant qu'une procédure à invoquer pour l'application cliente.
   - l'interaction entre un client et un service web RPC est centrée autour d'une interface spécifique.
   - le SOAP Body contient un élément désignant l'opération à invoquer, éventuellement avec des sous-éléments désignant ses paramètres d'entrée.
2. **processus** : une application cliente envoie une requête encapsulant un appel à une méthode distante, éventuellement avec des arguments, qui renverra une réponse contenant sa valeur de retour.

```
<!--../source/soap/xml/snippets/soap_rpc_example.xml-->

<!--Exemple d'une requête SOAP utilisant le mode RPC-->
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
        <tx:Transaction-id
        xmlns:tx="http://www.transaction.com/transactions"
        env:mustUnderstand="1">512
         </tx:Transaction-id>
    </env:Header>
```

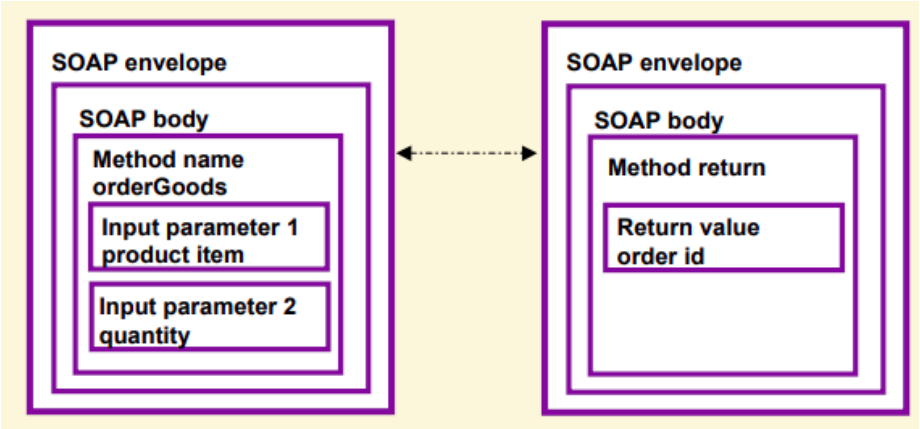

Figure 8: Modèle de communication RPC de SOAP

```
    <env:Body>
        <!--Nom de l'opération-->
        <m:GetProductPrice>
            <!--Argument d'entrée-->
            <product-id>450R6OP</product-id>
        </m:GetProductPrice>
    </env:Body>
</env:Envelope>

<!--Exemple d'une réponse SOAP utilisant le mode RPC-->
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
        <!--Optional context information-->
    </env:Header>
    <env:Body>
        <m:GetProductPriceResponse>
            <product-price>134.42</product-price>
        </m:GetProductPriceResponse>
    </env:Body>
</env:Envelope>
```

1.3.5.2 SOAP en mode Document

1. **description** :
    - le SOAP Body contient un ou plusieurs éléments XML sans aucune

règle de formatage SOAP (*i.e., le SOAP Body contient tout ce que l'expéditeur et le destinataire du message se mettent d'accord dessus*).
    - les éléments XML constituent un fragment d'un document XML qui ne reflète par forcément une structure XML explicite.
2. **processus** :
    - le moteur d'exécution de SOAP accepte le document encapsulé par le SOAP Body et l'envoie à l'application destinataire tel qu'il est.
    - il peut éventuellement avoir une réponse associée.

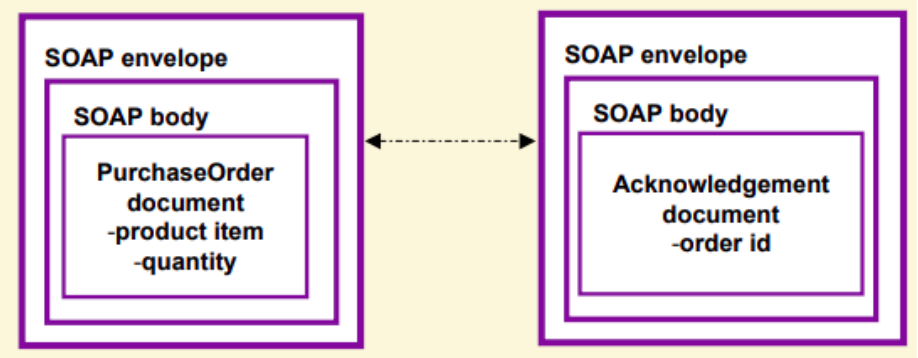

Figure 9: Modèle de communication document de SOAP

```
<!--../source/soap/xml/snippets/soap_document_example.xml-->

<!--Example d'une requête SOAP utilisant le mode document-->
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <tx:Transaction-id
        xmlns:tx="http://www.transaction.com/transactions"
        env:mustUnderstand="1">512
         </tx:Transaction-id>
    </env:Header>
    <env:Body>
        <po:PurchaseOrder orderDate="2004-12-02"
        xmlns:po="http://www.plastics_supply.com/POs">
            <po:from>
                <po:accountName>RightPlastics</po:accountName>
                <po:accountNumber>PSC-0343-02</po:accountNumber>
            </po:from>
            <po:to>
                <po:supplierName>Plastic Supplies Inc.</po:supplierName>
                <po:supplierAddress>Yara Valley Melbourne</po:supplierAddress>
            </po:to>
```

```
            <po:product>
                <po:product-name>injection molder</po:product-name>
                <po:product-model>G-100T</po:product-model>
                <po:quantity>2</po:quantity>
            </po:product>
        <:po:PurchaseOrder>
    </env:Body>
</env:Envelope>
```

1.3.6 Transport des messages SOAP

1. SOAP peut être utilisé avec n'importe quel protocole de transport (e.g., HTTP, FTP, SMTP, ...), mais il est utilisé souvent avec le protocole HTTP.
2. avec HTTP, les requêtes/réponses SOAP sont encapsulées dans des requêtes/réponses HTTP.
3. SOAP utilise les méthodes HTTP suivantes :
   - `GET` : seul la réponse contient un message SOAP.
   - `POST` : la requête et la réponse contiennent toutes les deux un message SOAP.
4. SOAP utilise les mêmes codes de status HTTP.

1.3.6.1 Composition d'une requête HTTP avec SOAP

1. **éléments header HTTP standard** :
   - le type de l'opération HTTP, le nom du programme à invoquer et la version de HTTP utilisée.
   - le nom du hôte.
   - le type du contenu : `text/xml; charset="UTF-8"`.
   - la longueur du contenu.
   - ...
2. **élément header HTTP SOAP** : `SOAPAction` spécifiant l'URI de l'action/méthode SOAP à invoquer.
3. **message SOAP**.

1.3.6.2 Composition d'une réponse HTTP avec SOAP

1. **éléments header HTTP standard** :
   - la version de HTTP utilisée, le code de retour et le message de retour.
   - le type du contenu : `text/xml; charset="UTF-8"`
   - la longueur du contenu.
   - ...
2. **message SOAP**.

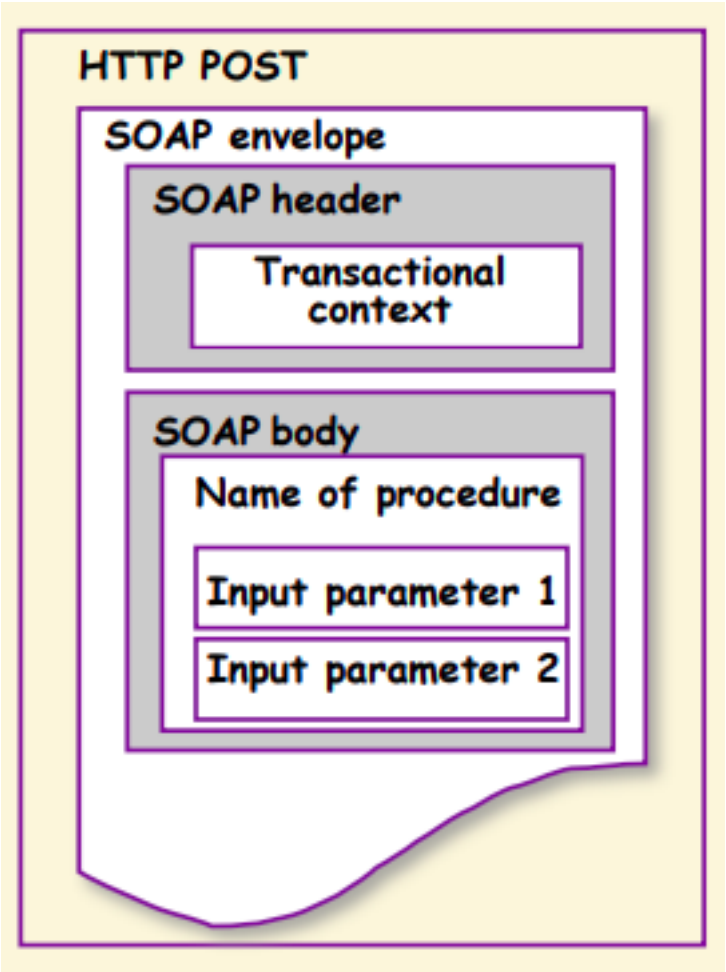

Figure 10: SOAP avec HTTP

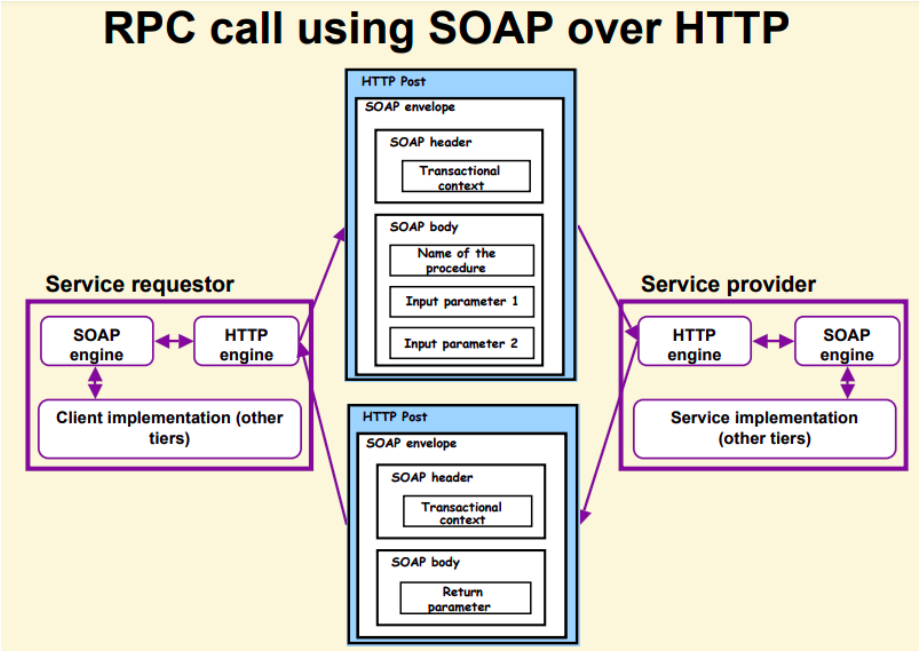

Figure 11: SOAP avec HTTP et RPC

1.3.7 Exemples

1.3.7.1 StockQuote

```
<!--../source/soap/xml/examples/stockquote.xml-->

<!--Exemple 1 : StockQuote, un ensemble de services
pour des informations sur des actions boursières-->

<!--REQUÊTE-->
<!-- début de la partie de portage sur HTTP -->
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset='utf-8'
Content-Length: nnnn
SOAPAction: "URI" <!-- entête HTTP spécifique pour qu'un
serveur puisse reconnaître la requête SOAP -->
<!-- fin de la partie HTTP -->

<!-- début du message SOAP -->
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!-- début de l'enveloppe SOAP -->
    <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="URI"> <!-- namespace utilisateur -->
                <symbol>DIS</symbol>
            </m:GetLastTradePrice>
        </SOAP-ENV:Body>
    <SOAP-ENV:Envelope>
<!-- fin de l'enveloppe SOAP -->

<!--RÉPONSE-->
<!-- début de la partie réponse HTTP -->
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<!-- fin de la réponse HTTP -->

<!-- début du message SOAP -->
<?xml version="1.0" encoding="UTF-8"?>

<!-- début du enveloppe SOAP transformé -->
    <SOAP-ENV:Envelope
        xmlns:SOAP-ENV="https://schemas.xmlsoap.org/soap/envelope"
        SOAP-ENV:encodingStyle="https://schemas.xmlsoap.org/soap/encoding">
        <SOAP-ENV:Body>
            <m:GetLastTradePriceResponse xmlns:m="URI">
                <Price>34.5</Price>
            </m:GetLastTradePriceResponse>
        </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
<!-- fin de l'enveloppe SOAP -->
```

1.3.7.2 BabelFish

```
<!--../source/soap/xml/examples/babelfish.xml-->

<!--Exemple 2 : WS de translation fourni par XMethods Babelfish
en utilisant SOAP et HTTP-->
<!--REQUÊTE-->
POST /perl/soaplite.cgi HTTP/1.0
Host: services.xmethods.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 538
```

```
SOAPAction: "urn:xmethodsBabelFish#BabelFish"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:BabelFish
            xmlns:ns1="urn:xmethodsBabelFish"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
            <translationmode xsi:type="xsd:string">en_fr</translationmode>
            <sourcedata xsi:type="xsd:string">Hello, world!</sourcedata>
        </ns1:BabelFish>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

<!--RÉPONSE-->
HTTP/1.0 200 OK
Date: Sat, 09 Jun 2001 15:01:55 GMT
Server: Apache/1.3.14 (Unix) tomcat/1.0 PHP/4.0.1pl2
SOAPServer: SOAP::Lite/Perl/0.50
Cache-Control: s-maxage = 60, proxy-revalidate
Content-Type: text/xml; charset="utf-8"
Content-Length: 539

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:BabelFishResponse
        xmlns:ns1="urn:xmethodsBabelFish">
            <return xsi:type="xsd:string">Bonjour, monde!</return>
        </ns1:BabelFish>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.3.7.3 XYZ Quotation

```
<!--../source/soap/xml/examples/xyz.xml-->
```

```
<!--Exemple 3 : WS de quotation fourni par xyz.org
en utilisant SOAP et HTTP-->
<!--REQUÊTE-->
POST /Quotation HTTP/1.1
Host: www.xyz.org
Content-Type: text/xml; charset="utf-8"
Content-Length: nnn

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
    SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <SOAP-ENV:Body
    xmlns:m="http://www.xyz.org/quotations">
        <m:GetQuotation>
            <m:QuotationsName>Microsoft</m:QuotationsName>
        </m:GetQuotation>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

<!--RÉPONSE-->
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnn

<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
    SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <SOAP-ENV:Body
    xmlns:m="http://www.xyz.org/quotations">
        <m:GetQuotationResponse>
            <m:Quotation>Here is the quotation</m:Quotation>
        </m:GetQuotationResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.3.8 Avantages et inconvénients

1. **avantages** :
   - portabilité;
   - facilité d'usage avec des firewalls;
   - protocol ouvert et standardisé;
   - interopérabilité.
2. **inconvénients** :
   - couplage fort à HTTP.

- *Statelessness*.
- sérialisation par valeur non par référence.

1.4 WSDL (Web Service Description Language)

1.4.1 Introduction

1.4.1.1 Définition

1. un **protocole standard** basé sur XML pour la description de services web XML : *comment y accéder et quelles opérations sont supportées*.
2. un **dialecte XML modulaire** : *un document WSDL importe d'autres documents WSDL*.
3. développé par IBM et Microsoft.
4. le langage utilisé par **Universal Description, Discovery, and Integration** (**UDDI**), un registre XML universel de services web.

1.4.1.2 Utilité

1. fournir un moyen à une entreprise de décrire ses services web XML d'une manière verbeuse.
2. fournir un moyen aux particuliers/entreprises pour consommer programmatiquement les services web XML offerts par d'autres particuliers/entreprises.

1.4.1.3 Objectif

1. utiliser un **protocole d'échange de données** (*souvent SOAP*) et un **système de typage** (*souvent des schémax XML*) pour **décrire** et **utiliser** des **services web XML**.
2. décrire les services web comme un ensemble d'**opérations et de messages abstraits** associés à des **protocoles et des serveurs réseaux**.

1.4.1.4 Processus général

1. un client souhaitant consommer un service web, consulte son WSDL afin de savoir quelles opérations sont offertes.
2. les types spécifiques au service web sont décrits dans le WSDL sous forme d'un schéma XML.
3. le client utilise alors SOAP pour invoquer l'une des opérations du WS décrites dans son WSDL.

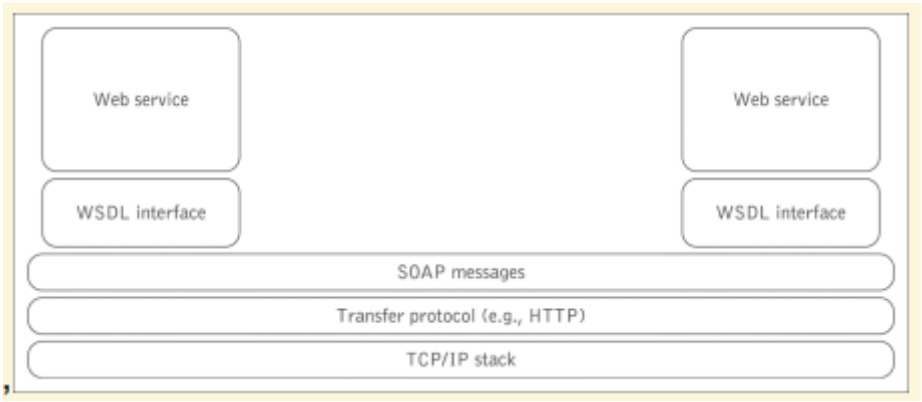

Figure 12: WSDL et SOAP

1.4.2 Structure

```
<!--../source/soap/xml/snippets/wsdl_generic_structure.xml-->

<?xml version="1.0" encoding="UTF-8"?>
<definitions>
    <types>
    <!--définition de types...-->
    </types>

    <message>
    <!--définition d'un message...-->
    </message>

    <portType>
        <!--définition d'un portType...-->
        <operation>
        <!--définition d'une opération...-->
        </operation>
    </portType>

    <binding>
    <!--définition d'un binding...-->
    </binding>

    <service>
    <!--définition d'un service...-->
        <port>
        <!--définition d'un port...-->
```

```
        </port>
    </service>
</definitions>
```

1.4.2.1 Principe

1. WSDL décompose les services web en trois éléments principaux : les types de données, les opérations, et les liaisons.
2. tous les autres éléments d'un document WSDL sont contenus dans ces trois éléments.
3. les éléments principaux peuvent être développés dans des documents séparés pouvant être combinés et réutilisés pour créer des documents WSDL complets.

1.4.2.2 Parties abstraite et concrète

1. **partie abstraite** :
   - les types de données utilisés par le service web ;
   - les messages élémentaires échangés par le service web ;
   - les collections d'opérations du service web.
2. **partie concrète** :
   - le(s) protocole(s) d'encodage des données ;
   - le(s) protocole(s) de transport ;
   - les adresses réseau associées aux opérations.

1.4.2.3 Éléments obligatoires

1.4.2.3.1 L'élément `<definitions>`

1. l'élément racine d'un document WSDL, contenant le reste des éléments utilisés pour la description d'un service web.
2. définit le nom du service web.
3. contient les espaces de noms utilisés dans le document.

1.4.2.3.2 L'élément `<types>`

Contient les définitions abstraites des types de données (*souvent sous forme de schémas XML*) utilisés dans les messages échangés du service web.

1.4.2.3.3 L'élément `<message>`

1. décrit d'une manière abstraite les données à échanger sous forme d'un message contenant des éléments `<part>`.
2. l'élément `<part>` :

- un document XML.
- un élément décrit en XML qui sera mappé à une/plusieurs opérations sous forme d'un paramètre d'entrée, valeur de retour/d'erreur, ...

1.4.2.3.4 L'élément `<operation>`

1. décrit une définition abstraite d'une opération : *e.g., signature d'une méthode, un message queue, un processus métier, ...*
2. chaque opération référence des éléments `<message>` comme paramètres d'entrée ou comme valeur de retour/d'erreur.

1.4.2.3.5 L'élément `<portType>`

1. décrit d'une manière abstraite une collection d'`<operation>` qui seront liées à un protocole de transport et d'encodage de données.
2. chaque `<portType>` peut avoir plusieurs liaisons à plusieurs protocoles de transport et d'encodage de données.

1.4.2.3.6 L'élément `<binding>`

Décrit une association concrète d'un `<portType>` à un protocole de transport et un protocole d'encodage de données.

1.4.2.3.7 L'élément `<port>`

Décrit un **point d'entrée/endpoint** consistant d'une association d'un `<binding>` à une adresse réseau de communication du service web.

1.4.2.3.8 L'élément `<service>`

Décrit une collection de `<port>` d'adresses relatives et d'éventuelles extensions de celles-ci.

1.4.2.4 Éléments facultatifs

1.4.2.4.1 L'élément `<documentation>`

1. fournit une documentation lisible par les humains.
2. peut être inclus dans n'importe quel élément d'un document WSDL.

1.4.2.4.2 L'élément `<import>`

Permet d'importer d'autres documents WSDL ou d'autres schémas XML dans le document WSDL courant.

1.4.3 Outils

1. **toolkits** :
   - vers Java et réciproquement : *Java2WSDL, WSDL2Java, ...* (*propriétaires*)
   - vers C# et réciproquement.
2. générateur WSDL à partir de déploiement SOAP ou EJB.
3. générateur de proxy SOAP à partir de WSDL.

1.4.4 Détails sur les éléments WSDL

1.4.4.1 `<definitions>`

1. l'élément racine d'un document WSDL, contenant le reste des éléments utilisés pour la description d'un service web.
2. définit le nom du service web.
3. contient les espaces de noms utilisés dans le document.

```
<!--../source/soap/xml/snippets/wsdl_definitions_structure.xml-->

<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
    targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!--targetNamespace : une convention de schémas XML
        permettant à un document WSDL de se référencer-->
    <!--xmlns : l'espace de noms par défaut du document WSDL courant,
        (celui du language WSDL).
        Ceci permet de références les éléments WSDL
        sans les préfixer par wsdl:-->
    <!--xmlns:soap : l'espace de noms de SOAP-->
    <!--xmlns:tns : l'espace de noms de tous les éléments
        personnalisés utilisés dans le document WSDL courant-->
    <!--xmlns:xsd : l'espace de noms de tous les types prédéfinis
        au sein de XMLSchema-->
    ...
</definitions>
```

1.4.4.2 `<types>`

1. contient les définitions abstraites des types de données utilisés dans les messages échangés du service web.

2. les types de données sont soit des documents XML, soit des parties d'un document XML.
3. le système de typage utilisé par WSDL par défaut est les schémas XML, mais WSDL n'est pas lié exclusivement à un système de typage spécifique.
4. si le service web décrit par le document WSDL courant utilise uniquement des types simples (*e.g., strings, integers, ...*) natifs aux schémas XML, alors `<types>` peut être omis.
5. les types de données définis au sein d'un document WSDL peuvent être réutilisés par d'autres documents WSDL.

```
<!--../source/soap/xml/snippets/wsdl_types_structure.xml-->

...
<types>
    <schema
        targetNamespace="http://example.com/stockquote.xsd"
        xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="TracePriceRequest">
            <complexType>
                <all>
                    <element name="tickerSymbol" type="string">
                </all>
            </complexType>
        </element>
            <complexType>
                <all>
                    <element name="price" type="float">
                </all>
            </complexType>
        <element>
        </element>
    </schema>
    <!--
    targetNamespace : l'espace de noms du schéma XML
    défini dans le document WSDL courant

    xmlns : l'espace de noms par défaut utilisé au sein du schéma XML défini,
    (celui du schéma XML standard contenant tous les types prédéfinis).
    Ceci permet de références les types prédéfinis
    sans les préfixer par xsd:
    -->
</types>
```

1.4.4.3 `<message>`

1. décrit d'une manière abstraite les données à échanger entre le fournisseur

du service web et ses consommateurs.

2. chaque `<message>` :
   - est référencé par une/plusieurs `<operation>` en tant qu'un élément `<input>` ou `<output>`.
   - est composé de 0..*n* éléments `<part>`.
3. les éléments `<part>` :
   - **description** :
     1. des documents XML ;
     2. des éléments décrits en XML qui seront mappés à une `<operation>` sous forme de paramètres d'entrée (*si* `<message>` *est un* `<input>`) ou valeurs de retour/d'erreur (*si* `<message>` *est un* `<output>`/`<fault>`).
   - **constitution** : chaque `<part>` possède un nom et un type de données.
   - **types** : **natifs** au système de typage ou **personnalisés**.

```
<!--../source/soap/xml/snippets/wsdl_message_structure.xml-->

<!--message qui sera utilisé en tant qu'<input>-->
<message name="SayHelloRequest">
    <!--paramètre d'entrée de type natif-->
    <part name="firstName" type="xsd:string" />
</message>

<!--message sera utilisé en tant qu'<output>-->
<message name="SayHelloResponse">
    <!--valeur de retour de type natif-->
    <part name="greeting" type="xsd:string" />
</message>

<!--message qui sera utilisé en tant qu'<input>-->
<message name="TelMsg1">
    <!--paramètre d'entrée de type natif-->
    <part name="areacode" type="xsd:int" />

    <!--paramètre d'entrée de type natif-->
    <part name="number" type="xsd:string" />
</message>

<!--message qui sera utilisé en tant qu'<input>-->
<message name="TelMsg2">
    <!--paramètre d'entrée de type personnalisé
    <phone> défini comme <element> dans <types>-->
    <part name="personne" element="tns:phone" />
</message>
```

1.4.4.4 `<operation>`

1. décrit une définition abstraite d'une opération : *e.g., signature d'une méthode, un message queue, un processus métier, ...*
2. chaque opération :
   - référence :
     1. 0..1 `<message>` comme paramètre d'entrée (*élement* ***<input>***) ;
     2. 0..1 `<message>` comme valeur de retour (*élément* ***<output>***) ;
     3. 0..*n* `<message>` comme valeur d'erreur (*élément* ***<fault>***) ;
   - peut éventuellement définir leur ordre via l'attribut `parameterOrder`.
3. **types** :
   - `One-Way` : l'opération reçoit un message en entrée (*i.e., un* ***<input>***).
   - `Request-Response` :
     1. l'opération reçoit un message en entrée (*i.e., un* ***<input>***) et retourne un message en sortie (*i.e., un* ***<output>***).
     2. l'opération peut éventuellement encapsuler des erreurs (*i.e., des* ***<fault>***).
   - `Solicit-Response` :
     1. l'opération envoie un message en sortie (*i.e., un* ***<output>***) et reçoit un message en entrée (*i.e., un* ***<input>***).
     2. l'opération peut éventuellement encapsuler des erreurs (*i.e., des* ***<fault>***).
   - `Notification` : l'opération envoie un message en sortie (*i.e., un* ***<output>***).

```
<!--../source/soap/xml/snippets/wsdl_operation_structures.xml-->

<!--Grammaire d'une opération One-Way-->
<?xml version="1.0" encoding="UTF-8"?>
<definitions>
    ...
    <portType>+
        <operation name="someNameToken">
            <input name="someNameToken"? message="messageQualifiedName" />
        </operation>
    </portType>
    ...
</definitions>

<!--Structure d'une opération Request-Response-->
<?xml version="1.0" encoding="UTF-8"?>
<definitions>
    ...
    <portType>+
        <operation name="someNameToken" parameterOrder="someNameToken1 someNameToken2 ..."?>
            <input name="someNameToken1"? message="messageQualifiedName" />
            <output name="someNameToken2"? message="messageQualifiedName" />
```

```
            <fault name="someNameTokenN"? message="messageQualifiedName" />*
        </operation>
    </portType>
    ...
</definitions>

<!--Structure d'une opération Solicit-Response-->
<?xml version="1.0" encoding="UTF-8"?>
<definitions>
    ...
    <portType>+
        <operation name="someNameToken" parameterOrder="someNameToken1 someNameToken2 ..."?>
            <output name="someNameToken1"? message="messageQualifiedName" />
            <input name="someNameToken2"? message="messageQualifiedName" />
            <fault name="someNameTokenN"? message="messageQualifiedName" />*
        </operation>
    </portType>
    ...
</definitions>

<!--Structure d'une opération Notification-->
<?xml version="1.0" encoding="UTF-8"?>
<definitions>
    ...
    <portType>+
        <operation name="someNameToken">
            <output name="someNameToken"? message="messageQualifiedName" />
        </operation>
    </portType>
    ...
</definitions>
```

1.4.4.5 `<binding>`

1.4.4.5.1 Principe

1. lier les descriptions abstraites d'opérations regroupées au sein d'un `<portType>` à un protocole de transport et à un protocole d'encodage des données échangées.
2. les liaisons peuvent être réalisées via SOAP, HTTP (GET et POST), MIME, ...
3. plusieurs liaisons peuvent être réalisées pour un même `<portType>`.

1.4.4.5.2 Attributs

1. `name` : le nom du `<binding>`.
2. `type` : le `<portType>` lié par le `<binding>`.

1.4.4.5.3 `<binding>` avec SOAP

1. WSDL1.1 contient des extensions built-in pour SOAP1.1 afin de spécifier :
   - les SOAP headers ;
   - les SOAP encoding styles ;
   - l'élement header HTTP `SOAPAction`.
2. les extensions built-in :
   - `<soap:binding>`.
   - `<soap:operation>`.
   - `<soap:body>`.

1.4.4.5.4 `<soap:binding>`

Indique que la liaison se fait via SOAP en spécifiant le style d'encodage des données échangées via les messages SOAP et leur protocole de transport.

```
<!--../source/soap/xml/snippets/wsdl_soap_binding_structure.xml-->

<binding name="SomeBindingName" type="tns:SomePortType">
    <soap:binding style="StyleTYPE" transport="TransportURI" />
    <!--
    StyleTYPE :
        1. rpc : SOAP en mode RPC.
        2. document : SOAP en mode Document.
    TransportURI :
        1. http://schemas.xmlsoap.org/soap/http : SOAP avec HTTP.
        2. http://schemas.xmlsoap.org/soap/smtp : SOAP avec SMTP.
        3. etc.
    -->
    ...
</binding>
```

1.4.4.5.5 `<soap:operation>`

1. associe une opération du `<portType>` lié à une implémentation fournie par l'élément header HTTP `SOAPAction`.
2. peut éventuellement redéfinir le style d'encodage des données utilisées par l'opération.

```
<!--../source/soap/xml/snippets/wsdl_soap_operation_structure.xml-->

<binding name="SomeBindingName" type="tns:SomePortType">
```

```
    <soap:binding style="StyleTYPE" transport="TransportURI" />
    <operation name="NameOfOperationInPortType">
        <soap:operation soapAction="operationURI" style="OperationStyleType"?>
        ...
    </operation>
    ...
</binding>
```

1.4.4.5.6 `<soap:body>`

Associe à chaque message d'une opération du `<portType>` lié un style d'encodage des données, un espace de noms, et un mode de sérialisation pour les données encodées.

```
<!--../source/soap/xml/snippets/wsdl_soap_body_structure.xml-->

<binding name="SomeBindingName" type="tns:SomePortType">
    <soap:binding style="StyleTYPE" transport="TransportURI" />
    <operation name="NameOfOperationInPortType">
        <soap:operation soapAction="operationURI" style="OperationStyleType"?>
            <input>
            <soap:body
                use="DataSerializationMode"
                encodingStyle="EncodingStyleURI"
                namespace="WebServiceTypeNamespace"/>
                <!--
                DataSerializationMode : déterminer la manière dont les données
                apparaîssent dans les messages SOAP échangés (encoded ou literal)
                EncodingStyleURI :
                    1. e.g., http://schemas.xmlsoap.org/soap/encoding
                    2. encodingStyle est utilisé uniquement si use="encoded"
                WebServiceTypeNamespace :
                    1. l'espace de noms du service web
                    2. namespace est utilisé uniquement si use="encoded"
                -->
            </input>
            ...
    </operation>
    ...
</binding>
```

1.4.4.5.7 Modèles de `<bindings>` avec SOAP

1. `rpc/encoded` : utilisation de SOAP en mode **RPC** avec encodage de types, tel que `<SOAP-ENV:Body>` contient un seul sous-élément désignant

l'opération invoquée, éventuellement encapsulant ses paramètres d'entrée avec leurs types.
2. `rpc/literal` : utilisation de SOAP en mode **RPC** sans encodage de types, tel que `<SOAP-ENV:Body>` contient un seul sous-élément désignant l'opération invoquée, éventuellement encapsulant ses paramètres d'entrée sans leurs types.
3. `document/encoded` : non utilisé en pratique car non conforme aux règles du **WS-I** (**Web Services Interoperability**).
4. `document/literal` : utilisation de SOAP en mode **Document** sans encodage de types, tel que `<SOAP-ENV:Body>` du message obtenu contient plusieurs sous-éléments.
5. `document/literal` **wrapped** : idem que `document/literal`, mais `<SOAP-ENV:Body>` du message obtenu contient un seul sous-élément encapsulant plusieurs éléments.

1.4.4.5.8 Le modèle `rpc/encoded`

1. **description** : utilisation de SOAP en mode **RPC** avec encodage de types, tel que `<SOAP-ENV:Body>` contient un seul sous-élément désignant l'opération invoquée, éventuellement encapsulant ses paramètres d'entrée avec leurs types.
2. **avantages** :
   - WSDL simple.
   - le nom de l'opération apparaît dans le message SOAP obtenu → facilité d'invoquer l'implémentation de la méthode correspondante.
3. **inconvénients** :
   - le type d'encodage de chaque élément de l'opération invoquée est défini via `xsi:type` et est explicité pour chaque message échangé → *overhead* sur la performance.
   - n'est pas conforme aux règles du WS-I.
   - on ne peut pas valider par un validateur XML le corps du message SOAP obtenu, vu que l'élément référençant l'opération n'est pas conforme à un système de typage (*schéma XML*), et est défini uniquement au niveau d'un `<operation>` dans le document WSDL.

```
// ../source/soap/java/snippets/myMethod.java

// signature d'une méthode qui sera invoquée via un service web
public void myMethod(int x, float y);
```

```
<!--../source/soap/xml/examples/wsdl_rpc_encoded_example.xml-->

<!--WSDL simplifié de myMethod utilisant le modèle rpc/encoded-->
...
<message name="myMethodRequest">
    <part name="x" type="xsd:int" />
```

```
    <part name="y" type="xsd:float" />
</message>
<message name="empty" />

<portType name="PT">
    <!--L'opération à invoquer-->
    <operation name="myMethod">
        <!--Paramètre d'entrée-->
        <input message="tns:myMethodRequest" />
        <!--Valeur de retour-->
        <output message="tns:empty" />
    </operation>
</portType>

<binding name="SomeBinding" type="tns:PT">
    <soap:binding
        style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="myMethodRequest">
        <soap:operation soapAction="myMethodRequestURI" />
        <input>
            <soap:body
                use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                namespace="WebServiceTypeNamespace" />
        </input>
        <output>
            <soap:body
                use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                namespace="WebServiceTypeNamespace" />
        </output>
    </operation>
</binding>

<!--
le message SOAP simplifié obtenu
en invoquant la méthode avec x = 5 et y = 5.0
-->
<SOAP-ENV:Envelope>
    <SOAP-ENV:Body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
```

```
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.4.4.5.9 Le modèle `rpc/literal`

1. **description** : utilisation de SOAP en mode **RPC** sans encodage de types, tel que `<SOAP-ENV:Body>` contient un seul sous-élément désignant l'opération invoquée, éventuellement encapsulant ses paramètres d'entrée sans leurs types.
2. **avantages** :
   - WSDL simple.
   - le nom de l'opération apparaît dans le message SOAP obtenu.
   - les types d'encodage des éléments de l'opération invoquée ne figurent pas dans le message SOAP obtenu.
   - conforme aux règles du WS-I.
3. **inconvénient** : on ne peut pas valider par un validateur XML le corps du message SOAP obtenu, vu que l'élément référençant l'opération n'est pas conforme à un système de typage (*schéma XML*), et est défini uniquement au niveau d'un `<operation>` dans le document WSDL.

```
// ../source/soap/java/snippets/myMethod.java

// signature d'une méthode qui sera invoquée via un service web
public void myMethod(int x, float y);
```

```
<!--../source/soap/xml/examples/wsdl_rpc_literal_example.xml-->

<!--WSDL simplifié de myMethod utilisant le modèle rpc/literal-->
...
<message name="myMethodRequest">
    <part name="x" type="xsd:int" />
    <part name="y" type="xsd:float" />
</message>
<message name="empty" />

<portType name="PT">
    <!--L'opération à invoquer-->
    <operation name="myMethod">
        <!--Paramètre d'entrée-->
        <input message="tns:myMethodRequest" />
        <!--Valeur de retour-->
        <output message="tns:empty" />
    </operation>
</portType>

<binding name="SomeBinding" type="tns:PT">
```

```
    <soap:binding
        style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="myMethodRequest">
        <soap:operation soapAction="myMethodRequestURI" />
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>

<!--
le message SOAP simplifié obtenu
en invoquant la méthode avec x = 5 et y = 5.0
-->
<SOAP-ENV:Envelope>
    <SOAP-ENV:Body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.4.4.6 Le modèle `document/literal`

1. **description** : utilisation de SOAP en mode **Document** sans encodage de types, tel que `<SOAP-ENV:Body>` du message obtenu contient plusieurs sous-éléments.
2. **avantages** :
   - les types d'encodage des éléments du document ne figurent pas dans le message SOAP obtenu.
   - conforme aux règles de WS-I, mais avec des restrictions.
   - on peut valider le message SOAP obtenu avec un validateur XML, vu que tout le contenu du corps du message est défini à travers le système de typage utilisé par le document WSDL (*Un schéma XML*).
3. **inconvénients** :
   - WSDL plus compliqué.
   - le nom de l'opération invoquée n'apparaît pas dans le message SOAP obtenu → difficulté (*voire impossibilité*) d'invoquer l'implémentation de la méthode correspondante.

- les règles WS-I indiquent que `<SOAP-ENV:Body>` doit contenir un seul sous-élément, alors qu'il peut contenir plusieurs.

```
// ../source/soap/java/snippets/myMethod.java

// signature d'une méthode qui sera invoquée via un service web
public void myMethod(int x, float y);
```

```
<!--../source/soap/xml/examples/wsdl_document_literal_example.xml-->

<!--WSDL simplifié de myMethod utilisant le modèle document/literal-->
...
<types>
    <schema>
        <element name="xElement" type="xsd:int" />
        <element name="yElement" type="xsd:float" />
    </schema>
</types>

<message name="myMethodRequest">
    <part name="x" element="tns:xElement" />
    <part name="y" element="tns:yElement" />
</message>
<message name="empty" />

<portType name="PT">
    <!--L'opération à invoquer-->
    <operation name="myMethod">
        <!--Paramètre d'entrée-->
        <input message="tns:myMethodRequest" />
        <!--Valeur de retour-->
        <output message="tns:empty" />
    </operation>
</portType>

<binding name="SomeBinding" type="tns:PT">
    <soap:binding
        style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="myMethodRequest">
        <soap:operation soapAction="myMethodRequestURI" />
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
```

```
        </output>
    </operation>
</binding>

<!--
le message SOAP simplifié obtenu
en invoquant la méthode avec x = 5 et y = 5.0
-->
<SOAP-ENV:Envelope>
    <SOAP-ENV:Body>
        <xElement>5</xElement>
        <yElement>5.0</yElement>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.4.4.7 Le modèle `document/literal` wrapped

1. **description** :
   - utilisation de SOAP en mode **Document** sans encodage de types, tel que `<SOAP-ENV:Body>` contient un seul sous-élément désignant l'opération invoquée, éventuellement encapsulant ses paramètres d'entrée sans leurs types.
   - l'élément `e` correspondant à l'opération désigne un élément défini au sein du schéma XML du document WSDL et éventuellement de type complex encapsulant ses paramètres.
   - l'opération définie par `<operation>` en WSDL prend en entrée un seul élément `<input>` référençant `e`.
2. **avantages** :
   - les types d'encodage des éléments du document ne figurent pas dans le message SOAP obtenu.
   - conforme aux règles de WS-I.
   - on peut valider le message SOAP obtenu avec un validateur XML, vu que tout le contenu du message est défini à travers le système de typage utilisé par le document WSDL (*Un schéma XML*).
3. **inconvénient** : WSDL plus compliqué.

```
// ../source/soap/java/snippets/myMethod.java

// signature d'une méthode qui sera invoquée via un service web
public void myMethod(int x, float y);

<!--../source/soap/xml/examples/wsdl_document_literal_wrapped_example.xml-->

<!--WSDL simplifié de myMethod utilisant le modèle document/literal wrapped-->
...
<types>
```

```
    <schema>
        <!--Element wrapper référencé par l'input de l'opération-->
        <element name="myMethod">
            <complexType>
                <sequence>
                    <element name="x" type="xsd:int" />
                    <element name="y" type="xsd:float" />
                </sequence>
            </complexType>
        </element>
        <!--Element wrapper référencé par l'output de l'opération-->
        <element name="myMethodResponse">
            <complexType />
        </element>
    </schema>
</types>

<message name="myMethodRequest">
    <part name="parameters" element="tns:myMethod" />
</message>
<message name="empty">
    <part name="parameters" element="tns:myMethodResponse" />
</message>

<portType name="PT">
    <!--L'opération à invoquer-->
    <operation name="myMethod">
        <!--Paramètre d'entrée-->
        <input message="tns:myMethodRequest" />
        <!--Valeur de retour-->
        <output message="tns:empty" />
    </operation>
</portType>

<binding name="SomeBinding" type="tns:PT">
    <soap:binding
        style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="myMethodRequest">
        <soap:operation soapAction="myMethodRequestURI" />
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
```

```
        </output>
    </operation>
</binding>

<!--
le message SOAP simplifié obtenu
en invoquant la méthode avec x = 5 et y = 5.0
-->
<SOAP-ENV:Envelope>
    <SOAP-ENV:Body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.4.4.8 `<port>`

Décrit un **point d'entrée/endpoint** unique et nommé consistant d'une association d'un `<binding>` à une adresse réseau de communication du service web.

```
<!--../source/soap/xml/snippets/wsdl_port_structure.xml-->

<port name="SomeEndPointName" binding="tns:SomeBindingName">
    <soap:address location="WebServiceAddress" />
</port>
```

1.4.4.9 `<service>`

1. décrit une collection de `<port>` d'adresses relatives et d'éventuelles extensions de celles-ci.
2. possède un nom et préférablement un élément `<documentation>` permettant de fournir des informations lisibles par les utilisateurs du service web.
3. permet à un client utilisant le service web de savoir :
   - comment y accéder ;
   - quel endpoint utiliser.

```
<!--../source/soap/xml/snippets/wsdl_service_structure.xml-->

<service name="SomeWebServiceName">
    <documentation>?
        Documentation describing the web service's access, endpoints,
        message formats, and other important details
```

```
    </documentation>
    <port name="SomeEndPointName" binding="tns:SomeBindingName">
        <soap:address location="WebServiceAddress" />
    </port>
    ...
</service>
```

1.4.5 Exemples

1.4.5.1 Exemple type d'un document WSDL

```
<!--../source/soap/xml/examples/foo_service.wsdl-->

<?xml version="1.0" encoding="UTF-8"?>
<!--NAMESPACES-->
<definitions name="FooSample"
    targetNamespace="http://tempuri.org/wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
    xmlns:tns="http://tempuri.org/xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!--TYPES-->
    <types>
        <schema
            targetNamespace="http://tempuri.org/xsd"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding"
            elementFormDefault="qualified">
        </schema>
    </types>

    <!--MESSAGES-->
    <!--message correspondant à un paramètre de type entier-->
    <message name="Simple.foo">
        <part name="arg" type="xsd:int" />
    </message>

    <!--message correspondant à un résultat de type entier-->
    <message name="Simple.fooResponse">
        <part name="result" type="xsd:int" />
    </message>

    <!--PORT TYPES-->
    <!--portType contenant l'opération foo-->
```

```
<portType name="SimplePortType">
    <!--l'opération foo-->
    <operation name="foo" parameterOrder="arg">
        <!--argument en entrée = le message Simple.foo-->
        <input message="tns:Simple.foo" />
        <!--valeur de retour = le message Simple.fooResponse-->
        <output message="tns:Simple.fooResponse" />
    </operation>
</portType>

<!--BINDINGS-->
<!--binding des opérations du portType SimplePortType
    au protocole SOAP/HTTP-->
<binding name="SimpleBinding" type="tns:SimplePortType">
    <!--transport = SOAP/HTTP et encodage = SOAP RPC-->
    <soap:binding
        style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <!--l'opération foo du portType SimplePortType-->
    <operation name="foo">
        <!--préciser l'URI de l'opération-->
        <soap:operation soapAction="http://tempuri.org/action/Simple.foo" />
        <!--message d'entrée-->
        <input>
            <!--comment sera créé le message d'entrée-->
            <soap:body
                use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                namespace="http://tempuri.org/message" />
        </input>
        <!--message de sortie-->
        <output>
            <!--comment sera créé le message de sortie-->
            <soap:body
                use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                namespace="http://tempuri.org/message" />
        </output>
    </operation>
</binding>

<!--SERVICE-->
<!--service exposant les opérations du portType "SimplePortType"
    liées par le binding "SimpleBinding"-->
<service name="FOOSAMPLEService">
    <!--port "SimplePort" liant "SimpleBinding" à une adresse réseau-->
```

```
        <port name="SimplePort" binding="tns:SimpleBinding">
            <soap:address location="http://localhost/FooSample/FooSample.asp" />
        </port>
    </service>
</definitions>
```

1.4.5.2 Exemple d'un document WSDL décrivant un WS consistant d'une opération affichant "Hello World!"

```
<!--../source/soap/xml/examples/hello_world.wsdl-->

<?xml version="1.0" encoding="UTF-8"?>
<!--NAMESPACES-->
<definitions name="HelloService"
    targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
    xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!--TYPES-->
    <!--Using the built-in types of XMLSchema-->

    <!--MESSAGES-->
    <message name="sayHelloRequest">
        <part name="firstName" type="xsd:string" />
    </message>

    <message name="sayHelloResponse">
        <part name="greeting" type="xsd:string" />
    </message>

    <!--PORT TYPES-->
    <portType name="tns:Hello_PortType">
        <operation name="sayHello">
            <input message="tns:sayHelloRequest" />
            <output message="tns:sayHelloResponse" />
        </operation>
    </portType>

    <!--BINDINGS-->
    <binding name="Hello_Binding" type="tns:Hello_PortType">
        <soap:binding
            style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http" />
```

```
        <operation name="sayHello">
            <soap:operation soapAction="sayHello" />
            <input>
                <soap:body
                    use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                    namespace="urn:examples:helloservice" />
            </input>
            <output>
                <soap:body
                    use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                    namespace="urn:examples:helloservice" />
            </output>
        </operation>
    </binding>

    <!--SERVICE-->
    <service name="Hello_Service">
        <documentation>WSDL File for HelloService</documentation>
        <port name="Hello_Port" binding="tns:Hello_Binding">
            <soap:address location="http://www.examples.com/SayHello" />
        </port>
    </service>
</definitions>
```

1.4.5.3 Exemple d'un document WSDL décrivant un WS permettant de manipuler un address book

```
<!--../source/soap/xml/examples/address_book.wsdl-->

<?xml version="1.0" encoding="UTF-8"?>
<!--NAMESPACES-->
<definitions name="Address Example"
    targetNamespace="urn:xml-soap-address-demo"
    xmlns="http://schemas.xmlsoap.org/wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
    xmlns:tns="http://tempuri.org/xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!--TYPES-->
    <types>
        <schema
            targetNamespace="http://tempuri.org/xsd"
```

```
        xmlns="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
        <complexType name="phone">
            <element name="areaCode" type="int" />
            <element name="exchange" type="string" />
            <element name="number" type="string" />
        </complexType>
        <complexType name="address">
            <element name="streetNum" type="int" />
            <element name="streetName" type="string" />
            <element name="city" type="string" />
            <element name="state" type="string" />
            <element name="zip" type="int" />
            <element name="phoneNumber" type="tns:phone" />
        </complexType>
    </schema>
</types>

<!--MESSAGES-->
<message name="AddEntryRequest">
    <part name="name" type="xsd:string" />
    <part name="address" type="tns:address" />
</message>

<message name="GetAddressFromNameRequest">
    <part name="name" type="xsd:string" />
</message>

<message name="GetAddressFromNameResponse">
    <part name="address" type="tns:address" />
</message>

<!--PORT TYPES-->
<portType name="AddressBook">
    <operation name="addEntry">
        <input message="tns:AddEntryRequest" />
    </operation>
    <operation name="getAddressFromName">
        <input message="tns:GetAddressFromNameRequest" />
        <output message="tns:GetAddressFromNameResponse" />
    </operation>
</portType>

<!--BINDINGS-->
<binding name="AddressBookSOAPBinding" type="tns:AddressBook">
    <soap:binding
```

```
            style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="addEntry">
            <soap:operation soapAction="" />
            <input>
                <soap:body
                    use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                    namespace="urn:AddressFetcher2" />
            </input>
        </operation>
        <operation name="getAddressFromName">
            <soap:operation soapAction="" />
            <input>
                <soap:body
                    use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                    namespace="urn:AddressFetcher2" />
            </input>
            <output>
                <soap:body
                    use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
                    namespace="urn:AddressFetcher2" />
            </output>
        </operation>
    </binding>

    <!--SERVICE-->
    <service name="AddressBookService">
        <port name="AddressBook" binding="tns:AddressBookSOAPBinding">
            <soap:address location="http://www.mycomp.com/soap/servlet/rpcrouter" />
        </port>
    </service>
</definitions>
```

1.5 UDDI (Universal Description, Discovery, and Integration)

1.5.1 Définition

1. UDDI est une spécification décrivant un modèle centralisé et réplicable pour la description, la publication, la découverte et l'intégration de services web.

2. il s'agit d'une spécification pour le développement d'un annuaire distribué de services web.
3. un registre UDDI est une implémentation de la spécification UDDI.
4. utilise SOAP, CORBA, ou RMI pour la communication.
5. utilise WSDL pour la description de ses interfaces.

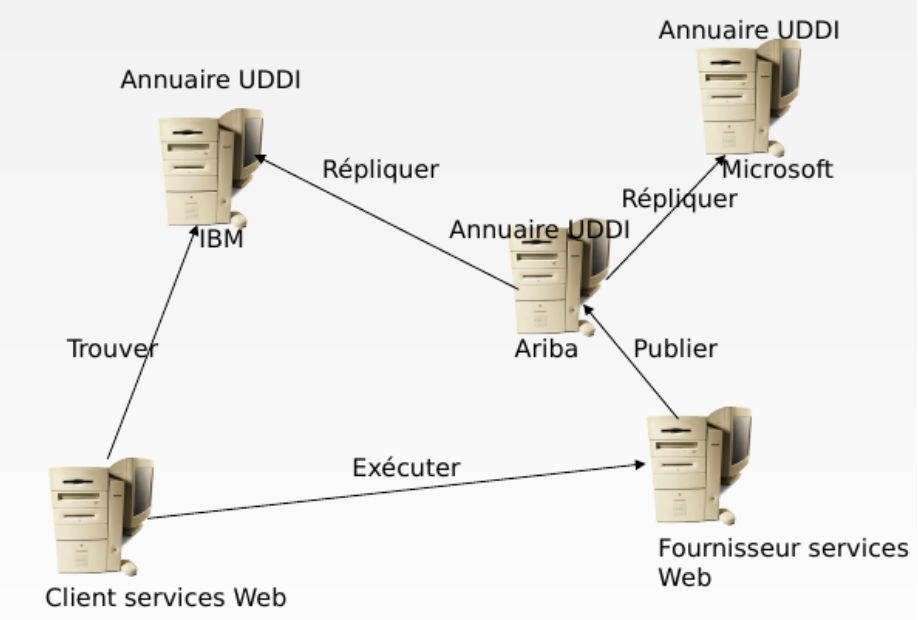

Figure 13: Schéma d'utilisation de UDDI

1.5.2 Motivation

1. les concepteurs d'applications web ont besoin de trouver les fournisseurs des services web.
2. les outils ont besoin d'obtenir les descriptions des services web pour coder les appels nécessaires.
3. les applications peuvent avoir besoin de trouver de nouveaux services web dynamiquement.

1.5.3 Origines

1. créé par Microsoft, IBM et Ariba en 2000.
2. actuellement maintenu par OASIS (Oracle, Microsoft, ...).

1.5.4 Schéma de fonctionnement

1. les programmeurs et les sociétés peuplent la section **Service Type Registrations** du registre UDDI avec des descriptions de types des services web publiés.
2. les sociétés peuplent la section **Business Registrations** du registre UDDI avec des métadonnées sur elles-mêmes ainsi que les services web publiés.
3. le registre UDDI affecte un **Universally Unique ID** (**UUID**) à chaque type de service dans **Service Type Registrations** et à chaque enregistrement dans **Business Registrations**.
4. les moteurs de recherche et applications interrogent la section **Service Type Registrations** du registre UDDI afin de découvrir les services web publiés par les sociétés et les utiliser pour satisfaire leurs besoins.

1.5.5 Architecture

1. **modèle de données** : schéma XML pour décrire les sociétés et leurs WS offerts.
2. **spécifications API** : interfaces pour la publication et la recherche de services web dans/depuis un registre UDDI.
3. **services cloud** : sites distants fournissant des implémentations pour les spécifications UDDI et synchronisant les données du data model régulièrement.

1.5.6 UBR (UDDI Business Registry)

1. **définition** :
   - un système constitués de plusieurs noeuds contenant des données UDDI répliquées et synchronisées.
   - le système est logiquement centralisé mais physiquement distribué.
   - les données enregistrées au sein d'un noeud racine du système sont répliquées vers ses autres noeuds.
2. **alias** : *Public Cloud*.
3. **remarques** :
   - la synchronisation des noeuds du UBR se fait actuellement tous les jours.
   - les registres UDDI privés ne sont pas synchronisés avec le UBR.

1.5.7 Modèles de données

1. **définition** : un schéma XML pour décrire les sociétés et leurs WS offerts.
2. **structure de données** : `<businessEntity>`, `<businessService>`, `<bindingTemplate>`, `<tModel>`, ...

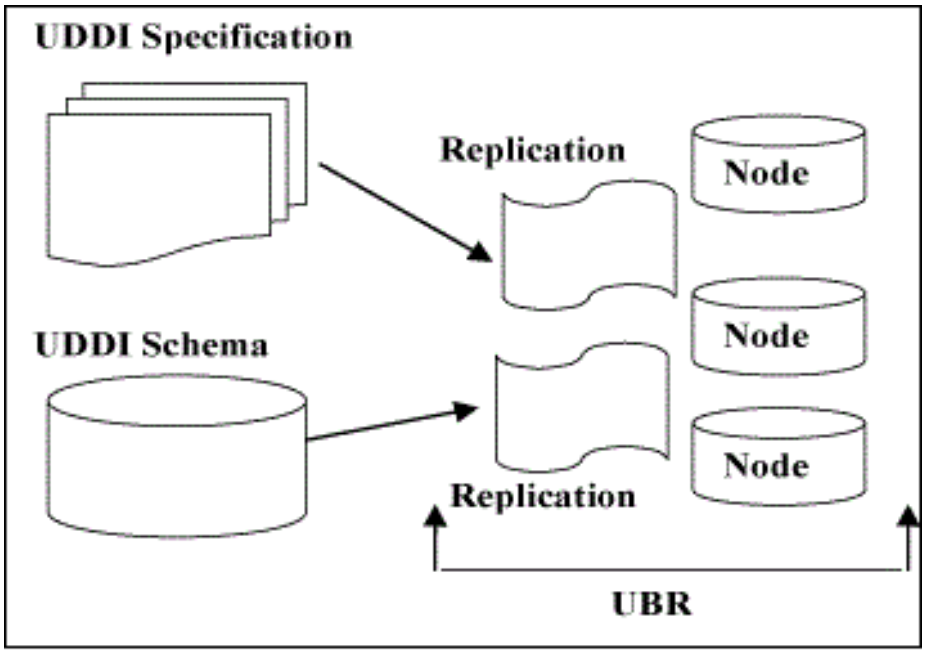

Figure 14: Architecture technique de UDDI

1.5.7.1 `<businessEntity>`

1. représente le fournisseur du service web.
2. contient :
   - le nom du fournisseur avec une courte description textuelle ;
   - les informations de contact ;
   - la catégorie industrielle du fournisseur ;
   - des informations taxonomiques pour catégoriser le fournisseur et faciliter sa découverte ;
   - la liste des services web fournis.

```
<!--../source/soap/xml/snippets/uddi_businessEntity_example.xml-->

<businessEntity
    businessKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40"
    operator = "http://www.ibm.com"
    authorizedName = "John Doe">
    <name>Acme Company</name>
    <description>We create cool Web services</description>

    <contacts>
        <contact useType = "general info">
            <description>General Information</description>
```

```
            <personName>John Doe</personName>
            <phone>(123) 123-1234</phone>
            <email>jdoe@acme.com</email>
      </contact>
    </contacts>

    <businessServices>
        ...
    </businessServices>
    <identifierBag>
        <keyedReference
            tModelKey = "UUID:8609C81E-EE1F-4D5A-B202-3EB13AD01823"
            name = "D-U-N-S"
            value = "123456789" />
    </identifierBag>

    <categoryBag>
        <keyedReference
            tModelKey = "UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2"
            name = "NAICS"
            value = "111336" />
    </categoryBag>
</businessEntity>
```

1.5.7.2 `<businessService>`

1. représente un service web fourni par le fournisseur représenté par l'élément `<businessEntity>` parent.
2. contient :
    - des informations descriptives, mais non techniques, d'un service web fourni ;
    - des informations taxonomiques pour catégoriser le service web fourni et faciliter sa découverte et utilisation ;
    - la liste des implémentations du service web fourni.

```
<!--../source/soap/xml/snippets/uddi_businessService_example.xml-->

<businessService
    serviceKey = "uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
    businessKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
    <name>Hello World Web Service</name>
    <description>A friendly Web service</description>

    <bindingTemplates>
        ...
    </bindingTemplates>
```

```
    <categoryBag />
</businessService>
```

1.5.7.3 `<bindingTemplate>`

1. représente les détails d'un binding d'un service web décrit par l'élément `<businessService>` parent.
2. contient :
   - des informations techniques sur le binding d'un service web ;
   - le point d'entrée du service web lié ;
   - un pointeur vers le document WSDL du service web lié ;
   - des processus métiers associés au service web lié.

```
<!--../source/soap/xml/snippets/uddi_bindingTemplate_example.xml-->

<bindingTemplate
    serviceKey = "uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
    bindingKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
    <description>Hello World SOAP Binding</description>
    <accessPoint URLType = "http">http://localhost:8080</accessPoint>

    <tModelInstanceDetails> <!--reference to an existing <tModel>-->
        <tModelInstanceInfo tModelKey = "uuid:EB1B645F-CF2F-491f-811A-4868705F5904">
            <instanceDetails>
                <overviewDoc>
                    <description>
                        references the description of the WSDL service definition
                    </description>

                    <overviewURL>http://localhost/helloworld.wsdl</overviewURL>
                </overviewDoc>
            </instanceDetails>
        </tModelInstanceInfo>
    </tModelInstanceDetails>
</bindingTemplate>
```

1.5.7.4 `<tModel>`

1. représente un modèle technique (*Technical Model*) d'une `<bindingTemplate>` d'un service web.
2. contient les spécifications de l'API UDDI du service web, incluant des informations taxonomiques et techniques.

```
<!--../source/soap/xml/snippets/uddi_tModel_example.xml-->
```

```
<tModel
    tModelKey = "uuid:xyz987..."
    operator = "http://www.ibm.com"
    authorizedName = "John Doe">
    <name>HelloWorldInterface Port Type</name>
    <description>An interface for a friendly Web service</description>

    <overviewDoc>
        <overviewURL>http://localhost/helloworld.wsdl</overviewURL>
    </overviewDoc>
</tModel>
```

1.5.8 Types de registres UDDI

1. **publiques** : *e.g., Microsoft, IBM, ...*
2. **privés ou locaux**.

1.5.9 Sections d'un registre UDDI

1.5.9.1 Business registrations

1. **contenu** : métadonnées sur les services web, incluant des pointeurs sur leurs descriptions WSDL.
2. **acteurs** : les entreprises fournissent des informations sur leur propre compte et sur leurs services web.

1.5.9.2 Service type registrations (tModels)

1. **contenu** : un ensemble de définitions d'éléments `<portTypes>` WSDL pour manipuler/rechercher un registre UDDI.
2. **acteurs** : les organismes de normalisation, les développeurs et les entreprises fournissent des informations sur les types de services publiés.

1.5.10 Structure d'un registre UDDI

1. **pages blanches** : une liste d'éléments `<businessEntity>`.
2. **pages jaunes** : une liste d'éléments `<businessService>`.
3. **pages vertes** : une liste d'éléments `<bindingTemplate>`.

1.5.11 Publier un service web dans un registre UDDI

1. UDDI offre une interface aux fournisseurs de services web afin de pouvoir publier leurs descriptions au sein d'un registre UDDI.

2. cette interface comporte des opérations, dont :
    - `save_business` : créer/modifier les informations d'un élément `<businessEntity>` contenu dans un registre UDDI.
    - `save_service` : créer/modifier les informations d'un élément `<businessService>` contenu dans un registre UDDI.
    - `save_binding` : créer/modifier les informations d'un élément `<bindingTemplate>` contenu dans un registre UDDI.
    - `save_tModel` : créer/modifier les informations d'un élément `<tModel>` contenu dans un registre UDDI.
    - `delete_business` : supprimer un élément `<businessEntity>` contenu dans un registre UDDI.
    - `delete_service` : supprimer un élément `<businessService>` contenu dans un registre UDDI.
    - `delete_binding` : supprimer un élément `<bindingTemplate>` contenu dans un registre UDDI.
    - `delete_tModel` : supprimer un élément `<tModel>` contenu dans un registre UDDI.
    - ...
3. l'interaction avec le registre UDDI peut se faire via SOAP, CORBA, RMI, ...

```
<!--../source/soap/xml/snippets/uddi_web_service_publishing_soap_message_example.xml-->

POST /save_business HTTP/1.1
Host: www.XYZ.com
Content-Type: text/xml; charset = "utf-8"
Content-Length: nnnn
SOAPAction: "save_business"

<?xml version = "1.0" encoding = "UTF-8" ?>
<Envelope
    xmlns = "http://schemas/xmlsoap.org/soap/envelope/">
    <Body>
        <save_business
            generic = "2.0"
            xmlns = "urn:uddi-org:api_v2">
            <businessKey = ""></businessKey>
            <name>XYZ, Pvt Ltd.</name>
            <description>
                Company is involved in giving Stat-of-the-art....
            </description>

            <identifierBag> ... </identifierBag>
            ...
      </save_business>
   </Body>
```

```
</Envelope>
```

1.5.12 Rechercher un service web dans un registre UDDI

1. UDDI offre une interface aux clients/applications pour la recherche/découverte de services web fournis et publiés par des fournisseurs au sein de registres UDDI.
2. cette interface comporte des opérations, dont :
   - `find_business` : retourner une liste de `<businessEntity>` contenus dans un registre UDDI selon les critères paramétrant l'opération.
   - `find_service` : retourner une liste de `<businessService>` contenus dans un registre UDDI selon les critères paramétrant l'opération.
   - `find_binding` : retourner une liste de `<bindingTemplate>` contenus dans un registre UDDI selon les critères paramétrant l'opération.
   - `find_tModel` : retourner une liste de `<tModel>` contenus dans un registre UDDI selon les critères paramétrant l'opération.
   - `get_businessDetail` : retourner des informations sur un `<businessEntity>` contenu dans un registre UDDI.
   - `get_serviceDetail` : retourner des informations sur un `<businessService>` contenu dans un registre UDDI.
   - `get_bindingDetail` : retourner des informations sur un `<bindingTemplate>` contenu dans un registre UDDI.
   - `get_tModelDetail` : retourner des informations sur un `<tModel>` contenu dans un registre UDDI.
   - ...
3. l'interaction avec le registre UDDI peut se faire via SOAP, CORBA, RMI, ...

```
<!--../source/soap/xml/snippets/uddi_web_service_discovery_soap_message_example.xml-->

POST /get_businessDetail HTTP/1.1
Host: www.XYZ.com
Content-Type: text/xml; charset = "utf-8"
Content-Length: nnnn
SOAPAction: "get_businessDetail"

<?xml version = "1.0" encoding = "UTF-8" ?>
<Envelope xmlns = "http://schemas/xmlsoap.org/soap/envelope/">
    <Body>
        <get_businessDetail
            generic = "2.0"
            xmlns = "urn:uddi-org:api_v2">
            <businessKey = "C90D731D-772HSH-4130-9DE3-5303371170C2">
            </businessKey>
        </get_businessDetail>
```

```
    </Body>
</Envelope>
```

1.5.13 Problèmes

1. pas de modérateurs pour les registres UDDI → risque d'entrées erronées, dupliquées ou frauduleuses.
2. pas de moyen pour assurer la qualité du service (*e.g. niveau de sécurité assuré, fiabilité/disponibilité, support des transactions, ...*)
3. centralisation excessive de certains registres UDDI.
4. registres UDDI majoritairement locaux et privés, et donc, fermés au public.

1.6 SOAP & sécurité

@To_Update

1.6.1 Introduction

1. besoins :
   - au niveau du client :
     a. authentification des utilisateurs
     b. gestion des autorisations
   - au niveau des messages :
     a. signatures digitales des messages
     b. cryptage des messages
2. solutions :
   - WSS (Web Service Security) :
     a. au début : pas de sécurité
     b. après : IBM, Microsoft et Verisign travaillaient sur WS-security
     c. enfin : Oasis adoptent WS-security sous le nom WSS
     d. versions :
        – version 1 : mars 2004
        – version 2 : février 2006
   - SOAP Message Security : assurer le respect des principes de confidentialité et d'intégrité des données

1.6.2 Terminologie

1. principe de confidentialité : assurer que les données ne sont accessibles que par les entités/processus/individus autorisés

2. principe d'intégrité : vérifier les données lors de leur réception pour s'assurer qu'il ne sont pas altérées au cours de leur circulation sur le réseau
3. signature digitale : - définition : une valeur hash unique associée à un message - utilité : a. authentification : permet au destinataire d'un message de s'assurer que le message est bien créé et envoyé par une entité émettrice b. non répudiation : permet au destinataire d'un message de s'assurer que l'entité émettrice ne peut pas nier le fait qu'elle a créé et envoyé le message c. intégrité : permet au destinataire de s'assurer qu'un message n'a pas été altéré au cours de sa circulation sur le réseau depuis son émission - processus : a. côté émetteur : * l'émetteur doit générer une paire de clés publique/privée * il applique un algorithme de hachage sur le message pour obtenir le digest du message * le digest du message est ensuite crypté avec sa clé privée pour obtenir la signature digitale * l'émetteur envoie ainsi le message signé par sa signature digitale b. côté receveur : * le receveur reçoit le message avec la signature digitale * il utilise la clé publique de l'émetteur pour déchiffrer la signature digitale pour obtenir le digest du message DPK (authentification) * il utilise ensuite le même algorithme de hachage utilisé par l'émetteur sur le message et compare le digest obtenu avec DPK (intégrité) - limitation : une attaque man-in-the-middle peut bloquer le message de l'émetteur et permettre au hacker d'envoyer son propre message avec sa propre signature digitale -> compromis de l'authentification
4. jeton de sécurité :
    - définition : une collection d'affirmations (claims)
    - affirmaton : une déclaration faite par une entité (nom, identité, privilège, etc.)
    - confirmation d'une affirmation : vérification qu'une affirmation s'applique à une entité (i.e. que l'affirmation déclarée par l'entité est valide)
    - jeton de sécurité signé : cryptage et confirmation des affirmations du jeton par une autorité dédiée, par exemple :
        a. certificat X.509
        b. tickets Kerberos

1.6.3 WSS

1. objectif :

    - utiliser des jetons de sécurité pour authentifier les clients du WS
    - assurer la confidentialité et l'intégrité des messages SOAP

2. besoins :

    - définir les formats des jetons de sécurité
    - avoir recours à des domaines de confiance (trust domains)

- choisir les formats et les technologies de cryptage
- assurer le cryptage de bout en bout (end-to-end encryption) de la communication et non seulement au niveau des messages SOAP

3. envelope SOAP générique utilisant WSS :

```
<SOAP-ENV:Header>
    <!--
    SOAP-ENV:actor="uri" est un attribut optionnel permettant de spécifier le rôle du n
    s'il y en a plus que deux interlocuteurs :
    si uri = "http://schemas.xmlsoap.org/soap/actor/next", alors il s'agit du client WS
    si uri est vide, alors il s'agit du fournisseur du WS
    -->
    <wsse:Security SOAP:actor="..." SOAP-ENV:mustUnderstand="true">
        ...
    </wsse:Security>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
    ...
</SOAP-ENV:Body>
```

4. authentification :
   - approche : codifier les jetons de sécurité au sein des messages SOAP
   - types d'authentification :
     a. jetons UserNameToken : une paire login/mot de passe classique
     b. jetons d'authentification binaires (BinarySecurityToken) : tickets Kerberos ou certificat X.509
   - schéma classique :
     a. processus :
        – avant de consommer un WS, un client WS envoie une requête à un fournisseur de jetons de sécurité
        – le fournisseur de jetons lui retourne un jeton de sécurité qui sera embarqué au sein de tous les messages SOAP échangés entre le client et le fournisseur du WS
        – le client envoie une requête SOAP signée par le token au serveur WS
        – le serveur WS vérifiera l'authenticité du client en validant le jeton embarqué au sein de sa requête auprès du fournisseur des jetons
        – si la validité du jeton est confirmée, le fournisseur WS renvoie une réponse SOAP au client
     b. figure : (cf. cours_fournis/02.1- Les services web SOAP.pdf, slide 97)

1.6.3.1 Jetons UserNameToken

1. définition : un jeton de sécurité permettant à un client WS de s'authentifier en utilisant une paire login/mot de passe
2. éléménts :
   - : l'élément racine désignant le jeton UserName (contenant les autres éléments)
   - : le nom de l'utilisateur associé au token
   - : le mot de passe de l'utilisateur
   - : attribut spécifiant le format du mot de passe :
     a. si type="PasswordText": transmission du mot de passe en clair (plain text)
     b. si type="PasswordDigest": transmission du digest du mot de passe (cipher text)
     c. Password digest :
        - définition : une valeur hashée du mot de passe, générée en utilisant l'algorithme de cryptage SHA-1 et l'encodage UTF-8
        - calcul : PasswordDigest = Base64(SHA-1(Password, Created, Nonce))
   - : une chaîne de chiffrement (hash) générée aléatoirement
   - : le datetime de création du token
3. schéma de fonctionnement avec un mot de passe haché : (cf. cours_fournis/02.1-Les services web SOAP.pdf, slide 102)
4. attaques possibles :
   - message SOAP capturé par un man-in-the-middle attack -> jeton de sécurité peut être attrapé et réutilisé
   - message SOAP bloqué sans possibilité d'arriver à la destination -> denial of service
5. mécanismes de défense contre les attaques :
   - en cas d'une attaque man-in-the-middle :
     a. utiliser un timestamp associé à un message SOAP avec une période d'expiration (dates de création et d'expiration du message)
     b. garder un historique des nonces reçus le temps que les timestamps expirent
6. exemples :
   - exemple de transmission en clair du mot de passe en utilisant un jeton UsernameToken <SOAP-ENV: Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
     Joe uselesspassword
     … </SOAP-ENV: Envelope>
   - exemple de transmission cryptée du mot de passe en utilisant un jeton UsernameToken <SOAP-ENV: Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
     Joe weYI3nXd8LjMNVksCKFV8t3rgHh3Rw== WScqanjCEAC4mQoBE07sAQ==
     2003-07-16T01:24:32Z

… </SOAP-ENV: Envelope>

1.6.3.2 Certificats X.509

1. problème : la signature digitale ne permet pas de vérifier l'identité d'une entité (authentification)
2. solution : utilisation de certificats digitals (digital certificates) issus d'un organisme de certification :
   - certificat : une association entre une entité et une clé publique
   - organisme de certification : une entité tierce auquelle on fait confiance (e.g. autorité de certification) émettant des certificats digitals
3. utilité : vérifier l'identité d'une entité et le fait qu'elle possède bien une clé publique qu'elle déclare comme étant la sienne
4. structure d'un certificat X.509 :
   - corps :
     a. version du certificat X.509
     b. numéro de série du certificat
     c. l'algorithme de cryptage utilisé pour signer le certificat
     d. le nom de l'autorité de certification
     e. la signature digitale de l'autorité de certification
     f. la période de validité du certificat (dates de début et de fin de validité du certificat)
     g. le nom de l'entité certifiée (i.e. le propriétaire)
     h. la clé publique de l'entité certifiée
   - signature : le digest du corps du certificat crypté par la clé privée de l'autorité de certification
5. processus :
   - un fournisseur WS doit être certifié par un organisme de certification
   - lorsqu'un client souhaite utiliser un WS du fournisseur, il doit vérifier le certificat digital associé à sa clé publique au préalable
   - pour vérifier le certificat digital, le client :
     a. déchiffre la signature du certificat associé en utilisant la clé publique de l'autorité de certification émettrice afin d'obtenir la valeur DC
     b. compare le corps du certificat associé à la clé publique du fournisseur WS avec DC, si elles sont égales, alors l'authentification est établie
   - une fois l'authentification établie, le processus d'utilisation du WS se déroule d'une manière classique
6. schémas : (cf. cours_fournis/02.1- Les services web SOAP.pdf, slides 105->106)

7. utilisation du certificat X.509 dans un message SOAP :

```
<SOAP-ENV:Header>
    <wsse:Security SOAP-ENV:mustUnderstand="true">
        <wsse:BinarySecurityToken
            ValueType="X509v3"
            EncodingType="wsse:type_d'encodage" <!--type_d'encodage='Base64' ou 'HexBin
            Id="SecurityToken-f49bd662-59a0-401a-ab23-1aa12764184f">
            MIIHdjCCB... <!--token value-->
        </wsse:BinarySecurityToken>
    </wsse:Security>
</SOAP-ENV:Header>
```

1.6.3.3 Kerberos

1. définition : un protocole d'authentification permettant à des noeuds se communiquant sur un réseau non sécurisé de s'authentifier l'un l'autre lors de leur communication, en utilisant des tickets, appelés tickets Kerberos
2. acteurs :
   - un client ayant une clé Kc
   - un serveur (Service Server our SS) offrant un service ayant un nom (Service Principal Name ou SPN) et ayant une clé Ks
   - un serveur d'authentification et de distribution de clés (Key Distribution Center ou KDC)
   - un serveur de tickets de sessions (Ticket Granting Service ou TGS)
3. schéma de fonctionnement : (cf. cours_fournis/02.1- Les services web SOAP.pdf, slides 107->109)
4. utilisation du protocole Kerberos dans un message SOAP :

```
<SOAP-ENV:Header>
    <wsse:Security SOAP-ENV:mustUnderstand="true">
        <wsse:BinarySecurityToken
            ValueType="Kerberosv5TGT" <!--ou Kerberosv5ST-->
            EncodingType="wsse:type_d'encodage" <!--type_d'encodage='Base64' ou 'HexBin
            Id="SecurityToken-f49bd662-59a0-401a-ab23-1aa12764184f">
            MIIHdjCCB... <!--token value-->
        </wsse:BinarySecurityToken>
    </wsse:Security>
</SOAP-ENV:Header>
```

1.6.3.3.1 Authentification du client par le serveur KDC

1. un client :

- envoie une requête au serveur KDC contenant son nom et le nom d'un serveur TGS,
- demandant un ticket (TGT ou ticket-granting ticket) lui permettant d'obtenir des tickets auprès du serveur TGS pour accéder à un service SPN d'un serveur SS sur le réseau

2. le serveur KDC :
   - vérifie l'identité du client et celle du serveur TGS en récupérant sa clé Ktgs et
   - renvoie au client :
     a. message A : une clé de session chiffrée par la clé du client Kc et
     b. message B : un ticket Ttgs timestamped et chiffré par la clé Ktgs du serveur TGS concerné
3. le client utilise sa clé pour déchiffrer la clé la session du message A

1.6.3.3.2 Obtention d'un ticket auprès du serveur TGS pour accéder à un service SPN d'un serveur SS

1. le client souhaitant utiliser un service SPN d'un serveur SS ayant une clé Ks :
   - demande un ticket auprès du serveur TGS en lui envoyant une requête contenant :
     a. message C : le ticket TGT Ttgs chiffré par Ktgs et l'identifiant du SPN qu'il souhaite utiliser
     b. message D : un authenticateur composé de l'identifiant du client et le timestamp d'émission de la requête, chiffré par la clé
2. le serveur TGS :
   - déchiffre le ticket Ttgs du message C en utilisant la clé Ktgs pour obtenir la clé
   - utilise la clé pour déchiffrer l'authenticateur du message D et obtenir l'identifiant du client
   - compare les identifiants du client des deux messages C et D, et renvoie en cas d'un match :
     a. message E : le ticket d'accès au serveur chiffré par la clé Ks du serveur SS
     b. message F : une clé de session chiffrée par la clé de session
3. le client utilise la clé de session pour déchiffrer la clé de session du message F

1.6.3.4 Requêtes client/serveur SS pour accéder au service SPN

1. après avoir obtenu un ticket de session pour se connecter au service SPN du serveur SS, le client envoie une requête contenant :
   - message G : le ticket d'accès au serveur chiffré par la clé Ks du serveur SS

- message H : un authenticateur composé de l'identifiant du client et d'une timestamp d'émission de la requête, chiffré par la clé

2. le serveur SS :
   - déchiffre le ticket d'accès au serveur en utilisant la clé Ks pour obtenir la clé
   - utilise la clé pour déchiffrer l'authenticateur du message H et obtenir l'identifiant du client
   - compare les identifiants du client des deux messages G et H, et renvoie en cas d'un match une réponse contenant :
     a. le timestamp de l'authenticateur + 1, chiffré avec la clé
3. le client :
   - reçoit la réponse du serveur SS et la déchiffre en utilisant la clé
   - vérifie la valeur du timestamp et en cas de match peut commencer à envoyer les requêtes WS pour utiliser le service

1.7 SOAP & interopérabilité

1.7.1 Aperçu

1. **standardisation** :
   - **WSDL** : *Qu'est ce qui est échangé ?*
   - **SOAP** : *Comment est-il échangé ?*
2. **profile** : ensemble de règles définies pour assurer l'interopérabilité des services web qui sont :
   - utilisées lors du développement d'un service web ;
   - plus ou moins suivies par les outils générant des requêtes SOAP ;
   - vérifiées par des outils monitorant les échanges entre un fournisseur d'un service web et ses clients.

1.7.2 WS-I (Web Services Interoperability)

1. **définition** : une organisation formée pour résoudre les inconsistantes et/ou ambiguïtés émergeant lors de la définition de services web afin d'assurer leur interopérabilité.
2. **rôle** : définir un nombre de profiles indiquant comment créer des services web interopérables.
3. **exemples de règles définies** :
   - un message doit être encodé en **UTF-8** ou **UTF-16** ;
   - si un client envoie un requête SOAP contenant un header **obligatoire** (*i.e. ayant l'attribut* `mustUnderstand="true"`) et que le receveur ne le reconnait pas, alors ce dernier doit générer un élément `<Fault>` ayant un `<faultCode>` de valeur `SOAP-ENV:MustUnderstand` dans sa réponse.

- un service ne doit pas exiger une acceptation des cookies côté client pour fonctionner correctement.
- un `<documentation>` peut apparaître comme un enfant d'un `<part>` dans un `<message>`.

2 Services web SOAP en Java, workflow, astuces, et bonnes pratiques

2.1 Aperçu de Java API for XML-Based Services (JAX-WS)

2.1.1 Principe

1. **définition** : le modèle de programmation de services Web complétant le modèle de base fourni par **Java API for XML-based RPC (JAX-RPC)**.
2. **utilité** : simplifier le développement des clients et services Web via l'utilisation de proxy dynamiques et d'annotations Java.

2.1.2 `wsimport`

1. **définition** : un outil CLI fourni avec la JDK permettant la génération automatique :
   - d'un service web à partir d'un document WSDL.
   - d'un client d'un service web à partir de son document WSDL.
2. les artefacts générés sont compatibles avec Java 5, ce qui les rend portables sur différentes versions et plateformes Java.

2.2 Créer un service web SOAP avec JAX-WS

2.2.1 Création du projet

Créer un projet Maven sans archetype :

1. `File` → `New` → `Project...`
2. sélectionner `Maven/Maven Project` puis `Next`.
3. cocher la case `Create a simple project (skip archetype selection)` puis `Next`.
4. remplir les informations de votre projet (au moins un Group Id, un artefact Id, et un nom) puis `Finish`.
5. **N.B.** : `Group Id` et `Artefact Id` suivent la même convention de nommage des packages.

2.2.2 Configuration

2.2.2.1 JDK

2.2.2.1.1 Windows

1. télécharger le dossier compressé de la openJDK8 : `https://jdk.java.net/java-se-ri/8-MR3` → `Windows 10 i586 Java Development Kit (md5) 92 MB`.
2. extraire le dossier compressé téléchargé dans `C:\Program Files\Java\`.
3. définir la variable d'environnement `JAVA_HOME` qui pointera sur le chemin du dossier extrait, et mettre à jour la variable d'environnement `PATH` en y ajoutant le dossier `bin/` du dossier extrait : `https://www.youtube.com/watch?v=104dNWmM6Rs`
4. tester l'installation :

```
java -version # donnera la version courante de la JRE (1.8)
wsimport -version # donnera la version courante de wsimport (2.2.9)
```

5. configurer le build path de votre projet afin de choisir la version openJDK8 :
   - clic droit sur le projet → `Build Path` → `Configure Build Path...`
   - dans l'onglet `Libraries`, clic sur `JRE System Library` → clic sur `Edit`
   - si la version par défaut n'est pas celle de openJDK8, clic sur `Alternate JRE` → clic sur `Installed JREs` → désélectionner la version actuelle puis clic sur `Add` → `Standard VM` → `Next` → `Directory...` → choisir le chemin `C:\Program Files\Java\<nom_dossier_extrait>\jre` → `Finish` → `Apply and Close`.
6. configurer le compliance level du compilateur à 1.8 :
   - clic droit sur le projet → `Build Path` → `Configure Build Path...`
   - si le niveau n'est pas déjà 1.8 par défaut alors clic sur l'onglet gauche `Java Compiler` → sélectionner 1.8 pour `Compiler compliance level`.

2.2.2.1.2 Mac OS X

1. installer la openJDK8 : `https://mkyong.com/java/how-to-install-java-on-mac-osx/`
2. tester l'installation :

```
java -version # donnera la version courante de la JRE (1.8)
wsimport -version # donnera la version courante de wsimport (2.2.9)
```

3. configurer le build path de votre projet afin de choisir la version openJDK8 :
   - clic droit sur le projet → `Build Path` → `Configure Build Path...`
   - dans l'onglet `Libraries`, clic sur `JRE System Library` → clic sur `Edit`

- si la version par défaut n'est pas celle de openJDK8, clic sur `Alternate JRE` → clic sur `Installed JREs` → désélectionner la version actuelle puis clic sur `Add` → `Standard VM` → `Next` → `Directory...` → choisir le chemin `/usr/local/opt/openjdk@8/libexec/openjdk.jdk` → `Finish` → `Apply and Close`.

4. configurer le compliance level du compilateur à 1.8 :
   - clic droit sur le projet → `Build Path` → `Configure Build Path...`
   - si le niveau n'est pas déjà 1.8 par défaut alors clic sur l'onglet gauche `Java Compiler` → sélectionner 1.8 pour `Compiler compliance level`.

2.2.2.1.3 Ubuntu

Installer la version 8 de openJDK en utilisant le gestionnaire de paquetages d'Ubuntu APT

```
sudo apt update # mettra à jour la liste des entrées dans le registre du gestionnaire de pa
sudo apt install openjdk-8-jdk # installera openJDK8
```

Tester l'installation

```
java -version # donnera la version courante de la JRE (1.8)
wsimport -version # donnera la version courante de wsimport (2.2.9)
```

Si la commande `java` et/ou `wsimport` n'est pas reconnue, alors il faut définir/modifier la variable d'environnement `JAVA_HOME` et l'ajouter à la variable d'environnement `PATH` :

```
# Depuis la ligne de commandes
# Ouvrir le fichier /etc/environment en mode super-utilisateur
sudo nano /etc/environment

# Dans le fichier /etc/environment
# Ajouter/modifier les variables JAVA_HOME et PATH
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
PATH="$JAVA_HOME:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us

# Dans le fichier ~/.bashrc
# Ajouter la ligne suivante à la fin
source /etc/environment

# Re-tester l'installation pour vérifier que le problème a été résolu
java -version # donnera la version courante de la JRE (1.8)
wsimport -version # donnera la version courante de wsimport (2.2.9)
```

Ensuite il faut configurer le build path de votre projet afin de choisir la version openJDK8 :

- clic droit sur le projet → `Build Path` → `Configure Build Path...`

- dans l'onglet `Libraries`, clic sur `JRE System Library` → clic sur `Edit`
- si la version par défaut n'est pas celle de openJDK8, clic sur `Alternate JRE` → clic sur `Installed JREs` → désélectionner la version actuelle puis clic sur `Add` → `Standard VM` → `Next` → `Directory...` → choisir le chemin `/usr/lib/jvm/java-8-openjdk-amd64/jre` → `Finish` → `Apply and Close`.

Enfin il faut configurer le compliance level du compilateur à 1.8 :

- clic droit sur le projet → `Build Path` → `Configure Build Path...`
- si le niveau n'est pas déjà 1.8 par défaut alors clic sur l'onglet gauche `Java Compiler` → sélectionner 1.8 pour `Compiler compliance level`.

2.2.2.2 Dépendances JAX-WS

```
<!--pom.xml-->
<dependencies>
    <dependency>
        <groupId>org.jboss.spec</groupId>
        <artifactId>jboss-javaee-6.0</artifactId>
        <version>1.0.0.Final</version>
        <type>pom</type>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>rt</artifactId>
        <version>2.3.1</version>
    </dependency>
</dependencies>
```

2.2.3 Conception

2.2.3.1 Définir les services web

Lors de la création de services web, il faut adopter une interface simple pour les opérations afin de les rendre facilement consommables par les clients.

En Java, il s'agit de créer une paire interface/classe annotées par `@WebService` pour définir l'interface/implémentation d'un service web. De plus, l'annotation `@WebService` de la classe aura l'attribut `endpointInterface="com.example.somepackage.ServiceWebInterf`
Les opérations exposées par le service web sont des méthodes déclarées au niveau de l'interface du service web, annotées par `@WebMethod`, et implémentées par sa classe d'implémentation.

Enfin, pour chaque opération du service web il faut prévoir les cas d'erreurs possibles et les traiter via le mécanisme d'exceptions en Java.

```java
// ../source/soap/java/snippets/web_service_soap_generic_example_jax_ws.java

package com.example.somepackage;

/* Interface générique d'un service web */
@WebService
public interface ServiceWebSoap {
    @WebMethod
    typeRetour1 operation1([params]);

    @WebMethod
    typeRetour2 operation2([params]);

    @WebMethod
    typeRetour3 operation3([params]);
    ...
}

/* Classe d'implémentation d'un service web */
@WebService(endpointInterface="com.example.somepackage.ServiceWebSoap")
public class ServiceWebSoapImpl implements ServiceWebSoap {
    /* attributs */
    /* méthodes */
    // méthodes de l'interface
    @Override
    public typeRetour1 operation1([params]){
        // implémentation
    }

    @Override
    public typeRetour2 operation2([params]) {
        // implémentation
    }

    @Override
    public typeRetour3 operation3([params]) {
        // implémentation
    }
    ...

    // autres méthodes
}
```

2.2.3.2 Publier le service web

Publier le service web consiste à l'associer à un endpoint et de générer son WSDL afin qu'il soit exploitable par des applications clientes. Pour ce faire, il suffit de créer une classe serveur qui déclarera une méthode `main()` contenant l'instruction `javax.xml.ws.Endpoint.publish("uri_service_web", new ServiceWebImpl())`.

Une fois `main()` exécutée, le service web sera consultable via son URI **"uri_service_web"** (*e.g., http://localhost:8080/employeeservice*) et prêt à être consommé par les application clientes.

```
// ../source/soap/java/snippets/web_service_soap_publisher_generic_example_jax_ws.java

package com.example.mypackage.server;

import javax.xml.ws.Endpoint;

public class WebServicePublisher {
    public static void main(String[] args) {
        Endpoint.publish("uri_service_web",
                new ServiceWebImpl());
        System.err.println("Server is ready");
    }
}
```

2.2.3.3 Définir les données échangées

Dans le cadre des web services, le modèle des données échangées comporte soit des types natifs, soit des types personnalisés. Les types de données sont encodés en XML en WSDL afin d'être échangés au sein de messages SOAP entre le fournisseur du service web et ses clients.

2.2.3.3.1 Exemple avec des types natifs

Supposons qu'on souhaite créer un service web pour faire des opérations arithmétiques d'une calculatrice. Les opérations du service sont l'addition, la soustraction, la multiplication, et la division, alors que les données échangées sont de type entier. Une exception sera levée si l'on essaye de diviser par zéro. Voici un exemple avec JAX-WS :

```
// ../source/soap/java/examples/MathService/MathService.java

package hai704i.tp2.exo2.server;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
```

```java
public interface MathService {
    @WebMethod
    int add(int a, int b);

    @WebMethod
    int subtract(int a, int b);

    @WebMethod
    int multiply(int a, int b);

    @WebMethod
    int divide(int a, int b) throws IllegalArgumentException;
}
```

```java
// ../source/soap/java/examples/MathService/MathServiceImpl.java

package hai704i.tp2.exo2.server;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService(endpointInterface="hai704i.tp2.exo2.server.MathService")
public class MathServiceImpl implements MathService {

    @WebMethod
    @Override
    public int add(int a, int b) {
        return a + b;
    }

    @WebMethod
    @Override
    public int subtract(int a, int b) {
        return a - b;
    }

    @WebMethod
    @Override
    public int multiply(int a, int b) {
        return a * b;
    }

    @WebMethod
    @Override
    public int divide(int a, int b) throws IllegalArgumentException {
        if (b == 0)
```

```
            throw new IllegalArgumentException("Error: Cannot divide by zero");
        return a / b;
    }
}
```

```
// ../source/soap/java/examples/MathService/MathServicePublisher.java

package hai704i.tp2.exo2.server;

import javax.xml.ws.Endpoint;

public class MathServicePublisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/mathservice",
                new MathServiceImpl());
        System.err.println("Server is ready");
    }
}
```

Lors de son publication, le service web aura le WSDL suivant :

```
<!--../source/soap/java/examples/MathService/mathservice.wsdl-->

<!--  Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.3.1
    svn-revision#6ef5f7eb9a938dbc4562f25f8fa0b67cc4ff2dbb.  -->
<!--  Generated by JAX-WS RI (http://javaee.github.io/metro-jax-ws).
    RI's version is JAX-WS RI 2.3.1 svn-revision#6ef5f7eb9a938dbc4562f25f8fa0b67cc4ff2dbb.
<definitions
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1
    xmlns:wsp="http://www.w3.org/ns/ws-policy"
    xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://server.exo2.tp2.hai704i/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://server.exo2.tp2.hai704i/"
    name="MathServiceImplService">
    <types>
        <xsd:schema>
            <xsd:import
                namespace="http://server.exo2.tp2.hai704i/"
                schemaLocation="http://localhost:8080/mathservice?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="subtract">
        <part name="parameters" element="tns:subtract"/>
```

```
</message>
<message name="subtractResponse">
    <part name="parameters" element="tns:subtractResponse"/>
</message>
<message name="add">
    <part name="parameters" element="tns:add"/>
</message>
<message name="addResponse">
    <part name="parameters" element="tns:addResponse"/>
</message>
<message name="divide">
    <part name="parameters" element="tns:divide"/>
</message>
<message name="divideResponse">
    <part name="parameters" element="tns:divideResponse"/>
</message>
<message name="multiply">
    <part name="parameters" element="tns:multiply"/>
</message>
<message name="multiplyResponse">
    <part name="parameters" element="tns:multiplyResponse"/>
</message>
<portType name="MathService">
    <operation name="subtract">
        <input
            wsam:Action="http://server.exo2.tp2.hai704i/MathService/subtractRequest"
            message="tns:subtract"/>
        <output
            wsam:Action="http://server.exo2.tp2.hai704i/MathService/subtractResponse"
            message="tns:subtractResponse"/>
    </operation>
    <operation name="add">
        <input
            wsam:Action="http://server.exo2.tp2.hai704i/MathService/addRequest"
            message="tns:add"/>
        <output
            wsam:Action="http://server.exo2.tp2.hai704i/MathService/addResponse"
            message="tns:addResponse"/>
    </operation>
    <operation name="divide">
        <input
            wsam:Action="http://server.exo2.tp2.hai704i/MathService/divideRequest"
            message="tns:divide"/>
        <output
            wsam:Action="http://server.exo2.tp2.hai704i/MathService/divideResponse"
            message="tns:divideResponse"/>
```

```
    </operation>
    <operation name="multiply">
        <input
            wsam:Action="http://server.exo2.tp2.hai704i/MathService/multiplyRequest"
            message="tns:multiply"/>
        <output
            wsam:Action="http://server.exo2.tp2.hai704i/MathService/multiplyResponse"
            message="tns:multiplyResponse"/>
    </operation>
</portType>
<binding name="MathServiceImplPortBinding" type="tns:MathService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="subtract">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="add">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="divide">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="multiply">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
```

```
            </output>
        </operation>
    </binding>
    <service name="MathServiceImplService">
        <port name="MathServiceImplPort" binding="tns:MathServiceImplPortBinding">
            <soap:address location="http://localhost:8080/mathservice"/>
        </port>
    </service>
</definitions>
```

2.2.3.3.2 Exemple avec des types personnalisés

Prenons un autre exemple plus complexe. Supposons que l'on dispose d'un ensemble d'employés ayant chacun un **identifiant** (*un entier*) et un **nom** (*un String*). Les employés sont des **Plain Old Java Object** (**POJO**), désignant des objets avec des attributs et des getters/setters.

On souhaite définir un service web permettant de manipuler cet ensemble de la manière suivante :

1. lister tous les employés ;
2. savoir le nombre des employés dans l'ensemble ;
3. récupérer un employé par son identifiant ;
4. ajouter un nouveau employé ;
5. supprimer un employé existant ;
6. modifier les informations d'un employé existant.

De plus, une erreur survient si l'on essaye :

1. de récupérer/modifier/supprimer un employé dont l'identifiant n'existe pas comme identifiant valide d'un employé dans l'ensemble.
2. d'ajouter un employé avec un identifiant déjà affecté à un autre employé dans l'ensemble.

Voici un exemple avec JAX-WS :

```java
// ../source/soap/java/examples/EmployeeService/Employee.java

package hai704i.tp2.demo.model;

public class Employee {
    /* ATTRIBUTES */
    private int id;
    private String name;

    /* CONSTRUCTORS */
    public Employee(int id, String name) {
        this.id = id;
```

```
        this.name = name;
    }

    /* METHODS */
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```
// ../source/soap/java/examples/EmployeeService/EmployeeNotFoundException.java

package hai704i.tp2.demo.exceptions;

public class EmployeeNotFoundException extends Exception {
    public EmployeeNotFoundException() {

    }

    public EmployeeNotFoundException(String message) {
        super(message);
    }
}
```

```
// ../source/soap/java/examples/EmployeeService/EmployeeAlreadyExistsException.java

package hai704i.tp2.demo.exceptions;

public class EmployeeAlreadyExistsException extends Exception {
    public EmployeeAlreadyExistsException() {

    }

    public EmployeeAlreadyExistsException(String message) {
        super(message);
    }
```

```
}
```

```
// ../source/soap/java/examples/EmployeeService/EmployeeService.java

package hai704i.tp2.demo.service;

import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebService;

import hai704i.tp2.demo.exceptions.EmployeeAlreadyExistsException;
import hai704i.tp2.demo.exceptions.EmployeeNotFoundException;
import hai704i.tp2.demo.model.Employee;

@WebService
public interface EmployeeService {

    /* METHODS */
    @WebMethod
    int count();

    @WebMethod
    List<Employee> getEmployees();

    @WebMethod
    Employee addEmployee(int id, String name) throws EmployeeAlreadyExistsException;

    @WebMethod
    Employee getEmployee(int id) throws EmployeeNotFoundException;

    @WebMethod
    Employee updateEmployee(int id, String name) throws EmployeeNotFoundException;

    @WebMethod
    boolean deleteEmployee(int id) throws EmployeeNotFoundException;
}
```

```
// ../source/soap/java/examples/EmployeeService/EmployeeServiceImpl.java

package hai704i.tp2.demo.service;

import java.util.List;

import javax.jws.WebService;

import hai704i.tp2.demo.exceptions.EmployeeAlreadyExistsException;
```

```java
import hai704i.tp2.demo.exceptions.EmployeeNotFoundException;
import hai704i.tp2.demo.model.Employee;
import hai704i.tp2.demo.repository.EmployeeRepository;
import hai704i.tp2.demo.repository.EmployeeRepositoryImpl;

@WebService(endpointInterface="hai704i.tp2.demo.service.EmployeeService")
public class EmployeeServiceImpl implements EmployeeService {

    /* ATTRIBUTES */
    private List<Employee> employees;

    /* CONSTRUCTORS */
    public EmployeeServiceImpl() {
        employees = new ArrayList<>();
        employees.addAll(Arrays.asList(
                new Employee(1, "Joe"),
                new Employee(2, "Jane"),
                new Employee(3, "Steve"),
                new Employee(4, "Alice"),
                new Employee(5, "Bob"),
                new Employee(6, "Alicia"),
                new Employee(7, "Tricia"),
                new Employee(8, "Paul"),
                new Employee(9, "Kevin"),
                new Employee(10, "Julia")
        ));
    }

    /* METHODS */
    @Override
    public int count() {
        return employees.size();
    }

    @Override
    public List<Employee> getEmployees() {
        return employees;
    }

    @Override
    public Employee addEmployee(int id, String name) throws EmployeeAlreadyExistsException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (!target.isEmpty())
```

```
            throw new EmployeeAlreadyExistsException(
                    "Error: An employee with ID "+id+" already exists");

        Employee employee = new Employee(id, name);
        employees.add(employee);
        return employee;
    }

    @Override
    public Employee getEmployee(int id) throws EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (target.isEmpty())
            throw new EmployeeNotFoundException(
                    "Error: No employee with ID "+id+" exists");

        return target.get();
    }

    @Override
    public Employee updateEmployee(int id, String name) throws EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (target.isEmpty())
            throw new EmployeeNotFoundException(
                    "Error: No employee with ID "+id+" exists");

        target.get().setName(name);
        return target.get();
    }

    @Override
    public boolean deleteEmployee(int id) throws EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (target.isEmpty())
            throw new EmployeeNotFoundException(
                    "Error: No employee with ID "+id+" exists");

        return employees.remove(target.get());
```

```
    }
}
// ../source/soap/java/examples/EmployeeService/EmployeeServicePublisher.java

package hai704i.tp2.demo.server;

import javax.xml.ws.Endpoint;

import hai704i.tp2.demo.service.EmployeeServiceImpl;

public class EmployeeServicePublisher {

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/employeeservice",
                new EmployeeServiceImpl());
        System.err.println("Server ready");
    }
}
```

Lors de son publication, le service web aura le WSDL suivant :

```
<!--../source/soap/java/examples/EmployeeService/employeeservice.wsdl-->

<!--  Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.3.1
    svn-revision#6ef5f7eb9a938dbc4562f25f8fa0b67cc4ff2dbb.  -->
<!--  Generated by JAX-WS RI (http://javaee.github.io/metro-jax-ws).
    RI's version is JAX-WS RI 2.3.1 svn-revision#6ef5f7eb9a938dbc4562f25f8fa0b67cc4ff2dbb.
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
    xmlns:wsp="http://www.w3.org/ns/ws-policy"
    xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://service.demo.tp2.hai704i/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://service.demo.tp2.hai704i/"
    name="EmployeeServiceImplService">
    <types>
        <xsd:schema>
            <xsd:import
                namespace="http://service.demo.tp2.hai704i/"
                schemaLocation="http://localhost:8080/employeeservice?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="count">
        <part name="parameters" element="tns:count"/>
```

```
</message>
<message name="countResponse">
    <part name="parameters" element="tns:countResponse"/>
</message>
<message name="getEmployees">
    <part name="parameters" element="tns:getEmployees"/>
</message>
<message name="getEmployeesResponse">
    <part name="parameters" element="tns:getEmployeesResponse"/>
</message>
<message name="addEmployee">
    <part name="parameters" element="tns:addEmployee"/>
</message>
<message name="addEmployeeResponse">
    <part name="parameters" element="tns:addEmployeeResponse"/>
</message>
<message name="EmployeeAlreadyExistsException">
    <part name="fault" element="tns:EmployeeAlreadyExistsException"/>
</message>
<message name="getEmployee">
    <part name="parameters" element="tns:getEmployee"/>
</message>
<message name="getEmployeeResponse">
    <part name="parameters" element="tns:getEmployeeResponse"/>
</message>
<message name="EmployeeNotFoundException">
    <part name="fault" element="tns:EmployeeNotFoundException"/>
</message>
<message name="updateEmployee">
    <part name="parameters" element="tns:updateEmployee"/>
</message>
<message name="updateEmployeeResponse">
    <part name="parameters" element="tns:updateEmployeeResponse"/>
</message>
<message name="deleteEmployee">
    <part name="parameters" element="tns:deleteEmployee"/>
</message>
<message name="deleteEmployeeResponse">
    <part name="parameters" element="tns:deleteEmployeeResponse"/>
</message>
<portType name="EmployeeService">
    <operation name="count">
        <input
            wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/countRequest"
            message="tns:count"/>
        <output
```

```
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/countResponse"
        message="tns:countResponse"/>
</operation>
<operation name="getEmployees">
    <input
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/getEmployeesReq
        message="tns:getEmployees"/>
    <output
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/getEmployeesRes
        message="tns:getEmployeesResponse"/>
</operation>
<operation name="addEmployee">
    <input
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/addEmployeeRequ
        message="tns:addEmployee"/>
    <output
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/addEmployeeResp
        message="tns:addEmployeeResponse"/>
    <fault
        message="tns:EmployeeAlreadyExistsException"
        name="EmployeeAlreadyExistsException"
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/addEmployee/Fau
</operation>
<operation name="getEmployee">
    <input
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/getEmployeeRequ
        message="tns:getEmployee"/>
    <output
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/getEmployeeResp
        message="tns:getEmployeeResponse"/>
    <fault
        message="tns:EmployeeNotFoundException"
        name="EmployeeNotFoundException"
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/getEmployee/Fau
</operation>
<operation name="updateEmployee">
    <input
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/updateEmployeeR
        message="tns:updateEmployee"/>
    <output
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/updateEmployeeR
        message="tns:updateEmployeeResponse"/>
    <fault
        message="tns:EmployeeNotFoundException"
        name="EmployeeNotFoundException"
        wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/updateEmployee/
```

```
    </operation>
    <operation name="deleteEmployee">
        <input
            wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/deleteEmployeeR
            message="tns:deleteEmployee"/>
        <output
            wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/deleteEmployeeR
            message="tns:deleteEmployeeResponse"/>
        <fault
            message="tns:EmployeeNotFoundException"
            name="EmployeeNotFoundException"
            wsam:Action="http://service.demo.tp2.hai704i/EmployeeService/deleteEmployee/
    </operation>
</portType>
<binding name="EmployeeServiceImplPortBinding" type="tns:EmployeeService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="count">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="getEmployees">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="addEmployee">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
        <fault name="EmployeeAlreadyExistsException">
            <soap:fault name="EmployeeAlreadyExistsException" use="literal"/>
        </fault>
    </operation>
```

```
        <operation name="getEmployee">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
            <fault name="EmployeeNotFoundException">
                <soap:fault name="EmployeeNotFoundException" use="literal"/>
            </fault>
        </operation>
        <operation name="updateEmployee">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
            <fault name="EmployeeNotFoundException">
                <soap:fault name="EmployeeNotFoundException" use="literal"/>
            </fault>
        </operation>
        <operation name="deleteEmployee">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
            <fault name="EmployeeNotFoundException">
                <soap:fault name="EmployeeNotFoundException" use="literal"/>
            </fault>
        </operation>
    </binding>
    <service name="EmployeeServiceImplService">
        <port name="EmployeeServiceImplPort" binding="tns:EmployeeServiceImplPortBinding">
            <soap:address location="http://localhost:8080/employeeservice"/>
        </port>
    </service>
</definitions>
```

2.2.3.4 Abstraction des détails internes

Parfois une opération exposée peut utiliser une logique métier complexe et/ou de bas niveau pour accomplir une sous-tâche de sa fonctionnalité complète. Dans ce cas, cette logique métier complexe peut être encapsulée au sein d'une autre opération interne et non exposée par le service web. On parle ainsi de l'abstraction des détails internes d'une classe.

```
// ../source/soap/java/snippets/web_service_soap_generic_example_with_operation_abstraction_

package com.example.somepackage;

/* Interface générique d'un service web */
@WebService
public interface ServiceWebSoap {
    @WebMethod
    typeRetour1 operation1([params]);

    @WebMethod
    typeRetour2 operation2([params]);

    @WebMethod
    typeRetour3 operation3([params]);
    ...
}

/* Classe d'implémentation d'un service web */
@WebService(endpointInterface="com.example.somepackage.ServiceWebSoap")
public class ServiceWebSoapImpl implements ServiceWebSoap {
    /* attributs */
    /* méthodes */
    // méthodes de l'interface
    @Override
    public typeRetour1 operation1([params]){
        // implémentation
        // éventuellement invocation de méthodes internes
    }

    @Override
    public typeRetour2 operation2([params]) {
        // implémentation
        // éventuellement invocation de méthodes internes
    }

    @Override
    public typeRetour3 operation3([params]) {
        // implémentation
        // éventuellement invocation de méthodes internes
```

```
    }

    // méthode d'implémentation internes
    visibilite typeRetour4 operationInterne1([params]) {
        // implémentation
        // en général visibilite = private ou protected
    }
    ...
}
```

2.2.3.5 Définir des DAOs

2.2.3.5.1 Motivation

Dans l'exemple du service web défini sur un ensemble d'employés, si l'on suppose maintenant que les employés sont stockés dans une base de données, on devra alors changer l'implémentation des méthodes de la classe implémentant le service web pour pouvoir refléter les changements au niveau de celle-ci. En effet, Le même scénario se répète si les employés sont récupérés depuis le système de fichiers.

On remarque ainsi que ces détails doivent être abstraites du service web, afin de pouvoir le réutiliser avec des données stockées sur des supports de persistance différents. Autrement dit, quelle que soit le support de persistance des données employés, l'implémentation du service web doit en rester indépendante.

2.2.3.5.2 Le design pattern Data Access Object (DAO)

Pour ce cas d'usage, le design pattern DAO est utilisé. Il s'agit de séparer le code métier d'une application cliente des détails bas niveau d'accès aux données qu'elle manipule. Autrement dit, une classe cliente utilisant une donnée ne doit pas se charger de la manière dont cette donnée est récupérée/stockée depuis/sur son support de persistance. Par conséquent, la seule chose qu'une classe cliente aura besoin de faire est de spécifier quelle implémentation du DAO à utiliser pour son cas d'usage. Ceci peut se faire via une **dépendance d'injection** classique via son constructeur ou via un setter.

Les participants dans ce design pattern sont :

1. `Model Object` : l'objet POJO stocké sur le support de persistance.
2. `DAO` : une interface définissant au moins les opérations **CRUD** (**Create, Read, Update, Delete**) permettant de créer/récupérer/modifier/supprimer des instances de `Model Object` depuis un support de persistance. D'autres opérations peuvent être définies aussi.
3. `DAOImpl` : une classe implémentant `DAO` pour une support de persistance spécifique (e.g., base de données, système de fichiers, ...).

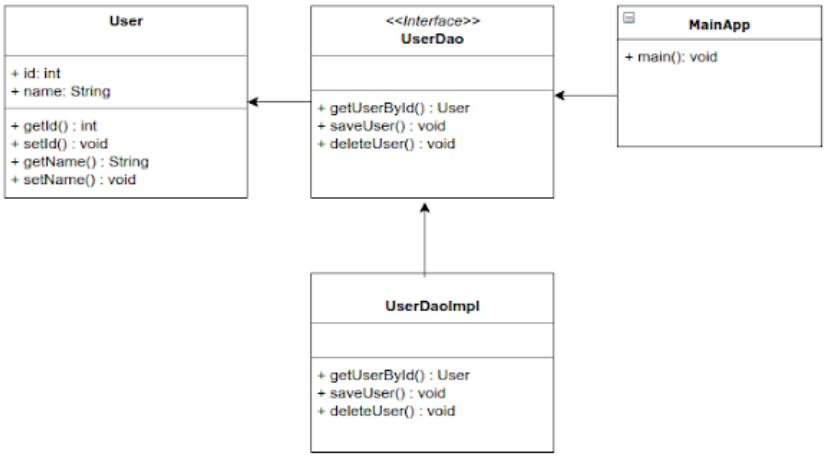

Figure 15: Diagramme UML du design pattern DAO sur un example

2.2.3.5.3 Exemple du service web des employés avec DAO

```
// ../source/soap/java/examples/EmployeeService/Employee.java

package hai704i.tp2.demo.model;

public class Employee {
    /* ATTRIBUTES */
    private int id;
    private String name;

    /* CONSTRUCTORS */
    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    /* METHODS */
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```java
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```java
// ../source/soap/java/examples/EmployeeService/EmployeeRepository.java

package hai704i.tp2.demo.repository;

import java.util.List;

import hai704i.tp2.demo.exceptions.EmployeeAlreadyExistsException;
import hai704i.tp2.demo.exceptions.EmployeeNotFoundException;
import hai704i.tp2.demo.model.Employee;

public interface EmployeeRepository {

    /* METHODS */
    int count();

    List<Employee> getEmployees();

    Employee addEmployee(int id, String name) throws EmployeeAlreadyExistsException;

    Employee getEmployee(int id) throws EmployeeNotFoundException;

    Employee updateEmployee(int id, String name) throws EmployeeNotFoundException;

    boolean deleteEmployee(int id) throws EmployeeNotFoundException;
}
```

```java
// ../source/soap/java/examples/EmployeeService/EmployeeRepositoryImpl.java

package hai704i.tp2.demo.repository;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import hai704i.tp2.demo.exceptions.EmployeeAlreadyExistsException;
import hai704i.tp2.demo.exceptions.EmployeeNotFoundException;
import hai704i.tp2.demo.model.Employee;
```

```java
public class EmployeeRepositoryImpl implements EmployeeRepository {

    /* ATTRIBUTES */
    private List<Employee> employees;

    /* CONSTRUCTORS */
    public EmployeeRepositoryImpl() {
        employees = new ArrayList<>();
        employees.addAll(Arrays.asList(
                new Employee(1, "Joe"),
                new Employee(2, "Jane"),
                new Employee(3, "Steve"),
                new Employee(4, "Alice"),
                new Employee(5, "Bob"),
                new Employee(6, "Alicia"),
                new Employee(7, "Tricia"),
                new Employee(8, "Paul"),
                new Employee(9, "Kevin"),
                new Employee(10, "Julia")
        ));
    }

    /* METHODS */
    @Override
    public int count() {
        return employees.size();
    }

    @Override
    public List<Employee> getEmployees() {
        return employees;
    }

    @Override
    public Employee addEmployee(int id, String name) throws EmployeeAlreadyExistsException {

        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (!target.isEmpty())
            throw new EmployeeAlreadyExistsException(
                    "Error: An employee with ID "+id+" already exists");

        Employee employee = new Employee(id, name);
```

```
        employees.add(employee);
        return employee;
    }

    @Override
    public Employee getEmployee(int id) throws EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (target.isEmpty())
            throw new EmployeeNotFoundException(
                    "Error: No employee with ID "+id+" exists");

        return target.get();
    }

    @Override
    public Employee updateEmployee(int id, String name) throws EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (target.isEmpty())
            throw new EmployeeNotFoundException(
                    "Error: No employee with ID "+id+" exists");

        target.get().setName(name);
        return target.get();
    }

    @Override
    public boolean deleteEmployee(int id) throws EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (target.isEmpty())
            throw new EmployeeNotFoundException(
                    "Error: No employee with ID "+id+" exists");

        return employees.remove(target.get());
    }
}

// ../source/soap/java/examples/EmployeeService/EmployeeService.java
```

```java
package hai704i.tp2.demo.service;

import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebService;

import hai704i.tp2.demo.exceptions.EmployeeAlreadyExistsException;
import hai704i.tp2.demo.exceptions.EmployeeNotFoundException;
import hai704i.tp2.demo.model.Employee;

@WebService
public interface EmployeeService {

    /* METHODS */
    @WebMethod
    int count();

    @WebMethod
    List<Employee> getEmployees();

    @WebMethod
    Employee addEmployee(int id, String name) throws EmployeeAlreadyExistsException;

    @WebMethod
    Employee getEmployee(int id) throws EmployeeNotFoundException;

    @WebMethod
    Employee updateEmployee(int id, String name) throws EmployeeNotFoundException;

    @WebMethod
    boolean deleteEmployee(int id) throws EmployeeNotFoundException;
}
```

```java
// ../source/soap/java/examples/EmployeeService/EmployeeServiceImplWithDAO.java

package hai704i.tp2.demo.service;

import java.util.List;

import javax.jws.WebService;

import hai704i.tp2.demo.exceptions.EmployeeAlreadyExistsException;
import hai704i.tp2.demo.exceptions.EmployeeNotFoundException;
import hai704i.tp2.demo.model.Employee;
```

```java
import hai704i.tp2.demo.repository.EmployeeRepository;
import hai704i.tp2.demo.repository.EmployeeRepositoryImpl;

@WebService(endpointInterface="hai704i.tp2.demo.service.EmployeeService")
public class EmployeeServiceImpl implements EmployeeService {

    /* ATTRIBUTES */
    private EmployeeRepository repository = new EmployeeRepositoryImpl();

    /* METHODS */
    @Override
    public int count() {
        return repository.count();
    }

    @Override
    public List<Employee> getEmployees() {
        return repository.getEmployees();
    }

    @Override
    public Employee addEmployee(int id, String name) throws EmployeeAlreadyExistsException {
        return repository.addEmployee(id, name);
    }

    @Override
    public Employee getEmployee(int id) throws EmployeeNotFoundException {
        return repository.getEmployee(id);
    }

    @Override
    public Employee updateEmployee(int id, String name) throws EmployeeNotFoundException {
        return repository.updateEmployee(id, name);
    }

    @Override
    public boolean deleteEmployee(int id) throws EmployeeNotFoundException {
        return repository.deleteEmployee(id);
    }
}
```

2.3 Consommer un service web SOAP à partir d'un client JAX-WS

2.3.1 Principe

Consommer un service web consiste à en créer un proxy qui sera utilisé par une application cliente afin d'invoquer ses opérations. Le proxy se chargera de réaliser les opérations réseaux, d'encodage et de décodage de données nécessaires avant d'invoquer la méthode souhaitée sur le service web.

Il faut donc générer la classe du proxy et les classes dont il dépend à partir de la WSDL du service web. Pour ce faire, il faut utiliser `wsimport` après la publication du service web (*i.e., le serveur est en cours d'exécution*) de la manière suivante :

```
wsimport -keep -p "com.example.package.clients" "uri_service_web?wsdl"
```

Cette commande générera les fichiers `.java` et `.class` nécessaires dans le package `chemin/dossier/courant/com.example.package.clients` à partir du WSDL du service web sur `"uri_service_web?wsdl"`

Ensuite, il faut importer les fichiers `.java` dans un package du projet client qui souhaite consommer le service web.

```
/* dans une méthode désirée du projet client */
// créer une instance de l'URI du service web à consommer
URL url = new URL("uri_service_web?wsdl");

// récupérer une instance de l'implémentation du service web
com.example.mypackage.client.ServiceWebImpl serviceImpl =
    new com.example.mypackage.client.ServiceWebImpl(url);

// récupérer un proxy du service web
com.example.mypackage.client.ServiceWeb proxy = serviceImpl.getServiceWebImplPort();

// invoquer les méthodes exposées souhaitées du service web sur le proxy
proxy.operation1([params]);
proxy.operation2([params]);
...
```

2.3.2 Exemple d'une CLI consommant un web service calculatrice

Dans cet exemple, l'idée consiste à consommer un service web exposant des opérations d'une calculatrice. Ces opérations sont les mêmes vues dans l'exemple du service web `MathService`, notamment l'addition, la soustraction, la multiplication, et la division sur des données entières. Ces opérations seront utilisées par une CLI permettant à un client de les consommer. La CLI aura

besoin ainsi de récupérer une instance du proxy du service web sur lequel elle invoquera l'opération choisie par le client.

La CLI est une application cliente héritant d'une classe abstraite `AbstractMain`. `AbstractMain` est une classe abstraite fournissant l'interface de base d'une CLI :

1. des attributs pour stocker l'URL du service web et terminer explicitement l'exécution de la CLI (*i.e., cette CLI continuera à s'exécuter tant que le choix la valeur de l'attribut `QUIT` n'a pas été choisie par le client*).
2. des méthodes pour saisir et vérifier l'URL du service web, mais aussi pour afficher le menu des interactions possibles.

Pour chaque CLI il y aura des méthodes supplémentaires pour : 1. récupérer une instance du proxy du service web ; 2. valider les valeurs saisies par l'utilisateur ; 3. invoquer les opérations correspondantes du service web avec éventuellement les valeurs saisies validées comme paramètres.

En particulier, la validation des valeurs saisies par l'utilisateur est réalisée par un processeur d'input défini dans la hiérarchie de classes enracinée par `ComplexUserInputProcessor<T>`. Par exemple, pour traiter une valeur saisie qui devrait correspondre à un entier ou à un entier non nulle, la CLI utilise une instance de `IntegerInputProcessor` ou `NonZeroIntegerInputProcessor` respectivement.

N.B. En général, une CLI n'a pas forcément besoin de suivre l'interface définie par `AbstractMain` ou d'utiliser un processeur d'input utilisateur. Il s'agit de concepts créés pour faciliter la réutilisation de code et simplifier la redondance, mais vous n'êtes pas forcément obligés de s'en servir ou d'en définir vous-mêmes.

```
// ../source/soap/java/examples/Common/AbstractMain.java

package hai704i.tp2.demo.main;

import java.io.BufferedReader;
import java.io.IOException;

public abstract class AbstractMain {
    public static String SERVICE_WSDL_URL;
    public static final String QUIT = "0";

    protected void setTestServiceWSDLUrl(BufferedReader inputReader)
            throws IOException {

        System.out.println("Please provide the URL to the web service to consume: ");
        SERVICE_WSDL_URL = inputReader.readLine();

        while(!validServiceWSDLUrl()) {
            System.err.println("Error: "+SERVICE_WSDL_URL+
```

```
                    " isn't a valid web service WSDL URL. "
                    + "Please try again: ");
            SERVICE_WSDL_URL = inputReader.readLine();
        }
    }

    protected abstract boolean validServiceWSDLUrl();

    protected abstract void menu();
}
// ../source/soap/java/examples/Common/ComplexUserInputProcessor.java

package hai704i.tp2.demo.main;

import java.io.BufferedReader;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.function.Predicate;

public abstract class ComplexUserInputProcessor<T> {
    /* ATTRIBUTES */
    protected BufferedReader inputReader;
    protected String message;
    protected Predicate<String> isValid;
    protected Method parser;
    protected T parameter;

    /* CONSTRUCTOR */
    public ComplexUserInputProcessor(BufferedReader inputReader) {
        this.inputReader = inputReader;
        setMessage();
        setValidityCriterion();
        setParser();
    }

    /* METHODS */
    protected abstract void setMessage();
    protected abstract void setValidityCriterion();
    protected abstract void setParser();

    public T process() throws IOException {
        System.out.println(message);
        String userInput = inputReader.readLine();
```

```
        while (!isValid.test(userInput)) {
            System.err.println("Sorry, wrong input. Please try again.");
            System.out.println();
            System.out.println(message);
            userInput = inputReader.readLine();
        }

        try {
            parameter = (T) parser.invoke(null, userInput);
        } catch (SecurityException | IllegalAccessException |
                IllegalArgumentException | InvocationTargetException e) {

            e.printStackTrace();
        }

        return parameter;
    }
}
```

```
// ../source/soap/java/examples/Common/IntegerInputProcessor.java

package hai704i.tp2.demo.main;

import java.io.BufferedReader;

public class IntegerInputProcessor extends ComplexUserInputProcessor<Integer> {

    public IntegerInputProcessor(BufferedReader inputReader) {
        super(inputReader);
    }

    @Override
    protected void setMessage() {
        message = "Please enter an integer:";
    }

    @Override
    protected void setValidityCriterion() {
        isValid = str -> {
            try {
                Integer value = Integer.parseInt(str);
                return value instanceof Integer;
            } catch (NumberFormatException e) {
                return false;
            }
        };
```

```
    }

    @Override
    protected void setParser() {
        try {
            parser = Integer.class.getMethod("parseInt", String.class);
        } catch (SecurityException | NoSuchMethodException e) {

            e.printStackTrace();
        }
    }
}
// ../source/soap/java/examples/Common/NonZeroIntegerInputProcessor.java

package hai704i.tp2.demo.main;

import java.io.BufferedReader;

public class NonZeroIntegerInputProcessor extends IntegerInputProcessor {

    public NonZeroIntegerInputProcessor(BufferedReader inputReader) {
        super(inputReader);
    }

    @Override
    protected void setMessage() {
        message = "Please enter a non-zero integer:";
    }

    @Override
    protected void setValidityCriterion() {
        isValid = str -> {
            try {
                int value = Integer.parseInt(str);
                return value != 0;
            } catch (NumberFormatException e) {
                return false;
            }
        };
    }

}
// ../source/soap/java/examples/CalculatorService/CalculatorServiceClientCLI.java

package hai704i.tp2.exo3.main;
```

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;

import hai704i.tp2.demo.main.AbstractMain;
import hai704i.tp2.demo.main.IntegerInputProcessor;
import hai704i.tp2.demo.main.NonZeroIntegerInputProcessor;
import hai704i.tp2.exo3.client.Calculator;
import hai704i.tp2.exo3.client.CalculatorSoap;

public class CalculatorServiceClientCLI extends AbstractMain {
    public static IntegerInputProcessor inputProcessor;

    public static void main(String[] args) {
        CalculatorServiceClientCLI main = new CalculatorServiceClientCLI();
        CalculatorSoap proxy = null;
        BufferedReader inputReader;
        String userInput = "";

        try {
            inputReader = new BufferedReader(
                    new InputStreamReader(System.in));
            main.setTestServiceWSDLUrl(inputReader);
            proxy = getProxy();

            do {
                main.menu();
                userInput = inputReader.readLine();
                main.processUserInput(inputReader, userInput, proxy);
                Thread.sleep(3000);

            } while(!userInput.equals(QUIT));
        } catch (MalformedURLException e) {
            System.err.println(SERVICE_WSDL_URL+" isn't a valid WSDL URL");

        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
```

```
protected boolean validServiceWSDLUrl() {
    return SERVICE_WSDL_URL.equals(
            "http://www.dneonline.com/calculator.asmx?WSDL");
}

private static CalculatorSoap getProxy()
        throws MalformedURLException {
    return new Calculator(new URL(SERVICE_WSDL_URL))
            .getCalculatorSoap();
}

@Override
protected void menu() {
    StringBuilder builder = new StringBuilder();
    builder.append(QUIT+". Quit.");
    builder.append("\n1. Add two integers: a + b.");
    builder.append("\n2. Subtract two integers: a - b.");
    builder.append("\n3. Multiply two integers: a * b");
    builder.append("\n4. Divide two integers: a / b");

    System.out.println(builder);
}

private void processUserInput(BufferedReader reader,
        String userInput, CalculatorSoap proxy) {
    try {
        switch(userInput) {
            case "1":
                invokeServiceOperation(
                        new IntegerInputProcessor(reader),
                        new IntegerInputProcessor(reader),
                        proxy,
                        "add");
                break;

            case "2":
                invokeServiceOperation(
                        new IntegerInputProcessor(reader),
                        new IntegerInputProcessor(reader),
                        proxy,
                        "subtract");
                break;

            case "3":
                invokeServiceOperation(
                        new IntegerInputProcessor(reader),
```

```
                        new IntegerInputProcessor(reader),
                        proxy,
                        "multiply");
                break;

            case "4":
                invokeServiceOperation(
                        new IntegerInputProcessor(reader),
                        new NonZeroIntegerInputProcessor(reader),
                        proxy,
                        "divide");
                break;

            case QUIT:
                System.out.println("Bye...");
                return;

            default:
                System.err.println("Sorry, wrong input. Please try again.");
                return;
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void invokeServiceOperation(IntegerInputProcessor
        param1Processor, IntegerInputProcessor param2Processor,
        CalculatorSoap proxy, String operationName) throws IOException {
    inputProcessor = param1Processor;
    int a = inputProcessor.process();

    inputProcessor = param2Processor;
    int b = inputProcessor.process();

    switch(operationName) {
        case "add":
            System.out.println("a + b = "+(proxy.add(a, b)));
            System.out.println();
            break;

        case "subtract":
            System.out.println("a - b = "+(proxy.subtract(a, b)));
            System.out.println();
            break;
```

```
            case "multiply":
                System.out.println("a * b = "+(proxy.multiply(a, b)));
                System.out.println();
                break;

            case "divide":
                System.out.println("a / b = "+(proxy.divide(a, b)));
                System.out.println();
                break;
        }
    }

}
```

2.3.3 Exemple d'une CLI consommant le service web `EmployeeService`

Dans cet exemple, l'idée consiste à consommer le service web `EmployeeService`. Ces opérations seront utilisées par une CLI permettant à un client de les consommer. La CLI est implémentée de la même manière que précédemment.

```
// ../source/soap/java/examples/EmployeeService/EmployeeServiceClientCLI.java

package hai704i.tp2.demo.main;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;

import hai704i.tp2.demo.client.EmployeeService;
import hai704i.tp2.demo.client.EmployeeServiceImplService;
import hai704i.tp2.demo.client.Employee;
import hai704i.tp2.demo.client.EmployeeAlreadyExistsException_Exception;
import hai704i.tp2.demo.client.EmployeeNotFoundException_Exception;

public class EmployeeServiceClientCLI extends AbstractMain {

    public static IntegerInputProcessor inputProcessor;

    public static void main(String[] args) {
        EmployeeServiceClientCLI main = new EmployeeServiceClientCLI();
        EmployeeService proxy = null;
        BufferedReader inputReader;
        String userInput = "";
```

```
    try {
        inputReader = new BufferedReader(
                new InputStreamReader(System.in));
        main.setTestServiceWSDLUrl(inputReader);
        proxy = getProxy();

        do {
            main.menu();
            userInput = inputReader.readLine();
            main.processUserInput(inputReader, userInput, proxy);
            Thread.sleep(3000);

        } while(!userInput.equals(QUIT));
    } catch (MalformedURLException e) {
        System.err.println(SERVICE_WSDL_URL+" isn't a valid WSDL URL");

    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

@Override
protected boolean validServiceWSDLUrl() {
    return SERVICE_WSDL_URL.equals(
            "http://localhost:8080/employeeservice?wsdl");
}

private static EmployeeService getProxy()
        throws MalformedURLException {
    return new EmployeeServiceImplService(new URL(SERVICE_WSDL_URL))
            .getEmployeeServiceImplPort();
}

@Override
protected void menu() {
    StringBuilder builder = new StringBuilder();
    builder.append(QUIT+". Quit.");
    builder.append("\n1. Get number of employees.");
    builder.append("\n2. Display all employees.");
    builder.append("\n3. Get employee by ID");
    builder.append("\n4. Add new employee");
    builder.append("\n5. Remove employee by ID");
    builder.append("\n6. Update existing employee");
```

```
        System.out.println(builder);

    }

    private void processUserInput(BufferedReader reader,
            String userInput, EmployeeService proxy) {
        try {
            switch(userInput) {
                case "1":
                    System.out.println("There are "+proxy.count()+" employees");
                    break;

                case "2":
                    proxy
                    .getEmployees()
                    .forEach(EmployeeServiceClientCLI::displayEmployee);
                    break;

                case "3":
                    inputProcessor = new IntegerInputProcessor(reader);
                    int id = inputProcessor.process();
                    Employee employee = proxy.getEmployee(id);
                    displayEmployee(employee);
                    break;

                case "4":
                    System.out.println("Employee ID: ");
                    inputProcessor = new IntegerInputProcessor(reader);
                    id = inputProcessor.process();
                    System.out.println();

                    System.out.println("Employee Name: ");
                    String name = reader.readLine();
                    System.out.println();

                    proxy.addEmployee(id, name);
                    System.out.println("Successfully added employee.");
                    break;

                case "5":
                    System.out.println("Employee ID: ");
                    inputProcessor = new IntegerInputProcessor(reader);
                    id = inputProcessor.process();
                    System.out.println();
```

```
                    if(proxy.deleteEmployee(id))
                        System.out.println("Successfully removed employee with ID "+id+".");
                    else
                        System.err.println("Failed to remove employee with ID "+id+".");

                    break;

                case "6":
                    System.out.println("Employee ID: ");
                    inputProcessor = new IntegerInputProcessor(reader);
                    id = inputProcessor.process();
                    System.out.println();

                    System.out.println("Employee Name: ");
                    name = reader.readLine();
                    System.out.println();

                    proxy.updateEmployee(id, name);
                    System.out.println("Successfully updated employee with ID "+id+".");
                    break;

                case QUIT:
                    System.out.println("Bye...");
                    return;

                default:
                    System.err.println("Sorry, wrong input. Please try again.");
                    return;
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (EmployeeNotFoundException_Exception e) {
            System.err.println(e.getMessage());
        } catch (EmployeeAlreadyExistsException_Exception e) {
            System.err.println(e.getMessage());
        }
    }

    private static void displayEmployee(Employee employee) {
        System.out.println("ID: "+employee.getId()+
                ", Name: "+employee.getName());
    }
}
```