**Université de Bouira**
**Département d'Informatique**
**Module : Les Architectures Orientées Services**
**Niveau M1-ISIL**

# Série TD / TP 7

**SAAJ – SOAP with Attachements API for Java**

## I. Buts

- ✓ Il est possible de créer et d'implémenter un Client qui communique/consomme un Service Web SOAP sans passer par JAX-WS (et wsimport) en utilisant l'API SAAJ.
- ✓ Décrire comment utiliser SAAJ dans une application.

**SAAJ**[1], pour SOAP with Attachments API for Java, est une API permettant l'envoi et la réception de messages XML sur le réseau depuis la plateforme Java. Ces messages obéissent à la spécification SOAP. Avec l'aide de SAAJ, un développeur peut produire, envoyer, et recevoir donc depuis son application des messages conformes à SOAP directement plutôt que d'utiliser JAX-WS.

Cette API propose un niveau d'abstraction assez élevé permettant de simplifier l'usage de SOAP. Ses classes sont regroupées dans le package *javax.xml.soap*.

SAAJ propose des classes qui encapsulent les différents éléments d'un message SOAP : SOAPMessage, SOAPPart, SOAPEnvelope, SOAPHeader et SOAPBody.

Tous les échanges de messages avec SOAP utilisent une connexion encapsulée dans la classe SOAPConnection. Cette classe permet la connexion directe entre l'émetteur et le receveur du ou des messages.

Dans ce qui suit, l'API SAAJ est utilisée afin de créer un Client Java qui consomme l'opération *makeHello* d'un service web Hello World (vu en TP2). Ce processus fonctionne en 5 étapes :

- **Création d'une connexion SOAP**
- **Création d'un message de requête SOAP**
- **Envoi de la requête SOAP au server**
- **Réception et traitement de la réponse SOAP**
- **Fermeture de la connexion SOAP**

## II. Implémentation d'un Client service web avec SAAJ

### 1. Création d'une connexion SOAP

Tous les messages SOAP sont envoyés et reçus via une connexion. Avec l'API SAAJ, la connexion est représentée par un objet **SOAPConnection**, qui va de l'expéditeur directement à sa destination. Ce type de connexion est appelé une connexion point à point, car il passe

[1] http://docs.oracle.com/javaee/5/tutorial/doc/bnbhf.html

d'un endpoint à un autre endpoint. Les messages envoyés à l'aide de l'API SAAJ sont appelés messages de demande-réponse.

Le bout de code suivant crée une connexion SOAP en créant un objet SOAPConnection :

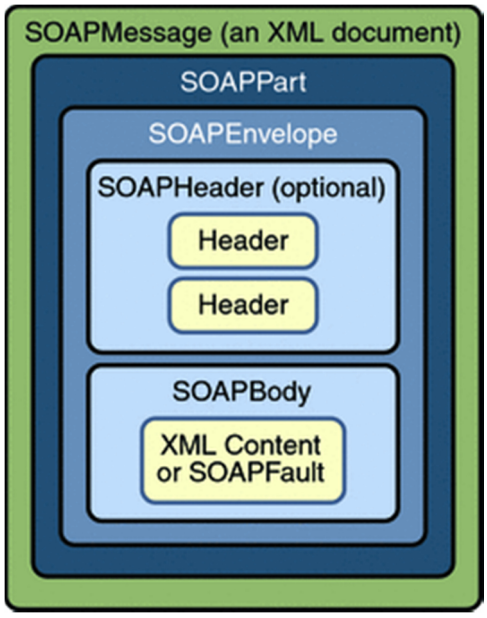
```
SOAPConnectionFactory soapConnectionFactory =
                                        SOAPConnectionFactory.newInstance();

SOAPConnection soapConnection = soapConnectionFactory.createConnection();
```

## 2. Création d'un message de requête SOAP

Les messages SAAJ, **SOAPMessage**, suivent les normes SOAP, qui prescrivent le format des messages et spécifient également certaines choses requises, facultatives ou non autorisées. Les deux principaux types de messages SOAP sont ceux qui ont des pièces jointes (attachements) et ceux qui n'en ont pas.

L'API SAAJ fournit la classe SOAPMessage pour représenter un message SOAP, la classe **SOAPPart** pour représenter la partie SOAP, l'interface **SOAPEnvelope** pour représenter l'enveloppe SOAP et ainsi de suite. La figure ci-dessous illustre la structure d'un message SOAP sans attachement (le type de messages qui nous intéresse ici).




Lorsqu'un nouvel objet SOAPMessage est créé, il aura automatiquement les parties qui doivent être dans un message SOAP. En d'autres termes, un nouvel objet SOAPMessage a un objet SOAPPart qui contient un objet SOAPEnvelope. L'objet SOAPEnvelope à son tour contient automatiquement un objet **SOAPHeader** vide suivi d'un objet **SOAPBody** vide. Si l'objet SOAPHeader n'est pas requis, qui est optionnel, sa suppression est possible.

Pour créer une requête SOAP, la première étape consiste à créer un message à l'aide d'un objet **MessageFactory**. L'API SAAJ fournit une implémentation par défaut de la classe MessageFactory, facilitant ainsi l'obtention d'une instance. Le fragment de code suivant

illustre l'obtention d'une instance de l'usine (factory) de messages par défaut, puis son utilisation pour créer un message :

```
MessageFactory messageFactory = MessageFactory.newInstance();

SOAPMessage soapRequest = messageFactory.createMessage();
```

L'étape suivante dans la création d'un message SOAP est d'accéder à ses parties afin que le contenu puisse être ajouté. D'abord, en utilisant la méthode *getSOAPPart* du message pour récupérer l'objet SOAPPart:

```
SOAPPart soapPart = soapRequest.getSOAPPart();
```

Ensuite, la méthode *getEnvelope* de *soapPart* est utilisée pour récupérer l'objet SOAPEnvelope qu'il contient. Aussi, la méthode *addNamespaceDeclaration* peut être appelée pour établir un namespace ainsi que son préfix:

```
SOAPEnvelope soapEnvelope = soapPart.getEnvelope();

soapEnvelope.addNamespaceDeclaration("hello", "http://tp2.ws.soa.org/");
```

Les méthodes *getHeader* et *getBody* de *soapEnvelope* peut désormais être utilisées pour récupérer ses objets SOAPHeader et SOAPBody vides :

```
SOAPHeader soapHeader = soapEnvelope.getHeader();

SOAPBody soapBody = soapEnvelope.getBody();
```

Cet exemple de client SAAJ n'utilise pas d'en-tête SOAP, l'objet *header* peut être supprimé :

```
SOAPHeader soapHeader = soapEnvelope.getHeader();

soapHeader.detachNode();
```

Pour ajouter du contenu au corps du message SOAP, un ou plusieurs objets **SOAPElement** peuvent être créés depuis *soapBody* pour contenir le contenu. Des sous-éléments aux objets SOAPElement peuvent notamment être ajoutés à l'aide de la méthode *addChildElement*. Enfin, pour chaque élément ou élément enfant, du contenu est ajouté en utilisant la méthode *addTextNode*.

```
SOAPBody soapBody = soapEnvelope.getBody();
SOAPElement soapElement =
                    soapBody.addChildElement("makeHello","hello");
SOAPElement element1 = soapElement.addChildElement("arg0");
element1.addTextNode("SAAJ API");

soapRequest.saveChanges();
```

Le contenu ajouté à *soapBody* ressemblera à ce qui suit lorsqu'il sera envoyé:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:hello="http://tp2.ws.soa.org/">
      <SOAP-ENV:Body>
            <hello:makeHello>
                  <arg0>SAAJ API</arg0>
            </hello:makeHello>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Pour afficher à la console le message de requête créé, utiliser ce code:

```
System.out.print("Request SOAP Message = ");
soapRequest.writeTo(System.out);
System.out.println();
```

Il est préférable de mettre tout le code permettant la création d'un message de SOAP dans une méthode, *createSOAPRequest*( ) par exemple, qui retourne un objet SOAPMessage, puis de l'appeler au besoin.

### 3. Envoi de la requête SOAP créée au service web

Les messages SOAP sont envoyés via l'objet SOAPConnection avec la méthode ***call***, qui envoie un message (une requête) puis bloque jusqu'à ce qu'il reçoive la réponse (une réponse). La méthode ***call*** prend deux arguments: le message de requête à envoyer et l'URL endpoint du service web SOAP à consommer.

La valeur de retour de la méthode *call* est un objet SOAPMessage représentant la réponse au message qui a été envoyé et elle prend comme paramètre

```
String url = "http://localhost:1234/helloworldws?wsdl";
SOAPMessage soapResponse = soapConnection.call(soapRequest, url);
```

Ci-dessous le message de la réponse SOAP reçu par le client suite à sa requête :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
            <ns2:makeHelloResponse xmlns:ns2="http://tp2.ws.soa.org/">
                  <return>Hello World, SAAJ API</return>
            </ns2:makeHelloResponse>
      </S:Body>
</S:Envelope>
```

### 4. Réception et traitement de la réponse SOAP

Les étapes initiales pour récupérer le contenu d'un message de réponse SOAP sont les mêmes que pour donner du contenu à un message SOAP: utiliser l'objet SOAPMessage pour obtenir l'objet SOAPBody, ensuite, accéder à l'objet SOAPBodyElement de ce dernier.

Pour obtenir le contenu de l'élément qui a été ajouté avec la méthode SOAPElement.*addTextNode*, appeler la méthode *getTextContent*. À noter que *getTextContent* renvoie la valeur de l'enfant immédiat de l'élément qui appelle la méthode.

Pour accéder à SOAPBodyElement, appeler la méthode *getChildElements* sur SOAPBody. Cette dernière méthode prend comme paramètre le namespace de l'élément qui l'identifie de façon unique. Pour ce faire, un objet QName est utilisé.

Passer l'objet QName à *getChildElements* renvoie un objet java.util.Iterator qui contient tous les éléments enfants identifiés cet objet. Dans cet exemple, il n'y en a qu'un seul enfant, donc appeler la méthode *next* renverra le SOAPBodyElement voulu.

```
SOAPBody soapBody = soapResponse.getSOAPBody();
QName bodyName = new QName("http://tp2.ws.soa.org/", "makeHelloResponse",
                 "ns2");
Iterator iterator = soapBody.getChildElements(bodyName);
SOAPBodyElement bodyElement = (SOAPBodyElement)iterator.next();
String helloResponse = bodyElement.getTextContent();
System.out.println("Response Element est : " + helloResponse);
```

### 5. Fermeture de la connexion SOAP

Une connexion SOAP utilise une quantité importante de ressources, il est donc conseillé de la fermer cette dernière dès que son utilisation n'est plus requise, comme suit :

```
soapConnection.close( );
```

## III. Partie TP

- Publier le Service Web SOAP Hello World (i.e. Run As …).
- Créer un nouveau projet Java : File -> New -> Other -> Java Project. Donner comme nom de projet *HelloWorldWSClientSAAJ*.
- Ajouter une classe main dans le projet *HelloWorldWSClientSAAJ*. L'appeler *HelloWorldClientSAAJ*.
- Modifier la classe *HelloWorldClientSAAJ* comme suit :