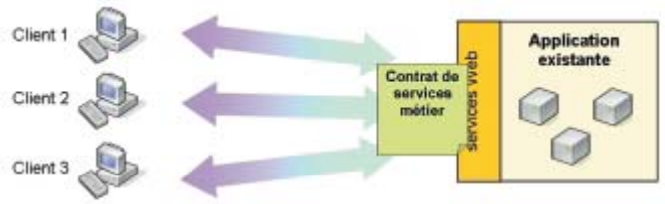
Les services web

Introduction

- Un service web est un composant logiciel qui permet la communication entre deux applications ou systèmes qui peuvent être dans un environnement hétérogène et / ou distribué.
- Contexte d'application
  - Interopérabilité entre plates-formes hétérogènes.
  - Intégration des applications existantes.
  - Client / serveur sur Internet.
  - Fournir des services métier.
  - Applications distribuées.
  - Autres technologies
    - CORBA
    - DCOM
    - RMI



Fournir des services à forte valeur ajoutée métier

Interopérabilité entre plates-formes

Etapes de conception des services web

- Etape 1 : Définition d'un contrat de services métier.
  - Spécification des besoins du client sous la forme d'un contrat de service métier.
- Etape 2 : Identification des Services Web (contrat technique).
  - Identifier les Services Web à partir du contrat de services métier
  - Décrire chaque service métier comme l'assemblage d'un ou plusieurs Services Web.
  - Lors de cette phase on s'intéresse aussi aux problèmes de performance: bande passante, consommation mémoire, montée en charge…
  - Granularité
    - Exemple: soit une application qui désire récupérer les informations d'un client.
    - Deux possibilités de définition:
      - Soit plusieurs services : getNom, getPrénom, getAdresse ;
      - Soit un seul : getClient ➜ Client (Couplage faible entre le client et le serveur).
  - Services Web indépendants du contexte client
- Etape 3: Identifier les Services Web asynchrones et ceux qui échangent de gros documents ou des données binaires.
  - Identifier les services web asynchrones:

    L'objectif de cette étape est de se prémunir contre des blocages et des problèmes de performance éventuels.

    <u>services web candidats:</u>

Etapes de conception des services web

Services web dont le traitement est long parce qu'ils exécutent une requête de base de données longue à traiter ou des appels à d'autres services Web.

Services web asynchrones

Si la plateforme ne supporte pas les services web asynchrones alors il faut publier deux opérations dans le document WSDL par Web Service asynchrone :

- Une première pour envoyer la requête et obtenir en retour un identifiant (le job id) ;
- Une deuxième pour interroger régulièrement le serveur (polling) avec cet identifiant.

Prévoir aussi une opération (commune à tous les Services Web asynchrones) pour annuler un Services Web lancé ou un paramètre supplémentaire lors de l'appel, pour définir un timeout.

Cas de gros documents ou des données binaires: utiliser des pièces jointes.

- Etape 4 : Ecriture du contrat (le fichier WSDL)
  - La plateforme génère automatiquement le fichier WSDL, en Java et .Net la plateforme se base sur les annotations pour générer le contrat wsdl.

Formats

- XML RPC
- SOAP
- REST

XML-RPC

- Présentation
  - Développé en 1998 juste après la publication de la première version du langage XML (XML 1.0)
  - *http://www.xmlrpc.com*
  - Protocole d'appel de procédure distante de type RPC qui utilise:
    - HTTP comme protocole de transport.
    - XML comme format d'échange de message.
- Principe de fonctionnement:
  - Une représentation XML de l'appel de procédure distante est incluse dans le contenu associé à une requête POST HTTP.
  - Une représentation XML du retour de l'appel est incluse dans le contenu associé à une réponse POST HTTP.

WS- SOAP

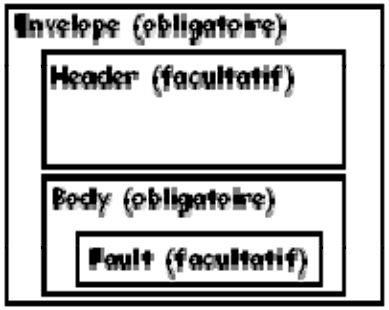
standards techniques définis par le W3C :

- SOAP (Simple Object Access Protocol ou Service Oriented Architecture Protocol);
- Historique:
  - SOAP 1.0: 1999 (Microsoft)
  - SOAP 1.1 : 2000 (IBM) contrats WSDL
  - 2001 Standard W3C: http://www.w3.org/TR/soap/
- WSDL (Web Services Description Language) : langage XML utilisé pour décrire :
  - le contrat de services : l'ensemble des opérations disponibles, ainsi que la structure des messages XML échangés (exprimée en XML Schema) ;
  - Comment transporter les messages XML sur 1 ou plusieurs protocoles (habituellement SOAP) ;
  - la localisation du service web.
- XML et HTTP.
- Le transfert se fait le plus souvent à l'aide du protocole HTTP mais peut être également effectué par d'autres protocoles (SMTP, FTP, HTTPS, TCP/IP, JMS…)

- La spécification SOAP prévoit deux modèles de messages:
  - Les messages de type RPC
    - La structure de la requête et de la réponse est imposée par la spécification
  - Les messages de type Document.
    - Le sens des messages véhiculés est laissé à l'appréciation des applications participant à l'échange
  - Dans SOAP 1.2, seul le support du modèle Document est obligatoire.
- Deux standards pour le support des pièces jointes:
  - Anciens Standards
    - DIME : mécanisme utilisé par .Net et requiert l'installation de WSE 2.0 (Web Services Extension)
    - SOAP With Attachments (W3C / SOAP 1.1) : mécanisme supporté par la plupart des autres plates-formes (Java notamment).
- Nouveau standard : XOP / MTOM (W3C / SOAP 1.2)
  - Il est supporté à la fois par .Net (requiert l'installation de WSE 3.0) et par les autres plates-formes.
  - XOP(XML-binary Optimized Packaging) :permet l'inclusion de *données binaires brutes dans un document XML 1.0 sans avoir recours au codage Base64*
  - *MTOM (Message Transmission Optimization Method): spécifie comment lier le format XOP à SOAP.*

Structure d'un message SOAP

- Un message SOAP est composé de trois sections:
  - L'enveloppe <soap:Envelope> : donne des indications sur le message et son traitement. Sa grammaire est décrite par le schéma XML : « http://schema.xmlsoap.org/soap/envelope » (qui est aussi utilisé pour l'espace de nommage).
  - L'en-tête <soap:Header> contient des informations diverses :
    - adresse d'envoi,
    - adresse source, signature électronique, ...Cette en-tête est facultative.
  - le corps <soap:Body> contient le message ; s'il s'agit d'un message de requête, la méthode invoquée et les paramètres correspondants doivent figurer ; s'il s'agit d'un message de réponse, on y place les résultats.
  - <Fault>: les messages d'erreur




```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/e
nvelope/">
<soap:Header> <!– Optionnel -->
<!-- en-tête -->
</soap:Header>
<soap:Body>
<!– informations ou message d'erreur -->
</soap:Body>
</soap:Envelope>
```

Exemple 1: message document-SOAP

```
<soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <message:transaction xmlns:message ="soap-transaction"
        soap:mustUnderstand ="true">
                <transactionID> 1010</transactionID>
        </message:transaction>
    </soap:Header>
    <soap:Body>
        <n:bonCommande xmlns:n="urn:CommandeService" >
            <emetteur>
                <client>Zidane</client>
                <service>Proudction</service>
            </emetteur>
            <destinataire>
                <vendeur>Hilal</vendeur>
                <service>Ventes</service>
            </destinataire>
            <commande>
                <quantite>20</quantite>
                <produit>Rames papier</produit>
            </commande>
        </n:bonCommande>
    </soap:Body>
</soap:Envelope>
```

Remarque:

L'attribut mustUnderstand="true" possède la signification suivante. Si le destinataire ne comprend pas ce qui se trouve dans l'en-tête (ici l'identifiant de la transaction), le message entier doit être rejeté.

Mise en œuvre d’un service web

- Java
- Java EE
- Java ME

- Pour définir une classe POJO comme un service web la spécification JAX-WS impose les conditions suivantes:
  - La classe doit être annotée par @javax.jws.WebService ou son équivalent XML dans le descripteur de déploiement
  - Pour définir un service web comme un point terminal EJB , la classe doit être annotée par @javax.ejb.Stateless.
  - La classe doit être publique
  - La classe ne doit pas être abstract ou final
  - La classe doit avoir un constructeur par défaut.
  - La classe ne doit pas contenir la méthode finalize()
  - Le service web ne doit pas gérer l'état du client.
- D'après la JSR 181 (WS-Metadata specification) , une classe qui satisfait les conditions précédentes peut être déployée comme un service web dans un conteneur de servlet (on l'appelle une servlet point terminal)

Mise en œuvre d'un service web en Java

- Implémentations des services web dans la plateforme Java
  - *JAX-RPC (JSR 101): dépréciée; remplacée par la JSR 224*
  - JSR 181 (Web Services Metadata for the JavaTM Platform) défini un format d'annotation facilitant la définition de services web
  - JSR 224 avec JAX-WS
  - JSR 222 JAXB (Java API for XML Binding): défini un ensemble d'API et d'annotations pour représenter des documents XML comme des artéfacts Java (ce qui permet la sérialisation (marshaling) dé-sérialisation (unmarshaling) entre documents XML et objets Java.
  - JSR 311 JAX-RS : services web REST (JEE 6.0)

Mise en œuvre en Java: Exemple 1

Création du Service Web: Le service web permet d'associer à un code donné le nom du client qui lui correspond

```
package erp;
import javax.jws.WebService;
@WebService
public class Client {
    public String
    getClient(String code) {
        return
    Clients.getClient(code);
    }
}
```

```
package erp;
public class Clients {
public static String getClient(String code) {
        if (code.equals("c01")) {
                return "Zindane";
        } else if (code.equals("c02")) {
                return "Hilali";
        } else {
                return null;
        }
}
}
```

Le fichier WSDL

<definitions>: racine du document WSDL, elle contient les déclarations globales et les espaces de noms visibles dans le document

```
<?xml version ='1.0' encoding ='UTF-8' ?>
<definitions
targetNamespace='http://clients.com/Client'
xmlns:tns=' http://clients.com/Client '
xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
xmlns='http://schemas.xmlsoap.org/wsdl/'>
```

<types>: définit les types de données qui seront utilisés dans les messages, dans cet exemple les types de données sont définis dans un schéma XML: ClientService?xsd=1

```
<types>
        <xsd:schema>
                <xsd:import namespace="http://erp/" schemaLocation="http://localhost:8080/wsClient/ClientService?xsd=1" />
        </xsd:schema>
</types>
```

Les sections <message> définissent les listes de messages qui peuvent être échangés avec le service. On y retrouve les différents types de requêtes ainsi que les différentes réponses, dans cet exemple la requête getClient et la réponse getClientResponse

```
<message name="getClient">
<part name="parameters" element="tns:getClient" />
</message>
<message name="getClientResponse">
<part name="parameters" element="tns:getClientResponse" />
</message>
```

La section <portType> spécifie les opérations du service Web.

```
<portType name='ClientPortType'>
        <operation name='getclient'>
                <input message='tns:getclientRequest'/>
                <output message='tns:getclientResponse'/>
        </operation>
</portType>
```

la section <binding> et <service> définit les détails d'implémentation (protocoles ou couches transport à utiliser).et le format des messages

```
<binding name="ClientPortBinding" type="tns:Client">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="getClient">
        <soap:operation soapAction="" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
</binding>
```

la section <service> contient une collection d'éléments de type port ; chaque élément <port> est associé à un point terminal (une adresse réseau ou une URL).

```
    <service name='Client_Service'>
        <documentation>WSDL qui permet de récupérer le nom d'un client à partir de son
        code</documentation>
        <port name='ClientPort' binding='ClientBinding'>
            <soap:address location='http://www.client.com/soap/server.php'/>
        </port>
    </service>
</definitions>
```

Le schéma XML

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:tns="http://erp/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        version="1.0" targetNamespace="http://erp/">
        <xs:element name="getClient" type="tns:getClient" />
        <xs:element name="getClientResponse" type="tns:getClientResponse" />
        <xs:complexType name="getClient">
                <xs:sequence>
                        <xs:element name="arg0" type="xs:string" minOccurs="0" />
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="getClientResponse">
                <xs:sequence>
                        <xs:element name="return" type="xs:string" minOccurs="0" />
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Création du consommateur du service web

```
import erp.*;
public class Main {
	public static void main(String[] args) {
		Client cl = new
		ClientService().getClientPort();
		String nom = cl.getClient(null);
		System.out.println(nom);
	}
}
```

Exemple 2: Création d'un service web qui permet la validation de carte de crédit

```
package banque;
import javax.jws.WebService;
// Service web qui permet la validation d'une carte
de crédit
@WebService
public class CCValidation {
	public boolean valider(CarteCredit cc) {
		String type;
		type = cc.getType();
		if (type.equals("MC")) {
			return true;
		}
		return false;
	}
}
```

```
package banque;
import javax.xml.bind.annotation.XmlRootElement;
//Sérialisation, dé-sérialisation XML
@XmlRootElement
public class CarteCredit {
	private String numero;
	private String dateExpiration;
	//valeur de vérification de la carte
	private Integer cvv;
	private String type;
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}
	// … méthodes set et get
}
```

Fichier wsdl

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
        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://banque/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        targetNamespace="http://banque/" name="CCValidationService">
        <types>
                <xsd:schema>
                <xsd:import namespace="http://banque/"
                schemaLocation="http://localhost:8080/wsClient/CCValidationService?xsd=1" />
                </xsd:schema>
        </types>
        <message name="valider">
                <part name="parameters" element="tns:valider" />
        </message>
        <message name="validerResponse">
                <part name="parameters" element="tns:validerResponse" />
        </message>
        <portType name="CCValidation">
                <operation name="valider">
                        <input wsam:Action="http://banque/CCValidation/validerRequest" message="tns:valider" />
                        <output wsam:Action="http://banque/CCValidation/validerResponse" message="tns:validerResponse" />
                </operation>
        </portType>
        <binding name="CCValidationPortBinding" type="tns:CCValidation">
                <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
                <operation name="valider">
                        <soap:operation soapAction="" />
                        <input>
                                <soap:body use="literal" />
                        </input>
                        <output>
                                <soap:body use="literal" />
                        </output>
                </operation>
        </binding>
                <service name="CCValidationService">
                <port name="CCValidationPort" binding="tns:CCValidationPortBinding">
                <soap:address location="http://localhost:8080/wsClient/CCValidationService" />
                </port>
        </service>
</definitions>
```

Schéma XML

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:tns="http://banque/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
targetNamespace="http://banque/">
        <xs:element name="carteCredit" type="tns:carteCredit" />
        <xs:element name="valider" type="tns:valider" />
        <xs:element name="validerResponse" type="tns:validerResponse" />
        <xs:complexType name="valider">
                <xs:sequence>
                        <xs:element name="arg0" type="tns:carteCredit" minOccurs="0" />
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="carteCredit">
                <xs:sequence>
                        <xs:element name="cvv" type="xs:int" minOccurs="0" />
                        <xs:element name="dateExpiration" type="xs:string" minOccurs="0" />
                        <xs:element name="numero" type="xs:string" minOccurs="0" />
                        <xs:element name="type" type="xs:string" minOccurs="0" />
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="validerResponse">
                <xs:sequence>
                        <xs:element name="return" type="xs:boolean" />
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Création du client

```
package TWS1;
public class Main {
	public static void main(String[] args) {
		CarteCredit cc = new CarteCredit();
		cc.setCvv(123);
		cc.setDateExpiration("10/10/2011");
		cc.setNumero("2142140114024254");
		cc.setType("MC");
		CCValidation ccv = new CCValidationService().getCCValidationPort();
		boolean b = ccv.valider(cc);
		System.out.println(b);
	}
}
```

Annotations des services web

- La spécification WS-Metadata specification (JSR 181) définit deux types d'annotations:
  - Annotations de mapping WSDL: ces annotations définies dans le package javax.jws permettre de réaliser le mapping WSDL/Java; les annotations @WebMethod, @WebResult, @WebParam et @OneWay sont utilisées pour personnaliser la signature des méthodes publiées
  - Annotations de binding SOAP: définies dans le package javax.jws.soap sont utilisées pour personnaliser les liaisons soap (@SOAPBinding et @SOAPMessageHandler).
- Remarque:
  - Les annotations des services web peuvent être redéfinies dans un descripteur de déploiement optionnel webservices.xml.

Les annotations WSDL

- @WebService: marque une classe ou une interface Java comme un service web, si l'annotation est utilisée avec une classe alors le processeur des annotations du conteneur générera l'interface correspondante:
  - Les deux blocs de code suivants sont équivalents:
    - **@WebService**
      public class CCValidation{
    - @WebService
      public interface ICCValidation{
      public class CCValidation implements ICCValidation{
  - Attributs optionnel de l'annotation **@WebService**
    - name:Nom de l'élément <wsdl:portType> par défaut le nom de la classe ou de l'interface sera utilisé
    - targetNamespace: espace de noms du fichier WSDL par défaut le nom du package contenant le service web est utilisé
    - serviceName: nom de l' élément < wsdl:service>, par défaut : nom_de_la_classe + "Service".
    - portName() : nom de l'élément <wsdl:port> par défaut: nom_de_la_classe + "Port".
    - wsdlLocation: définit l'adresse du fichier WSDL

@WebMethod

- Attributs:
  - operationName: définit le nom de l'élément <wsdl:operation>, par défaut le nom de la méthode est utilisé.
  - action: définit l'action de cette opération
  - exclude: si exlude== true alors l'opération ne sera pas publiée (valeur par défaut:false).
- Toutes les méthodes publiques du service web qui ne sont pas exclues explicitement à l'aide de l'attribut exclude seront publiées.
- L'annotation @WebResult(name = "nom_retour") permet de définir le nom de la variable de retour. dans le fichier wsdl