Services Web SOAP

Version 3, 2017-2018

El Habib NFAOUI, Ph.D.
Full Professor
Department of Computer Science,
Faculty of Sciences Dhar El Mahraz,
Sidi Mohamed Ben Abdellah University, Fez
elhabib.nfaoui@usmba.ac.ma

Contenu

- Introduction à l'architecture orientée service
- Services web SOAP et leurs spécifications
- WSDL et SOAP
- Développement de services web SOAP à l'aide de JAX-WS

INTRODUCTION A L'ARCHITECTURE ORIENTÉE SERVICE

Plan

1. Motivation et enjeux du «B to B»
2. Application distribuée
3. Architecture orientée service et services web

1. Motivation et enjeux du «B to B»

- Quelques besoins des entreprises en terme d'applications informatiques:
  - Applications distribuées
    - Les sociétés sont présentes sur les différents continents, fonctionnent 24 heures sur 24, 7 jours sur 7 via Internet et dans de nombreux pays ;
    - Leurs systèmes doivent communiquer avec des systèmes externes.
  - Interopérabilité entre applications par un couplage faible (couplage lâche)
    - Tout le monde n'a pas la même organisation et les mêmes systèmes.
  - Flexibilité et indépendance
    - La possibilité de réorganiser le système de façon rapide, efficace et peu chère
  - Partage d'informations/d'applications
    - Entre partenaires, clients ou services
  - Sécurité et confidentialité

2. Application distribuée

- Une application distribuée est définie par un ensemble de composants qui:
  - collaborent pour l'exécution de tâches communes.
  - sont distants géographiquement.
  - sont interconnectés via un réseau de communication.
  - sont hétérogènes.

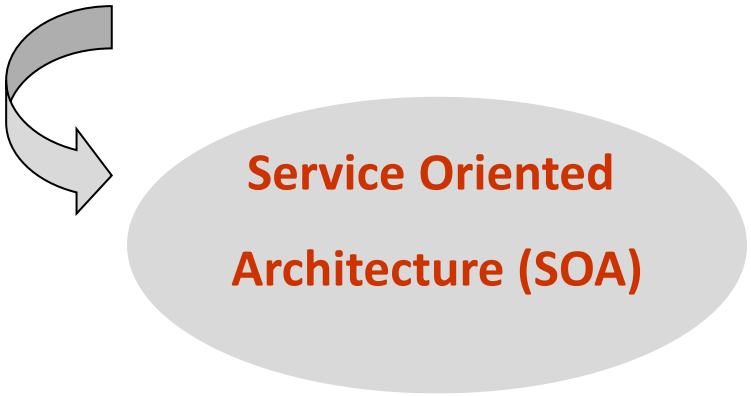
- Solutions (architectures) à base de composants logiciels répartis qui ont fait leur preuve:

  DCOM, CORBA, RMI, EJB, .Net Remoting (trop proche de RMI), …

  - CORBA (OMG)
    - Multilangage, multiplateforme, MultiVendeurs.
  – Java RMI (Oracle)
    - Monolangage : Java, multiplateforme.
  – DCOM (Propriétaire Microsoft)
    - Multilangage, plateforme Windows.

2. Application distribuée

- Limites de ces solutions
  - Interopérabilité limitée : par exemple Java RMI est monolangage.
  - Le protocole de transport est spécifique.



3. Architecture orientée service et services web

- SOA (Service Oriented Architecture) qui est traduit comme « Architecture Orientée Service » **définit une architecture logicielle à base de services.** Le service (ou composant) désigne le fondement de ce modèle d'interaction entre applications. Un Service est un composant logiciel distribué, exposant les fonctionnalités à forte valeur ajoutée d'un domaine métier [XEBIA BLOG, 2009]. Une autre définition est proposée dans [F. Xavier et all., 2017]: Un service, au sens SOA, est un composant d'un système informatisé qui met à disposition de ses clients (acteurs humains, matériels ou logiciels intervenants dans des processus métiers) un accès centralisé à une ou plusieurs fonctions métiers.

  Parmi les aspects qui caractérisent un service, on trouve entre autres: contrat standardisé, couplage lâche, sans état, réutilisabilité, composabilité, etc.

- L'architecture orientée service est un concept et une approche de mise en œuvre des **architectures réparties (distribuées)** centrée sur la notion de **relation de service entre applications et sur la formalisation de cette relation dans un contrat**. Bien qu'elle soit souvent confondue avec les services web, l'architecture orientée services **est en principe un concept indépendant de la technologie des services Web** que nous allons détailler ultérieurement, mais cette **dernière représente son plus important moyen d'implémentation** et fournit la base technologique pour sa diffusion. Les applications orientées services peuvent être également implémentées avec d'autres technologies comme le framework OSGi (Open Services Gateway initiative). La solution basée sur les services Web fait l'**objectif de notre cours.**

3.1 Le paradigme SOA : Publier, Chercher et Consommer

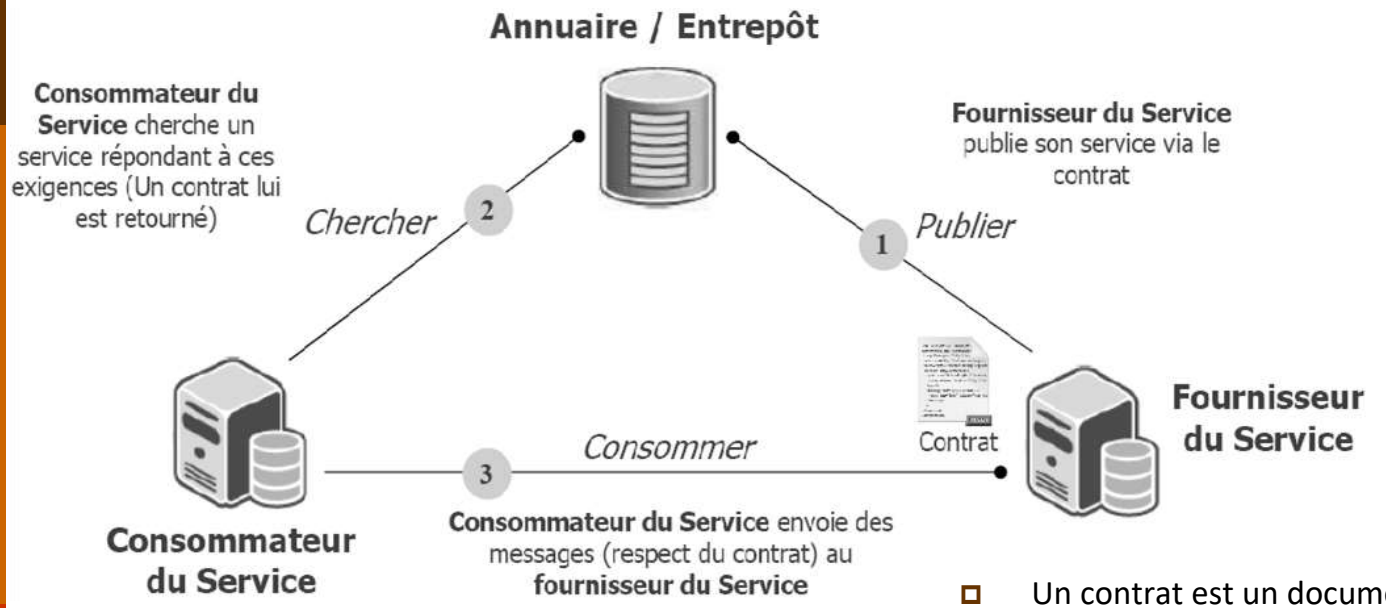


Figure 1. Paradigme SOA [Mickaël BARON, 2010]

- Un contrat est un document organisé en plusieurs parties, dont les plus importantes sont :
  - la description des *fonctions du service ;*
  - la description de l'*interface du service ;*
  - la description de la *qualité du service.*

- Une application logicielle qui exerce une activité dont les résultats sont directement ou indirectement exploitables par d'autres applications, éventuellement réparties sur un réseau, joue le rôle de *fournisseur de services* (ou *prestataire de services)*. L'ensemble des résultats exploitables de l'activité est appelé *prestation de services*, et les applications qui en bénéficient jouent le rôle de *consommateur du service* (ou *client du service)*. Une application peut être en même temps prestataire de plusieurs services distincts et cliente de différents services.

SERVICES WEB SOAP ET LEURS SPÉCIFICATIONS

Plan

1. Introduction
2. Caractéristiques des services Web
3. Exemples
4. Technologies de services Web
5. Architecture orientée service Web
6. Résumé de la spécification des services web

1. Introduction

- Les services web sont un **moyen standard** permettant aux entreprises de communiquer sur un réseau. Plus exactement, les services web sont une sorte de **logique métier** offerte à une application cliente (c'est-à-dire un consommateur de service) via une interface de service. Leurs précurseurs s'appellent CORBA (Common Object Request Broker Architecture), initialement utilisée par les systèmes Unix et DCOM (Distributed Component Object Model), son rival Microsoft. A un niveau plus bas, on trouve RPC (Remote Procedure Call) et, plus près du monde Java, RMI (Remote Method Invocation).

2. Caractéristiques des services Web

- Les services Web sont indépendants de:
  - la plate-forme (UNIX, Windows, …)
  - du langage utilisé pour l'implémentation (Java, C#,…)
  - la plate-forme de développement sous-jacente (.NET, JEE, JAX-WS, Axis…)

- On dit que les services web sont "**faiblement couplés**" car leurs clients **n'ont pas besoin de connaitre les détails d'implémentation** (le langage utilisé pour les développer ou les signatures des méthodes, notamment). Le consommateur peut invoquer un service web à l'aide d'une interface intuitive décrivant les méthodes métiers disponibles (paramètres et valeur de retour). L'implémentation sous-jacente peut être réalisée avec n'importe quel langage de programmation (Java, C#, C, C++, etc.). Avec un couplage faible, un consommateur et un service peuvent quand même échanger des données : en utilisant des documents XML. Un consommateur envoie une requête à un service web sous la forme d'un document XML et, éventuellement, reçoit une réponse également en XML.

- Les services Web sont réutilisables

2. Caractéristiques des services Web

- Les services web concernent également la distribution. Les logiciels distribués existent depuis longtemps, mais les services web sont conçus pour le Web : leur protocole réseau par défaut est HTTP, un protocole sans état bien connu et robuste.

- Les services web sont partout et peuvent s'exécuter sur des machines de bureau ou intervenir dans une intégration B2B (*business-to-business)*. Ils intègrent des applications utilisées par différentes organisations sur Internet ou au sein de la même société – désignées par le terme EAI (*Enterprise Application Integration).* Dans tous les cas, les services web constituent un moyen standard de connecter différents composants logiciels.

3. Exemples

- Un service d'agence de voyages vend des formules de vacances et utilise d'autres services Web : compagnie de carte de crédit, hôtel, compagnie aérienne, etc.

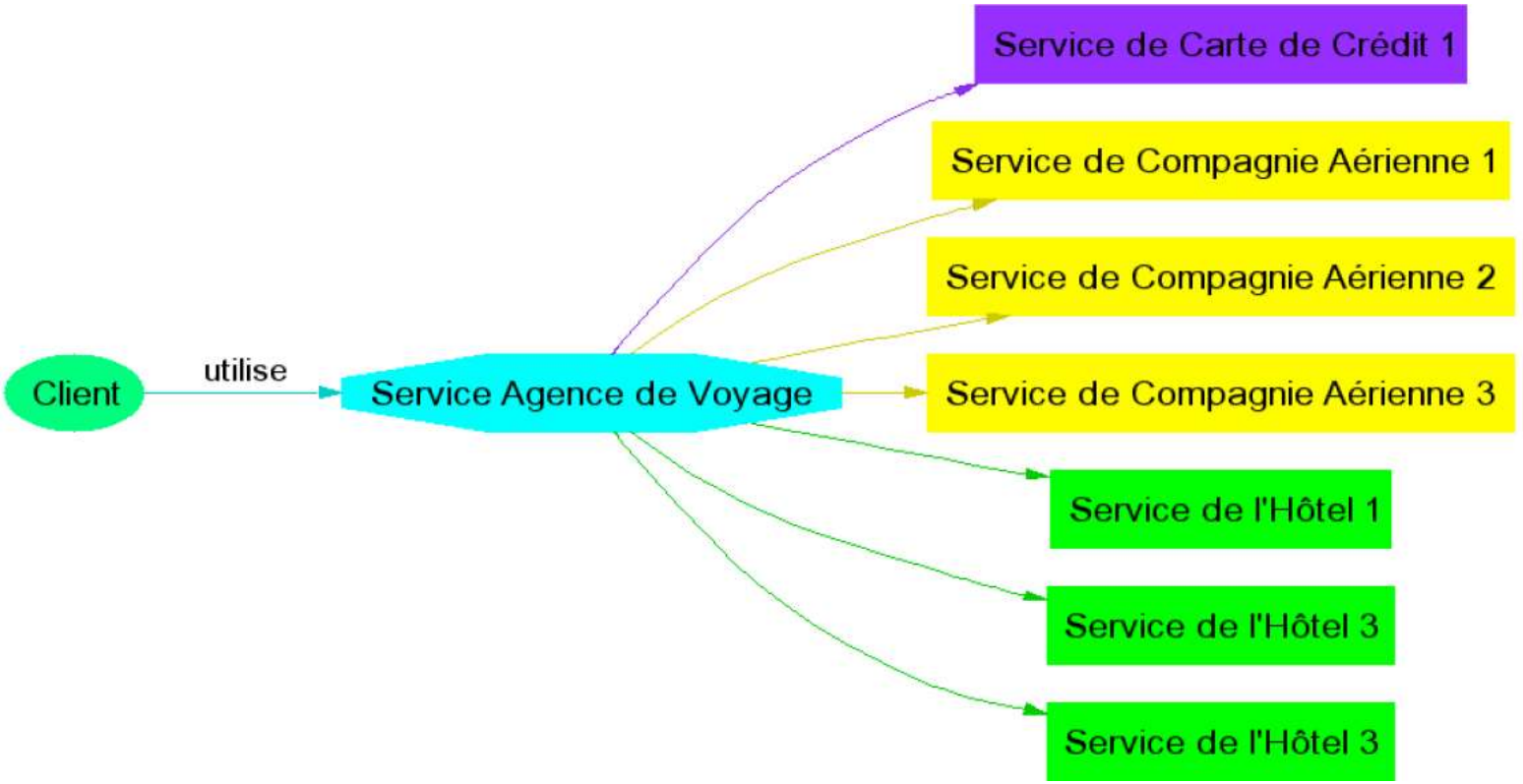


Figure 1. Exemple de service web
(http://www.w3.org/2002/Talks/1203-hh-inria/slide7-0.html)

3. Exemples

- **Avantages:**
  - Le service d'agence de voyage a accès à plus de services.
  - Les clients ont accès à de meilleures offres.
  - Bas coût pour de nouvelles compagnies aériennes, hôtels,
  - Compagnies de carte de crédit: mettre en vente leurs produits et services.

- Google Web Services
- Amazon Web Services (AWS).
- Facebook Web Services.
- CDYNE: http://www.cdyne.com

  Fournisseur mondial d'API SOAP et REST offrant des services web de différents métiers (SMS notify, Phone notify, Phone verify, Data quality APIs)

4. Technologies de services Web

- Le terme « technologies de services Web » désigne un ensemble de technologies basées sur des standards ouverts (non propriétaires) et aptes à la mise en œuvre d'architectures orientées services.

- Définition du W3C:
  - " A Web service is a *software application* identified by a *URI*, whose *interfaces* and *binding** are capable of being *defined, described and discovered* by XML artefacts and supports direct interactions with other software applications using *XML based messages* via *Internet-based protocols* ".

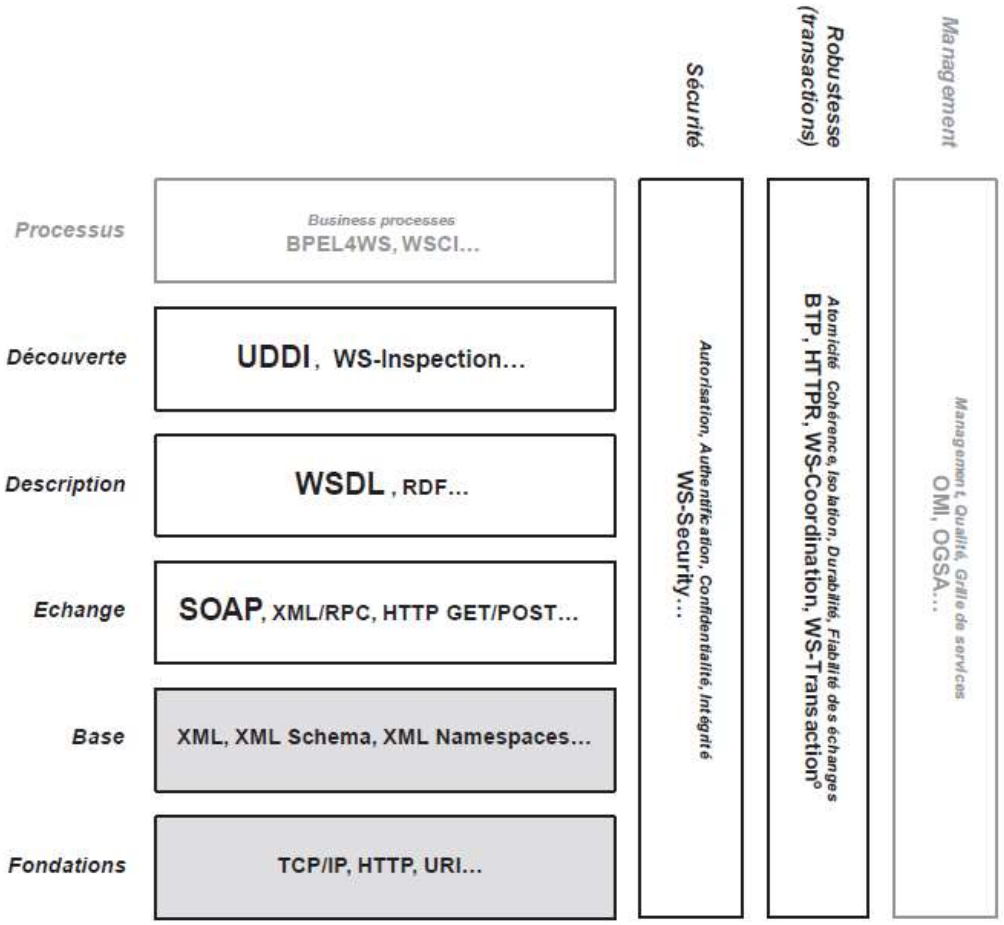
  - Un service Web est une application logicielle:
    1. identifiée par un URI dont les interfaces et les liaisons sont définies, décrites et découvertes avec des mécanismes XML, et
    2. supporte une interaction directe avec les autres applications logicielles en utilisant des messages XML via un protocole Internet.

(*) An association between an Interface, a concrete protocol and a data format

- Pour résumer, les services web sont une sorte de logique métier offerte à une application cliente (c'est-à-dire un consommateur de service) via une interface de service. À la différence des objets ou des EJB du langage Java, les services web fournissent une interface faiblement couplée en se servant de XML. Les standards précisent que l'interface à laquelle on envoie un message doit définir le format du message de requête et de réponse, ainsi que les mécanismes pour publier et pour découvrir les interfaces du service (une base de registres).

4. Technologies de services Web

- Le diagramme général des technologies de services Web est présenté sur la figure ci-dessous. Chaque brique technologique représentée dans le diagramme joue un rôle précis dans une architecture orientée services. **L'architecture orientée services est donc une spécification « générique » d'une famille de systèmes répartis dont les technologies de services Web constituent un moyen d'implémentation privilégié.**



La notation abrégée WS - * est utilisée pour désigner les standards se rapportant aux services web.

Figure 2. Le diagramme des technologies de services Web.

5. Architecture orientée service Web

- La Figure ci-dessous représente l'interaction d'un service web de façon très abstraite. Le service peut **éventuellement** enregistrer son interface dans une base de registres (UDDI) afin qu'un consommateur puisse le trouver. Une fois que le consommateur connait l'interface du service et le format du message, il peut envoyer une requête et recevoir une réponse.

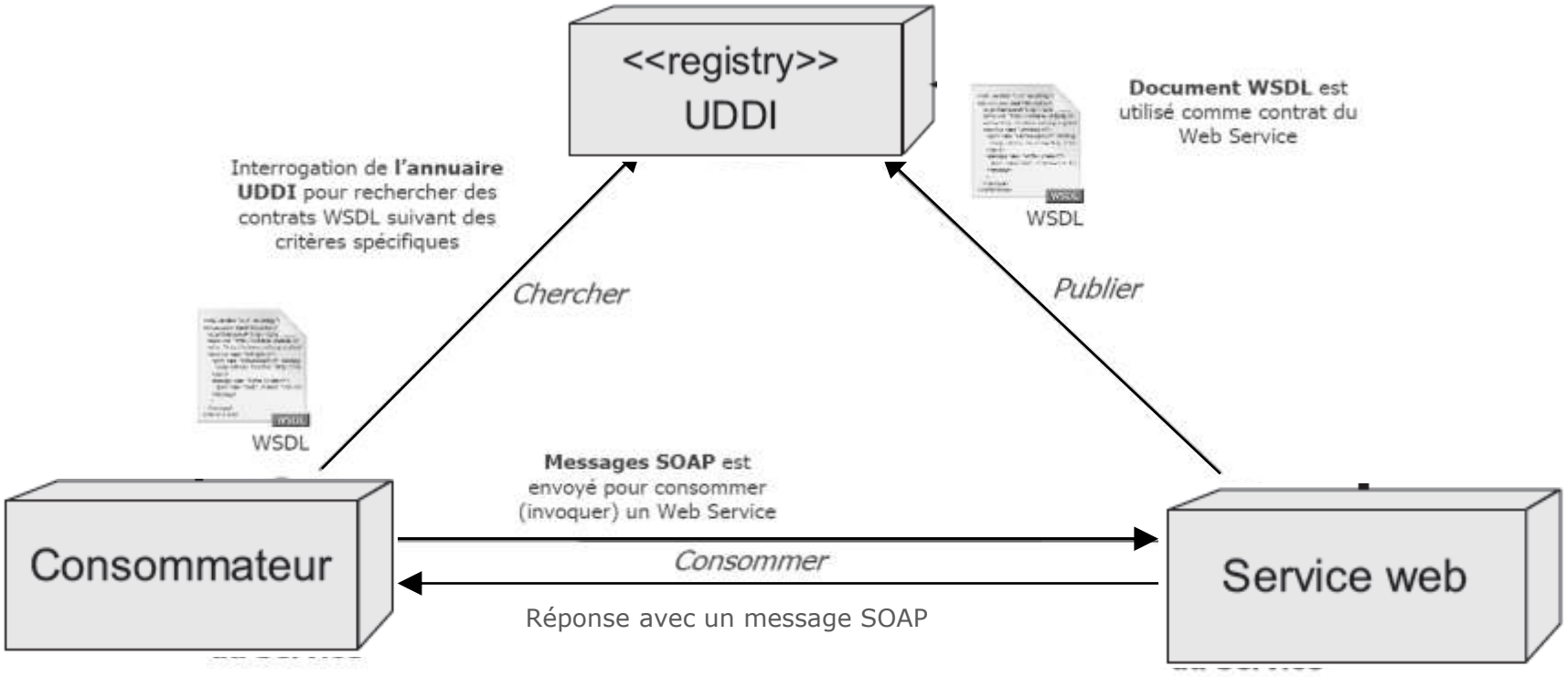


Figure 3. Architecture utilisant le protocole SOAP et le langage WSDL.

5. Architecture orientée service Web

- Les partenaires :
  - Le **fournisseur de services** crée le service Web, puis publie son interface ainsi que les informations d'accès au service, dans un annuaire de services Web.
  - L'**annuaire de services** rend disponible l'interface du service ainsi que ses informations d'accès, pour le demandeur potentiel de service.
  - Le **demandeur de services (consommateur)** accède à l'annuaire de service pour effectuer une recherche afin de trouver les services désirés. Ensuite, il se lie au fournisseur pour invoquer le service.
- Les services web nécessitent plusieurs technologies et protocoles pour transporter et transformer les données d'un client vers un service de façon standard. Les plus courants sont les suivants :
  - **UDDI (Universal Description Discovery and Integration)** est une base de registres et un mécanisme de découverte qui ressemble aux pages jaunes. Il sert à stocker et à classer les interfaces des services web.
  - **WSDL (Web Services Description Language)** définit l'interface du service web, les données et les types des messages, les interactions et les protocoles.
  - **SOAP (Simple Object Access Protocol)** est un protocole d'encodage des messages reposant sur les technologies XML. Il définit une enveloppe pour la communication des services web.

    Les messages sont échangés à l'aide d'un protocole de transport. Bien que HTTP (*Hypertext Transfer Protocol)* soit le plus utilisé, d'autres comme SMTP ou JMS sont également possibles.
  - **XML (Extensible Markup Language)** est la base sur laquelle sont construits et définis les services web (SOAP, WSDL et UDDI).

  Grâce a ces technologies standards, les services web ont un potentiel quasiment illimité. Les clients peuvent appeler un service qui peut être associé à n'importe quel programme et accommoder n'importe quel type et structure de données pour échanger des messages via XML.

5.1 UDDI

- Les programmes qui interagissent avec un autre via le Web doivent pouvoir trouver les informations leur permettant de s'interconnecter. UDDI fournit pour cela une approche standardisée permettant de trouver les informations sur un service web et sur la façon de l'invoquer.

  UDDI est une base de registres de services web en XML pour publier et utiliser des services web sur Internet. L'idée est que, puisque beaucoup de services sont utiles sur Internet, comme un service de vérification des numéros de carte de crédit, des sociétés pourraient proposer leur service à d'autres sociétés clientes, un peu comme les professionnels peuvent enregistrer leurs services dans les pages jaunes. Cet enregistrement inclut le type du métier, sa localisation géographique, le site web, le numéro de téléphone, etc. Les autres métiers peuvent ensuite parcourir cette base et retrouver les informations sur un service web spécifique, qui contiennent des métadonnées supplémentaires décrivant son comportement et son emplacement. Ces informations sont stockées sous la forme de document WSDL : les clients peuvent lire ce document afin d'obtenir l'information et invoquer le service.

5.1 UDDI

- UDDI comprend trois parties :
  - Des pages blanches : informations sur les sociétés qui ont publié des services (adresse…).
  - Des pages jaunes : il s'agit d'une répartition des services en catégories.
  - Des pages vertes : elles regroupent des informations techniques et les services disponibles.

- Le registre XML est interrogeable comme un service web classique, c'est-à-dire avec SOAP et un descripteur WSDL. Voici quelques serveurs UDDI :
  - jUDDI est une implémentation open source réalisée par le groupe Apache (*http://ws.apache.org/juddi/*).
  - Oracle, Microsoft, IBM fournissent aussi des serveurs UDDI

5.2 WSDL

- La base de registres UDDI pointe vers un fichier WSDL sur Internet, qui peut être téléchargé par les consommateurs potentiels. WSDL est un langage de définition d'interfaces (IDL) permettant de définir les interactions entre les consommateurs et les services (voir Figure ci-dessous). C'est donc le composant central d'un service web puisqu'il décrit le type du message, le port, le protocole de communication, les opérations possibles, son emplacement et ce que le client peut en attendre. Vous pouvez considérer WSDL comme une interface Java, mais écrite en XML.

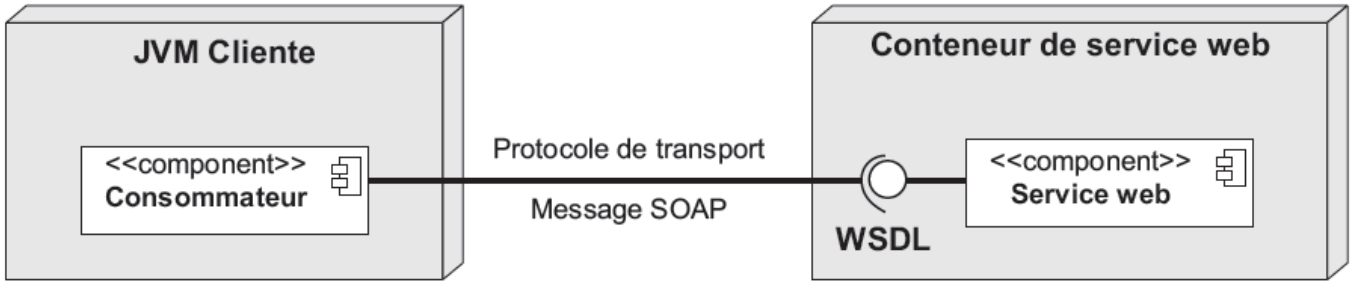


Figure 4. Interface WSDL entre le consommateur (par exemple un client Java) et le service web.

- Pour garantir l'interopérabilité, l'interface standard du service web doit être standardisée, afin qu'un consommateur et un producteur puissent partager et comprendre un message. C'est le rôle de WSDL ; SOAP, de son coté, définit la façon dont le message sera envoyé d'un ordinateur à l'autre.

5.3 SOAP et protocole de transport

- SOAP (Simple Access Object Protocol) est le protocole standard des services web. Il fournit le mécanisme de communication permettant de connecter les services qui échangent des données au format XML au moyen d'un protocole réseau – HTTP le plus souvent. Comme WSDL, SOAP repose fortement sur XML : un message SOAP est un document XML contenant plusieurs éléments (une enveloppe, un corps, etc.). SOAP est conçu pour fournir un protocole indépendant et abstrait, permettant de connecter des services distribués. Ces services connectés peuvent être construits à l'aide de n'importe quelle combinaison de matériels et de logiciels reconnaissant un protocole de transport donné.

- Pour qu'un consommateur puisse communiquer avec un service web, il faut qu'ils puissent s'envoyer des messages. Les services web étant essentiellement utilisés sur le Web, ils utilisent généralement HTTP, mais d'autres protocoles, comme HTTPS (Secure HTTP), TCP/IP, SMTP (Simple Mail Transport Protocol), FTP (File Transfer Protocol), sont également possibles.

5.4 XML

- Pour les services web, XML sert de technologie d'intégration pour résoudre les problèmes d'indépendance des données et d'interopérabilité. Il est utilisé non seulement pour le format des messages mais également pour définir les services (WSDL) ou la façon dont ils sont échangés (SOAP). Des schémas sont associés à ces documents XML pour valider les données échangées.

6. Résumé de la spécification des services web

- Les spécifications des services Web proviennent de standards différents (W3C (World Wide Web Consortium), d'OASIS (Organization for the Advancement of Structured Information Standards), du JCP (Java Community Process), etc.

  - Le W3C est un consortium qui développe et gère les technologies web comme HTML, RDF, CSS, etc. Il s'occupe également des technologies des services web : XML, XML Schema, SOAP et WSDL.

  - OASIS héberge plusieurs standards liés aux services web, comme UDDI, WS-Addressing, WS-Security, WS-Reliability et bien d'autres.

6. Résumé de la spécification des services web

- **Spécifications Java EE et Implémentation de référence**

  les services web sont utilisés par de nombreux autres langages de programmation, ses spécifications ne sont pas directement liées au JCP.

  - Le JCP propose un ensemble de spécifications faisant partie de Java EE et Java SE – notamment JAX-WS, JAXB, Web Services Metadata, etc.
  - **Metro n'est pas une spécification Java EE mais une implémentation de référence open-source des spécifications des services web Java**. Metro permet de créer et de déployer des services web et des consommateurs sécurisés, fiables, transactionnels et interopérables. Bien que la pile Metro soit produite par la communauté GlassFish, elle peut être utilisée en dehors de celui-ci, dans un environnement Java EE ou Java SE.

- **Spécifications pour les autres langages**

  les services web sont utilisés par de nombreux autres langages de programmation, à titre d'exemple, PHP fournit une extension native nommée « SOAP ». Elle permet de meilleures performances et profite des toutes dernières fonctionnalités objet de PHP. Le framework .Net fournit aussi des API pour les services Web.

WSDL ET SOAP

Plan

1. Introduction
2. WSDL (Web Services Description Language)
3. SOAP (Simple Object Access Protocol)
4. Exemple complet

1. Introduction

- Même si l'on ne manipule pas explicitement des documents SOAP et WSDL lorsque l'on développe avec des API ou des frameworks, il est important de comprendre un peu leur structure.

  Les services web fournissent deux ingrédients essentiels : un langage de définition d'interface (WSDL) et un standard de messages (SOAP). Lorsque l'on développe des services web, on peut se servir de WSDL pour définir leur interface, c'est-a-dire définir les paramètres d'entrée et de sortie du service en termes de schéma XML. Les messages SOAP, quant à eux, permettent de transporter les paramètres d'entrée et de sortie précisés par le document WSDL.

2. WSDL

- WSDL (Web Services Description Language) est standardisé par le W3C, il permet de définir l'interface d'un service web. Cette description est suffisante pour utiliser le service sans connaître son implémentation.

- Un fichier WSDL est un document XML qui décrit :
  - **Ce que fait le service** : la liste des opérations, leur sens (requête seule ou requête-réponse), leur signature (le type des arguments).
  - **Où le trouver** : le nom du service, et le point d'attache (« endpoint ») pour l'invocation (le plus souvent l'URL du serveur Web hébergeur).
  - **Comment l'invoquer** : les règles d'encodage des arguments, le protocole de transport utilisé.

- Les documents WSDL sont hébergés dans le conteneur de service web.

2. WSDL

La figure ci-dessous illustre les différentes parties d'un document WSDL.

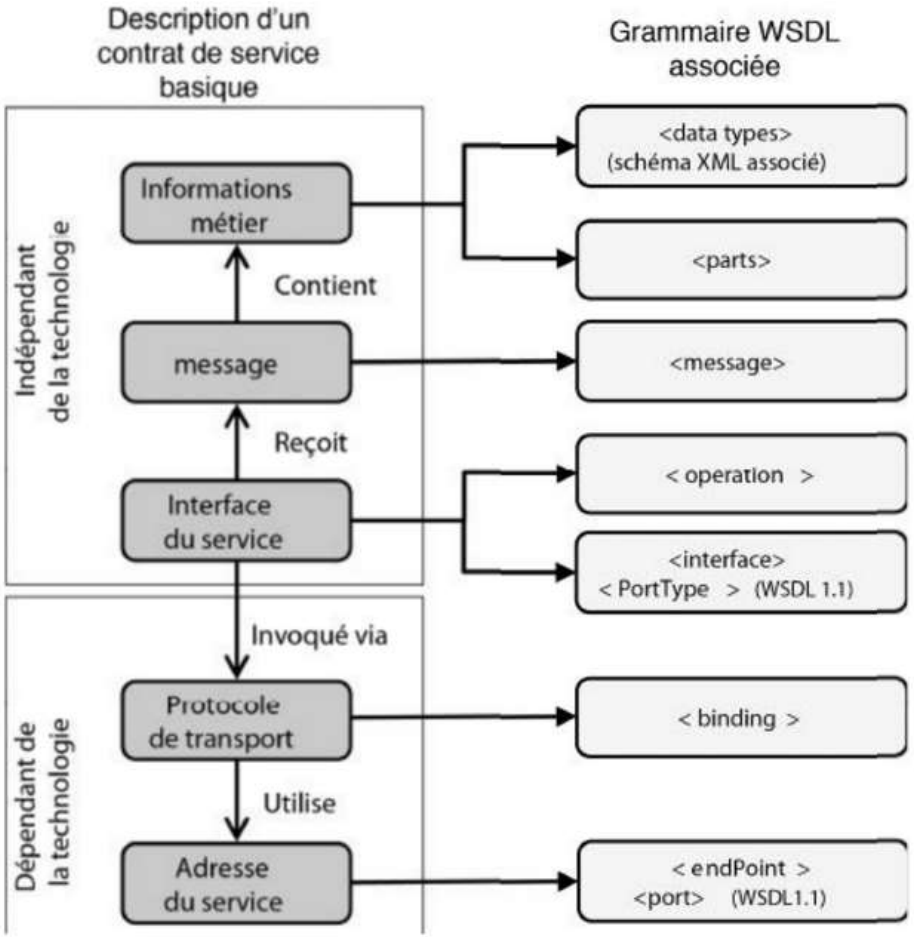


Figure 1. Parties d'un document WSDL

2. WSDL

- La **partie abstraite** d'un service rassemble les données et les opérations du service. Les données échangées, leur format, leurs noms, leur structure et leurs contraintes sont exprimés selon une grammaire XML et reposent sur le standard XML Schema. Il est possible d'importer un schéma existant au sein d'un WSDL pour réutiliser un modèle pivot canonique. Les opérations sont également formalisées et indiquent les données liées en entrée et en sortie, ainsi que les cas d'exception.
- La **partie concrète** d'un service indique des informations techniques comme les normes d'encodage et de transport, et les adresses d'invocation du service associé.

- Les IDE de développement actuels masquent le langage XML associé à un WSDL. Ils peuvent également générer le WSDL associé à un service à partir de son code, par exemple une interface Java annotée via JAX-WS pour la plate-forme Java ou une configuration WCF pour la plate-forme .NET.

2.1 Structure d'un document WSDL

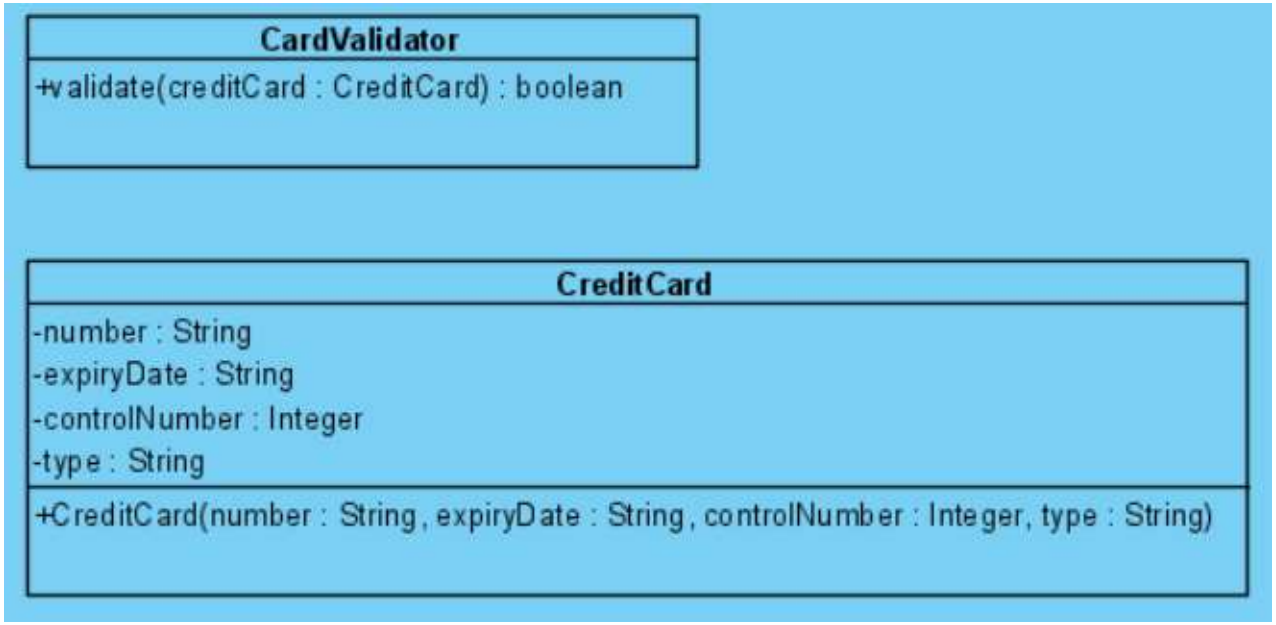
- Un document WSDL est tout d'abord un document XML. Il respecte une structure bien établie, formée de plusieurs parties comme le montre la figure ci-dessous:


```
<definitions>
        <types>
                définition des types
        </types>
        <message>
                définition des messages
        </message>
        <portType>
                définition des interfaces
        </portType>
        <binding>
                définition des bindings
        </binding>
        <service>
                définition de endpoint
        </service>
</definitions>
```

Figure .2. Structure d'un document WSDL

2.1 Structure d'un document WSDL

- Pour bien comprendre le rôle de chaque élément d'un document WSDL, nous considérons à titre d'exemple le contrat WSDL d'un service web (classe CardValidator) qui permet de valider des cartes de crédit (classe CreditCard).



Le code suivant montre le contrat WSDL pour le service web CardValidator.

Nous pouvons remarquer que le WSDL n'a pas été conçu pour être décrit « à la main », mais généré automatiquement par un outil de développement.

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

    <types>
        <xsd:schema>
            <xsd:import namespace="http://projet.org/"
                schemaLocation="http://localhost:8080/projet/CardValidatorService?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="validate">
        <part name="parameters" element="tns:validate"/>
    </message>
    <message name="validateResponse">
        <part name="parameters" element="tns:validateResponse"/>
    </message>
    <portType name="CardValidator">
        <operation name="validate">
            <input message="tns:validate"/>
            <output message="tns:validateResponse"/>
        </operation>
    </portType>
    <binding name="CardValidatorPortBinding"
        type="tns:CardValidator">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <operation name="validate">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="CardValidatorService">
        <port name="CardValidatorPort"
            binding="tns:CardValidatorPortBinding">
            <soap:address location ="http://localhost:8080/projet/CardValidatorService"/>
        </port>
    </service>
</definitions>
```

2.1.1 Élément <definitions>

- C'est l'élément racine de WSDL. Il contient le nom du service décrit et les espaces de noms (namespaces) faisant référence aux types utilisés dans le document.

Exemple:

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

    ....
</definitions>
```

2.1.2 Élément <types> : Les types de données

- Il définit les types de données (paramètres) utilisés par les messages.
- Utilise généralement la grammaire XSD car cette dernière supporte un grand nombre de type de données. Les types peuvent être tirés de la spécification XML Schema (comme xsd:string) ou être définis par l'utilisateur par composition de types simples.
- Ne peut être présent qu'une fois, mais peut contenir plusieurs définitions.

Exemple:

C'est la définition du schéma XML (CardValidatorService?xsd=1) qui décrit le type des paramètres de la requête adressée au service (un objet CreditCard) et le type de la réponse (un Boolean).

```
<types>
    <xsd:schema>
      <xsd:import namespace="http://projet.org/"
        schemaLocation="http://localhost:8080/projet/CardValidatorService?xsd=1"/>
    </xsd:schema>
</types>
```

L'élément <xsd:import namespace> fait référence à un schéma XML qui doit être disponible sur le réseau pour les clients du WSDL. Le code ci-dessous montre que ce schéma définit les types utilisés par le service web (structure d'un objet CreditCard avec un numéro, une date d'expiration, etc.).

```
<xs:schema version="1.0"
   targetNamespace="http://projet.org/"
   xmlns:tns="http://projet.org/"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:element name="creditCard" type="tns:creditCard"/>
   <xs:element name="validate" type="tns:validate"/>
   <xs:element name="validateResponse" type="tns:validateResponse"/>
   <xs:complexType name="validate">
      <xs:sequence>
         <xs:element name="arg0" type="tns:creditCard" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="creditCard">
      <xs:sequence>
         <xs:element name="controlNumber" type="xs:int" minOccurs="0"/>
         <xs:element name="expiryDate" type="xs:string" minOccurs="0"/>
         <xs:element name="number" type="xs:string" minOccurs="0"/>
         <xs:element name="type" type="xs:string" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="validateResponse">
      <xs:sequence>
         <xs:element name="return" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

2.1.3 Élément <message>

- Chaque message caractérise grâce à l'élément <message>, un accès à une méthode en entrée ou en sortie. Il est constitué d'un ensemble d'éléments <part>. L'élément <part> permet de décrire les arguments et la valeur de retour.
- Le nom d'un message est unique dans l'ensemble des noms des messages définis dans le document.

```
<definitions .... >
        <message name=" nmtoken ">
                <part name="nmtoken" element="qname" type="qname"/>
                …
        </message>
        …
</definitions>
```

  - l'attribut **name** est unique parmi les parties du message
  - l'attribut **element** référence un élément de schéma XML par un nom qualifié ;
  - l'attribut type référence un **simpleType** ou un **complexType** de schéma XML par un nom qualifié .

- Les parties de message sont utiles pour définir les contenus logiques abstraits d'un message et permettre ainsi de les référencer directement par les éléments de liaison.

2.1.3 Élément <message>

Exemple:

Dans notre code, il s'agit de la requête (la méthode validate) et de la réponse (validateResponse).

```
<message name="validate">
   <part name="parameters" element="tns:validate"/>
</message>
<message name="validateResponse">
   <part name="parameters" element="tns:validateResponse"/>
</message>
```

2.1.4 Élément <portType>

- Cet élément précise les opérations du service.
- Chaque type de port contient un ensemble d'opérations. Chaque opération, étant une association de messages en entrée et en sortie, est en quelque sorte un appel de méthode complet avec passage des paramètres et récupération d'une valeur. On utilise l'élément <portType pour le caractériser>.
- **Les types de ports** (proche d'une interface au sens Java)
  - Le type de port définit un ensemble d'opérations (une opération est proche d'une méthode au sens Java) abstraites et indique les messages impliqués dans ces opérations (Nb: en WSDL version 2, le portType est rebaptisé Interface). Cet élément se situe comme suit dans la hiérarchie du document WSDL :

```
<definitions .... >
    <portType name="nmtoken">
        <operation name="nmtoken" ... />
        …
    </portType>
</definitions>
```

2.1.4 Élément <portType>

- Une opération est un ensemble de messages qui constitue une unité d'interaction (*transmission primitive*) avec le service Web.
- Il existe plusieurs types d'opérations:
  - Interaction à sens unique: **One-way**
    - L'interaction à sens unique correspond à une situation où le nœud de communication ne fait que réceptionner un message (<input>) :

```
<operation name="nmtoken">
   <input name="nmtoken«  message="qname"/>
</operation>
```

  - **Request-response**
    - Le point d'entrée reçoit un message (<input>) et retourne un message corrélé (<output>) ou un ou plusieurs messages de faute (<fault>). Cette configuration s'exprime ainsi :

```
<operation name="nmtoken" parameterOrder="nmtokens">
      <input name="nmtoken "  message="qname"/>
      <output name="nmtoken" message="qname"/>
      <fault name="nmtoken" message="qname"/>
</operation>
```

2.1.4 Élément <portType>

- **Solicit-response**
  - Le point d'entrée envoie un message (<output>) et reçoit un message corrélé (<input>) ou un ou plusieurs messages de faute (<fault>).
- **Notification**
  - Le point d'entrée envoie un message de notification (<output>)

- Paramètres
  - Les champs des messages constituent les paramètres (in, out, inout) des opérations

2.1.4 Élément <portType>

Exemple :

Dans notre code, il s'agit de déclarer l'opération fournie par le service (la méthode validate).

```
<portType name="CardValidator">
    <operation name="validate">
        <input message="tns:validate"/>
        <output message="tns:validateResponse"/>
    </operation>
</portType>
```

2.1.5 Élément <binding>: Liaison

- Il décrit le protocole concret (SOAP, ici) et les formats des données pour les opérations et les messages définis pour un type de port particulier. Autrement dit, la liaison (binding) indique comment les opérations d'un type de port sont encodés et circulent. La structure générique de ces éléments de liaison est représentée de la manière suivante :

```
<binding name="nmtoken" type="qname">
        <operation name="nmtoken">
                <input name="nmtoken">
                </input>
                <output name="nmtoken">
                </output>
                <fault name="nmtoken">
                </fault>
        </operation>
</binding>
```

2.1.5 Élément <binding>: Liaison

Exemple:

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

L'attribut *transport* indique par quel moyen les données circulent (HTTP, FTP, SMTP…) alors que l'attribut *style* indique comment les messages sont exploités : soit sous la forme de document, soit sous une forme RPC (la différence est minime et concerne le format des messages). Chaque partie du message SOAP en entrée (input) et sortie (output) est ensuite définie. On indique comment sont structurés l'en-tête header (en option) et le corps body. L'attribut *use* peut prendre les valeurs literal ou encoded (en réalité, comme le style est document, seule la valeur literal a un sens).

2.1.6 Élément <service>

- Un service est un point d'accès à un ensemble de ports, chaque port étant relié à un élément de liaison et une adresse d'accès pour les clients.
- Cet élément contient une collection d'éléments <port> associés, chacun, à une extrémité (une adresse réseau ou une URL).
  - <port>

    Un port définit un nœud de communication (ou point d'entrée (endpoint)), et donc un URI, pour une liaison particulière. Dans un document WSDL, cet élément se décrit ainsi :

```
<port name="nmtoken" binding="qname">
</port>
```

  - <service>
    - Une collection de points d'entrée (endpoint) relatifs. Un service est décrit dans un document WSDL de la manière suivante :

```
<service name="nmtoken">
    <port .... />
</service>
```

Exemple:

```
<service name="CardValidatorService">
    <port name="CardValidatorPort"
      binding="tns:CardValidatorPortBinding">
      <soap:address location ="http://localhost:8080/projet/CardValidatorService"/>
    </port>
</service>
```

2.1.7 Résumé de la structure d'un document WSDL

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

3. SOAP

- SOAP (Simple Object Access Protocol) fait partie de la couche de communication des Web Services. Il définit la structure des messages XML utilisés par les applications pour dialoguer entre elles. La force de ce protocole réside dans son universalité et sa flexibilité. C'est une recommandation du W3C

- SOAP est un protocole qui utilise XML pour sérialiser les méthodes et leurs arguments, ainsi que les valeurs de retour. Avec SOAP, XML transite en général au-dessus du protocole HTTP. Ce dernier est un protocole de communication au-dessus de TCP/IP qui permet d'effectuer des requêtes sur un serveur Web.

3.1 Structure des messages SOAP

- Alors que WSDL décrit une interface abstraite du service web, SOAP fournit une implémentation concrète en définissant la structure XML des messages échangés. Dans le cadre du Web, SOAP est une structure de messages pouvant être délivrés par HTTP (ou d'autres protocoles de communication) – la liaison HTTP de SOAP contient quelques en-têtes d'extension HTTP standard. Cette structure de message est décrite en XML. Au lieu d'utiliser HTTP pour demander une page web à partir d'un navigateur, SOAP envoie un message XML *via une requête HTTP* et reçoit *une réponse* HTTP. Un message SOAP est un document XML contenant les éléments suivants :
  - <Envelope> : Définit le message et l'espace de noms utilisés dans le document. Il s'agit de l'élément racine obligatoire.
  - <Header>: Contient les attributs facultatifs du message ou l'infrastructure spécifique à l'application, comme les informations sur la sécurité ou le routage réseau.
  - <Body> : Contient le message échangé entre les applications (Nom d'une procédure, valeurs des paramètres, valeur de retour).
  - <Fault> : Fournit des informations sur les erreurs qui surviennent au cours du traitement du message. Cet élément est facultatif.

Seuls l'enveloppe et le corps sont obligatoires.

3.1 Structure des messages SOAP

La figure ci-dessous récapitule les éléments d'un message SOAP.

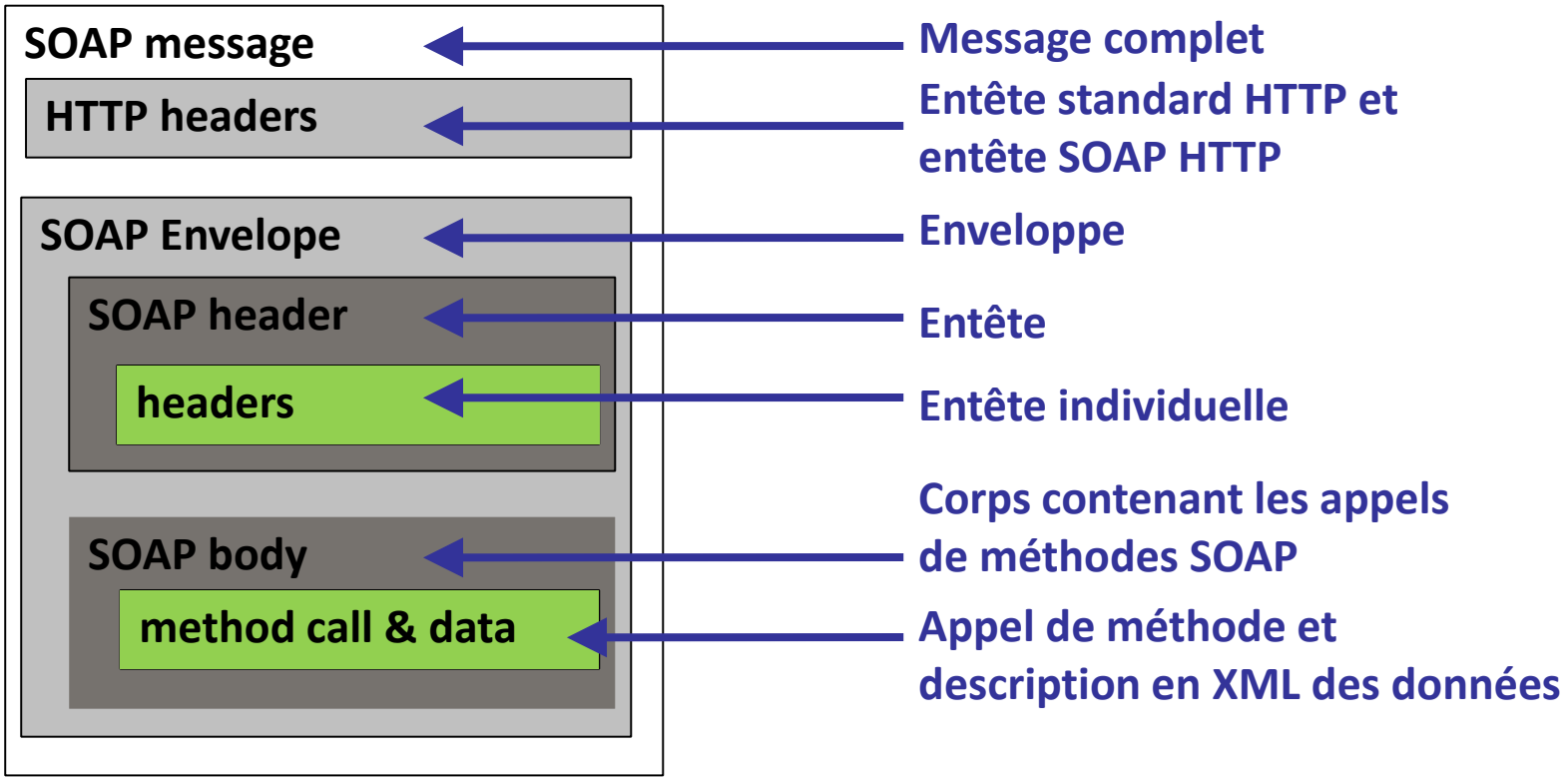


Figure 3. Eléments constituant un message SOAP

3.1 Structure des messages SOAP

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

      <soap:Header>
          ...
      </soap:Header>
      <soap:Body>
          ...
          <soap:Fault>
    .     ..
          </soap:Fault>
      </soap:Body>
</soap:Envelope>
```

3.2 Exemple d'un entête de l'enveloppe

Exemple d'un entête qui spécifie un compte d'accès au service (nom et mot de passe).

```
<soap:Header>
<a:Authentication>
    <Username b:type='c:string'>user@example.org</Username>
    <MD5 b:type='c:string'>9b3e64e326537b4e8c0ff19e953f9673</MD5>
</a:Authentication>
</soap:Header>
```

3.3 Exemple d'un corps SOAP

- Dans notre exemple, une application cliente appelle le service web pour valider une carte de crédit (une enveloppe SOAP pour la requête) et reçoit un booléen indiquant si cette carte est valide ou non (une autre enveloppe SOAP pour la réponse). Les listings de code ci-dessous montrent les structures de ces deux messages SOAP.

Enveloppe SOAP de la requête:

```
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:cc="http://projet.org/">
   <soap:Header/>

   <soap:Body>
      <cc:validate>
         <arg0>
            <controlNumber>1234</controlNumber>
            <expiryDate>10/10</expiryDate>
            <number>9999</number>
            <type>VISA</type>
         </arg0>
      </cc:validate>
   </soap:Body>
</soap:Envelope>
```

Enveloppe SOAP de la réponse:

```
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:validateResponse
         xmlns:ns2="http://projet.org/">
         <return>true</return>
      </ns2:validateResponse>
   </soap:Body>
</soap:Envelope>
```

3.4 Exemple d'utilisation de <Fault>

- Si le traitement de l'appel n'avait pas pu être correctement effectué, un code d'erreur (code) aurait été retourné par l'élément *Fault* avec un message explicite (Reason). Voici un exemple:

```
<soap:Body>
   <soap:Fault>
      <soap:Code><soap:Value>soap:MustUnderstand</soap:Value></soap:Code>
      <soap:Reason><soap:Text>Exception .....</soap:Text>
      </soap:Reason>
   </soap:Fault>
</soap:Body>
```

4. Exemple complet

Le code ci-dessous montre une classe annotée @WebService() (API JAX-WS), il décrit un service web nommé CalculatorWS proposant une méthode add qui retorne la somme de deux entiers passés en paramètres.

```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService()
public class CalculatorWS {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "add")
    public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j) {
        int k = i + j;
        return k;
    }
}
```

4. Exemple complet

La figure ci-dessous montre la page de test du service Web CalculatorWS affichée par le testeur intégré dans l'IDE NetBeans:

Figure 4. Page de test

Le code ci-dessous montre le fichier WSDL généré automatiquement par l'API JAX-WS

```
- <!--
    Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.1-hudson-28-.
  -->
- <!--
    Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.1-hudson-28-.
  -->
- <definitions targetNamespace="http://calculator.me.org/" name="CalculatorWSService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://calculator.me.org/" schemaLocation="http://localhost:8080/CalculatorWSApplication/CalculatorWSService?xsd=1"/>
      </xsd:schema>
    </types>
  - <message name="add">
      <part name="parameters" element="tns:add"/>
    </message>
  - <message name="addResponse">
      <part name="parameters" element="tns:addResponse"/>
    </message>
  - <portType name="CalculatorWS">
    - <operation name="add">
        <input wsam:Action="http://calculator.me.org/CalculatorWS/addRequest" message="tns:add"/>
        <output wsam:Action="http://calculator.me.org/CalculatorWS/addResponse" message="tns:addResponse"/>
      </operation>
    </portType>
  - <binding name="CalculatorWSPortBinding" type="tns:CalculatorWS">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="add">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
```

4. Exemple complet

Fichier WSDL (suite)

```
        − <output>
            <soap:body use="literal"/>
          </output>
        </operation>
      </binding>
    − <service name="CalculatorWSService">
      − <port name="CalculatorWSPort" binding="tns:CalculatorWSPortBinding">
          <soap:address location="http://localhost:8080/CalculatorWSApplication/CalculatorWSService"/>
        </port>
      </service>
  </definitions>
```

4. Exemple complet

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:add xmlns:ns2="http://calculator.me.org/">
            <i>2</i>
            <j>3</j>
        </ns2:add>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:addResponse xmlns:ns2="http://calculator.me.org/">
            <return>5</return>
        </ns2:addResponse>
    </S:Body>
</S:Envelope>
```

DÉVELOPPEMENT DE SERVICES WEB SOAP À L'AIDE DE JAX-WS

Plan

1. Introduction
2. API JAX-WS : Généralités
3. Modèle JAX-WS et Extrémités d'un service web
4. Générer les artefacts serveur et les artefacts nécessaires au consommateur
5. Appel d'un service Web
6. Annotations
7. JAXB : Java Architecture for XML Binding

1. Introduction

- La majorité des langages de programmation supportent le développement de services Web
  - Java, PHP, C#, C++, …

- En JAVA, il existe différents frameworks et API de développement de Services Web:
  - **JAX-WS** (Java API for XML-Based Web Services): Spécification Oracle (https://jax-ws.dev.java.net/). **Metro** est une implémentation de référence open-source des spécifications des services web Java. Bien que la pile Metro soit produite par la communauté GlassFish, elle peut être utilisée en dehors de celui-ci, dans un environnement Java EE ou Java SE.
  - **Apache CXF** (http://cxf.apache.org/)
  - **JBoss** WS (http://www.jboss.org/jbossws)
  - **XFire** Codehaus (xfire.codehaus.org)

2. API JAX-WS : Généralités

- Le document WSDL étant le contrat qui lie le consommateur et le service, il peut servir à écrire le code Java pour ces deux parties : c'est ce que l'on appelle la *méthode descendante,* également *méthode par contrat* car elle part du contrat (le WSDL) en définissant les opérations, les messages, etc. Lorsque le consommateur et le fournisseur sont d'accord sur le contrat, on peut implémenter les classes Java en fonction de celui-ci. Les implémentations de JAX-WS fournissent quelques outils permettant de produire des classes à partir d'un document WSDL.

  Dans l'autre approche, la *méthode ascendante,* la classe de l'implémentation existe déjà et il suffit de créer le WSDL. Là encore, les implémentations disposent d'outils pour effectuer cette opération. Dans les deux cas, le code doit parfois être ajusté pour correspondre au WSDL ou *vice versa.*

- Les services web suivent le paradigme de facilité de développement de Java EE et n'obligent pas à écrire le moindre code WSDL ou SOAP. Un service web est simplement un **POJO* annoté** qui doit être déployé dans un **conteneur de service web**.

* : POJO (Plain Old Java Object que l'on peut traduire en français par *bon vieil objet Java*), c'est un objet qui n'a besoin d'aucun héritage ni d'aucune interface particulière, si ce n'est simplement les opérations et attributs nécessaires à son fonctionnement.

2. API JAX-WS : Généralités

Exemple:

- Dans l'exemple ci-dessous, une classe Java utilise des annotations JAX-WS qui vont permettre par la suite de générer le document WSDL. Le document WSDL est auto-généré par le serveur d'application au moment du déploiement :

```
@WebService
public class CardValidator {
        @WebMethod
        public boolean validate(CreditCard creditCard) {
            ...
        }
            ...
}
```

3. Modèle JAX-WS

- Comme la plupart des composants de Java EE, les services web s'appuient sur le paradigme de la configuration par exception. Seule l'annotation @WebService est nécessaire pour transformer un POJO en service web, mais cette classe doit respecter les règles suivantes :
  - Elle doit être annotée par @javax.jws.WebService ou son équivalent XML dans un descripteur de déploiement.
  - Pour transformer un service web en EJB, la classe doit être annotée par @javax.ejb.Stateless.
  - Elle doit être publique et ne doit pas être finale ni abstraite.
  - Elle doit posséder un constructeur par défaut public.
  - Elle ne doit pas définir la méthode finalize().
  - Un service doit être un objet sans état et ne pas mémoriser l'état spécifique d'un client entre les appels de méthode.
- La spécification énonce que, tant qu'il respecte ces règles, un POJO peut être utilisé pour implémenter un service web déployé dans le conteneur de servlet – il est alors souvent désigné sous **le terme d'extrémité de servlet (Servlet end point).** Un **bean de session sans état** peut également servir à implémenter un service web qui sera déployé dans un conteneur EJB – il est alors appelé **extrémité EJB (EJB end point)**.

- Il est aussi possible de développer des Web Services en dehors d'un serveur d'application en mode autonome (Déployer directement l'application via Java SE).

3.1 Extrémités d'un service web

- Les codes d'un service web POJO et d'un service web EJB (voir exemples ci-dessous) sont peu différents : la seule différence est que le second a une annotation @Stateless supplémentaire. En outre, l'assemblage est différent : un service web POJO est assemblé dans un module web (un fichier war) et est appelé extrémité de servlet, tandis qu'un service web EJB est assemblé dans un fichier jar et est appelé extrémité EJB. Le premier est déployé dans un conteneur de servlet, le second, dans un conteneur EJB.

  Ces deux extrémités ont un comportement quasiment identique, mais les extrémités EJB ont quelques avantages supplémentaires car, le service web étant en ce cas également un EJB, il bénéficie automatiquement des transactions et de la sécurité qui sont gérées par le conteneur. En outre, il peut utiliser des intercepteurs, ce qui n'est pas possible avec les extrémités de servlets. Le code métier peut être exposé simultanément comme un service web et comme un EJB, ce qui signifie qu'il peut être exposé à la fois v*ia SOAP et RMI en ajoutant une interface distante.*

3.1 Extrémités d'un service web

Exemple:

Le service web CardValidator : extrémité Servlet

```
import javax.jws.WebService;
@WebService
public class CardValidator {
    public boolean validate(CreditCard creditCard) {
        //code de la méthode
    }
}
```

Le service web CardValidator : extrémité EJB

```
import javax.jws.WebService;
Import javax.ejb.Stateless;
@WebService
@Stateless
public class CardValidator {
    public boolean validate(CreditCard creditCard) {
        //code de la méthode
    }
}
```

3.2 Annotations

- JAX-WS fournit un ensemble d'annotations qui simplifient le développement des services Web SOAP. Si les valeurs par défaut vous conviennent, vous pouvez même limiter les annotations à la seule @WebService. Les différentes annotations seront étudiées ultérieurement dans ce chapitre.

3.3 Cycle de vie et méthodes de rappel

- Le cycle de vie des services web (figure ci-dessous) ressemble à celui des beans sans état et des MDB. Comme avec tous les composants qui ne mémorisent pas l'état, soit ils n'existent pas, soit ils sont prêts à traiter une requête. Ce cycle de vie est géré par le conteneur.

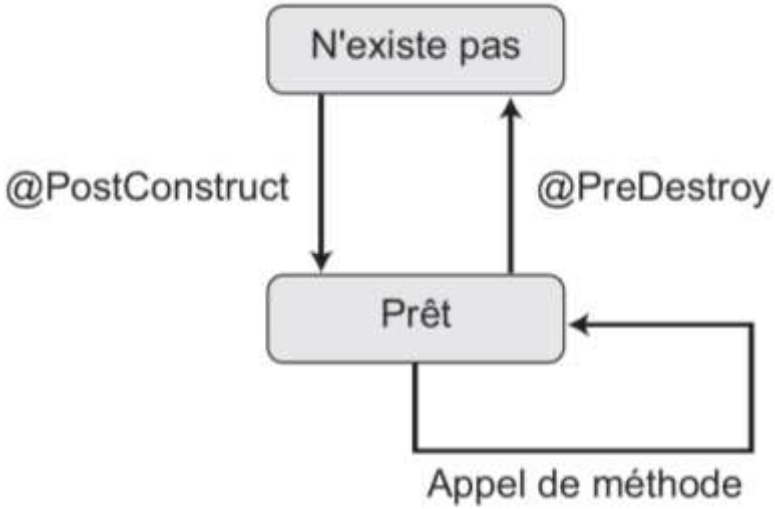


Figure 1. Cycle de vie des services web et méthodes de rappel (callback methods)

3.3 Cycle de vie et méthodes de rappel

- Comme elles s'exécutent dans un conteneur, les extrémités de servlet et EJB autorisent l'injection des dépendances et les méthodes de rappel (**callback methods**) du cycle de vie. C'est-à-dire que le conteneur appellera la méthode de rappel @PostConstruct (si elle existe) lorsqu'il crée une instance d'un service web et la méthode de rappel @PreDestroy (si elle existe) lorsqu'il la détruit.

  Une différence entre les extrémités de servlet et les extrémités EJB est que ces dernières peuvent utiliser des intercepteurs, qui sont l'implémentation Java EE du concept de programmation orientée aspects (POA).

Exemple:

```
@WebService()
public class CardValidator  {
        public  boolean validate(CreditCard creditCard){
        ...
        }
        //callback methods
        @PostConstruct
        void beginValidation() {
            ...
        }
        @PreDestroy
        void endValidation() {
            ...
        }
}
```

3.4 Contexte d'un service web

- Un service a un contexte d'environnement auquel il peut accéder en injectant une référence à javax.xml.ws.WebServiceContext au moyen d'une annotation @Resource. Dans ce contexte, le service peut obtenir des informations d'exécution comme la classe qui implémente l'extrémité, le contexte du message et des informations concernant la sécurité relative à la requête qui est traitée.

```
@Resource
private WebServiceContext context;
```

Le tableau ci-dessous énumère les méthodes définies dans l'interface javax.xml.ws.WebServiceContext.

Tableau 1. Méthodes de l'interface javax.xml.ws.WebServiceContext.

Méthode	*Description*
`getMessageContext`	Renvoie le `MessageContext` pour la requête en cours de traitement au moment de l'appel. Permet d'accéder aux en-têtes du message SOAP, etc.
`getUserPrincipal`	Renvoie le principal qui identifie l'émetteur de la requête en cours de traitement.
`isUserInRole`	Teste si l'auteur authentifié appartient au rôle logique indiqué.
`getEndpointReference`	Renvoie l'`EndpointReference` associé à cette extrémité.

4. Générer les artefacts serveurs et les artefacts nécessaires au consommateur

- Un artefact est composé de l'ensemble des documents nécessaires à un service web. Nous pouvons citer par exemple le document WSDL ou encore les classes Java qui formeront les messages SOAP d'échanges XML.

- JDK fournit des outils pour manipuler les Web Services du côte serveur et du côté consommateur.
  - **wsgen:**
    Lit une classe extrémité de service web et produit des artefacts (notamment le fichier WSDL). L'utilisation de cet outil n'est pas obligatoire puisque cette génération est implicite lors de l'exécution.
    Exemple: la commande ci-dessous génère le docment wsdl

    ```
    wsgen -cp . CardValidator –keep –wsdl
    ```

  - **wsimport :**
    Prend en entrée un fichier WSDL et produit les artefacts JAX-WS nécessaire au consommateur. En effet, Le fichier WSDL établit le contrat entre le consommateur et le service, l'outil wsimport est un outil de conversion WSDL vers Java, il produit des classes et des interfaces Java à partir du code WSDL – ces interfaces sont appelées SEI (*service endpoint interfaces)* car ce sont des représentations Java d'une extrémité de service web (servlet ou EJB). Cette **SEI agit comme un proxy** (Java relais ou stubs) qui route l'appel Java local vers le service web distant via HTTP ou d'autres protocoles de transport.

Exemple: La commande ci-dessous génère les artefacts côté client. L'utilitaire wsimport prend en paramètres l'**URL** du **WSDL** du service web :

```
wsimport http://localhost:8080/servicewebtep1/CardValidator<b>Service</b>?WSDL
```

Exemple

- On considère le service Web « CardValidator » qui propose une méthode appelée **validate** permettant de valider une carte de crédit dont les données sont envoyées sous forme d'un objet CreditCard par le consommateur (une autre application) sur le réseau.

```
import javax.jws.WebService;
@WebService
public class CardValidator {

   public boolean validate(CreditCard creditCard) {
      //votre code de validation
    }
}
```

Si les valeurs par défaut vous conviennent, vous pouvez même limiter les annotations à la seule @WebService.

Exemple

- Classe CreditCard.java

```
4  import javax.xml.bind.annotation.XmlRootElement;
5
6  @XmlRootElement
7  public class CreditCard {
8      private String number;
9      private String expiryDate;
10     private Integer controlNumber;
11     private String type;
12
13     public CreditCard() {
14     }
15
16     public CreditCard(String number, String expiryDate, Integer controlNumber, String type) {
17         this.number = number;
18         this.expiryDate = expiryDate;
19         this.controlNumber = controlNumber;
20         this.type = type;
21     }
22
23     public String getNumber() {
24         return number;
25     }
26
27     public void setNumber(String number) {
28         this.number = number;
29     }
```

Exemple

- A partir de la classe **CreditCard**, l'utilitaire **wsgen** génère les éléments suivants :
  - Le document **WSDL** décrivant le service web et son schéma XSD.
  - Ce fichier fait ensuite référence à la définition des types de données et porte l'extension <.xsd>. Ce fichier est également auto-généré par le serveur d'applications.
  - Les différentes classes qui vont permettre les échanges de messages XML suivant le protocole SOAP.

Contrat WSDL:

CardValidatorService.wsdl

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

    <types>
        <xsd:schema>
            <xsd:import namespace="http://projet.org/"
                schemaLocation="http://localhost:8080/projet/CardValidatorService?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="validate">
        <part name="parameters" element="tns:validate"/>
    </message>
    <message name="validateResponse">
        <part name="parameters" element="tns:validateResponse"/>
    </message>
    <portType name="CardValidator">
        <operation name="validate">
            <input message="tns:validate"/>
            <output message="tns:validateResponse"/>
        </operation>
    </portType>
    <binding name="CardValidatorPortBinding"
        type="tns:CardValidator">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <operation name="validate">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="CardValidatorService">
        <port name="CardValidatorPort"
            binding="tns:CardValidatorPortBinding">
            <soap:address location ="http://localhost:8080/projet/CardValidatorService"/>
        </port>
    </service>
</definitions>
```

CardValidatorService.xsd_1.xsd

```
<xs:schema version="1.0"
  targetNamespace="http://projet.org/"
  xmlns:tns="http://projet.org/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="creditCard" type="tns:creditCard"/>
  <xs:element name="validate" type="tns:validate"/>
  <xs:element name="validateResponse" type="tns:validateResponse"/>
  <xs:complexType name="validate">
    <xs:sequence>
      <xs:element name="arg0" type="tns:creditCard" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="creditCard">
    <xs:sequence>
      <xs:element name="controlNumber" type="xs:int" minOccurs="0"/>
      <xs:element name="expiryDate" type="xs:string" minOccurs="0"/>
      <xs:element name="number" type="xs:string" minOccurs="0"/>
      <xs:element name="type" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="validateResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

5. Appel d'un service Web

- L'appel d'un service web ressemble *à* l'appel d'un objet distribué avec RMI (Remote Method Invocation) : comme ce dernier, JAX-WS permet au programmeur d'utiliser un appel de méthode local pour invoquer un service se trouvant sur un autre hôte. La différence est qu'un service web sur l'hôte distant peut être écrit dans un autre langage de programmation. Nous rappelons que le fichier WSDL établit le contrat entre le consommateur et le service, et l'implémentation de JAX-WS fournit un outil de conversion WSDL vers Java (**wsimport**) qui produit des classes et des interfaces Java à partir du code WSDL – ces interfaces sont appelées SEI (*service endpoint interfaces)* car ce sont des représentations Java d'une extrémité de service web (servlet ou EJB). Cette SEI agit comme un **proxy** qui route l'appel Java local vers le service web distant via HTTP ou d'autres protocoles de transport. Lorsqu'une **méthode de ce proxy** est appelée (voir un exemple sur la figure ci-dessous), elle convertit ses paramètres en message SOAP (la requête) et l'envoie à l'extrémité du web service.

  Pour obtenir le résultat, la réponse SOAP est reconvertie en une instance du type renvoyé. Pour l'utiliser**, il n'est pas nécessaire de connaitre le fonctionnement interne du proxy ni d'étudier son code**. Avant de compiler le consommateur client, il faut produire la SEI afin d'obtenir la **classe proxy** pour l'appeler dans le code.

5. Appel d'un service Web

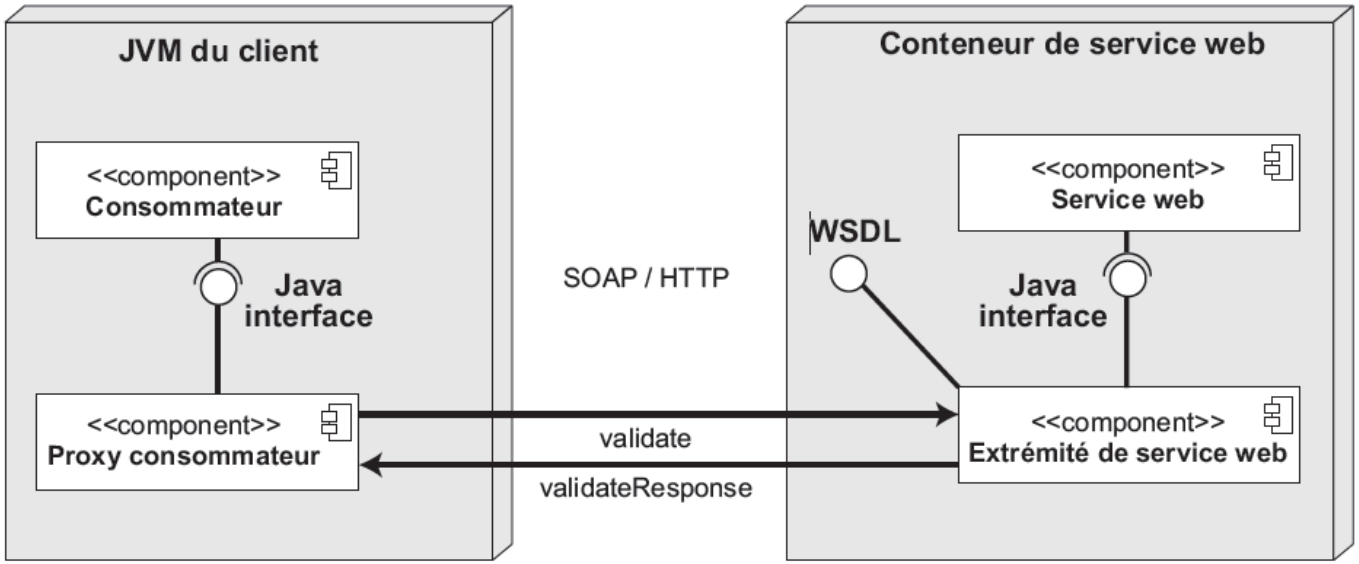


Figure 2. Un consommateur appelle un service web via un proxy.

5. Appel d'un service Web

- Le consommateur peut obtenir **une instance du proxy par injection** ou **en la créant par programme**. On injecte un client de service web à l'aide de l'annotation @javax. xml.ws.WebServiceRef. Lorsqu'elle est appliquée à un attribut (ou à une méthode getter), le conteneur injecte une instance du service web lorsque l'application est initialisée. Le code est de la forme suivante :

```
@WebServiceRef
private CardValidatorService   cardValidatorService;
// ...
CardValidator      cardValidator = cardValidatorService.getCardValidatorPort();
cardValidator.validate(creditCard);
```

Commentaire:

à partir du WSDL, ***wsimport*** *génère plusieurs éléments. Parmi eux, nous trouvons les classes:*

- ***CardValidatorService*** *: c'est la classe principale qui est utilisée dans le code de l'application cliente. Celle-ci possède la méthode* ***getCardValidatorPort****() qui retourne un objet de l'interface* ***CardValidator*** *possédant la même signature que le service web.*
  - *la signature de cette méthode est toujours de la forme suivante :*

getNomDeLaClasseDuServiceWebPort()

- *La classe fabrique* ***ObjectFactory****: elle est utilisée pour créer les classes générant les messages XML suivant le protocole SOAP.*

5. Appel d'un service Web

- La classe CardValidatorService est la SEI, pas le service web lui-même. Vous devez ensuite obtenir la classe proxy CardValidator pour invoquer localement les méthodes métiers. On appelle localement la méthode validate() du proxy, qui, à son tour, invoquera le service web distant, créera la requête SOAP, sérialisera les messages, etc.

  Pour que cette injection fonctionne, ce code doit s'exécuter dans un **conteneur** (de **servlet**, d'**EJB** ou de **client d'application**) : dans le cas contraire, on ne peut pas utiliser l'annotation @WebServiceRef et il faut alors passer par la **programmation** comme le montre le code ci-dessous.

  Les classes produites par l'outil wsimport peuvent être directement utilisées de la façon suivante :

```
CardValidatorService    cardValidatorService = new CardValidatorService();
CardValidator cardValidator = cardValidatorService.getCardValidatorPort();
cardValidator.validate(creditCard);
```

  Ce code crée une instance de CardValidatorService à l'aide du mot-clé new ; le reste est identique.

5. Appel d'un service Web

Résumé:

- L'écriture et la consommation d'un service web comporte quatre phases :
  1. Développement du service web proprement dit (la clase du Web service).
  2. Génération des artefacts côté serveur.
  3. Génération des artefacts côté client (artefacts du consommateur).
  4. Appel du service web chez le client (consommer le service Web).

- Au niveau codage, que ce soit sur le serveur, ou que ce soit chez le client, le codage d'un service web est extrêmement simple. Toute l'architecture de bas niveau est automatiquement construite par les différents utilitaires.

6. Annotations JAX-WS

- Au niveau du service, les systèmes sont définis en termes de messages XML, d'opérations WSDL et de messages SOAP. Cependant, au niveau Java, les applications sont décrites en termes d'objets, d'interfaces et de méthodes. Il est donc nécessaire d'effectuer une traduction des objets Java vers les opérations WSDL. L'environnement d'exécution de JAXB (Java Architecture for XML Binding) utilise les annotations pour savoir comment sérialiser/désérialiser une classe vers/à partir de XML. Généralement, ces annotations sont cachées au développeur du service web. De même, JWS se sert d'annotations pour déterminer comment sérialiser un appel de méthode vers un message de requête SOAP et comment désérialiser une réponse SOAP vers une instance du type du résultat de la méthode.
- La spécification WS-Metadata (JSR 181) définit deux sortes d'annotations :
  - **Les annotations de traduction WSDL**. Elles appartiennent au paquetage javax.jws et permettent de modifier les associations entre WSDL et Java. Les annotations @WebMethod, @WebResult, @WebParam et @OneWay sont utilisées sur le service web pour adapter la signature des méthodes exposées.
  - **Les annotations de traduction SOAP.** Elles appartiennent au paquetage javax.jws.soap et permettent d'adapter la liaison SOAP (@SOAPBinding et @SOAPMessageHandler).
- Comme toutes les autres spécifications de Java EE, ces annotations peuvent être redéfinies par un descripteur de déploiement XML facultatif (webservices.xml). Les sections qui suivent décrivent plus précisément chacune d'elles.
- Remarque: JAX-WS repose sur l'utilisation massive d'annotations pour la configuration d'un Web Service, cependant, seule l'utilisation de l'annotation @WebService est nécessaire si les valeurs par défaut vous conviennent.

6.1 Annotation @javax.jws.WebService

- L'annotation @javax.jws.WebService marque une classe ou une interface Java comme étant un service web. Lorsqu'elle est utilisée directement sur la classe, le processeur d'annotations du conteneur produira l'interface – les deux extraits de code qui suivent sont donc équivalents.

  Voici l'annotation sur une classe :

```
@WebService
public class CardValidator {
        …..
}
```

  Voici l'annotation sur une interface implémentée par une classe :

```
@WebService
public  interface CardValidatorEndPoint {
        ….
}
@WebService(endpointInterface = " CardValidatorEndPoint ")
public class CardValidator  implements CardValidatorEndPoint {
        …
}
```

6.1 Annotation @javax.jws.WebService

- Cette annotation possède un certain nombre d'attributs (voir la déclaration ci-dessous) permettant de personnaliser le nom du service web dans le fichier WSDL (éléments <wsdl:portType> ou <wsdl:service>) et son espace de noms, ainsi que l'emplacement du fichier WSDL lui-même (attribut wsdlLocation).

API de @WebService:

```
@Retention(RUNTIME)
@Target(TYPE)
public @interface WebService {
        String name() default "";
        String targetNamespace() default "";
        String serviceName() default "";
        String portName() default "";
        String wsdlLocation() default "";
        String endpointInterface() default "";
}
```

- *@Retention* est une méta-annotation standard définie dans le package java.lang.annotation. ELLe permet d'influer sur la "durée de vie« d'une annotation. La valeur RUNTIME passée en paramètre signifie que l'annotation sera conservée dans les fichiers .class, (en plus des fichiers sources) et sera accessible à la machine virtuelle. L'instrospection sera possible.
- la méta-annotation *@Target* précise à quelles entités il sera possible d'appliquer l'annotation.

Lorsque l'on utilise @WebService, toutes les **méthodes publiques du service web** qui n'utilisent pas l'annotation @WebMethod sont exposées.

6.1 Annotation @javax.jws.WebService

- Attributs de l'annotation @WebService
  - **name** : définit le nom du **service web**. Ce nom se trouve alors dans le fichier **WSDL** dans l'attribut **name** de la balise **<portType>**. La valeur par défaut est le nom de la classe d'imlémentation. Dans notre exemple, il s'agit de CardValidator.
  - **wsdlLocation** : définit l'**URL**, qu'elle soit relative ou absolue, d'un fichier **WSDL** existant. Par défaut, ce fichier est **auto-généré** lors du **déploiement**.
  - **endpointInterface** : définit le nom complet de l'interface **"endpoint"** définissant les **méthodes** du **service web**. Cela permet au développeur de séparer le **"contrat" (l'interface)** de l'implémentation. L'implémentation de cette interface par le service n'est pas requise. L'intérêt de cette solution est également d'affecter des annotations de mapping Java vers le WSDL dans l'interface et non dans la classe d'implémentation.

6.2 Annotation @WebMethod

- Par défaut, toutes les méthodes publiques d'un service web (nous parlons d'opération dans le protocole **SOAP**) sont exposées dans le WSDL et utilisent toutes les règles d'association par défaut.

  L'annotation @javax.jws.WebMethod permet de personnaliser certaines de ces associations de méthodes. Son API est assez simple et permet de renommer une méthode ou de l'exclure du WSDL.

- Quelques attributs de l'annotation @WebMethod
  - **String operationName** : définit la valeur de l'attribut **name** de la balise **<operation>** dans le fichier **WSDL**. Celui-ci représente le nom de l'opération.
  - **Boolean exclude** : marque une méthode comme étant non disponible par le **service web**. Ce paramètre est utilisé lorsque nous souhaitons masquer une méthode héritée, par exemple.

6.2 Annotation @WebMethod

- **Exemple:**

Le code ci-dessous montre comment le service web CardValidator peut renommer sa première méthode en ValidateCreditCard et exclure la seconde.

```
@WebService
public class CardValidator {
    @WebMethod(operationName = "ValidateCreditCard")
    public boolean validate(CreditCard creditCard) {
        // Logique métier
    }
    @WebMethod(exclude = true)
    public void validate(String ccNumber) {
        // Logique métier
    }
}
```

6.3 Annotation @javax.jws.WebResult

- L'annotation @javax.jws.WebResult fonctionne en relation avec @WebMethod pour contrôler le nom de la valeur renvoyée par le message dans le WSDL. Elle décrit la relation entre le paramètre de sortie d'une méthode et un message part d'une opération.

- Attributs de l'annotation
  - **boolean header** : précise si le paramètre de sortie doit être transmis dans l'en-tête du message (true) ou dans le corps (false)
  - **String name** : nom du paramètre de sortie
  - **String partName** : le nom du wsdl:part représentant ce paramètre de sortie
  - **String targetNamespace** : l'espace de nommage de ce paramètre de sortie

- Exemple: Renommage du résultat de la méthode

Dans le code ci-dessous, le résultat de la méthode validate() est renommé en IsValid.

```
@WebService
public class CardValidator {
        @WebMethod
        @WebResult(name = "IsValid")
        public boolean validate(CreditCard creditCard) {
            // Logique métier
        }
}
```

6.3 Annotation @javax.jws.WebParam

- L'annotation @javax.jws.WebParam permet de personnaliser la génération du WSDL qui concerne les paramètres de la méthode. Elle décrit la relation entre un paramètre d'entrée d'une méthode et un message part d'une opération.

- L'API de cette annotation permet de modifier le nom du paramètre dans le WSDL, l'espace de noms et le mode de passage des paramètres : IN, OUT ou INOUT.

API de @WebParam

```
@Retention(RUNTIME) @Target(PARAMETER)
public @interface WebParam {
        String name() default "";
        public enum Mode {IN, OUT, INOUT};
        String targetNamespace() default "";
        boolean header() default false;
        String partName() default "";
}
```

6.3 Annotation @javax.jws.WebParam

- Attributs :
  - **String name** : définit le nom du paramètre qui sera utilisé dans le fichier **WSDL**. Par défaut, le nom de l'argument (du code Java) est utilisé.
  - **String partName** : le nom du **wsdl:part** représentant ce paramètre
  - **Boolean header** :précise si le paramètre doit être transmis dans l'en-tête du message (true) ou dans le corps (false). La valeur par défaut étant **false.**
  - **WebParam.Mode mode** : indique si le paramètre est utilisé en **entrée**, en **sortie** ou les **deux**. Pour spécifier ce type, il faut utiliser l'énumération **WebParam.MODE** (Valeurs : IN, OUT, BOTH).
  - **String targetNamespace** : définit le **namespace** à utiliser. Par défaut, nous utilisons un namespace vide. Cet attribut est à utiliser si le paramètre est mappé dans l'en-tête.
- **Exemple : Renommage du paramètre de la méthode**

```
@WebService
public class CardValidator {
        @WebMethod
        public boolean validate ( @WebParam(name ="Credit-Card") CreditCard cc){
           // Logique métier
        }
}
```

6.4 Annotation @javax.jws.OneWay

- L'annotation @OneWay peut être utilisée avec les méthodes qui ne renvoient aucun résultat, comme celles de type void. Cette annotation n'a aucun élément et peut être considérée comme une interface de marquage informant le conteneur que l'appel de cette méthode peut être optimisé (en utilisant un appel asynchrone, par exemple) puisqu'il n'y a pas de valeur de retour.

Exemple:

```
package photos;
import javax.jws.*;

@WebService()
public class StockerPhotos {
        @WebMethod(operationName = "stocker")
        @OneWay
        public void stocker(@WebParam(name = "nomFichier") String nomFichier,
            @WebParam(name = "octets") Byte [] octets) {
            ...
        }
        ...
}
```

7. JAXB : Java Architecture for XML Binding

- XML est utilisé pour échanger les données et définir les services web via WSDL et les enveloppes SOAP. Pourtant, comme nous avons vu dans les codes précédents, un consommateur invoquait un service web sans qu'il n'y ait trace de XML car ce consommateur ne manipulait que des interfaces et des objets Java distants qui, à leur tour, gèrent tous les détails XML et les connexions réseau. On manipule des classes Java à un endroit de la chaîne et des documents XML à un autre. En réalité, JAX-WS s'appuie sur l'API **JAXB** pour faciliter cette correspondance bidirectionnelle entre document XML et objets Java.

- L'API JAXB, définie dans le paquetage javax.xml.bind, fournit un ensemble d'interfaces et de classes permettant de produire des documents XML et des classes Java – en d'autres termes, elle relie les deux modèles. Le framework d'exécution de JAXB implémente les opérations de sérialisation et de désérialisation. La sérialisation (ou marshalling) consiste à convertir les instances des classes annotées par JAXB en représentations XML. Inversement, la désérialisation (unmarshalling) consiste à convertir une représentation XML en arborescence d'objets.

- Les données XML sérialisées peuvent être validées par un schéma XML – JAXB peut produire automatiquement ce schéma à partir d'un ensemble de classes et vice versa. La Figure ci-dessous montre les interactions possibles entre une application et JAXB.

7. JAXB : Java Architecture for XML Binding

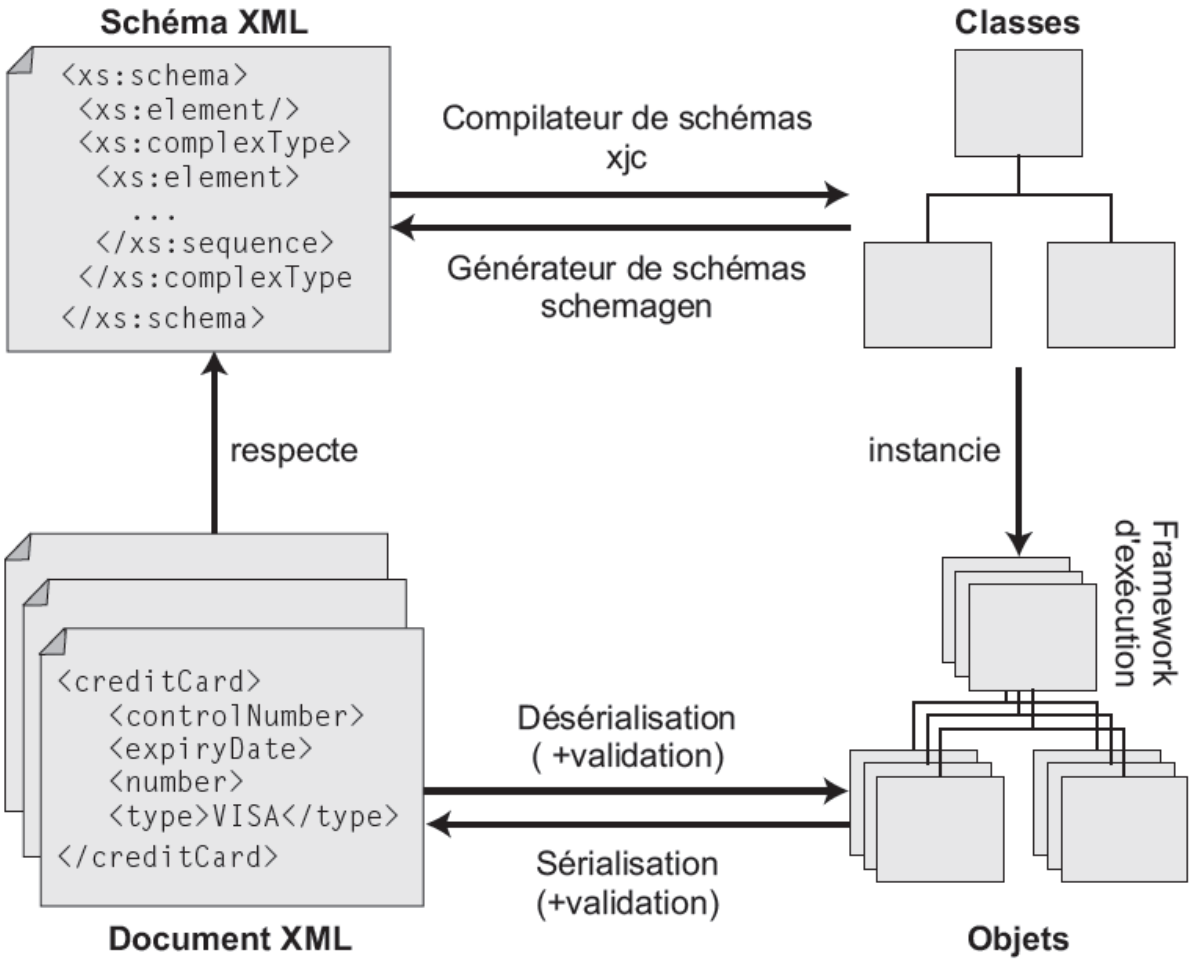


Figure 3. Architecture JAXB.

7. JAXB : Java Architecture for XML Binding

Exemple:

Considérons la classe CreditCard annotée par l'annotation @javax.xml.bind.annotation.XmlRootElement de l'API JAXB.

```
@XmlRootElement
public class CreditCard {
        private String number;
        private String expiryDate;
        private Integer controlNumber;
        private String type;
          // Constructeurs, getters, setters
}
```

- Grâce à l'annotation @XmlRootElement et à un mécanisme de sérialisation, JAXB est capable de créer une représentation XML d'une instance de CreditCard, comme celle qui est présentée ci-dessous.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<creditCard>
        <controlNumber>6398</controlNumber>
        <expiryDate>12/09</expiryDate>
        <number>1234</number>
        <type>Visa</type>
</creditCard>
```

7. JAXB : Java Architecture for XML Binding

- JAXB peut aussi produire automatiquement le schéma qui validerait automatiquement la structure XML de la carte de crédit afin de garantir qu'elle est correcte et que les types des données conviennent. Le code ci-dessous montre la définition du schéma XML (XSD) de la classe CreditCard.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0“
        xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="creditCard type="creditCard"/>

    <xs:complexType name="creditCard">
        <xs:sequence>
            <xs:element name="controlNumber" type="xs:int"  minOccurs="0"/>
            <xs:element name="expiryDate" type="xs:string"  minOccurs="0"/>
            <xs:element name="number" type="xs:string" minOccurs="0"/>
            <xs:element name="type" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Ce schéma est constitué d'éléments simples (controlNumber, expiryDate, etc.) et d'un type complexe (creditCard).

Bibliographie and Webographie

- Architectures réparties en Java: RMI, CORBA, JMS, sockets, SOAP, services web. *Annick Fron*. Dunod, 2007. ISBN: 2100511416, 9782100511419
- Informatique répartie, Développement d'applications. Les services web. Last access: September 2015.

  http://programmation-java.1sur1.com/Java/Tutoriels/J2EE/PDF/ServicesWeb.pdf
- Java API for XML-Based Web Services (JAX-WS) 2.3. *Jitendra Kotamraju and Lukas Jungmann*. Maintenance Release, July 11, 2017. Oracle Corporation.
- Java EE6 et GlassFish 3. *Antonio Goncalves.* Editeur: Pearson, 2010. ISBN: 978-2-7440-4157-0
- [Mickaël BARON, 2010]. Cours SOA –Services Web.

  https://fr.scribd.com/document/94133706/Intro-So-A

  https://fr.scribd.com/document/49423425/jaxws
- Services Web avec J2EE et .NET, Conception et implémentations. *Libero Maesano, Christian Bernard, Xavier Le Galles.* Edition: Eyrolles, 2003. ISBN : 2-212-11067-7
- SOA , microservices et API management : Le guide de l'architecte des SI agiles. *Fournier-Morel Xavier, Grojean Pascal, Plouin Guillaume*. Editeur: Dunod. Publication: 2017. ISBN: 978-2-10-076730-4.
- [XEBIA BLOG, 2009]. http://blog.xebia.fr/category/soa
- XML Cours et exercices. *Alexandre Brillant.* Edition: Eyrolles, 2007. ISBN : 978-2-212-12151-3