SOA Resumé

Service-Oriented Architecture (SOA) & Web Services

Notion de Service

Un service est un composant logiciel qui exécute une action pour le compte d’un client. Il traduit le niveau logique d’accès aux traitements, plutôt que le niveau physique d’implémentation (EJB, Servlet...). On distingue cinq types de services :

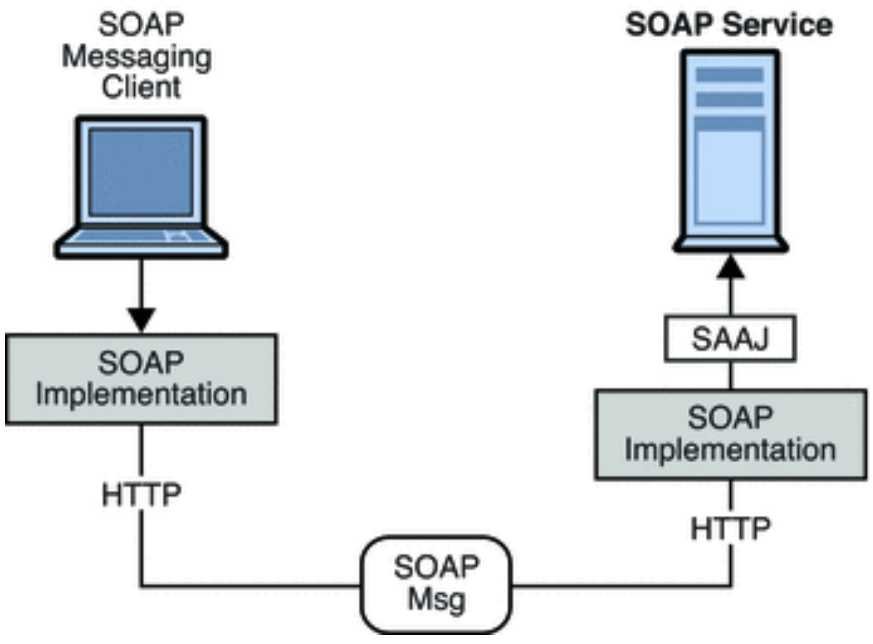
1. **Applicatif**: Traduit la logique applicative d’une application, exprimée par les uses cases ou les processus métier.
2. **CRUD**: Service élémentaire permettant de créer, rechercher, lire, mettre à jour ou exporter vers un format.
3. **Transverse (Infrastructure)**: Exécute un traitement métier spécifique (services de log, gestion du contexte utilisateur...).
4. **Host**: Permet aux applications distribuées d’utiliser une application MainFrame du Host de l’entreprise.
5. **Fonctionnel**: Exécute un traitement métier et peut être invoqué par différents services applicatifs. Il invoque des services CRUD et/ou Transverses pour pouvoir manipuler des objets métiers. Il peut aussi invoquer des services plus élémentaires ou externes pour assurer l’orchestration et peut servir à la gestion de la sécurité, des règles métiers, etc.

Architecture SOA

L’architecture orientée service constitue un style d’architecture basée sur le principe de séparation de l’activité métier en une série de services. Ces services peuvent être assemblés et liés entre eux selon le principe de couplage lâche pour exécuter l’application désirée. Ils sont définis à un niveau supérieur de la traditionnelle approche composants.

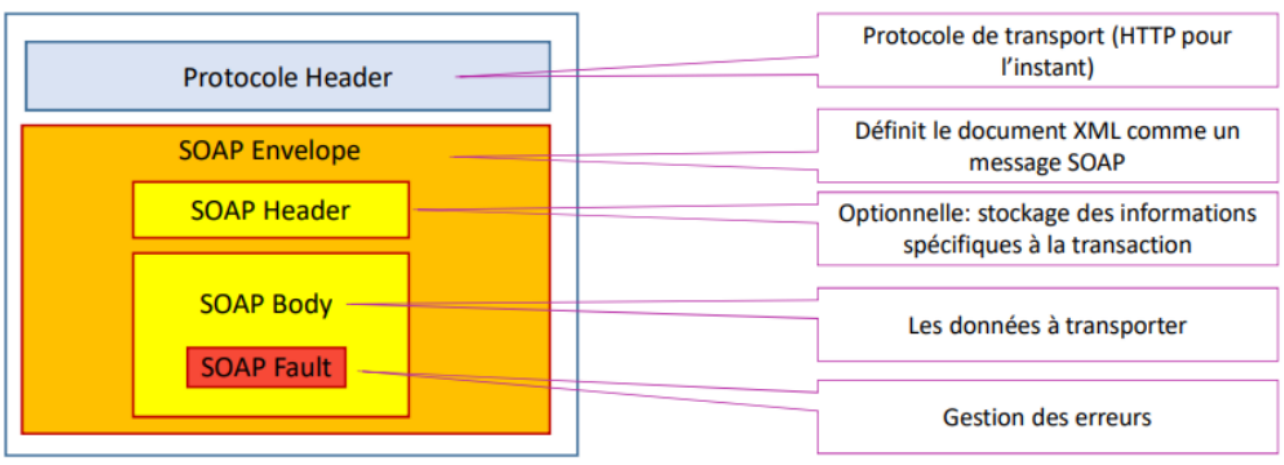
Web Service avec SOAP

Un service web est un composant logiciel identifié par une URI, dont les interfaces publiques sont définies et appelées en XML. Sa définition peut être découverte par d'autres systèmes logiciels. Les services Web peuvent interagir entre eux d'une manière prescrite par leurs définitions, en utilisant des messages XML portés par les protocoles Internet.



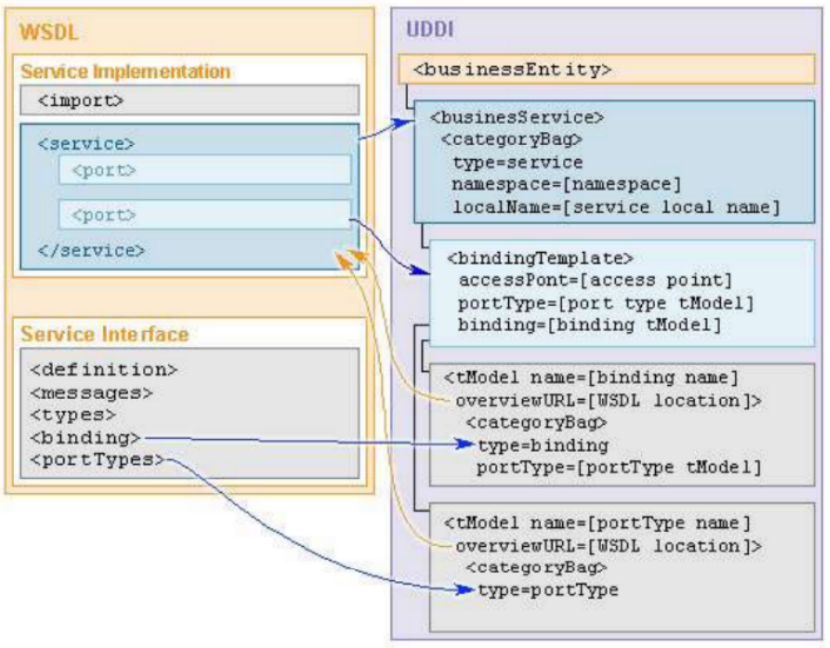
Le protocole SOAP

Le protocole SOAP (Simple Object Access Protocol) est un protocole standard basé sur XML pour échanger des messages structurés dans l'environnement distribué du World Wide Web. Il définit un format pour la structure du message SOAP et les règles de traitement des messages sur la base des formats XML.



Relation entre UDDI & WSDL

UDDI (Universal Description, Discovery, and Integration) est un registre de services qui permet aux fournisseurs de publier des descriptions de services et aux consommateurs de découvrir ces services via ces descriptions. WSDL (Web Services Description Language) est un langage XML utilisé pour décrire les services web et leurs fonctionnalités.



Etapes d'implementation de service web

1. **Implementation de service:**

```
import javax.jws.*;

@WebService( [endpointInterface = "<packageName.InterfaceName>"] ) // attribute if we add int
public class ServiceClass [implements InterfaceName] {
    [@Override]
    @WebMethod(operationName = "<label>")
    public String func(@WebParam(name = "p1") String p1) {
        return "hello :" + p1;
    }
}
```

2. **Publication du service**:

```
import javax.xml.ws.Endpoint;
import ws.ServiceClass;

public class Main {
    public static void main(String[] args) {
        try {
            Endpoint.publish("<http://localhost:8080/ws/ServiceClass>", new ServiceClass
());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. **Accès au WSDL**: Le WSDL du service web est accessible à l'URL : `http://localhost:8080/ws/ServiceClass?wsdl`
4. **Client du service**:

```
import ws.ServiceClass;
import ws.ServiceClassService;

public class Main {
    public static void main(String[] args) {
        try {
                ServiceClassServiceLocator service = new ServiceClassServiceLocator();
                        ServiceClass port = service.getServiceClass();
```

```
                        /* // ou selon version
                                ServiceClassService service = new ServiceClassService();
                                ServiceClass port = service.getSalutationPort();
                        */
            System.out.println(port.operationName("abdo"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Exemple d'implémentation de service web

Voici un exemple d'implémentation d'un service web en Java à l'aide de JAX-WS :

- **Définition de l'interface du service** (IMath.java):

```
import javax.jws.*;

@WebService
public interface IMath {
    @WebMethod(operationName = "addition")
    public int add(@WebParam(name = "a") int a, @WebParam(name = "b") int b);
}
```

- **Implémentation du service** (Math.java):

```
import javax.jws.WebService;

@WebService(endpointInterface = "ws.IMath")
public class Math implements IMath {
    @Override
    public int add(int a, int b) {
        return a + b;
    }
}
```

- **Publication du service** (Main.java):

```
import javax.xml.ws.Endpoint;
import ws.Math;

public class Main {
    public static void main(String[] args) {
        try {
            Endpoint.publish("<http://localhost:3000/ws/Math>", new Math());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **Client du service** (Main.java):

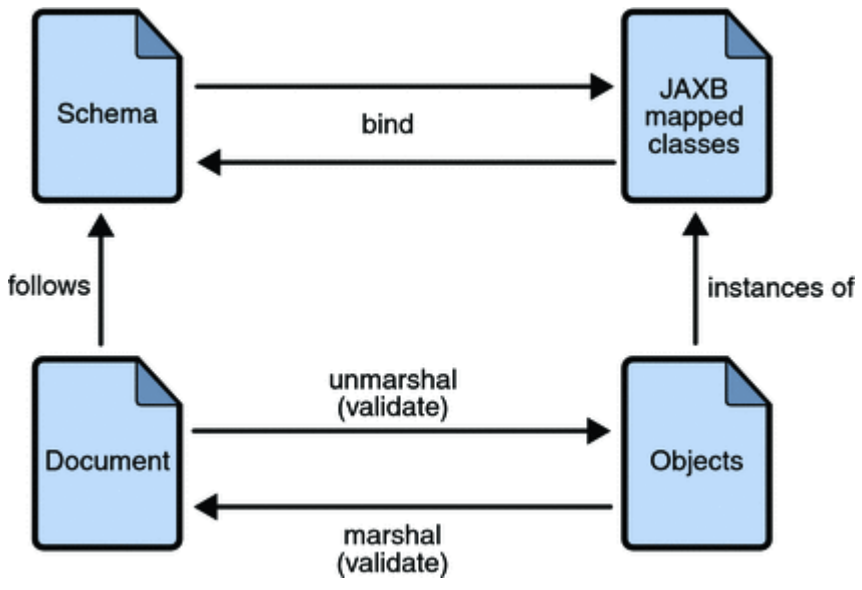
```
import ws.MathService;
import ws.IMath;

public class Main {
```

```
    public static void main(String[] args) {
        try {
            MathService service = new MathService();
            IMath port = service.getMathPort();
            System.out.println(port.add(7, 5));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **Accès au WSDL**: Le WSDL du service web est accessible à l'URL : `http://localhost:3000/ws/Math?wsdl`

Java API pour la Liaison XML (JAXB)

Java Architecture for XML Binding (JAXB) est une API Java qui permet aux développeurs de travailler avec des données XML dans les applications Java. JAXB offre des fonctionnalités pour sérialiser des objets Java en documents XML et désérialiser des documents XML en objets Java.



Annotations

JAXB fournit un ensemble d'annotations pour personnaliser la représentation XML des objets Java. Certaines annotations couramment utilisées incluent :

- `@XmlRootElement(name = "name")` : Spécifie le nom de l'élément racine pour la représentation XML d'une classe Java.
- `@XmlAccessorType(XmlAccessType.FIELD)` : Spécifie comment JAXB doit accéder aux champs (directement ou via les getters/setters) pour la sérialisation et la désérialisation.
- `@XmlElement(name = "userID")` : Spécifie le nom de l'élément XML pour un champ ou une propriété.
- `@XmlTransient` : Spécifie qu'un champ ou une propriété doit être ignoré lors du traitement XML.
- `@XmlAttribute(name = "userID")` : Spécifie qu'un champ ou une propriété doit être associé à un attribut XML.
- `@XmlElementWrapper(name = "wrapper")` : Enveloppe une propriété de collection dans un élément XML parent.
- `@XmlList` : Spécifie qu'un champ ou une propriété est une liste d'éléments séparés par des espaces.
- `@XmlSchema` : Spécifie des informations de schéma pour un package.
- `@XmlAccessorOrder(XmlAccessOrder.UNDEFINED)` : Spécifie l'ordre dans lequel les propriétés sont sérialisées.
- `@XmlType(propOrder={"id", "prenom", "nom"})` : Spécifie l'ordre dans lequel les propriétés doivent apparaître dans la représentation XML.

Exemple

```
import javax.xml.bind.annotation.*;

// Specifies the name of the root element for the XML representation of this class
```

```java
@XmlRootElement(name = "user")
// Specifies that JAXB should access fields directly for serialization/deserialization
@XmlAccessorType(XmlAccessType.FIELD)
// Specifies the order in which the properties are serialized
@XmlType(propOrder = {"id", "firstName", "lastName", "email", "roles", "notes"})
public class User {

    // Specifies that this field should be an attribute in the XML representation
    @XmlAttribute(name = "userID")
    private int id;

    // Specifies the name of the XML element for this field
    @XmlElement(name = "firstName")
    private String firstName;

    @XmlElement(name = "lastName")
    private String lastName;

    @XmlElement(name = "email")
    private String email;

    // This field should be ignored during XML processing
    @XmlTransient
    private String password;

    // Wraps the collection in a parent XML element
    @XmlElementWrapper(name = "roles")
    // Specifies that this field is a list of elements
    @XmlElement(name = "role")
    private Set<String> roles;

        // Specifies that this field is a space-separated list in the XML
    @XmlElement(name = "notes")
    @XmlList
    private List<String> notes;

    // Constructors, getters, and setters
}
```

Sérialisation / Marshalling

La sérialisation est le processus de conversion d'objets Java en documents XML. Avec JAXB, vous pouvez prendre une hiérarchie d'objets Java et la convertir en une représentation XML équivalente. Cela est utile, par exemple, lorsque vous devez persister des objets Java sur disque au format XML ou lorsque vous devez envoyer des objets Java sur un réseau en utilisant des protocoles basés sur XML comme SOAP.

Pour sérialiser un objet Java en un document XML en utilisant JAXB, suivez ces étapes :

1. Créez un `JAXBContext` pour la classe cible.
2. Créez un `Marshaller` à partir du `JAXBContext`.
3. Définissez des propriétés pour formater la sortie (facultatif).
4. Sérialisez l'objet Java vers un flux de sortie.

```java
public static String serialize(Person person) throws JAXBException {
    JAXBContext context = JAXBContext.newInstance(Person.class);
    Marshaller marshaller = context.createMarshaller();

    ByteArrayOutputStream baos = new ByteArrayOutputStream();
```

```
    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
    marshaller.marshal(person, baos);
    return baos.toString();
}
```

Désérialisation / Unmarshalling

La désérialisation est le processus de conversion de documents XML en objets Java. JAXB vous permet de prendre un document XML et de le convertir en une hiérarchie d'objets Java équivalente. Cela est utile, par exemple, lorsque vous devez lire des données XML dans votre application Java et y travailler en tant qu'objets Java.

Pour désérialiser un document XML en un objet Java en utilisant JAXB, suivez ces étapes :

1. Créez un `JAXBContext` pour la classe cible.
2. Créez un `Unmarshaller` à partir du `JAXBContext`.
3. Désérialisez le document XML à partir d'un `Reader` ou d'un `InputStream`.

```
public static Person deserialize(String XMLString) throws JAXBException {
    JAXBContext context = JAXBContext.newInstance(Person.class);
    Unmarshaller unmarshaller = context.createUnmarshaller();
    StringReader XMLReader = new StringReader(XMLString);
    return (Person) unmarshaller.unmarshal(XMLReader);
}
```

Exemple d'utilisation

Voici un exemple d'utilisation de JAXB pour sérialiser et désérialiser un objet `Person` :

```
public class Main {
    public static void main(String[] args) throws JAXBException {
        // Create a sample Person object
        Person person = new Person();
        person.setId(1);
        person.setName("John");
        person.setAge(30);
        person.setIsMarried(true);
        person.setSiblings(Arrays.asList(new Sibling("Jane"), new Sibling("Jack")));

        // Serialize the Person object to XML
        String serializedXml = serialize(person);
        System.out.println("Serialized XML:");
        System.out.println(serializedXml);

        // Deserialize the XML back to a Person object
        Person deserializedPerson = deserialize(serializedXml);
        System.out.println("\\\\nDeserialized Person:");
        System.out.println("Name: " + deserializedPerson.getName());
        System.out.println("Age: " + deserializedPerson.getAge());
        System.out.println("Is Married: " + deserializedPerson.getIsMarried());
        System.out.println("Siblings: " + deserializedPerson.getSiblings());
    }
}
```

Sortie

```
// Sortie de la sérialisation
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<human>
```

```
    <age>30</age>
    <isMarried>true</isMarried>
    <name>John</name>
    <siblings>Jack Jane</siblings>
</human>

// Sortie de la désérialisation
\\nDeserialized Person:
Name: John
Age: 30
Is Married: true
Siblings: [Sibling{name='Jane'}, Sibling{name='Jack'}]
```

Java API pour la Liaison JSON (JSON-B)

JSON Binding (JSON-B) est une API Java qui permet de travailler avec des données JSON dans les applications Java. JSON-B offre des fonctionnalités pour sérialiser des objets Java en documents JSON et désérialiser des documents JSON en objets Java.

- JSON Object : **{}**
- JSON Array of Objects : **[{}, {}, ...]**

Sérialisation / Marshalling

La sérialisation est le processus de conversion d'objets Java en documents JSON. Avec JSON-B, vous pouvez prendre une hiérarchie d'objets Java et la convertir en une représentation JSON équivalente.

```
public static String serialize(Personne personne) {
    JsonbConfig config = new JsonbConfig().withFormatting(Boolean.TRUE);
    Jsonb builder = JsonbBuilder.create(config);
    return builder.toJson(personne);
}
```

Désérialisation / Unmarshalling

La désérialisation est le processus de conversion de documents JSON en objets Java. JSON-B vous permet de prendre un document JSON et de le convertir en une hiérarchie d'objets Java équivalente.

```
public static Personne deserialize(String jsonString) {
    Jsonb builder = JsonbBuilder.create();
    return builder.fromJson(jsonString, Personne.class);
}
```

Exemple d'Utilisation

Voici un exemple d'utilisation de JSON-B pour sérialiser et désérialiser un objet `Personne` :

```
public class Main {
    public static void main(String[] args) {
        // Créer un objet Personne
        Personne personne = new Personne("Jean", 30, true);

        // Sérialiser l'objet Personne en JSON
        String jsonSerialise = serialize(personne);
        System.out.println("JSON sérialisé:");
        System.out.println(jsonSerialise);

        // Désérialiser le JSON en un objet Personne
```

```
        Personne personneDeserialise = deserialize(jsonSerialise);
        System.out.println("\nPersonne désérialisée:");
        System.out.println("Nom: " + personneDeserialise.getNom());
        System.out.println("Âge: " + personneDeserialise.getAge());
        System.out.println("Marié: " + personneDeserialise.estMarie());
    }
}
```

Sortie

```
// Sortie de la sérialisation
{
    "nom": "Jean",
    "age": 30,
    "marie": true
}

// Sortie de la désérialisation
Personne désérialisée:
Nom: Jean
Âge: 30
Marié: true
```

Web Service avec REST API architecture

- REpresentational State Transfert (REST) est un style d'architecture pour les systèmes distribués.
- Les services REST sont sans états (Stateless)
  - Chaque requête envoyée au serveur doit contenir toutes les informations relatives à son état et est traitée indépendamment de toutes autres requêtes
  - Minimisation des ressources systèmes (pas de gestion de session, ni d'état)
- Méthodes (verbes) permettant de manipuler les ressources (identifiants)
- Interface uniforme basée sur les méthodes *HTTP* (*GET, POST, PUT, DELETE*)
  - **RETRIEVE → GET**
  - **CREATE → POST**
  - **UPDATE → PUT**
  - **DELETE → DELETE**
- Les architectures **RESTful** sont construites à partir de ressources uniquement identifiées par des **URI**(s)
  - Ressources Identifiée par un *URI* ex (*https://example.com/items*)
- URL : *http://localhost:8080/library/category*
  - localhost : address server
  - 8080 : nombre de port
  - library : context/nom d'application
  - category : ressource
- Format d'échanges entre le client et le serveur *(XML, JSON, text/plain,...)*

Java API for RESTful Web Services (JAX-RS)

- Annotation essentials

```
@Get     // Indique que cette méthode gère les requêtes HTTP GET.
@Post    // Indique que cette méthode gère les requêtes HTTP POST.
```

```
@Put     // Indique que cette méthode gère les requêtes HTTP PUT.
@Delete  // Indique que cette méthode gère les requêtes HTTP DELETE.
@Path("add({a},{b})")  // Spécifie le modèle de chemin pour cette méthode, où {a} et {b} sont
des paramètres de chemin.
@Consumes(MediaText.TEXT_PLAIN)  // Spécifie que cette méthode consomme le type de média tex
t/plain.
@Produces(MediaType.APPLICATION_[XML|JSON])  // Spécifie que cette méthode produit soit le ty
pe de média XML, soit JSON en fonction de la demande.
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  // Spécifie que cette mét
hode produit à la fois les types de médias XML et JSON.
@PathParam("name")  // Spécifie un paramètre de chemin nommé "name".
```

Server / Web Service

pour déployer un service web RESTful en utilisant Java EE, vous devez disposer d'un serveur d'applications Java EE, tel que Apache Tomcat, WildFly ou GlassFish, Ces serveurs d'applications fournissent l'environnement nécessaire et le support d'exécution pour exécuter les applications Java EE, y compris les services web RESTful.

```
import javax.ws.rs.*;

@Path("/example")
public class ExampleService {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/hello")
    public String sayHello() {
        return "Hello, World!";
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/echo")
    public String echoMessage(String message) {
        return message;
    }

    @PUT
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/update/{id}")
    public String updateResource(@PathParam("id") int id) {
        return "Resource with ID " + id + " updated successfully!";
    }

    @DELETE
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/delete/{id}")
    public String deleteResource(@PathParam("id") int id) {
        return "Resource with ID " + id + " deleted successfully!";
    }
}
```

```
package ws;

import javax.ws.rs.*;
```

```
@Path("math")
public class Math {

    @GET
    @Path("functions/add({a},{b})")
    @Produces(MediaType.TEXT_PLAIN)
    public int add(@PathParam("a") int a, @PathParam("b") int b) {
        return a + b;
    }
}
```

Client

```
package client;

import javax.ws.rs.client.*;
import javax.ws.rs.core.MediaType;


public class Client {

    public static void main(String[] args) {
        final String url = "<http://localhost:8080/jax-rs101/webresources/>";
        Client client = ClientBuilder.newClient();
        String response = client.target(url)
                .path("math/functions/add(5,6)")
                .request(MediaType.TEXT_PLAIN)
                .get(String.class);
        System.out.println(response);
    }
}
```

Web Application Description Language [wadl]

- Le WADL du service web est accessible à l'URL : `/application.wadl`

```
<http://localhost:8080/jax-rs101/webresources/application.wadl?detail=true>
```

Composition de Services Web

La composition de services est le mécanisme qui permet l'intégration des services pour construire un nouveau web service à valeur ajoutée *(service composite)*.

Approches de composition:

- Orchestration
- Chorégraphie