Université Sorbonne Paris Nord

École d'Ingénieurs Sup Galilée

RMI et Services Web SOAP

Rapport de Travaux Pratiques – TP n°0

Matière : Du développement au déploiement d'applications web

Étudiant : BENHAMMADA Ahmed Amine 12107981

Promotion : INGINFOA2 (École d'Ingénieurs Sup Galilée)

Encadrant : Samir Youcef

Date de rendu : 23 novembre 2025

1 Introduction

Ce TP vise à implémenter deux technologies de communication distribuée en Java : le **RMI** (Remote Method Invitation) et les **Services Web SOAP** (Simple Object Access Protocol). L'environnement utilise **JDK 25.0.1** et **Maven**, nécessitant l'intégration des dépendances Jakarta EE pour assurer la compatibilité avec l'API JAX-WS. Les travaux sont disponibles dans le dépôt Git `code-to-deployment-course` (`TP0_RMI_SOAP_Services_Web`), sur la branche : `tp0-rmi-soap`

2 RMI (Remote Method Invocation) : L'Approche Java-Spécifique

Le RMI permet l'appel de méthodes sur des objets distants entre deux JVM. C'est une technologie orientée objet, mais son efficacité est limitée aux environnements Java homogènes.

2.1 Concepts et Mécanisme

Le RMI nécessite une **Interface Distante**, une **Implémentation du Serveur** qui s'enregistre dans le **Registry** (`rmiregistry`), et un **Client** qui obtient une référence via un **Stub**. La communication s'appuie sur la sérialisation pour transférer les objets.

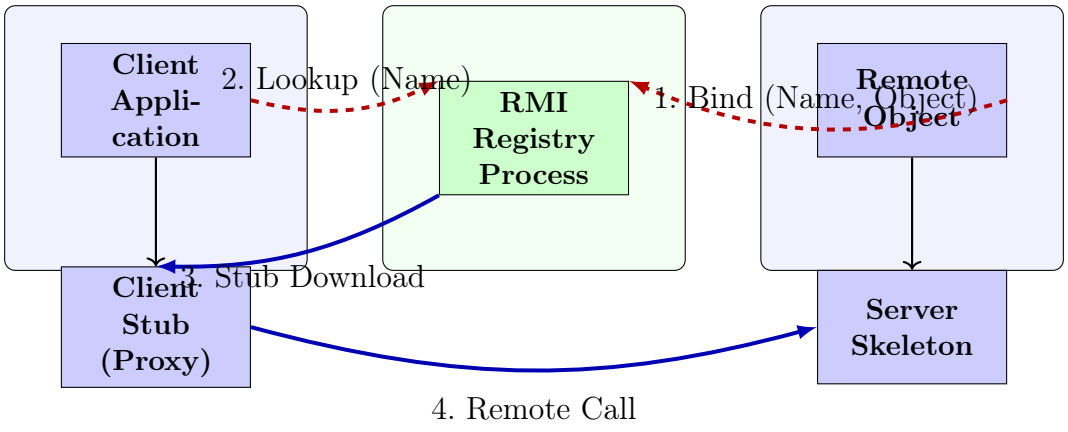

FIGURE 1 – Architecture RMI. Diagramme illustrant les interactions Client (Stub), Registry et Serveur (Skeleton).

3 Services Web SOAP : L'Interopérabilité par XML (JAX-WS/Jakarta)

SOAP utilise le format XML pour l'échange de messages, assurant une interopérabilité totale entre plateformes. L'implémentation est réalisée avec JAX-WS (Jakarta EE) pour le support JDK 25.

3.1 Configuration du Projet

L'utilisation de JDK 25 impose l'ajout explicite des dépendances `jakarta.xml.ws-api` et `jaxws-rt` dans le `pom.xml` afin de résoudre les imports et d'inclure le runtime du service.

3.2 Développement du Serveur SOAP

Le service est implémenté selon l'approche Bottom-Up, avec des annotations Jakarta (ex : `@WebService`) sur la classe métier.

3.2.1 Définition des Opérations (`MonServiceWeb.java`)

La classe implémente les opérations `convertir` et `additionner`, cette dernière utilisant `@WebParam` pour renommer les arguments en `parametre1` et `parametre2`.

```java
package service;

import jakarta.jws.WebMethod;
import jakarta.jws.WebParam;
import jakarta.jws.WebService;

@WebService(targetNamespace = "http://polytech.fr/")
public class MonServiceWeb {

    @WebMethod(operationName = "convertir")
    public double conversion(double r) {
        return r * 0.9;
    }

    @WebMethod(operationName = "additionner")
    public double somme(
        @WebParam(name = "parametre1") double a,
        @WebParam(name = "parametre2") double b) {
        return a + b;
    }
}
```

Code 1 – Extrait de MonServiceWeb.java (Operations)

3.2.2 Déploiement Autonome et Client Structure

Le service est publié localement via `Endpoint.publish()`.

```java
package service;

import jakarta.xml.ws.Endpoint;

public class Application {
    public static void main(String[] args) {
        String url = "http://localhost:8080/MonServiceWebWeb";
        Endpoint.publish(url, new MonServiceWeb());
        System.out.println("Service deploye sur " + url + "?wsdl");
           System.out.println("Acc dez au XSD via : " + url +
           "?xsd=1");
    }
}
```

Code 2 – Extrait de Application.java (Deploiement)

3.2.3 Définition de la Classe de Données Complexes (`Etudiant.java`)

Pour démontrer la capacité de SOAP à échanger des objets Java structurés, la classe `Etudiant` a été créée. Elle est annotée avec `@XmlRootElement` (faisant partie de l'API Jakarta JAXB) pour permettre sa sérialisation et désérialisation automatiques en XML, ce qui est essentiel pour les échanges dans le corps du message SOAP.

```
package service;

import jakarta.xml.bind.annotation.XmlRootElement;
import java.io.Serializable;

@XmlRootElement
public class Etudiant implements Serializable {
    private int id;
    private String nom;
    private double moyenne;

    // Constructeur sans argument requis pour JAXB
    public Etudiant() {
    }

    public Etudiant(int id, String nom, double moyenne) {
        this.id = id;
        this.nom = nom;
        this.moyenne = moyenne;
    }

    // Getters et Setters (Omis ici pour la concision)
    public int getId() { return id; }
    public String getNom() { return nom; }
    public double getMoyenne() { return moyenne; }

    // ... Setters ...
}
```

Code 3 – Classe Etudiant.java (Objet Complexe JAXB)

3.2.4 Opération de Manipulation d'Objet (`creerEtudiant`)

L'opération `creerEtudiant` a été ajoutée à `MonServiceWeb.java`. Elle prend en paramètre un objet `Etudiant` complet et le renvoie (ou le stocke dans un contexte réel). Cette méthode force JAX-WS à utiliser JAXB pour générer le schéma XML complexe correspondant dans le WSDL (élément `xsd:complexType`).

```
package service;
import jakarta.jws.WebMethod;
import jakarta.jws.WebParam;
import jakarta.jws.WebService;

@WebService(targetNamespace = "http://polytech.fr/")
public class MonServiceWeb {
```

```java
    // ... convertir et additionner ...

    /**
     * Cr e un   tudiant   fictif (exemple de s rialisation d'objet)
     * @param id identifiant de l' tudiant
     * @return un objet Etudiant
     */
    @WebMethod(operationName = "creerEtudiant")
    public Etudiant getEtudiant(@WebParam(name = "id") int id) {
        // Ici, on retourne un etudiant fictif pour l'exemple (pas de
           bdd pour ce tp0 mais    mon avis on pourrait aussi ici
           sauvegarder l     objet)
        return new Etudiant(1, "Amine", 18);
    }
}
```

Code 4 – Extrait de MonServiceWeb.java (Opération creerEtudiant)

3.3 Inspection du Contrat WSDL

Le WSDL (Web Service Description Language) est généré automatiquement par JAX-WS. Il décrit les opérations, les messages (Requêtes/Réponses) et les bindings du service.

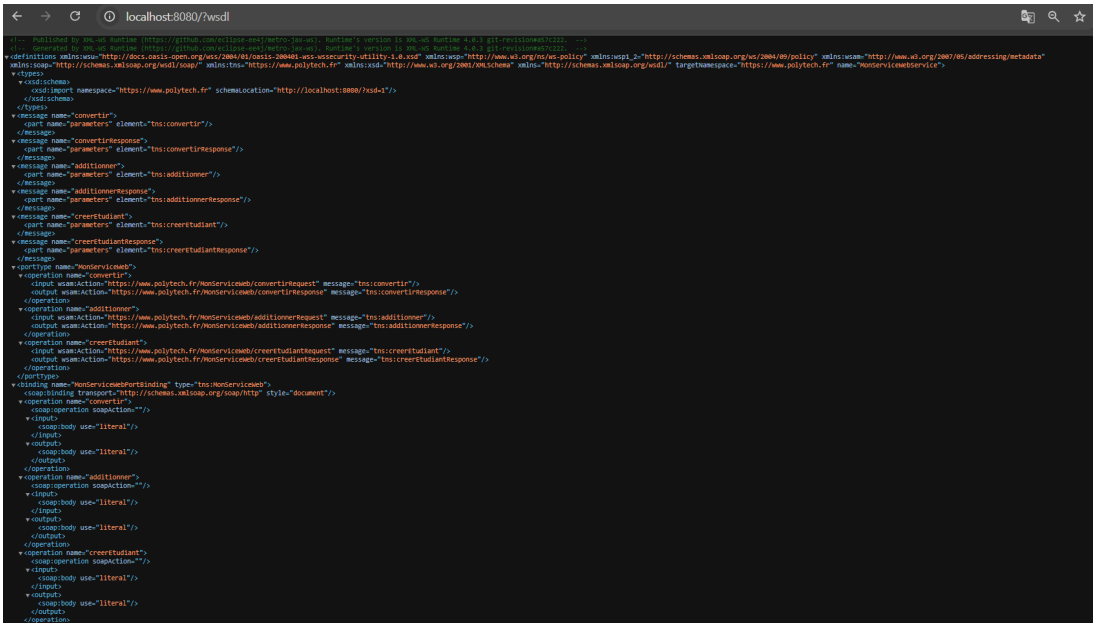

FIGURE 2 – Structure du WSDL. Le fichier décrit les messages (convertir, additionner, creerEtudiant), le portType (`MonServiceWeb`) et les bindings SOAP.

3.4 Vérification des Échanges de Messages (SoapUI)

SoapUI a été utilisé pour tester la conformité du service en inspectant les messages SOAP (XML).

3.4.1 Structure du Projet Client

L'outil client SoapUI interprète le WSDL pour générer la structure d'appel des opérations.

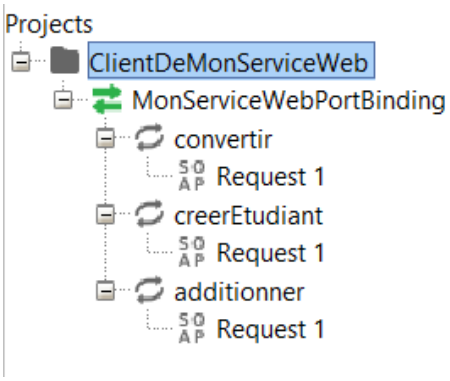

FIGURE 3 – Structure du service chargé dans SoapUI. Affichage des opérations `convertir`, `creerEtudiant` et `additionner` sous `MonServiceWebWebPortBinding`.

3.4.2 Opération de Conversion Simple

L'opération `convertir` utilise un argument par défaut (`arg0`) et retourne la valeur traitée.

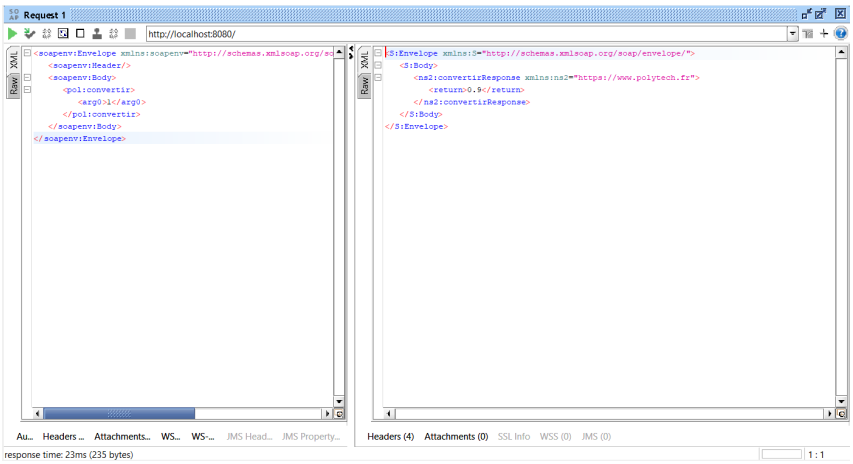

FIGURE 4 – Requête et Réponse SOAP pour l'opération `convertir`.

3.4.3 Opérations Simples et Paramètres Nommés

La requête pour l'opération `additionner` valide le renommage des arguments, essentiel pour la clarté du contrat de service, tel que spécifié par `@WebParam`.

3.4.4 Gestion des Objets Complexes (JAXB)

La classe `Etudiant` est sérialisée en XML par JAXB (Jakarta JAXB). La capture confirme l'échange d'objets structurés.

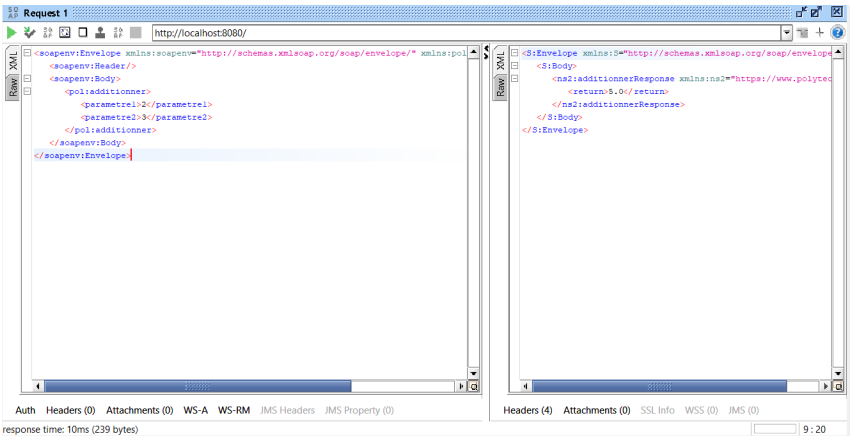

FIGURE 5 – Requête et Réponse SOAP pour l'opération `additionner`. Message XML montrant `parametre1=2`, `parametre2=3` et la réponse avec `5.0`.

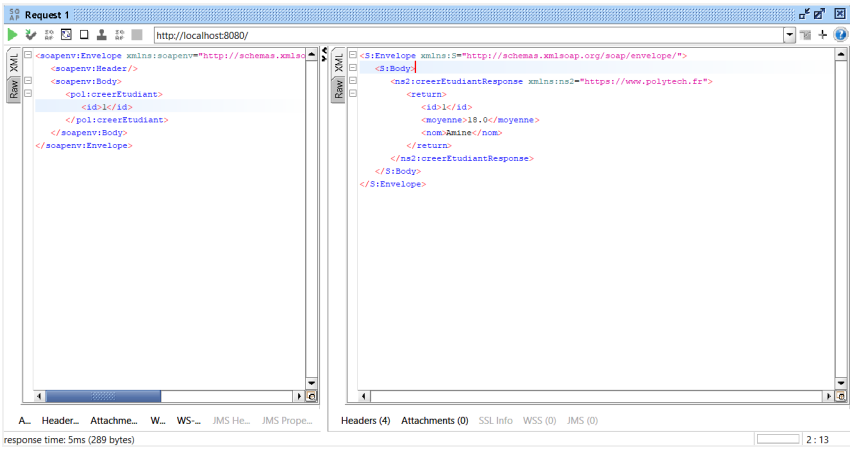

FIGURE 6 – Réponse SOAP contenant l'objet `Etudiant` sérialisé en XML. La réponse montre les champs de l'objet (`id`, `nom`, `moyenne`).

3.4.5 Détail de l'Échange d'Objet `Etudiant`

En se basant sur la figure 6, le message SOAP généré par SoapUI et validé par le service confirme la sérialisation des objets complexes. La requête contient tous les champs de la classe `Etudiant` encapsulés dans des balises XML, validant le bon fonctionnement de JAXB et du contrat WSDL/XSD généré.

Cette étape confirme l'interopérabilité des structures de données complexes, un avantage clé des Services Web SOAP basés sur des schémas XSD.

4 Conclusion et Perspectives

Le TP n°0 a permis d'acquérir une compréhension des communications distribuées Java. La migration vers Jakarta EE sous JDK 25 a été maîtrisée, assurant la pérennité de la solution. Les Services Web SOAP offrent une base solide sur l'interopérabilité via XML, préparant la transition vers les **API REST** du TP n°1, qui exploitent **la simplicité du format JSON.**