TD/TP nº 4
Services Web (WSDL)
Master Ingénierie Informatique
U.E. « Architectures logicielles orientées service »

F. BERTRAND

Année universitaire 2019–2020

1 Environnement de travail

Pour ce TP nous allons utiliser l'image virtualisée Linux. Il est possible d'installer les outils sur votre poste de travail mais aucun support ne sera assuré dans ce cas.

L'environnement de développement utilisé sera Netbeans, ce qui va permettre un gain de productivité quant au développement et au déploiement de services Web.

Il existe généralement trois approches pour développer des services Web avec un environnement de développement :

— soit on part d'une classe métier existante et on générera la spécification WSDL correspondante. L'environnement se chargera alors de générer l'ensemble des classes nécessaires au déploiement et à l'appel du service ;
— soit la classe métier sera créée directement avec des annotations indiquant les méthodes qui seront exportées comme services Web.
— soit on dispose d'une spécification WSDL et on génère directement les classes ;

Un des intérêts d'un environnement de développement c'est qu'il se charge d'assurer une certaine cohérence entre les différents fichiers nécessaires à la mise en œuvre et à l'appel du service.

Nous allons tester les deux dernières approches car, concernant la première approche, Netbeans nécessite que la classe métier soit déjà écrite comme un composant EJB.

2 Création d'un service Web *ab initio*

2.1 Spécification et déploiement

Soit la classe `Calculette` :

```
1  public class Calculette
2  {
3      public int additionner(int x, int y)
4      {
5          return x+y;
6      }
7  }
```

1. Tout d'abord installer le serveur d'application WildFly (dans `/tmp` par exemple). C'est un serveur d'applications relativement « léger » et moins gourmand en ressources que Glassfish.
2. Dans Netbeans, créer un projet `Calculette`, catégorie « Java Web », projet « Web application » et indiquer le répertoire d'installation du serveur WildFly lorsque celui est demandé.
3. Dans l'onglet « Projects » (en haut à gauche), accéder au menu contextuel du projet `CalculetteWS` et choisir « New », « Web Services... ». Nommer le service `CalculetteWS` puis indiquer l'option « Create Web Service from Scratch ».
4. Une fois le fichier généré, passer en mode « Conception » (bouton « Design ») et ajouter l'opération `additionner` avec ses paramètres. Puis repasser en mode « Source » pour voir le code annoté. Compléter le code pour qu'il soit similaire à la classe `Calculette` présentée ci-dessus.
5. Puis compiler et déployer le projet.
6. Observer la structure du fichier WSDL (son adresse est indiquée dans la console lors du déploiement). Puis utiliser l'annotation `@SOAPBinding(style=Style.RPC)` sur la classe implémentant le service et notez les modifications.

2.2 Accès au service via un programme Java

1. Créer un nouveau projet `CalculetteClient`, catégorie « Java », projet « Java Application » et laisser toutes les options par défaut.
2. Sélectionner `CalculetteClient` dans l'onglet « Projects » et accéder au menu contextuel en choisissant « New », « Web Service Client... ». Puis, pour le fichier WSDL, choisir le projet `Calculette` et sélectionner le service Web `CalculetteWS`.
3. Dans le `main`, accéder au menu contextuel et choisir « Insert Code... » et « Call Web Service Operation ». Choisir l'opération `additionner` de `CalculetteWS`.
4. Puis compiler et exécuter pour tester.

2.3 Visualisation des messages SOAP échangés

Pour visualiser les messages SOAP échangés avec TCPmon, il est nécessaire de modifier le client pour que celui-ci passe par TCPmon qui fera le relais et permettra de visualiser les messages échangés.

Le code client sera modifié de la manière suivante pour que celui-ci « passe » par TCPmon pour accéder au fichier `.wsdl` du serveur :

```
1 URL wsdlLocation = new URL(URL du fichier WSDL avec le numéro de port écouté par TCPmon);
2 QName serviceName = new QName(Espace de nom par défaut du service, Nom du service);
3 Nom_du_service_Service service
4         = new Nom_du_service_Service(wsdlLocation, serviceName);
5 Nom_du_service port = service.getNom_du_servicePort();
```

Vous trouverez l'URL original dans la barre de votre navigateur lorsque vous testez le service web (menu contextuel).

Éléments ci-dessous optionnels...

Puis, en partant de la classe `Calculette`, créer deux nouveaux services :

— *un service utilisant la délivrance fiabilisée des messages (*Reliable Message Delivery*) en activant l'option correspondante lorsque la classe assurant le service est visualisée en mode « Design ».*

— *un service sécurisé (*Secure Service*) permettant de crypter le contenu des messages SOAP en activant l'option correspondante lorsque la classe assurant le service est visualisée en mode « Design ». Il sera alors nécessaire d'éditer les attributs du service web et de choisir le mécanisme « Mutual Certificates Security ». Le détail de la configuration du client est précisé ici.*

Pour chaque service, veillez à faire deux projets ***différents*** *avec les clients correspondants. Vous rendrez, avec vos sources, les différents messages échangés.*

3 Création d'un service Web à partir d'une spécification WSDL

Dans cette approche, l'environnement va générer, à partir d'un fichier WSDL, le squelette des classes à compléter. L'avantage d'une spécification WSDL par rapport à une classe métier est qu'elle est neutre au niveau langage. On peut ainsi envisager, à partir de cette spécification, de générer la partie client en Python et la partie serveur en Java.

Pour tester cette approche il sera nécessaire de créer un fichier `serviceAnnonce.wsdl` décrivant les services Web développés par un éditeur pour gérer des annonces (dépôt, édition, recherche). Le service sera constitué des quatre opérations CRUD (Create, Retrieve, Update, Delete) et d'une opération de recherche générale :

```
1  // retourne l'identifiant de l'annonce créée
2  int creerAnnonce(String texte, String dateParution);
3  // recherche une annonce existante
4  Annonce rechercheAnnonce(int id);
5  // remplace le texte d'une annonce existante
6  boolean modifieAnnonce(int id, String nouveauTexte);
7  // supprime une annonce existante
8  boolean supprimeAnnonce(int id)
9  // retourne l'ensemble des annonces existantes
10 List<Annonce> donneAnnoncesExistantes();
```

3.1 Création du serveur et du client en Java

1. Créer un projet `ServiceAnnonce`, catégorie « Java Web », projet « Web application » et laisser toutes les options par défaut.
2. Dans l'onglet « Projects » (en haut à gauche), accéder au menu contextuel du projet `ServiceAnnonce` et choisir « New », « Web Services from WSDL... ». Nommer le service `ServiceAnnonceWS` puis indiquer l'emplacement du fichier `serviceAnnonce.wsdl`.
3. Dans un projet séparé, développer les classes métier en utilisant la classe fournie `ImplServiceAnnonce`. Écrire la classe `Annonce` et une classe `ServiceAnnonce` simulant le service permettant de compiler et de tester le `main` de la classe `ImplServiceAnnonce`. Le but ici est de pouvoir tester « localement » les classes métiers car leur mise au point en mode client-serveur est toujours plus délicat et plus long (en cas de problème nécessité d'aller regarder les logs sur le serveur).
4. Éditer et compléter le fichier `ServiceAnnonceWSDL.java` en vous aidant des classes générées dans le dossier « Generated Sources » (en grisé).
5. Inclure les classes métier développées dans le projet `ServiceAnnonce`, puis compiler et déployer le service.
6. Puis créer un client exécutant les mêmes actions que celle présentes dans le `main` de `ImplServiceAnnonce`.

3.2 Interopérabilité : création d'un client en Python

À l'aide de la bibliothèque `zeep` créer un client Python qui effectue les mêmes actions que le client Java créé précédemment.