UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

Rapport
Projet Services Web

Université Paris-Est Marne-la-Vallée

Master 2 - Logiciel

Olivier Bonin
obonin@etud.u-pem.fr

Raphaël Bohorodycz
rbohorod@etud.u-pem.fr

Elaad Furreedan
efurreed@etud.u-pem.fr

25 Novembre 2018

Table Des Matières

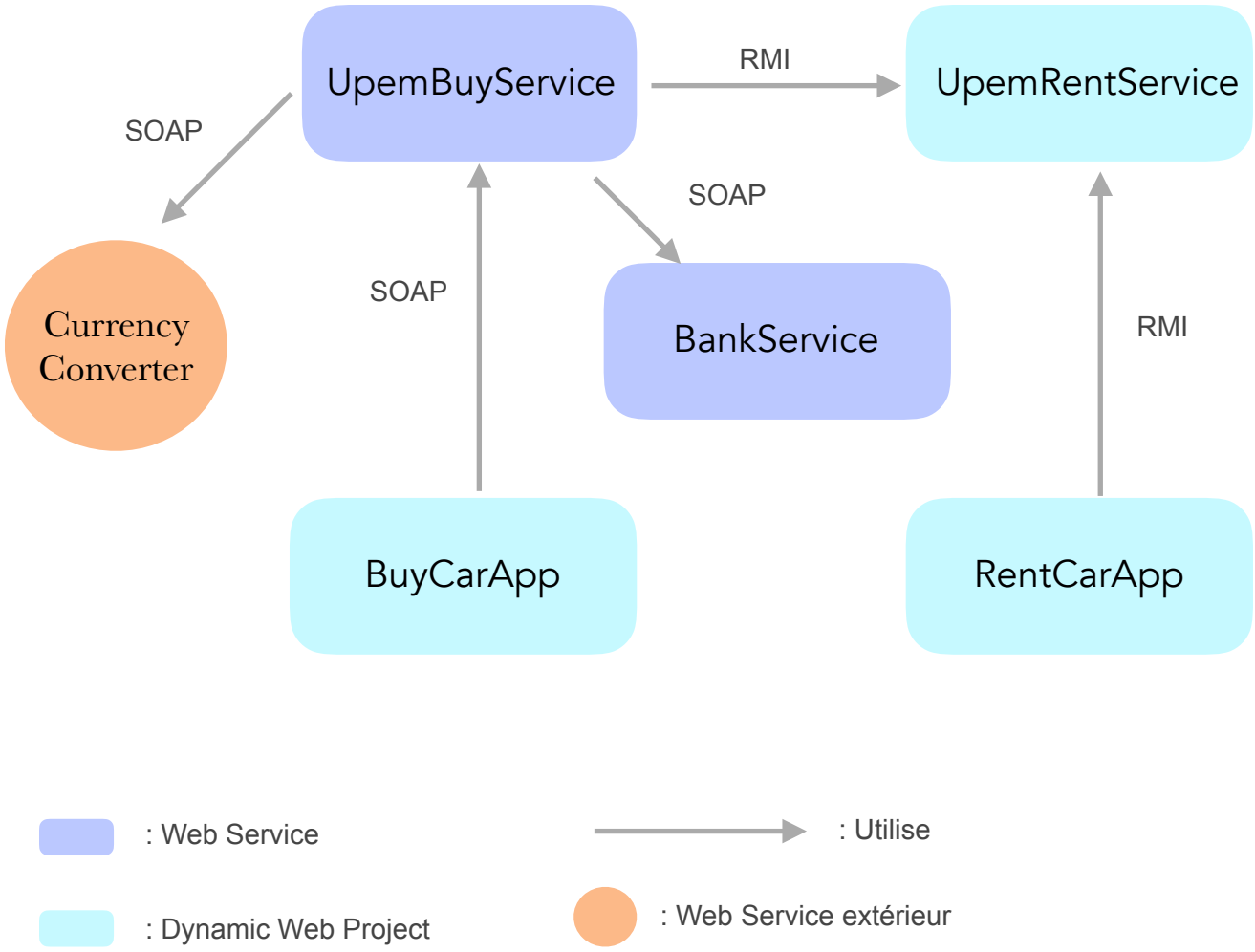
A. Introduction

Le projet a pour but d'implémenter un service de location pour les employés d'une entreprise, d'achat de véhicule, un système de notation et de commentaire ainsi qu'un service de gestion de compte bancaire simple.

Nous devions ainsi créer tous ces services en utilisant la technologie RMI afin de louer une voiture et la technologie SOAP et/ou REST pour l'achat et la gestion d'un compte bancaire.

Notre priorité dans ce projet a été de garantir l'isolation des données, en effet le système qui gère les employés et le service de location ne doivent pas se connaître mais doivent pouvoir fonctionner malgré tout. Nous nous sommes donc efforcer de garantir l'isolation entre chaque service, que se soit en SOAP ou en RMI.

B. Organisation des sources



1. UpemRentService

Le projet qui permet de gérer les voitures présentent dans le système et la location de voiture. Chaque objet voiture est associé à un id et chaque id de voiture est associé à une BlockingQueue qui représente la file d'attente pour louer la voiture en question. Le premier client de la BlockingQueue représente celui qui la possède actuellement. Au moment de la rendre il sort de la Queue et le prochain client prend alors la voiture.

Le système de file d'attente correspond au principe d'observateur et d'observé, ici les observateurs sont les clients, qui vont s'inscrire eux même à la file d'attente, lorsque la ressource est disponible c'est l'observé, ici les voitures qui vont notifier le client qui prend de suite la ressource.

HahMap qui permet gérer la liste des voitures louées.

Hashmap qui permet de gérer le dépôt de véhicule.

Hashmap qui permet de gérer la liste des reviews associées à un véhicule

`public void rent(long idVehicle, IClient client)`

Cette method permet de louer une voiture et de s'inscrire dans la file d'attente si jamais la ressource est prise (Queue non vide), elle représente la méthode subscribe du design pattern Observer.

`public void render(long idVehicle, IClient client)`

Cette method permet de rendre une voiture louée, elle représente la méthode notify du design pattern Observer car elle permet de prévenir le prochain client qu'il peut prendre la ressource,

tout ceci est gérer de manière évènementielle, c'est à dire qu'un fois la voiture rendu par le premier client alors le prochain récupère immédiatement la voiture.

C'est aussi directement dans ce projet que l'on ajoute la liste des voitures disponible.

2. BankService

Web Service qui gère le compte en banque d'un client, et qui permet d'interroger la solvabilité d'un client lors de l'achat d'un véhicule, de déposer de l'argent, de préserver une somme sur le compte ainsi que de créer un compte client.

Tous les client qui veulent acheter un voiture doivent créer un compte avant d'acheter une voiture, à la création le solde est de 0€, il faut donc déposer une certaine somme.

Le montant du solde est toujours affiché en euros, et les prélèvements se font aussi en euros

3. UpemBuyService

Web Service qui permet l'achat de véhicule présente dans la base de donnée. Il utilise lui même deux autres web service que sont : BankService et CurrencyConverter (http://currencyconverter.kowabunga.net/converter.asmx?WSDL) qui est un web service extérieur au projet.

Pour fonctionner il doit utiliser en RMI les méthodes dans UpemRentService, pour vérifier la disponibilité des voitures et accéder à liste des voitures.

Ce service permet entre autre de :

- Ajouter des voitures disponible dans un panier
- Acheter le contenu de son panier
- Gérer le contenu du panier
- Avoir un prix du panier mis à jour
- Avoir le prix du panier en la devise qu'il souhaite (via CurrencyConverter)

4. BuyCarApp & RentCarApp

Les deux application qui permettent d'interagir, il permettent respectivement d'acheter et de louer une voiture. RentCarApp communique avec le service de location en RMI tandis que BuyCarApp communique en SOAP avec le service d'achat.

C. RMI

Comme attendu dans le sujet, le service qui permet de louer des voitures (UpemRentService) est utilisé en RMI par l'application (RentCarApp) afin de permettre à des employés de louer des voitures et publier des reviews une fois la voiture rendue.

Nous avons décidé de masquer le maximum d'informations à UpemRentService concernant les employés, c'est pourquoi nous n'utilisons uniquement un id de client afin de faire référence aux employés qui louent des voitures. Les employés dans le service de location ne sont donc représenté par un id unique ce qui permet à d'autres programme d'utiliser le service en RMI, nécessitant qu'un id pour pouvoir fonctionner.

Concernant le service permettant d'acheter des voitures (UpemBuyService) il communique en RMI avec le service de location (UpemRentService) pour accéder à la liste des voitures et la liste des voitures louées actuellement.

Nous avons opté pour le passage de class par valeur plutôt que par référence afin d'éviter des effets de bords et de simplifier la gestion des données.

D. SOAP

Nous avons choisi d'utiliser la technologie SOAP pour faire communiquer notre service d'achat et notre service de gestion de compte en banque simplifié (BankService). UpemBuyService utilise BanService afin de vérifier le solde du client, pour des raisons de simplification nous avons décidé que l'id soit le même entre l'achat et son compte bancaire. Un client est donc représenté par un même id que se soit du côté de la banque que du service d'achat.

UpemBuyService utilise aussi un web service extérieur au projet qui permet d'obtenir le prix d'achat de son panier en la devise qu'il souhaite, ceci en temps réel.

Enfin l'application d'achat (BuyCarApp) utilise le service d'achat en SOAP afin de consulter la liste des voiture, la liste des reviews et acheter une voiture.

E. Problèmes rencontrés

Le problème majeur a été la conception du projet, en effet nous voulions minimiser les informations accessible des deux côtés que se soit du côté employé que du côté serveur de location.

Nous voulions aussi effectuer une interface web en JSP, mais n'ayant pas de réel base de donnée persistante ne pouvons pas afficher dynamiquement les données du système, c'est pourquoi nous avons décidé de nous d'implémenter des interaction directement via le terminal d'Eclipse JEE.

F. Manuel d'utilisation

1. Importer tous les projet dans le workspace (mis à part Server)
2. Créer un server en utilisant le server Tomcat présent dans le dossier « Server » (windows -> show View -> Server)
3. Créer un web service pour BankService et BuyCarService (file -> new -> web service)
4. Lancer le main dans RentCarService (java application)
5. Lancer les main de BuyCarApp et RentCarApp afin d'interagir

Le main du RentCarApp permet de gérer la location d'une voiture par un employé. Lorsqu'on lance le main, un menu s'affiche sur le terminal avec les différentes possibilités à notre disposition pour la location d'une voiture, il faudra entré :

- 1 pour louer une voiture
- 2 pour rendre une voiture
- 3 pour rendre une voiture et la notée
- 4 pour voir la liste des voitures
- 0 pour quitter l'application

Pour louer une voiture on demande quelle voiture l'employée veut louer, il faut entrer la marque de la voiture qu'on veut louer, une fois loué un message nous confirme que c'est fait.

Après avoir rendu une voiture, un message nous confirme que la voiture a bien été rendue. On peut rendre la voiture en ajoutant une review à la voiture, une note de 0 à 5 est demandée ainsi qu'un commentaire.

La liste des voitures s'affiche avec toutes les informations des voitures, marque, modèle, nombre de fois louées, prix d'achat, prix de location, âge requis.

Le main du BuyCarApp permet de gérer l'achat d'une voiture. Lorsqu'on lance le main, un menu s'affiche sur le terminal avec les différentes possibilités à notre disposition pour l'achat d'une voiture, il faudra entré :

- 1 pour déposer de l'argent sur son compte
- 2 pour ajouter une voiture au panier
- 3 pour acheter le ou les voitures dans le panier
- 4 pour voir le prix du panier en fonction d'une devise
- 5 pour voir la liste des reviews d'une voiture
- 6 pour voir la liste des voitures
- 0 pour quitter

Pour déposer de l'argent sur son compte, il est demandé le montant que l'on veut déposer.

Pour ajouter une voiture au panier il est demandé d'entrée l'Id de la voiture que l'on veut ajouter(Id que l'on peut voir en affichant la liste des voitures)

Lorsqu'on achète le ou les voitures du panier, un message de confirmation est affiché.

Pour voir les reviews d'une voiture, il faut entrer l'Id de la voiture souhaité.

G. Conclusion

Le projet avait pour but de nous faire manipuler les technologies SOAP et RMI vues en cours, bien que la technologie SOAP soit moins utilisé de nos jours, au profit de la technologie REST qui utilise protocole HTTP, nous avons pu nous confronter aux problèmes de conception et d'implémentation inhérents à ces deux technologies. N'ayant pas utiliser la technologie REST, nous pensons essayer d'implémenter le même projet en utilisant cette technologie car plus répandue afin d'avoir un réel comparatif en therme d'implémentation.

En utilisant l'IDE Eclipse nous nous sommes concentré sur l'implémentation du code, ce qui nous a déléguer d'une grosse partie de la configuration nécessaire, attisant notre curiosité sur le moyen de reproduire ce projet en d'autre langages.