SÉRIALISATION

Laboratoire n°4
Christophe Peretti
Samuel Darcey

Table des matières

1 Introduction

Le laboratoire SER – Plex consistant à simuler les activités d'un complexe cinématographique, est constitué de 4 laboratoires constituant les différentes parties du projet. Le présent document porte sur le quatrième laboratoire : Echange d'informations entre serveurs et clients grâce à RMI.

Dans ce laboratoire, il s'agissait concevoir plusieurs classes java, afin de mettre en place un serveur RMI du côté du World Film Center (WFC) afin de pouvoir mettre à jour les bases de données des différents gestionnaires de cinémas (PlexAdmin). De plus, ce PlexAdmin fait aussi office de serveur RMI afin que les différents médias (PlexMedia) puissent venir chercher les informations des différentes projection de film.

2 WFC

L'applicatif instancie un RmiServer, qui est une classe implémentant l'interface IServerApi (disponible dans le WFC_Interface. On crée aussi le Registre lié à cette classe, qui va être disponible sur le réseau avec le nom « RmiService ». Pour ce serveur, le port 9999 est utilisé.

```
try {
    Registry rmiRegistry = LocateRegistry.createRegistry(9999);
    server = new RmiServer(lastData);
    IServerApi rmiService = (IServerApi) UnicastRemoteObject.exportObject(server, 9999);
    rmiRegistry.bind("RmiService", rmiService);
} catch (Exception ex) {
    ex.printStackTrace();
}
```

Le serveur met à disposition une classe permettant à un client de s'enregistrer comme observer.

```
@Override
public void addObserver(IClientApi client) throws RemoteException {
    WrappedObserver wo = new WrappedObserver(client);
    addObserver(wo);
    System.out.println("Added observer: " + wo);
}
```

De plus, nous avons implémenté une fonction public permettant à l'applicatif de notifier les différents clients (et donc appeler leur fonction update).

```
public void notifyClients(Data data){
    this.data = data;
    setChanged();
    notifyObservers(data);
}
```

Lorsque l'utilisateur choisi une option, les dernières données sont transmises au serveur, et donc notifiées aux clients (PlexAdmin).

3. PlexAdmin

3.1 Client RMI

Le PlexAdmin fait office de client RMI : lorsque le serveur envoie les nouvelles données, le client doit automatiquement vider sa base de donnée de projections, et mettre à jour la liste des films avec les

nouvelles données. Dans le ControlleurWFC, nous avons instancié un client RMI, après avoir récupéré le RmiService du serveur.

```
try {
    //we connect to server
    IServerApi remoteService = (IServerApi) Naming.lookup("//localhost:9999/RmiService");

    //we start client
    RmiClient client = new RmiClient(remoteService, ctrGeneral);
    client.initialConnection();
    client.startCheckingThread();

} catch (Exception ex) {
    ex.printStackTrace();
}
```

Le RmiClient reçoit en paramètre le remoteService (afin de pouvoir s'y enregistrer en tant qu'observer), ainsi que le ControleurGeneral de l'applicatif, qui va pouvoir mettre à jour les données reçues.

Ainsi, lorsque le serveur met à jour ses données, cela notifie le client, appelle donc la fonction update, qui elle appelle la fonction de mise à jour des données du client.

```
@Override
public void update(Object observable, Signal signalType, Data newData) throws RemoteException {

    cont.initBaseDeDonneesAvecNouvelleVersion(newData);
    System.out.println("Update request: ");

}
```

3.2 Serveur RMI

L'applicatif PlexAdmin, fait aussi office de serveur RMI. Son fonctionnement est un peu différent de celui de WFC. Ici, ce n'est pas le serveur qui force les clients de faire une action, mais il met simplement à disposition une méthode qui leur transmet la liste des projections au format JSON. Cette méthode est appelée grâce à RMI, directement par les clients eux-mêmes, lorsqu'ils en ont envie.

```
@Override
public String getData() throws RemoteException {
    return this.cont.sendJSONToMedia();
}
```

Nous avons dû modifier l'interface du serveur RMI, afin que la méthode getData() retourne un String. Cela permet d'envoyer directement le JSON à travers cette méthode. Nous avons donc aussi dû modifier de deux manières la fonction sendJSONToMedia() : Tout d'abord, celle-ci ne s'exécute plus dans un Thread à part, afin d'être sûr que le JSON à la fin de la méthode est bien finit d'être écrit dans le fichier, et ensuite, ce même JSON devient la valeur de retour de la fonction.

4. PlexMedia

L'applicatif PlexMedia, fait office de client RMI, qui s'enregistre en tant qu'observer auprès de PlexAdmin, tout comme celui-ci s'enregistre auprès du WFC. Pour ce média, nous avons créé une vue ClientGUI, qui ouvre une petite fenêtre JFrame, avec un bouton unique au centre. Lors que le bouton est pressé, cela appelle la fonction getData() du RmiClient, qui fait office de relai et appelle la fonction à distance getData(), du RMIServer PlexAdmin.

```
public void getData(){
    try {
        // On affiche dans la console le JSon des projections
        System.out.println(remoteService.getData());
    } catch (RemoteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

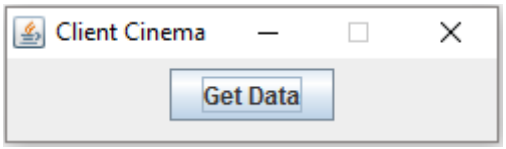
Et voici la vue du média :

```
public class ClientGUI {
    private static JPanel mainPanel;
    private static RmiClient controleurClient;
    private static JLabel warningLabel  = new JLabel();
    private static JLabel errorMessage  = new JLabel();
    private JButton getData                 = null;

    public ClientGUI(final RmiClient _controleurClient) {
        controleurClient = _controleurClient;
        mainPanel = new JPanel();
        // DÃ©finition de la barre d'outils
        getData = new JButton("Get Data");
        getData.addActionListener (new ActionListener(){
            public void actionPerformed(ActionEvent e){
                controleurClient.getData();
            }
        });
        mainPanel.add(getData);
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                //Turn off metal's use of bold fonts
                UIManager.put("swing.boldMetal", Boolean.FALSE);
                createAndShowGUI();
            }
        });

    }
    private void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("Client Cinema");
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(200, 150);
        //Add content to the window.
        frame.setContentPane(mainPanel);
        //Display the window.
        frame.pack();
        frame.setVisible(true);
        frame.setResizable(false);
    }

}
```

De plus, nous avons ajouté une petite méthode qui désérialise le Json afin d'en avoir un aperçu un peu plus intuitif :

```
public void deserialize(String json){
    int a = 1;
    int b;
    do{
        b = json.indexOf('"',a + 1);
        String name = json.substring(a + 1, b);
        a = json.indexOf('"',b + 1);

        switch(name){
            case "cinema":
            case "Projections":
                System.out.println(name);
                break;
            case "titre":
            case "date":
                System.out.print("\t" + name + " \t\t\t: ");
                break;
            case "acteurs":
                System.out.println("\t" + name);
                break;
            case "Place 1":
            case "Place 2":
                System.out.print("\t\t" + name + " \t: ");
                break;
            case "id":
                System.out.print("\t" + name + " \t\t\t: ");
                System.out.println(json.substring(b + 2, b + 3));
                break;
            default:
                System.out.println(name);
        }
    }while(b > 0 && a > 0);
}
```

5. Aperçu de l'applicatif PlexMedia




```
Press button to get Data:
{"cinema":{"Projections":[{"id":4,"titre":"Casino Royale","date":"2016-06-15 15:20:44","acteu
cinema
Projections
        id                      : 4
        titre                   : Casino Royale
        date                    : 2016-06-15 15:20:44
        acteurs
                Place 2         : Green, Eva
                Place 1         : Craig, Daniel
        id                      : 5
        titre                   : Django Unchained
        date                    : 2016-06-15 15:48:34
        acteurs
                Place 1         : Foxx, Jamie
                Place 2         : Waltz, Christoph
```

6 Conclusion

Grâce à ce laboratoire, nous avons eu un aperçu de la puissance de RMI. Pouvoir appeler des méthodes à distance est très pratique, et permet de s'affranchir d'une communication par Socket qui nécessite une sérialisation et désérialisation de chaque côté, parfois très couteuse est lourde à mettre en place.