

Filière : 1^{ème} année cyber-security

Semestre : 6

Module : Applications réparties

Rapport:

réalisation d'un système de gestion des événements en JAVA RMI et JAVA swing

ENCADRÉ PAR :
PR A.AMAMOU

PRÉPARÉ
PAR L'ÉTUDIANTE:
HAJAR BABA AHMED

Sommaire

- 01.** Introduction
- 02.** Analyse des besoins
- 03.** Conception du système
- 04.** Implémentation
- 05.** Déploiement
- 06.** conclusion

Introduction

Java RMI (Remote Method Invocation) est une technologie avancée qui facilite la communication entre objets distants en Java. En offrant une transparence de la communication, une simplicité d'utilisation et des mécanismes de sécurité intégrés, Java RMI permet aux développeurs de concevoir des applications distribuées sans se soucier des détails de la communication réseau. Grâce à son mécanisme d'invocation distante, les développeurs peuvent interagir avec des objets distants de manière similaire à des objets locaux, simplifiant ainsi le développement d'applications distribuées. Java RMI gère efficacement les exceptions liées à la communication entre objets distants, en utilisant des exceptions spécifiques telles que `RemoteException`. De plus, Java RMI offre des fonctionnalités avancées telles que la gestion des transactions distribuées et la sécurité renforcée à travers des mécanismes de signature numérique et de chiffrement. En constante évolution, Java RMI s'adapte aux architectures modernes en offrant une interopérabilité avec d'autres technologies de communication telles que les services web, ce qui permet de créer des systèmes distribués complexes et flexibles adaptés aux environnements cloud et microservices. En explorant ces aspects approfondis de Java RMI, les développeurs peuvent exploiter pleinement cette technologie pour concevoir des applications distribuées robustes et efficaces en Java.

Introduction

1 Contexte du projet

Avec l'avènement des technologies distribuées, il est désormais possible de créer des systèmes de gestion d'événements accessibles à distance via des réseaux. Java RMI (Remote Method Invocation) est une technologie qui permet la communication entre objets situés sur différentes machines virtuelles Java, facilitant ainsi le développement d'applications distribuées. Grâce à RMI, les objets peuvent être créés et enregistrés sur un serveur, et les clients peuvent accéder à ces objets en invoquant leurs méthodes à distance, tout en ignorant la localisation physique des objets distants. Cela permet de créer des applications robustes et scalables, où les objets peuvent être déployés sur différents serveurs et accédés par des clients à travers un réseau.

2 Objectifs du Projet

Ce projet vise à développer un système distribué basé sur une architecture client/serveur, où les clients peuvent interagir avec un serveur centralisé pour gérer des événements. En utilisant Java RMI pour la communication entre le client et le serveur, l'objectif est de permettre aux utilisateurs d'ajouter de nouveaux événements. De plus, le système offrira la fonctionnalité de suppression d'événements existants. En combinant la puissance de Java RMI avec la gestion des événements, ce projet vise à fournir une solution robuste et flexible pour la gestion efficace des événements à travers un environnement distribué.

Analyse des besoins

1 Ajouter des Événements

Permettre aux utilisateurs de créer de nouveaux événements, Pour effectuer cette opération, le système distribué utilisant Java RMI devrait offrir une interface utilisateur intuitive pour les utilisateurs. Le client du système distribué affichera une interface utilisateur qui permet aux utilisateurs de fournir des 'événement. le client envoie les informations de l'événement au serveur centralisé via Java RMI. Le serveur affiche dans une interafece un message indique que l'événement est ajouté avec succès, s'il trouve un problème lors de l'ajoute de l'événement, il va afficher un message d'erreur.

2 Supprimer des Événements

Le client du système distribué affichera une interface utilisateur qui permet aux utilisateurs de rechercher et de sélectionner les événements à supprimer, le client envoie alors une requête de suppression au serveur centralisé via Java RMI. Si l'opération de suppression est valide, le serveur supprime l'événement de la liste qui contient les événements ajoutés, et l'ajoute dans une liste des événements supprimés.

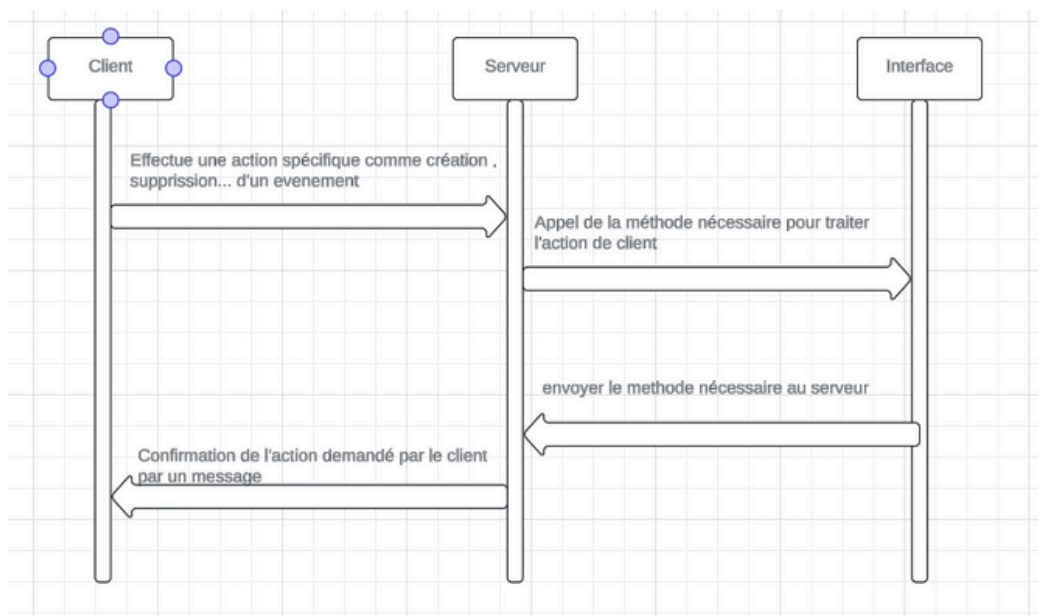
3 Lister les Événements

Après avoir ajouté un evenement par le client via l'interface graphique , ce événement va etre stocker et afficher dans une liste dans le coté serveur et aussi dans l'interface clinet, et le meme chose pour la suppression de l'événement.

Conception du système

1 Architecture générale

L'architecture de ce projet c'est l'architecture Client/Serveur, où le client envoie des requêtes (ajoute, suppression) au serveur via java RMI, et le serveur traite ces requêtes en utilisant les méthodes déclarées dans l'interface (Remote Interface), après avoir traité les requêtes envoyées par le client, le serveur affiche un message comme quoi la requête est bien traitée.



2 méthodes RMI

Les méthodes RMI exposées par le serveur et utilisées par le client sont deux méthodes, méthode qui permet de ajouter un événement et une méthode pour la suppression, les deux méthodes sont définies comme cela:

```
public void ajouterEvent(String event) throws RemoteException;
public void supprimerEvent(String event) throws RemoteException;
```

Implémentation

1 Serveur RMI

est un programme Java qui permet d'exécuter des méthodes d'un objet distant, c'est-à-dire appartenant à une autre machine virtuelle (JVM). Le serveur RMI est responsable de gérer les appels de méthodes distantes et de les déléguer à l'objet distant approprié. Pour notre projet, le serveur fait appel aux deux méthodes (ajouterEvent, supprimerEvent), et après avoir terminé le traitement des requêtes envoyées par le client, il utilise une interface graphique en JAVA swing pour afficher la liste des événements ajoutés et supprimés et les messages qui indiquent que l'opération est bien exécutée. Le serveur utilise des gestionnaires des exceptions comme RemoteException qui permet de gérer les problèmes de communication entre le client et le serveur, il utilise aussi NotBoundException qui permet de rechercher d'un objet non enregistré dans le registre RMI. La gestion des exceptions est essentielle pour garantir la robustesse et la fiabilité de notre application de gestion d'événements. Elle permet de gérer les erreurs de manière appropriée, d'informer les utilisateurs des problèmes, et de maintenir le système dans un état cohérent.

2 Extrait de code Serveur:

```
root@ubuntu: /home/hajarbabaahmed
GNU nano 4.8
RMIserver.java

import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class RMIserver extends JFrame implements RMIInterface {
    JComboBox<String> listEventAjouter;
    JComboBox<String> listEventSupprimer;
    JTextArea zoneMessage;

    JLabel labelEventAjouter;
    JLabel labelEventSupprimer;

    JScrollPane paneAjouter;
    JScrollPane paneSupprimer;
    JScrollPane paneMessage;

    public RMIserver() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        listEventAjouter = new JComboBox<>();
        listEventSupprimer = new JComboBox<>();
        zoneMessage = new JTextArea(10,30);

        labelEventAjouter = new JLabel("Evénements ajoutés");
        labelEventSupprimer = new JLabel("Evénements supprimés");

        paneAjouter = new JScrollPane(listEventAjouter);
        paneSupprimer = new JScrollPane(listEventSupprimer);
        paneMessage = new JScrollPane(zoneMessage);

        JPanel ajoutPanel = new JPanel(new BorderLayout());
        ajoutPanel.add(labelEventAjouter, BorderLayout.NORTH);
        ajoutPanel.add(paneAjouter, BorderLayout.CENTER);

        JPanel supprPanel = new JPanel(new BorderLayout());
        supprPanel.add(labelEventSupprimer, BorderLayout.NORTH);
        supprPanel.add(paneSupprimer, BorderLayout.CENTER);

        public static void main(String[] args) {
            RMIserver serverObject = new RMIserver();
            System.out.println("serveur pret");
            Registry reg = null;
            try {
                reg = LocateRegistry.createRegistry(1098);
            } catch (Exception e) {
                System.out.println("ERROR: Could not create the registry.");
                e.printStackTrace();
            }
            try {
                reg.rebind("server", (RMIInterface) UnicastRemoteObject.exportObject(serverObject, 0));
            } catch (Exception e) {
                System.out.println("ERROR: Failed to register the server object.");
                e.printStackTrace();
            }
        }
    }
}
```

Implémentation

1 Client RMI

est un programme Java qui invoque des méthodes d'objets distants hébergés sur un serveur RMI, donc le client pour ce projet invoque les deux méthodes (ajouterEvent, supprimerEvent) qui sont hébergés dans le serveur. Le client doit connaître l'interface distante qui définit les méthodes exposées par le serveur. Cette interface est partagée entre le client et le serveur, le client RMI récupère une référence sur l'objet distant via le registre RMI, puis invoque ses méthodes comme s'il était local. Le stub se charge de déléguer les appels au serveur distant de manière transparente pour le code client.

Le client contient une interface graphique où il effectue les différents actions (ajouter et supprimer), quand le client saisie l'événement qu'il veut l'ajouter et clique sur le bouton ajouter, un message indique que le client ajoute un événement va être affiché dans l'interface serveur, ainsi dans l'interface client, et il va être stocker dans deux liste, l'une dans l'interface serveur et l'autre dans l'interface client.

2 Extrait de code Client:

```
GNU nano 4.8 RMIClientGUI.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.util.ArrayList;

public class RMIClientGUI extends JFrame {

    JScrollPane outputAreaPane;
    JScrollPane listPane;

    JPanel inputPanel;
    JPanel buttonPanel;
    JPanel panelList;
    JPanel pane;

    JTextField eventField;
    JTextArea outputArea;

    JList<String> eventList;
    DefaultListModel<String> list;

    JLabel labelList_Evenements;

    JButton addButton;
    JButton deleteButton;

    private RMIInterface rmi;
    public RMIClientGUI() {

        eventField = new JTextField(20);
        outputArea = new JTextArea(10, 30);
        outputArea.setEditable(false);
        list = new DefaultListModel<>();
        eventList = new JList<>(list);

        try {
            Registry registry = LocateRegistry.getRegistry("192.168.254.141", 1098);
            rmi = (RMIInterface) registry.lookup("server");
            outputArea.append("Connecté au serveur RMI\n");
        } catch (RemoteException e) {
            outputArea.append("Erreur de connexion au serveur RMI: " + e.getMessage() + "\n");
            e.printStackTrace();
        } catch (NotBoundException e) {
            outputArea.append("Le serveur RMI n'est pas lié: " + e.getMessage() + "\n");
            e.printStackTrace();
        }

        public static void main(String[] args) {

            new RMIClientGUI().setVisible(true);
        }
    }
}
```


Implémentation

3

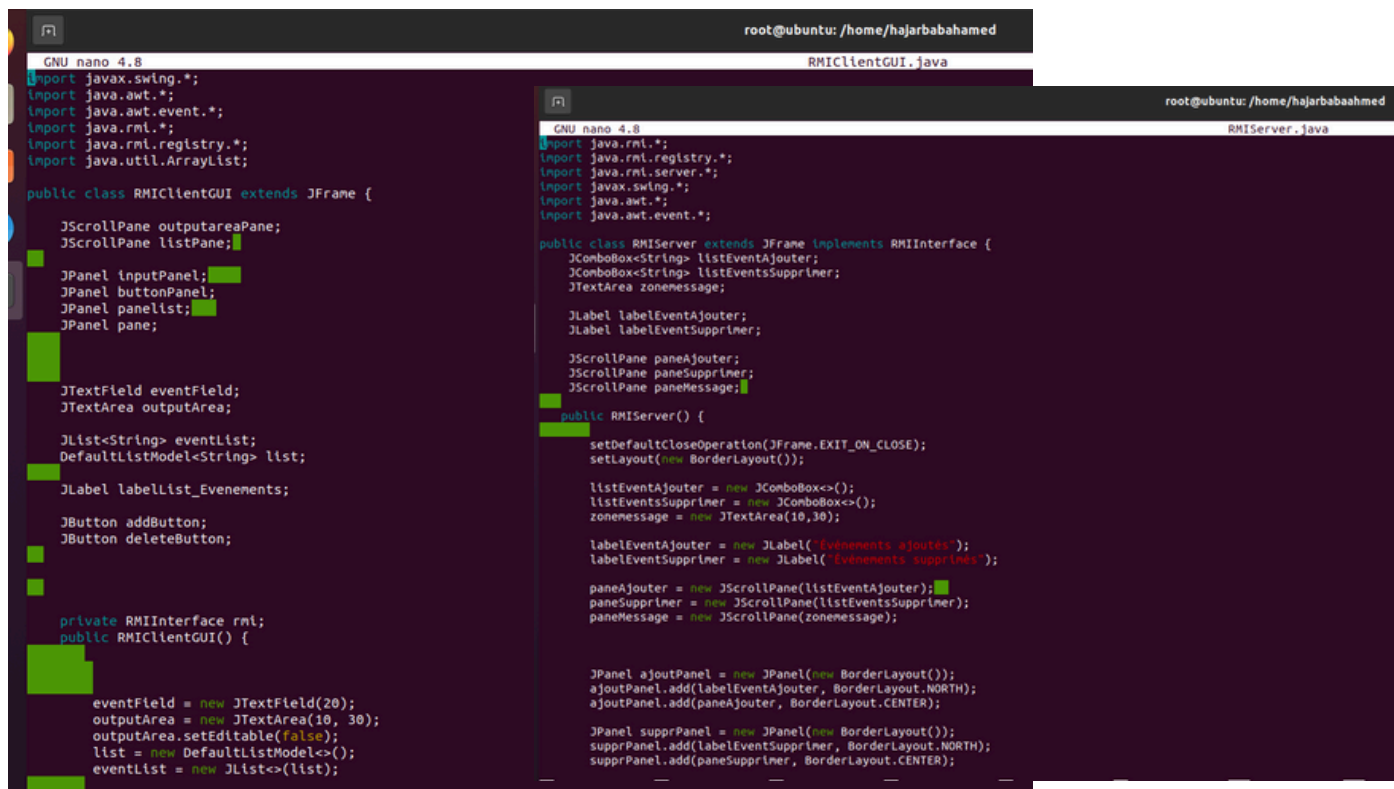
Interface graphique (client et serveur)

Une interface graphique (GUI pour Graphical User Interface) est un type d'interface utilisateur qui permet à l'utilisateur d'interagir avec un programme informatique à l'aide d'éléments graphiques tels que des fenêtres, des boutons, des menus, des barres de défilement. Pour notre projet on a réalisé deux interfaces graphique en java swing une pour le client et l'autre pour le serveur, et on a utilisée plusieurs fonctions pour créer les différents éléments d'une interface:

- JFrame : La classe JFrame est utilisée pour créer une fenêtre pour contenir d'autres composants graphiques. Elle est souvent utilisée comme la fenêtre principale d'une application
- JButton : La classe JButton est utilisée pour créer un bouton graphique. Elle peut être utilisée pour créer des boutons avec des étiquettes et des actions associées
- JPanel : La classe JPanel est utilisée pour créer un panneau graphique qui peut contenir d'autres composants. Elle est souvent utilisée pour organiser les composants dans une fenêtre.

On a utilisé les fonctionnalités de Listner qui écoute des événements générés par un composant graphique, comme un bouton, un champ de texte. Dans le contexte de Java Swing, les listeners sont utilisés pour gérer les interactions entre l'utilisateur et les composants graphiques. Dans notre cas on a utiliser les Listneres pour les buttons qui sont résponsale de l'ajoute et de suppression, on a relire entre le button et l'action correspondant a ce button (clique sur le button ajouter permet d'ajouter l'événement dans la liste).

Extrait de code de l'interface graphique pour le Client et le serveur:



```
GNU nano 4.8
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.util.ArrayList;

public class RMIClientGUI extends JFrame {

    JScrollPane outputAreaPane;
    JScrollPane listPane;

    JPanel inputPanel;
    JPanel buttonPanel;
    JPanel panelList;
    JPanel pane;

    JTextField eventField;
    JTextArea outputArea;

    JList<String> eventList;
    DefaultListModel<String> list;

    JLabel labelListEvenements;

    JButton addButton;
    JButton deleteButton;

    private RMIInterface rmi;
    public RMIClientGUI() {

        eventField = new JTextField(20);
        outputArea = new JTextArea(10, 30);
        outputArea.setEditable(false);
        list = new DefaultListModel<>();
        eventList = new JList<>(list);

        RMIServerGUI.java

GNU nano 4.8
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class RMIServer extends JFrame implements RMIInterface {
    JComboBox<String> listEventAjouter;
    JComboBox<String> listEventsSupprimer;
    JTextArea zoneMessage;

    JLabel labelEventAjouter;
    JLabel labelEventSupprimer;

    JScrollPane paneAjouter;
    JScrollPane paneSupprimer;
    JScrollPane paneMessage;

    public RMIServer() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        listEventAjouter = new JComboBox<>();
        listEventsSupprimer = new JComboBox<>();
        zoneMessage = new JTextArea(10,30);

        labelEventAjouter = new JLabel("Evénements ajoutés");
        labelEventSupprimer = new JLabel("Evénements supprimés");

        paneAjouter = new JScrollPane(listEventAjouter);
        paneSupprimer = new JScrollPane(listEventsSupprimer);
        paneMessage = new JScrollPane(zoneMessage);

        JPanel ajoutPanel = new JPanel(new BorderLayout());
        ajoutPanel.add(labelEventAjouter, BorderLayout.NORTH);
        ajoutPanel.add(paneAjouter, BorderLayout.CENTER);

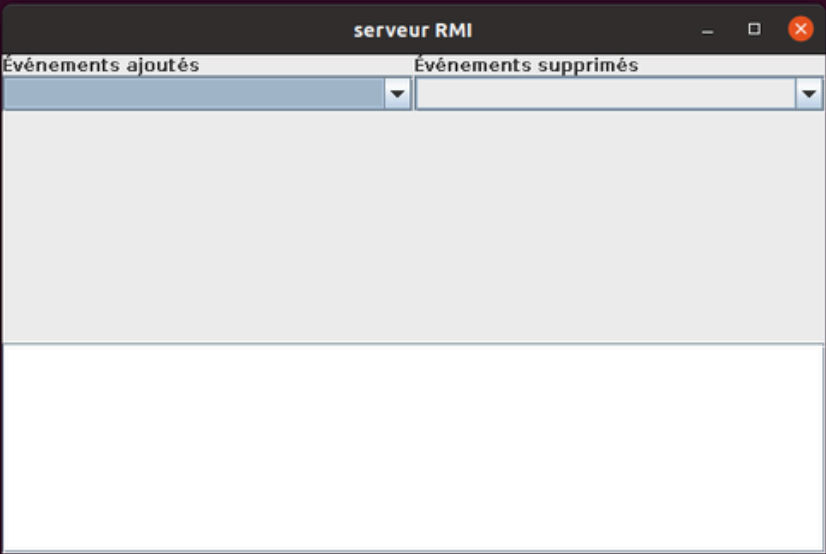
        JPanel supprPanel = new JPanel(new BorderLayout());
        supprPanel.add(labelEventSupprimer, BorderLayout.NORTH);
        supprPanel.add(paneSupprimer, BorderLayout.CENTER);
```

Déploiement

1 Configuration du serveur

Pour configurer et lancer un serveur RMI, on doit d'abord définir l'interface distante qui définit les méthodes distantes que le serveur exposera aux clients (ajouter et supprimer l'événement). Ensuite, on implémente l'interface distante en créant une classe qui fournit l'implémentation des méthodes. Enregistrez l'objet distant dans le registre RMI (rmiregistry) en utilisant `Naming.rebind()`. on peut également configurer le port du serveur RMI et désactiver le pare-feu pour ce port pour n'a pas avoir des problèmes de communication entre le client et le serveur. Pour compiler le code serveur on utilise `javac` "le nom de fichier serveur", cette commande indique les erreurs dans le code serveur. Pour lancer le serveur on utilise `java` "nom de fichier serveur", après cette commande on doit voir l'interface graphique de serveur et un message indique que le serveur est prêt pour les requêtes de client.

```
hajarbabaahmed@ubuntu:~$ sudo su
[sudo] password for hajarbabaahmed:
root@ubuntu:/home/hajarbabaahmed# javac RMIServer.java
root@ubuntu:/home/hajarbabaahmed# java RMIServer
serveur pret
```



The image shows a terminal window with the following commands and output:

```
hajarbabaahmed@ubuntu:~$ sudo su
[sudo] password for hajarbabaahmed:
root@ubuntu:/home/hajarbabaahmed# javac RMIServer.java
root@ubuntu:/home/hajarbabaahmed# java RMIServer
serveur pret
```

Below the terminal window, there is a GUI window titled "serveur RMI". It has two tabs: "Événements ajoutés" and "Événements supprimés". The "Événements ajoutés" tab is currently selected, showing a list of events. The "Événements supprimés" tab is also visible, showing a list of events. The GUI window is empty, indicating that no events have been added or removed yet.

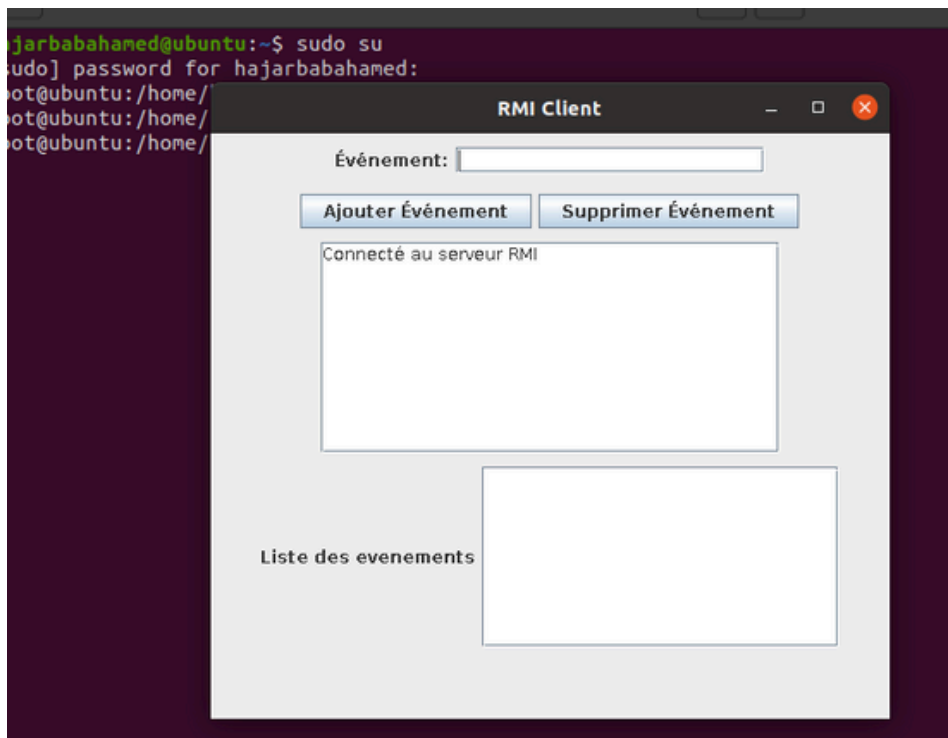
**Année Universitaire:
2024-2025**

Déploiement

1

Configuration du Client

Pour configurer un client RMI, on commence par définir une interface distante spécifiant les méthodes que le client pourra appeler sur le serveur (ajouter et supprimer événements). Ensuite, on implémente cette interface dans une classe qui fournit l'implémentation des méthodes distantes. On doit Enregistrer l'objet distant dans le registre RMI à l'aide de Naming.rebind dans la méthode principale du serveur. Pour le client, on doit initialisé la communication RMI en spécifiant l'adresse et le port du serveur, dans cette partie on doit verifier plusieurs choses pour assurer que la communication entre le client et le serveur va passé, c'est pour ça on doit verifier que l'adresse IP qu'on utilise est correcte , et que le pare-fue ne bloque pas la communication via le port utilisé, puis utilisez Naming.lookup() pour obtenir une référence distante à l'objet distant. Enfin, appelez les méthodes distantes sur cet objet comme si elles étaient locales. Compilez le code client avec javac et exécutez-le avec java, en vous assurant que le serveur RMI est actif



**Année Universitaire:
2024-2025**

Conclusion

Le développement de système de gestion des événements en utilisant Java RMI et une architecture client/serveur a été un succès. Ce projet a permis d'explorer et de mettre en pratique des concepts fondamentaux de la programmation distribuée, tout en illustrant les avantages de Java RMI pour la communication entre applications distantes.

L'utilisation de Java RMI a facilité la communication entre le client et le serveur, permettant au client d'interagir avec l'interface graphique pour ajouter ou supprimer des événements, tandis que le serveur affiche des messages confirmant ces actions. Cette approche a démontré l'efficacité de Java RMI pour développer des applications distribuées robustes et évolutives.