Project Report: Java RMI Chat Application

Narta Matteo

February 18, 2026

1 Utilisation

Clone the github : git@github.com:Narta1/ChatServer.git. Then go in code repository, compile with "javac *.java". Start the server using "java Server" and the clients using "java Client username" or "java ClientGUI username" depending if you want the graphical user interface or not.

2 Introduction

This project implements a distributed chat application using Java RMI (Remote Method Invocation). The system allows multiple clients to connect to a central server, exchange messages, and maintain a shared chat history. The architecture follows a client-server model where remote objects are used to enable communication between distributed components.

3 System Architecture

The application is divided into the following main components:

- **Chat Interface (Chat.java)**: Defines the remote methods that clients can invoke on the server. These include joining the chat, leaving, writing messages, reading messages, and saving the chat history.
- **Chat Implementation (ChatImpl.java)**: Implements the remote interface and extends `UnicastRemoteObject`. It manages connected users, message history, synchronization, and persistence of chat data.
- **Server (Server.java)**: Starts the RMI registry and binds the chat service under a specific name (e.g., "chatServer"). This allows clients to look up and access the remote object.
- **Client and GUI (Client.java, ClientGUI.java)**: The client connects to the RMI registry, retrieves the remote chat object, and interacts with it. The graphical interface provides user-friendly interaction for sending and receiving messages.
- **ChatClient Interface and Implementation**: Allows the server to call back clients (callback mechanism), enabling real-time message updates.

4 Main Functionalities

4.1 User Management

Users can join and leave the chat using the `join()` and `leave()` methods. The server maintains a `HashMap` of connected participants, mapping usernames to their corresponding remote client objects.

4.2 Messaging

When a user sends a message using `write()`, the server stores it in a shared message history (a `List<String>`). Clients can retrieve messages using the `read()` method. The callback mechanism allows the server to notify clients when new messages are available.

4.3 Concurrency Control

Since multiple clients may access the server simultaneously, synchronization mechanisms are used. A `ReentrantLock` ensures thread-safe access to shared resources such as the participants list and message history.

4.4 Persistence

The chat history is saved to a file, allowing messages to be restored when the server restarts. This ensures that previous conversations are not lost.

5 Technologies Used

- Java RMI for distributed communication
- Java Collections Framework (HashMap, ArrayList)
- Concurrency utilities (ReentrantLock)
- Java Swing for graphical user interface

6 Conclusion

This project demonstrates the implementation of a distributed system using Java RMI. It covers key distributed system concepts such as remote interfaces, object binding in a registry, client-server communication, synchronization, and persistence. The use of callbacks enhances real-time interaction, making the chat application efficient and responsive.