Computer Networks Lab Report Template

Francesco Mazzini, 19080; Alice Frezza, 19229; Michael Von Troyer, 19625;

[fmazzini@unibz.it](mailto:fmazzini@unibz.it), [alfrezza@unibz.it](mailto:alfrezza@unibz.it), mvontroyer@unibz.it

1. INTRODUCTION

**Assignment 2**

* 1. Purpose

Lo scopo è di sviluppare un Online Chat (facendo uso della tecnologia RMI di java), in cui multipli Clients possono comunicare tra di loro. Devono poter essere in grado di connettersi, disconnettersi dalla chat, di poter inviare e ricevere messaggi agli/dagli altri client che sono connessi nella Chat Room.

1.2 User needs and Dependencies

Chiunque può essere un utente e non ci sono distinzioni di privilegi. L’applicazione deve essere eseguibile su Windows e MacOS, così come qualsiasi distribuzione di Linux. Ciò sarà possibile data la versatilità del linguaggio in cui sarà scritto: Java. Il progetto non è soggetto ad alcun fattore esterno se non dalle tecnologie di cui sarà composto: Maven (gestore di dipendenze), Git, librerie importate per Maven (JavaFX, altri plugin di Maven riguardanti il suo lifecycle).

1.3 Functional Requirements

L’utente deve poter connettersi alla chat digitando l’indirizzo IP e nickname con la quale vuole esser riconosciuto nella conversazione. L’utente deve poter inviare messaggi e poter leggerli. L’utente deve poter essere in grado di disconnettersi dalla conversazione in qualsiasi momento chiudendo la finestra di dialogo o digitando ‘/quit’.

1.4 Non-Functional Requirements

L’applicazione deve garantire il funzionamento delle suddette funzioni per ogni singolo utente tenendo conto che qualsiasi possibile problema non deve interferire con altre connessioni con altri utenti. Per motivi di sicurezza il server deve mantenere un file di log in cui viene scritto tutto ciò che accade nella chat room compreso l’orario e ogni possibile informazione necessaria.  
I messaggi inviati devono essere recapitati agli altri utenti senza possibilità che essi vengano persi e che vengano recapitati in un ordine diverso da quelli invati. Ciò è già assicurato e intrinseco nella tecnologia RMI di Java, essendo che si basa sul protocollo TCP, che garantisce le suddette proprietà.

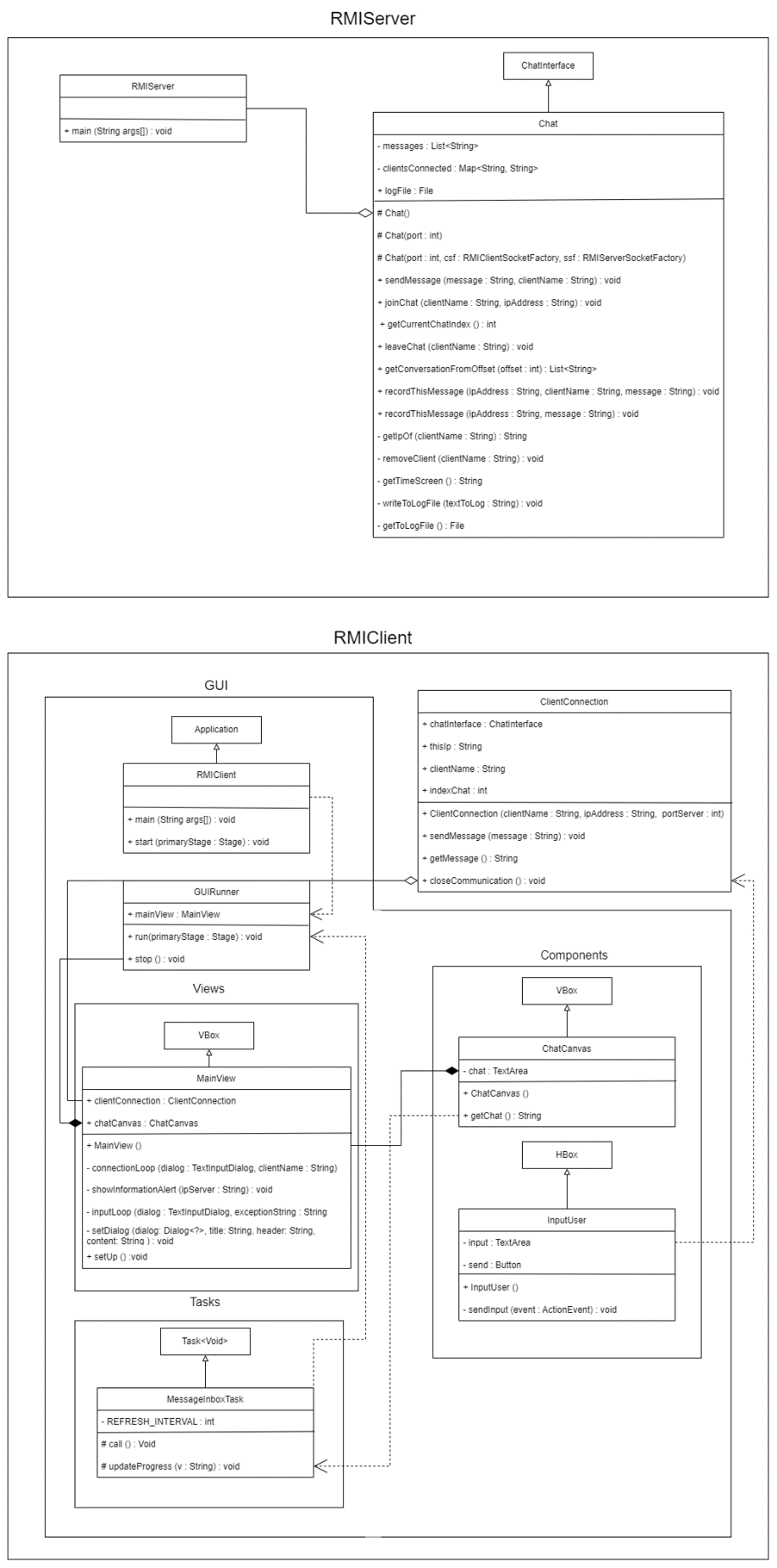
2. METHODOLOGY

Per soddisfare i requisiti richiesti, si è sviluppata un’applicazione Maven scritta in Java, la quale contiene sia il programma del server che il programma del client, in due folder separati.

Il programma del server gestisce la connessione offrendo semplicemente dei metodi, definiti dall’interfaccia ChatInterface. Tramite ciò il server sarà in grado (in modo non esplicito) di inviare i messaggi ricevuti dalla connessione a tutti gli altri utenti connessi e di salvare nel file di log i messaggi inviati da quella connessione.

Il programma del client si occupa con la classe RMIClient di avviare la GUI gestita da GUIRunner. Essa definisce le caratteristiche principali dell’interfaccia utente da un punto di vista funzionale, andando poi ad avviare la MainView che sarà la prima vera finestra che apparirà all’utente. MainView appena prende il controllo dell’esecuzione, chiede all’utente tramite dei Dialogs con quale nome si voglia connettere e a quale IP. Ciò gli permette di istanziare una ClientConnection (class dedicata a mantenere tutte le informazioni della connessione col server) per poi riprendere il flow di esecuzione e creare in maniera definita l’interfaccia utente che verrà usata durante la chat. Questo viene fatto istanziando all’interno della finestra la ChatCanvas (che gestisce graficamente l’update dei messaggi in arrivo) e l’InputUser che invece gestisce graficamente e funzionalmente l’invio di messaggi da parte dell’utente al server. Siccome il flow di esecuzione è mantenuto dalla classe InputUser per l’invio di messaggi, ChatCanvas farà riferimento a un thread parallelo di tipo grafico, chiamato MessageInboxTask. Esso assume una struttura leggermente diversa dal solito Thread, perché appunto in grado di modificare componenti grafiche durante l’esecuzione, cosa non possibile tramite i normali thread. Si metterà quindi in ascolto di messaggi e eseguirà l’update grafico non appena saranno ricevuti.

È possibile visualizzare la struttura del programma nel diagramma UML che segue.



1. PROGETTAZIONE ALGORITMO

Per rendere possibile un’interazione tra server e chat client, l’applicazione server deve essere avviato prima che il client si connetti. Il server crea un’istanza di un oggetto remote che implementa l’interfaccia ChatInterface e un’istanza di Registry che ascolta sul port 7896. Dopodiché il server associa l’oggetto remote con l’istanza di Registry. Il port sul quale questa istanza ascolta è impostato in modo statico.

Quando un utente lancia l’applicazione client, viene chiesto di scegliere un nickname e di specificare l’IP del server a cui si vuole connettere. Il client invoca un metodo dell’oggetto remote per passare il suo nickname e la sua IP. Questi dati vengono salvati sul server. In seguito, il server si salva nella propria memoria, tra i messaggi, la notifica che un nuovo cliente si è connesso alla chat, in modo che gli altri client saranno in grado di saperlo non appena richiederanno la chat.

Una limitazione dell’applicazione è che i dati di un cliente vengono sovrascritti se un altro utente si connette alla chat con un nickname già registrato. Ogni client richiede periodicamente la parte della chat che non ha ancora ricevuto. Questo costituisce un overhead di computazione siccome potrebbe essere il caso che per la maggior parte delle richieste non ci sono messaggi nuovi, tuttavia l’aggiornamento lato Client avviene ovviamente soltanto quando questa richiesta presenta un contenuto di risposta (quindi quando ci sono nuovi messaggi). Questo sistema è stato fatto per ovviare al fatto che il server non può contattare direttamente i client per una notifica, a meno che non siano anche loro dei simil Server. A scopo dell’esercitazione, si è preferito mantenerli Client (aggiungendo questo possibile overhead) per una questione concettuale di ruoli dei due.

Quando una finestra client dell’utente X viene chiusa, o un user manda il messaggio “/quit”, i dati associati al client vengono rimossi dal server e l’oggetto remote manda il messaggio “<client name> has left the chat” per informare tutti gli utenti ancora online del fatto che X si è disconnesso dalla chat. Questo atto termina il rapporto tra il cliente che si è disconnesso e il server.

1. TESTING FEEDBACK

Durante l’implementazione del progetto non siamo stati confrontati con tanti problemi o bugs. Il maggior problema che abbiamo incontrato era che la prima versione del client che avrebbe dovuto prendere solo la parte nuova dei messaggi della chat non funzionava perché il tipo ritornato non era una lista ma un View di una lista. Un semplice casting al tipo ArrayList ha risolto il problema.

In più all’inizio la connessione falliva perché mancava il port nella stringa che rappresenta il nome associato all’oggetto remote. Una breve ricerca e alcuni esperimenti sono bastati per risolvere questo problema.

Un’altra questione che ha suscitato un po’ di dubbi e discussioni iniziali è stata quella del fatto (anche già accennato) che tramite RMI il Server non può contattare di sua spontanea volontà un Client a meno che anch’esso non implementi un’interfaccia di un oggetto Remote, sul quale essere contattato. Avevamo trovato due possibili soluzioni, quella di rendere il Client anch’esso Server e l’altra di aggiungergli un altro Thread che si occupasse della richiesta della parte di Chat in maniera continua. Abbiamo optato per la seconda per seguire il concetto del paradigma Client Server, anche se in una situazione reale probabilmente opteremmo per la prima visto che è più efficiente.

1. Chat Room TCP vs RMI

Questa versione della Chat Room basata su RMI (Remote Method Invocation) presenta numerose differenze rispetto alla precedente, basata invece direttamente su TCP/IP. Nella versione precedente dovevamo specificare esattamente quale socket usare, se vogliamo TCP o UDP, gestire la formattazione dei messaggi che passano da client a server, ecc. Mentre con RMI la maggior parte del codice riguardo la rete, come appunto la formattazione dei messaggi e quale porta utilizzare (anche se comunque si può specificare) viene gestito automaticamente.

I miglioramenti dell’usare RMI sono:

* l’implementazione è più semplice e veloce. Nel nostro codice si può infatti notare che abbiamo scritto molto meno codice nell’applicazione basata su RMI;
* è più semplice espandere e modificare il livello di communicazione rete del programma.

Per quando riguardo i svantaggi di RMI:

* RMI gestisce soltanto la comunicazione tra programmi Java. Invece con TCP/IP se si vuole interfacciare un programma esistente che comunica attraverso socket non importa il linguaggio, ma che la formattazione del messaggio è corretta;
* RMI ha una efficienza minore rispetto a TCP/IP.