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 3**

* 1. Purpose

Lo scopo è di sviluppare un software distribuito che si occupi, dato un valore hash, di trovare il valore originario, dividendo il carico del compito a diversi client che si connettono per contribuire alla risoluzione del task.

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.

1.3 Functional Requirements

Un Client deve potersi connettere per la risoluzione del problema. Potrà essere un Client Master, cioè colui che prende il problema e lo divide tra gli altri ma collabora comunque alla sua risoluzione, oppure un Client Slave, che invece si occuperà solo di ricevere una porzione del problema da risolvere. La connessione deve essere fatta specificando IP e tipo di client.

1.4 Non-Functional Requirements

L’applicazione deve garantire il funzionamento delle suddette funzioni per ogni singolo utente tenendo conto che qualsiasi possibile problema deve essere contenuto il più possibile, in modo da non estendersi in maniera diretta a tutto la “rete” dei dispositivi che partecipano a risolvere il problema. Chiaramente data la natura del task, che fa sì che ognuno abbia una porzione del task, garantire un isolamento totale è relativamente non possibile e, sebbene dei miglioramenti potrebbero esser fatti in tal senso, si è deciso volontariamente di non intraprenderli perché non si vuole appesantire maggiormente il software, che è pensato per girare su dispositivi con limitate capacità di calcolo.

Pertanto si tiene anche in considerazione che il software deve essere relativamente leggero e in grado di girare il più velocemente possibile su dispositivi con limitate capacità di calcolo (ad esempio Raspberry Pi1).

2. METHODOLOGY

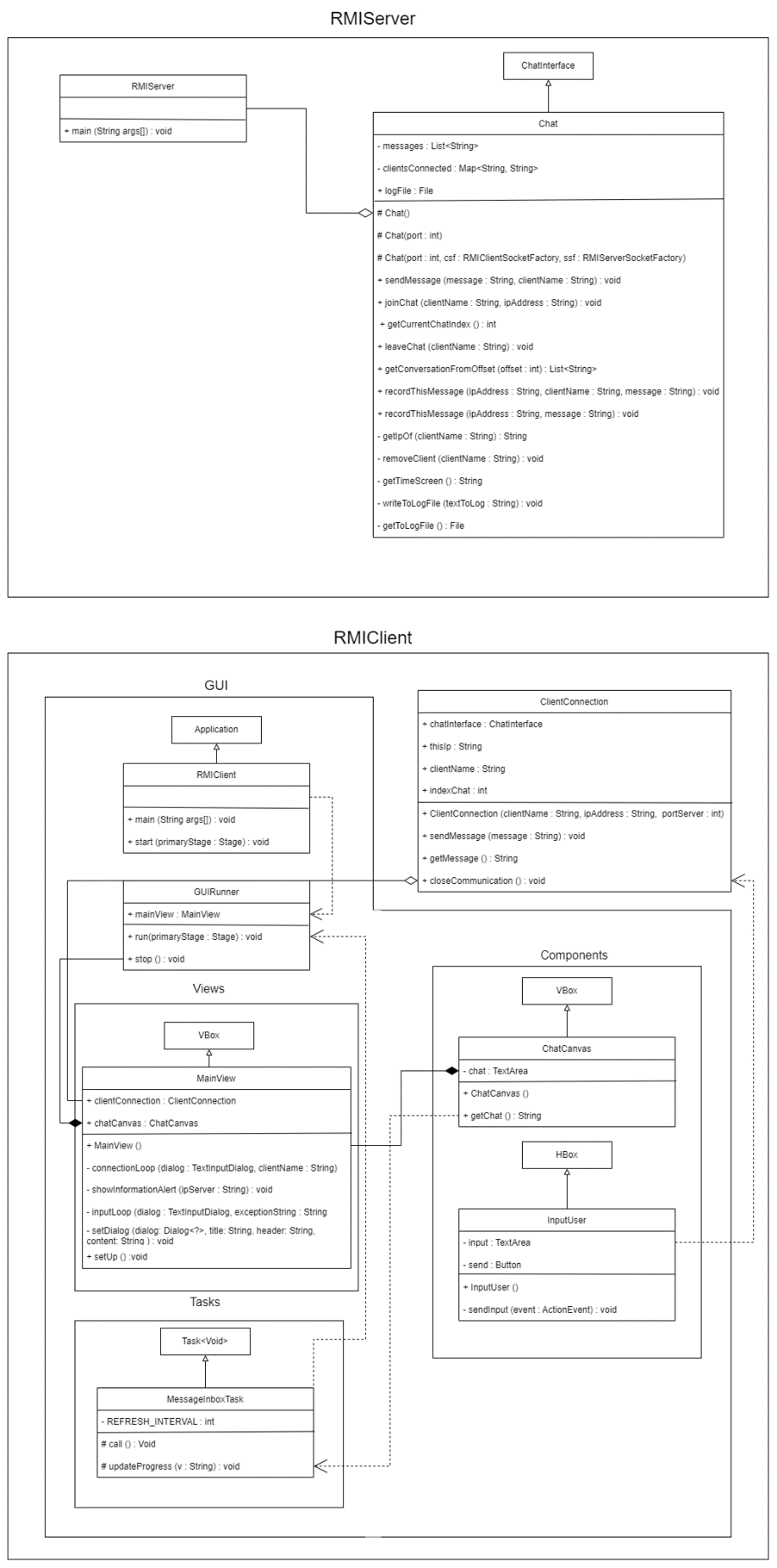
Per soddisfare i requisiti richiesti, si è sviluppata un’applicazione scritta in Java, la quale contiene un programma Client che gestisce sia il caso in cui esso sia Master che Slave.

La classe Client si occuperà dell’esecuzione principale del programma gestendo il comportamento del client, sia in caso Master che Slave. ClientCommHandler definisce invece l’implementazione dei metodi RMI che offre il client nei confronti del Server che gestisce la gara di risoluzione di problemi. Questi metodi sono dichiarati nell’Interface ClientCommInterface.

Il Client contiene anche le implementazioni dei metodi RMI che offre il Client a se stesso, una volta in quanto master e una volta in quanto slave. Questi metodi sono dichiarati nelle Interfaces MasterCommInterface e SlaveCommInterface.

------------DA RIVEDERE-------------

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



--------------------- UML NON CORRETTO PER QUESTO ASSIGNMENT-----------------------------

1. PROGETTAZIONE ALGORITMO

Il programma deve essere avviato specificando negli args l’ip del server che restituisce il problema, l’ip proprio del dispositivo, il nome del team (che parteciperà alla risoluzione del problema) e con che modalità di tipologia di client deve essere avviato. Fatto ciò, se master si occuperà di ricevere iscrizioni da parte di slave, attendere l’arrivo del problema dal server e di suddividere il problema, assegnandolo a tutti gli Slave iscritti, per poi partecipare a risolverlo. Se slave, si occuperà di iscriversi al master e provare a risolvere una porzione del problema comunicando, in caso la soluzione.

L’algoritmo specifico per la risoluzione e suddivisione del problema opera nel seguente modo. Il master suddivide equamente tra gli slave il problema in varia porzioni, ciò che avanza (per via del quoziente non intero) viene preso da esso insieme alla sua porzione. Sebbene ciò non è “giusto” e considerabile il modo più efficiente (in quanto il master ha già del carico in più di esecuzione, ovvero quello di gestione), è stato pensato tuttavia che una gestione più “equa” della suddivisione caricherebbe maggiormente il Master, per un guadagno in termini di efficienza di poco conto. Fatta la divisione tutti i Client controllano che il problema non sia già presente in una loro HashMap che si gestiscono autonomamente. Se ciò non è il caso, iniziano un algoritmo di Brute Force per risolvere il problema, partendo dal left edge dell’intervallo a loro assegnato e aumentando di un numero questo numero per poi salvare il risultato nell’HashMap, per poi controllare se corrisponde alla soluzione del problema. In questo caso gli Slave lo passano al master e il Master lo passa al Server avvertendo tutti che la soluzione è stata trovata. Ciò avviene anche nel caso in cui sia stato un altro team a trovare la soluzione e quindi il server comunica un nuovo problema. Ciò si ripete costantemente.

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

-possiamo scrivere HashMap se non va più

--- ?