Relazione Progetto 2017/208
Laboratorio Programmazione di Rete

Setzu Simone [508366] Vatteroni Francesco [468134]

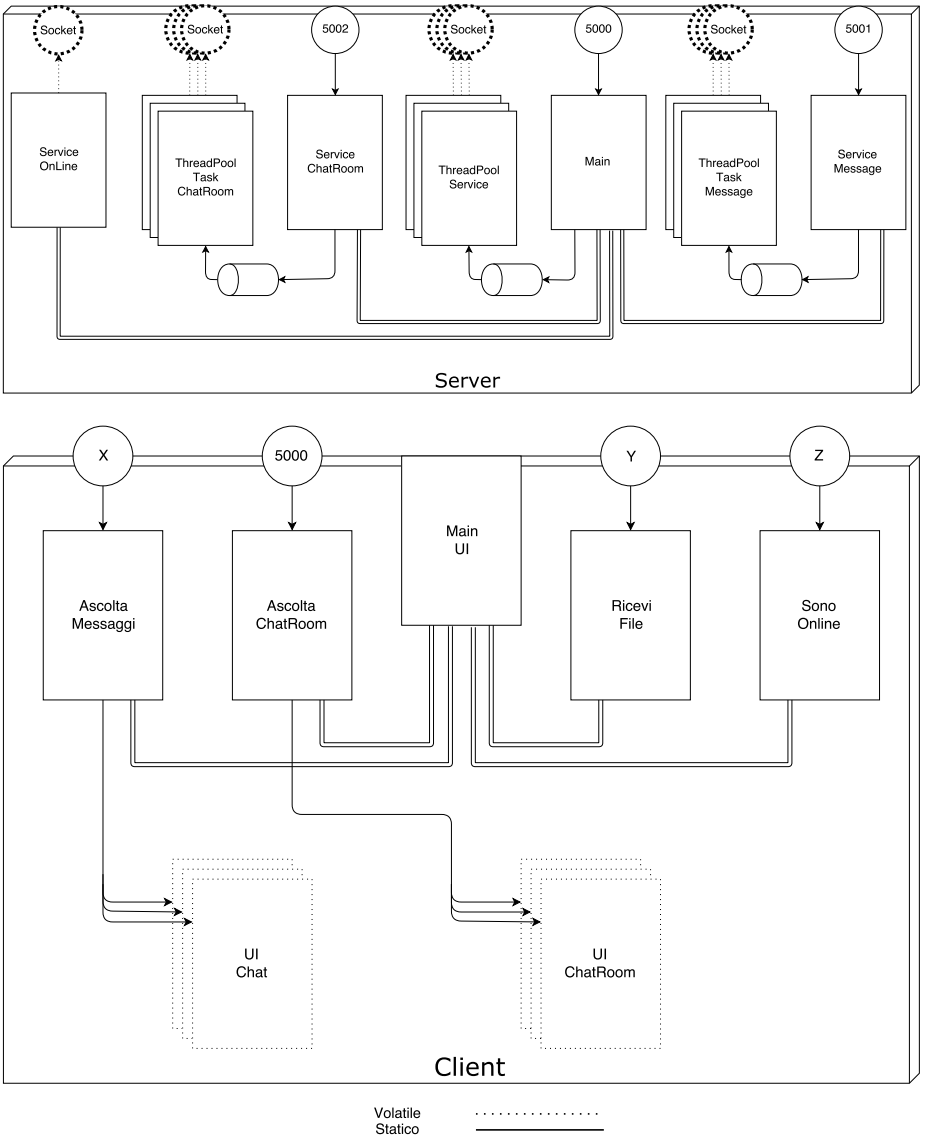
2018/01/12 v1.0

Indice

1 Intro

Lo abbiamo fatto in laboratorio tra un caffè da 0.25€ e l'altro, buona lettura.

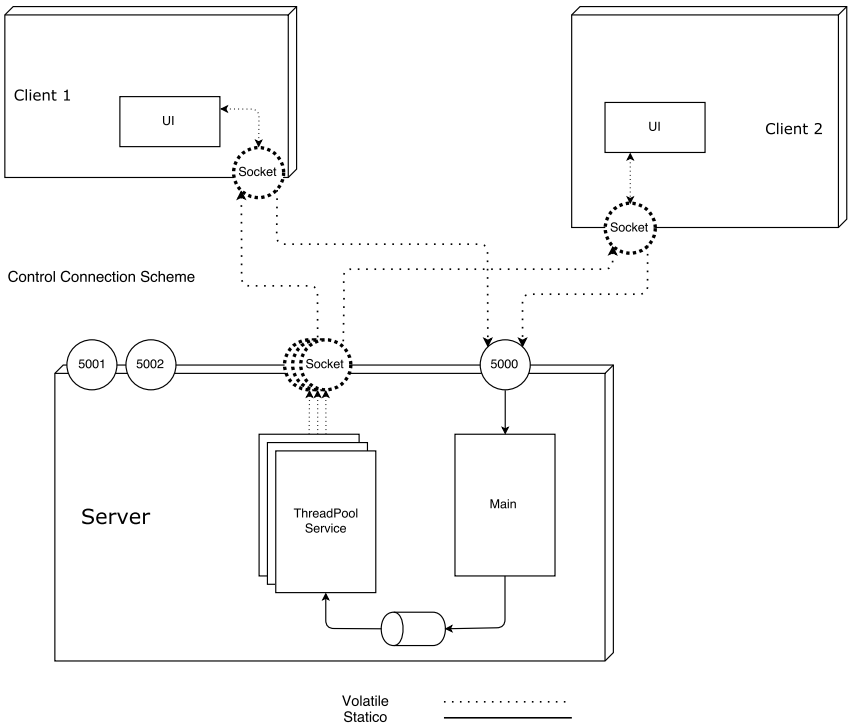
2 Overview



La realizzazione di questo progetto è stata strutturata in modo tale che le comunicazioni tra Client e Server avvengano tramite connessioni volatili. Il Server riceve richieste su porte predefinite, le quali sono necessarie per soddisfare la ricezione di operazioni di controllo, messaggi privati e messaggi destinati a gruppi. Tale preferenza comporta un utilizzo della rete ridotto rispetto all'uso di connessioni stabili, in quanto non verrà saturata la rete da connessioni che non richiedono alcuna operazione. Di contro, il tempo per ogni operazione è maggiore, in quanto si ha un overhead dovuto dall'apertura di una nuova connessione per ogni operazione. L'utilizzo di questo tipo di connessioni comporta il problema di testare lo Status di un determinato Client poichè, non essendoci connessioni stabili, un Client online potrebbe non avere connessioni attive. Abbiamo risolto

questo problema instaurando ciclicamente tra Server e Client (più precisamente ogni 10 secondi), una connessione TCP su una porta data, la quale, in caso di fallimento, causa il setting dello Status a offline. Questa operazione è gestita da un thread apposito, creato dal Server in fase di inizializzazione. Tale thread ripete l'operazione per ogni utente registrato. Inoltre, vengono utilizzati tre ThreadPool dedicati per le operazioni su chat private, chatroom e operazioni di controllo. Un ulteriore problema è quello dato dal conflitto delle porte in locale, risolto aprendo i socket sulle prime porte disponibili ed inoltrando al Server le porte su cui il Client sta in ascolto, dopo aver generato 4 thread per la ricezione delle comunicazioni dal Server. Si è scelto di salvare tutti gli Utenti e le ChatRoom in due ConcurrentHashMap per evitare problemi di race conditions, dovuti dall'accesso simultaneo di più Thread del Pool alle strutture. Si utilizza un unico server RMI che invia notifiche a tutti i Client. A seconda dei parametri di tali notifiche, sono i client stessi a filtrarle e visualizzarle. Per poter rendere disponibile l'elenco delle lingue nel pannello di registrazione viene scaricata la pagina `http://www.loc.gov/standars/iso639-2/php/code_list.php` attraverso una richiesta HTTP e viene successivamente parsato l'html utilizzando la libreria esterna JSOUP. Per poter utilizzare il servizio di traduzione si effettua una richiesta HTTP all'indirizzo `https://api.mymemory.translated.net/get?=`, includendo i parametri opportuni. Il messaggio, prima di essere inviato come parametro, viene parsato, sostituendo tutti gli spazi con la stringa %20. La risposta che viene ricevuta è in formato JSON: da questa, dopo l'opportuno parsing, viene estratto il messaggio tradotto, il quale viene inoltrato al destinatario. La manipolazione degli indirizzi di multicast avviene mantenendo fissi i primi 3 byte e generando in modo incrementale il restante byte, in modo che possano essere aperte contemporaneamente fino a 256 chatroom.

3 Connessione di Controllo

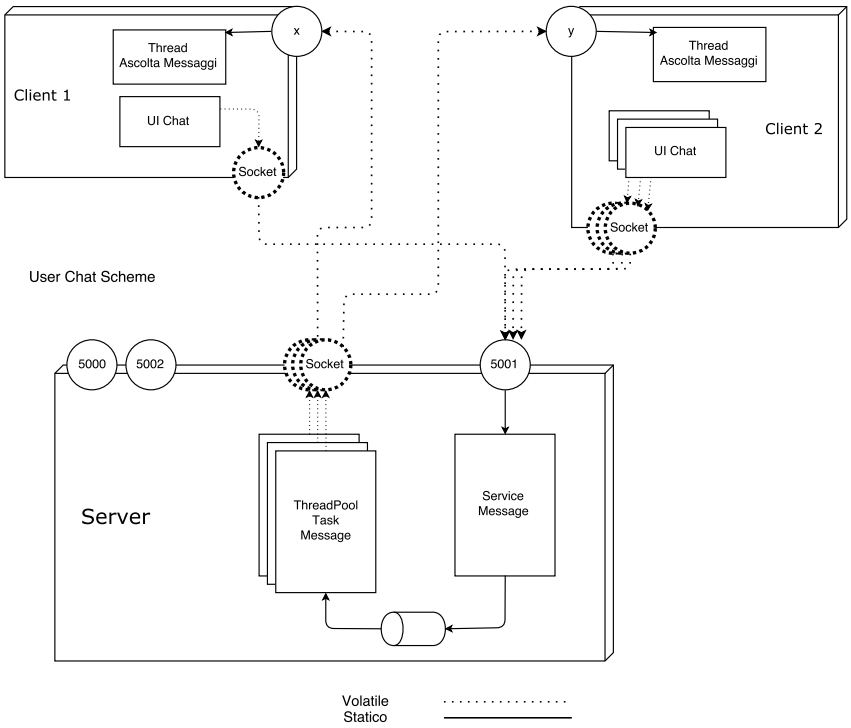


La connessione di controllo avviene tramite l'utilizzo di una connessone TCP, accettata dal main Thread del Server. Quest'ultimo crea un Thread Service che si occupa di soddisfare la richiesta entrante sulla connessione e, successivamente, di chiudere quest'ultima implementando, di fatto, una comunicazione a domanda-risposta. In questo caso è stato scelto di usare un ThreadPool perchè si presuppone che la gestione delle richieste di operazioni di controllo abbiano un costo computazionale molto importante per il Server, tale che avrebbe generato lo spawning di un numero eccessivo di thread. Il formato dei messaggi è di tipo JSON, strutturato in maniera dinamica a seconda della chiave che identifica il tipo di operazione richiesta. Le operazioni di controllo sono le seguenti:

- Login: riceve la tupla <nome utente, password, IP, porta messaggi, porta file, porta online>. Dopo una verifica della correttezza di nome utente e password, l'operazione di login effettua un aggiornamento dei campi relativi ad IP e porte ed imposta lo stato dell'Utente a online;
- Registrazione: riceve la tripla <nome utente, password, lingua> e, dopo aver controllato l'unicità del nome utente, lo aggiunge alla HashMap degli utenti;
- Ricerca Utente: riceve una stringa e restituisce, facendo pattern matching, la lista degli utenti di cui è prefisso;

- Richiesta Amicizia: prende come parametro la coppia <mittente, destinatario> che, dopo opportuni controlli, comporta l'aggiunta di un arco nel grafo delle amicizie tra il destinatario e il chiamante. La notifica viene inviata al destinatario attraverso RMI;
- Lista Amici: riceve il nome utente e restitisce la lista di utenti con cui è amico;
- Lista ChatRoom: restituisce la lista delle chatroom che sono attive in quel momento;
- Creazione ChatRoom: riceve la coppia <nome chatroom, creatore>. Dopo aver verificato l'unicità del nome la aggiunge alla HashMap e restituisce l'IP su cui è stata registrata;
- Lista Iscritti Chatroom: riceve la coppia <nome chatroom, richiedente> e viene restituita la lista degli utenti e le informazioni se il richiedente è il creatore o/e se è iscritto;
- Iscrizione ChatRoom: riceve la coppia <nome chatroom, richiedente> , aggiunge il richiedente alla lista di utenti iscritti alla chatroom e restituisce l'IP su cui dovrà registrarsi il client per ricevere i messaggi;
- Chiusura ChatRoom: riceve la coppia <nome chatroom, richiedente> , rimuove la chatroom dalla HashMap, ne restituisce l'IP su cui il client dovrà deregistrarsi ed invia una notifica di chiusura a tutti gli utenti iscritti;
- Lascia ChatRoom: riceve la coppia <nome chatroom, richiedente> , rimove l'utente dalla lista degli iscritti alla chatroom e restituisce l'IP su cui dovrà deregistrarsi il client;
- InviaFile: riceve la coppia <mittente, destinatario> e restituisce la coppia <IP, porta> sulla quale poi il client dovrà aprire una connessione per trasmettere il file;
- Exit: riceve nome utente che porta ad impostare lo status a offline.

4 Chat Utenti



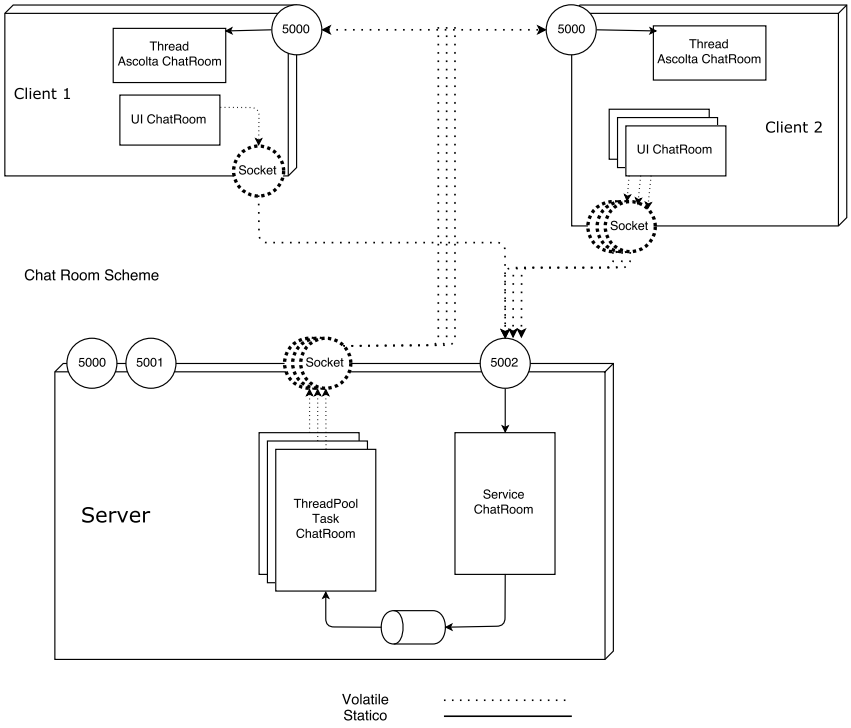
La comunicazione privata tra due utenti avviene tramite l'utilizzo di differenti connessioni TCP, strutturate come segue:

- Il Server è in ascolto con un Thread sulla porta 5001. Questo Thread ha il compito di accettare le connessioni e di passare il messaggio ad un ThreadPool, il quale si occupa dell'inoltro del messaggio al destinatario;
- Il Client è in ascolto con un Thread su una porta scelta all'avvio dell'applicazione tra quelle ancora disponibili. Tale Thread ha il compito di accettare le connessioni e di passare il messaggio all'Interfaccia Grafica, la quale si occupa di visualizzarlo;
- Il Client, per inviare un messaggio, apre una connessione sulla porta 5001 del Server, invia il messaggio ed infine chiude la connessione.

Il formato dei messaggi è di tipo JSON ed è strutturato come segue:

- messaggio
  - corpo
  - mittente
  - destinatario

5 Chat Gruppi



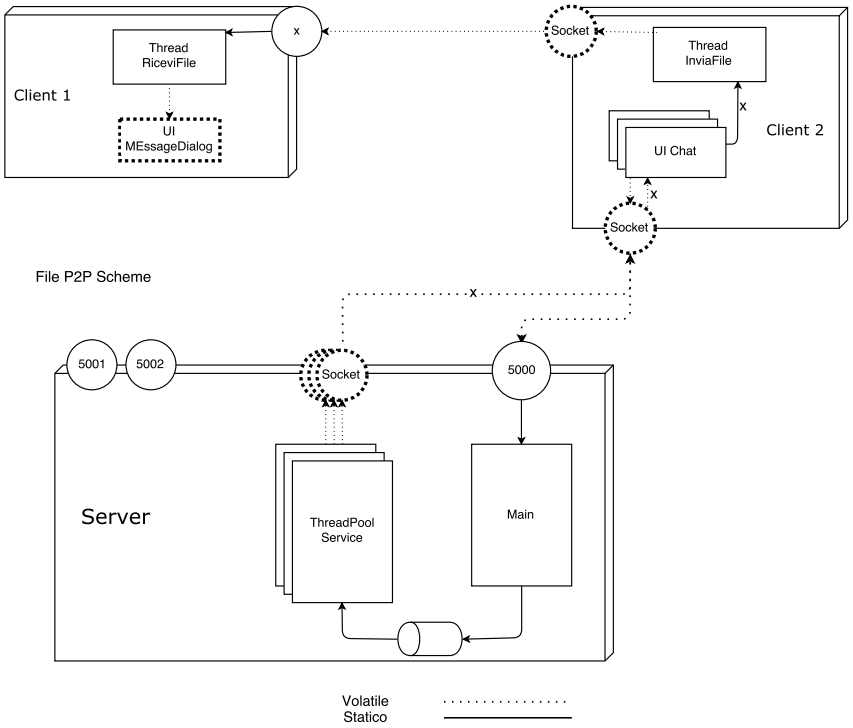
La comunicazione privata all'interno di una ChatRoom avviene tramite l'utilizzo del protocollo UDP ed è strutturata come segue:

- Il Server è in ascolto con un Thread sulla porta 5002. Questo Thread ha il compito di accettare pacchetti e di passare il messaggio ad un ThreadPool, il quale si occupa dell'inoltro del messaggio sull'IP Multicast relativo a quella ChatRoom;
- Il Client è in ascolto con un Thread sulla porta 5000. Tale Thread ha il compito di accettare pacchetti e di passare il messaggio all'Interfaccia Grafica, la quale si occupa di visualizzarlo nel campo apposito;
- Il Client, per inviare un messaggio, invia il pacchetto sulla porta 5002 del Server.

Il formato dei messaggi è di tipo JSON ed è strutturato come segue:

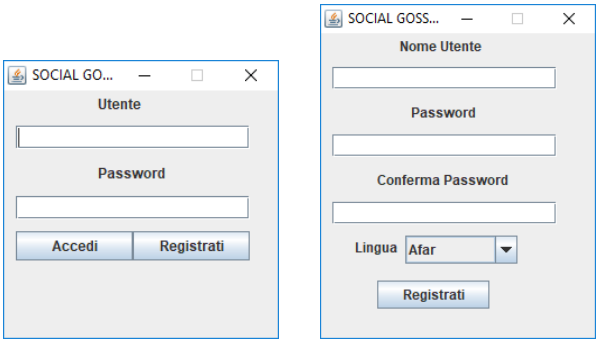
- messaggio chatroom
  - mittente
  - body
  - destinatario

6 Trasferimento File



Per inviare un File il Client richiede, attraverso una connessione di controllo, la coppia (IP,Porta) del Client destinatario. Successivamente, il Client richiedente crea un thread allo scopo di aprire una connessione utilizzando le informazioni ricevute. Il thread, infine, spedisce il File utilizzando NIO. La scelta di creare un nuovo thread per ogni trasmissione di file deriva dall'intento di non lasciare l'interfaccia del Client in attesa su tale operazione per un tempo che risulterebbe eccessivo e fastidioso all'utilizzatore. Ogni Client è in ascolto con un Thread su una porta scelta all'avvio dell'applicazione tra le porte disponibili. Il Client accetta quindi connessioni sulla porta scelta per ricevere File. Il File ricevuto viene salvato nella cartella Download dell'utente con il nome SOCIAL-GOSSIP TimeStamp.

7 Interfaccia Utente



All'avvio dell'applicazione Client, si presenta una finestra da cui è possibile inserire credenziali ed effettuare il login o, premendo sul bottone registrati, generare una nuova finestra da cui è possibile effettuare la registrazione.

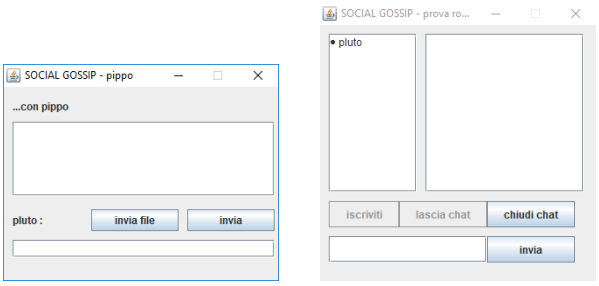
Una volta effettuata l'operazione di login si presenta una finestra da cui è possibile consultare:

- la propria lista amici;
- la lista di tutte le chatroom;
- il pannello delle notifiche;
- il pannello di ricerca utenti, in cui la ricerca avviene facendo pattern matching sulla parte del nome già inserita;

e si può interagire con essa

- facendo doppio click su un nome di un amico: in questo caso si apre una finestra di chat privata con esso
- facendo doppio click su un nome di una chatroom: si apre la finestra relativa a quella chatroom
- premendo il tasto crea chatroom: appare una dialog per poter inserire il nome della chatroom da creare

- premendo il tasto chiedi amicizia: in quest'ultimo caso, viene inviata la richiesta di amicizia all'utente selezionato.



Le finestre chat e chatroom sono organizzate in due ConcurrentHashMap, in modo che, alla ricezione di un messaggio, questo venga indirizzato alla finestra corretta. Qualora tale finestra ancora non esista, viene creata all'uopo. Nella finestra di chat, oltre alla ovvia possibilità di inviare e ricevere messaggi istantanei, si può inviare un file all'utente con cui si sta chattando. Dalla finestra di chatroom si possono inviare messaggi e leggere gli interventi degli altri partecipanti nelle discussioni cui si è iscritti. Solo nel caso in cui si sia il creatore della chatroom, la si può chiudere. Ogni iscritto può abbandonare la chatroom in qualunque momento. Nel momento in cui un gruppo viene chiuso dal suo creatore, viene inviata una notifica RMI ad ogni client attivo sulla chatroom. Oltre a notificare l'avvenuta cancellazione della chatroom, la chiamata RMI chiude anche la finestra su cui essa è aperta per ogni client.

8 Istruzioni per l'Uso

Il Server si fa partire dalla classe Server.
I Client si fanno partire dalla classe GUIclient.
Si è scelto di tenere tutti i file nello stesso package perchè eseguendoli su progetti differenti si andava incontro ad un problema di sicurezza legato all' esportazione del classpath. Siamo consapevoli del fatto che il Client e il Server debbano essere eseguiti su macchine differenti ma dopo svariati tentativi e ricerche online non siamo riusciti a risolvere questo problema.