Relazione Progetto di reti TURING

Marcello Protopapa, Mat. 517874

Agosto 2019

1 Architettura del sistema

Il sistema implementa un'architettura client server con due eseguibili separati (MainClass_Client e MainClass_Server). Il client implementa un'interfaccia grafica in Java Swing con cui l'utente interagisce per mandare richieste al server.

1.1 Architettura del Client

Il client implementa il pattern Model View Controller, in cui la view è data da Java Swing tramite le classi del paccehtto `client.gui`, il modello viene inviato dal server e la classe TURINGClient fa da controller. Dall'interfaccia grafica, infatti, partono richieste per TURINGClient, che vengono inoltrate al server. La risposta arriva alle classi ChannelReader o ChatServer e viene passata a TURINGClient che aggiorna l'interfaccia grafica.

1.2 Architettura del Server

Il server utilizza un Selector per la lettura e scrittura da socket. Le richieste arrivate dal socket sono passate ad un thread in un threadpool che le risolve e prepara una risposta, che poi è spedita al destinatario dal main thread.
La classe FilesManager mantiene informazioni su quali file sono attualmente aperti dagli utenti.

1.2.1 Filesystem

Tutta la parte dell'interazione con il filesystem è gestita dalla classe (statica) UsersManager e utilizza `java.nio`. I file sono salvati nella directory TURING-Files generata nel percorso di esecuzione del server, in cui ogni utente ha una sottocartella con il suo nome. La cartella di un utente contiene i documenti di cui l'utente è autore e dei dati che riguardano l'utente stesso.
I documenti sono formati da una cartella "nomefile.TURINGFile" che a sua volta contiene dei file di testo "n.section" con n numeri progressivi da 0 al numero delle sezioni nel documento - 1. Essendo le sezioni salvate su file separati su disco, non si pone il problema di scritture concorrenti/sovrascritture.
I dati dell'utente sono salvati nel file "nomeutente.data", che è un file JSON

con due campi stringa ("username" e "password") e un campo array di stringhe "invites" che mantiene una lista dei documenti che l'utente è stato invitato a modificare. In generale, un documento di cui un utente non è proprietario è descritto nel programma come "nome del proprietario/nome del documento".

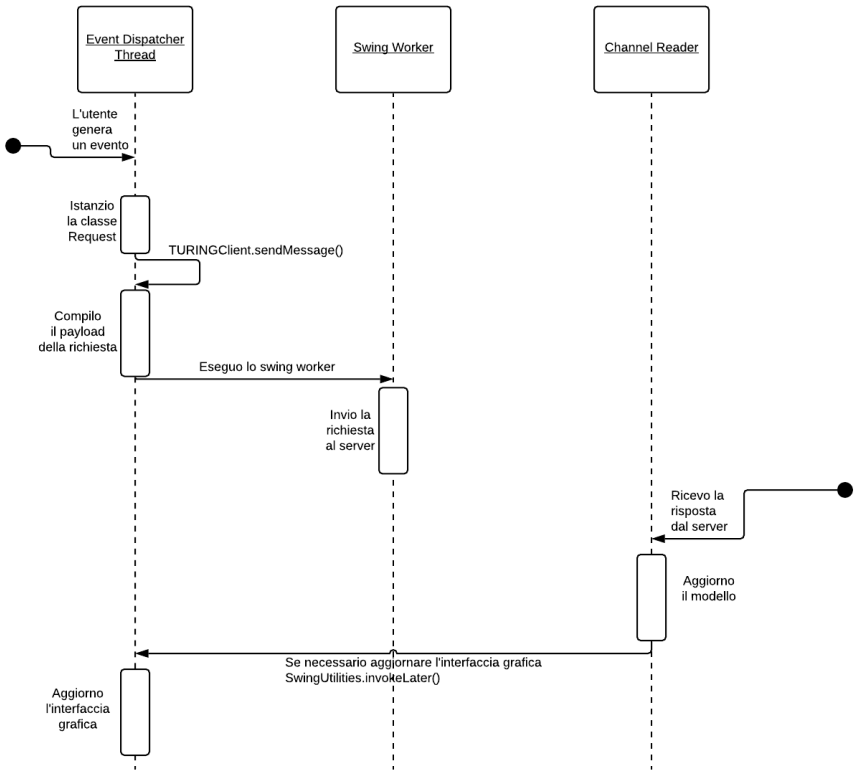
1.3 Comunicazione

La registrazione è fatta come da specifica tramite RMI. L'interfaccia `share.TURINGRegister` è condivisa tra client e server. L'implementazione esposta effettivamente dal server è `server.RegistrationServer`.
I messaggi sono istanze della classe Request. La comunicazione avviene scrivendo sul canale TCP il numero di caratteri che verranno inviati, seguito dalla stringa JSON corrispondente alla relativa istanza di Request (questo per rendere la comunicazione più leggera rispetto alla serializzazione di Java ed indipendente dal linguaggio di programmazione). Sia il client che il server utilizzano la classe `share.NetworkInterface` per lo scambio di messaggi, in modo che il formato sia sempre lo stesso a prescindere dalle modifiche fatte al protocollo.
La chat è implementata con multicast UDP. Nel momento in cui un utente apre un file, il server gli invia l'indirizzo di multicast della relativa chat, eventualmente generandone uno se nessuno sta già modificando il file. Gli indirizzi sono generati sequenzialmente nel range 239.255/16. Dopo questa fase, il client esegue un'istanza del chat server che utilizza per mandare e ricevere messaggi.

2 Thread e strutture dati

2.1 Client

Al lancio dell'applicazione il client fa partire l'event dispatch thread di Java Swing istanziando ClientGUIHandler (che estende JFrame). Dopodichè al momento del login viene anche fatto partire il thread che si occuperà della lettura dal canale (che istanzia la classe ChannelReader). Thread worker di Java Swing vengono attivati al momento dell'invio del messaggio come mostrato in figura:



In questo modo il thread event dispatcher si occupa solo di gestire l'interfaccia grafica e le comunicazioni di rete (per loro natura lente) vengono delegate ad appositi thread. Anche la richiesta di registrazione viene fatta in uno SwingWorker. La chat infine funziona in modo simile ai messaggi del server, ma con un thread che che esegue la classe ChatServer e che legge i messaggi dal canale di multicast UDP anzichè dal server. Il client non utilizza strutture dati a cui accedono più thread.

2.2 Server

Al lancio dell'applicazione il thread principale crea un thread pool, dopodichè entra nel loop del selector. Da questo punto in poi il thread principale si occupa solo di gestire le comunicazioni di rete fino alla chiusura.
Le richieste sottomesse al thread pool vengono processate ed inserite in coda per la scrittura tramite `key.attach()` (in un array di Request convertite in stringhe JSON) dove key è la SelectionKey che ha fatto partire la richiesta, che viene poi marchiata perchè il selettore la consideri per la scrittura. Il thread principale scriverà poi sul canale il contenuto dell'attachment un messaggio alla volta nel momento in cui esso verrà scelto dal selettore perchè è diventata pos-

sibile l'operazione di scrittura. Poichè la chiave può essere utilizzata da più thread contemporaneamente, le operazioni su di essa sono sincronizzate. Per quanto riguarda le strutture dati, gli utenti connessi sono salvati in una ConcurrentHashMap che va da stringhe (nomi utenti) a LoggedUser (classe che contiene solo nome utente e SelectionKey) perchè vengono acceduti dai thread del thread pool. Per lo stesso motivo, la classe FilesManager mantiene la lista dei file aperti in una ConcurrentHashMap che va da stringhe (i nomi dei file) a ConcurrentLinkedQueue di ServerFile (che contiene file con lo stesso nome ma autori diversi).

3 Classi definite

3.1 client

MainClass_Client Classe che contiene il main. Non fa altro che istanziare TURINGClient.

TURINGClient La classe principale del client. Rappresenta il controller nel pattern Model View Controller. Contiene metodi per mandare messaggi ed aggiornare l'interfaccia grafica, oltre che per aprire e chiudere la connessione.

NetworkHandler Implementa le operazioni di rete ad alto livello appoggiandosi a `share.NetworkInterface`.

ChannelReader Classe che estende thread e che si occupa solo della lettura di messaggi dal server. Il metodo run è un loop che legge dal server finchè non trova un errore e, se la lettura ha avuto successo, aggiorna il modello chiamando `TURINGClient.update`.

ChatServer Classe che si occupa delle comunicazioni di chat. Estende Thread e il metodo run è un loop in cui vengono letti messaggi dal MulticastSocket, che vengono poi passati alla classe TURINGClient. Ha inoltre un metodo per inviare un messaggio di chat sul socket.

ClientActionListener* ClientActionListenerLogin, ClientActionListenerMenu e ClientActionListenerDocument sono i listener utilizzati dall'interfaccia grafica delle relative schermate. Implementano `Java.awt.event.ActionListener` e il metodo actionPerformed crea una richiesta (il cui tipo dipende dal bottone premuto) e chiama `TURINGClient.sendMessage()`.

3.1.1 client.gui

ClientGUIElement Interfaccia che definisce una schermata dell'UI.

ClientGUILogin Schermata relativa al login. Il costruttore aggiunge gli elementi all'interfaccia. Contiene metodi getter per i campi di testo.

ClientGUIMenu Schermata relativa al menu dell'applicazione. Il costruttore aggiunge gli elementi all'interfaccia. Contiene un setter per aggiornare la lista dei file selezionabili dall'utente e un getter per prendere quello che l'utente ha selezionato.

ClientGUIDocument Schermata relativa ad un documento. Può essere generata in modalità sola lettura o lettura e scrittura. Nel primo caso, non vengono mostrati il bottone per salvare le modifiche e la schermata di chat. Contiene metodi per leggere il documento corrente, leggere il messaggio scritto nel campo di testo della chat dall'utente e aggiornare i messaggi di chat ricevuti.

ClientGUIHandler Classe che gestisce l'interfaccia grafica. Estende `javax.swing.JFrame`. Ha un metodo sincronizzato per cambiare schermata (sincronizzato perchè può succedere che più richieste provino a cambiare l'interfaccia contemporaneamente) e un metodo per creare la finestra di popup.

3.1.2 client.resources

ClientStrings Classe che contiene le stringhe costanti usate dal client, estende ListResourceBundle.

3.2 server

MainClass_Server Classe che contiene il main. Istanzia il server e fa partire il loop del selector.

TURINGServer Classe principale del server. Funzioni ad alto livello per l'inizializzazione, la gestione dei socket, degli utenti connessi e dei file, che si appoggiano sulle altre classi.

ServerTask Task del thread pool. Viene inizializzato con una richiesta e la SelectionKey da cui è stata letta. Contiene uno switch con una voce per tipo di richiesta ed implementa la logica delle risposte.

RegistrationServer Implementazione dell'interfaccia TURINGRegister, esposta da RMI per la registrazione di un utente.

LoggedUser Classe che rappresenta un utente connesso. Contiene nome utente e SelectionKey. Usata per mantenere la lista di utenti online.

FilesManager Classe che si occupa di mantenere informazioni sui quali file sono aperti in modifica dagli utenti.

ServerFile Classe che rappresenta un file di TURING. mantiene informazioni sul nome del file, chi lo ha creato, quali sezioni sono aperte e qual è l'indirizzo di multicast associato al file per la chat.

FileSection Classe che rappresenta una sezione di un file.

UsersManager Classe con metodi statici per gestire le interazioni con il filesystem a vari livelli di astrazione. Ha metodi per leggere tutte le informazioni di un utente o dei suoi file che servono al server.

User Classe che rappresenta un utente di TURING. è il corrispondente nel programma del file utente.data di cui si è parlato nella sezione sul filesystem.

3.3 share

NetworkInterface Classe che gestisce le comunicazioni su socket (lettura e scrittura) a basso livello.

Request Classe che rappresenta un messaggio scambiato tra client e server. Presenta un tipo (RequestType) e un payload, che è un'HashMap da stringhe a stringhe.

RequestType Enumerazione dei tipi di richieste possibili.

TURINGRegister Interfaccia usata per la registrazione, in modo da astrarre l'implementazione al client.

4 Esecuzione

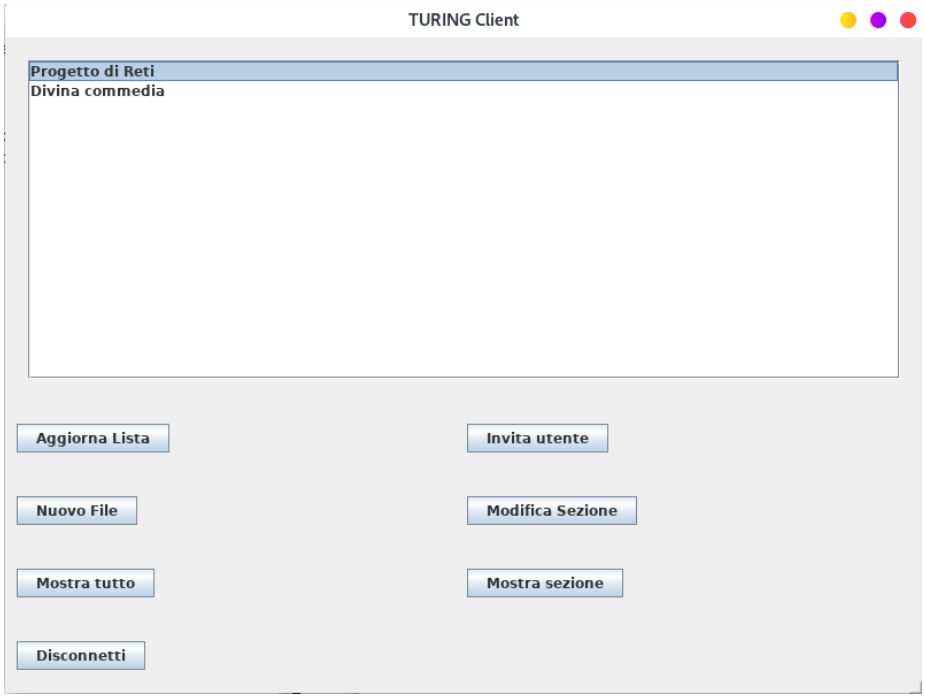
Sono stati preparati 3 script: build.sh per la compilazione, runServer.sh per far partire il server e runClient.sh per il client. Per compilare ed eseguire il progetto è sufficiente estrarre i file, spostarsi nella cartella "files", eseguire `chmod +x *.sh` per dare i permessi di esecuzione ed eseguire prima build.sh e poi gli altri due script. La compilazione e l'esecuzione richiedono l'utilizzo di Jackson, i cui file jar è incluso nella cartella lib.

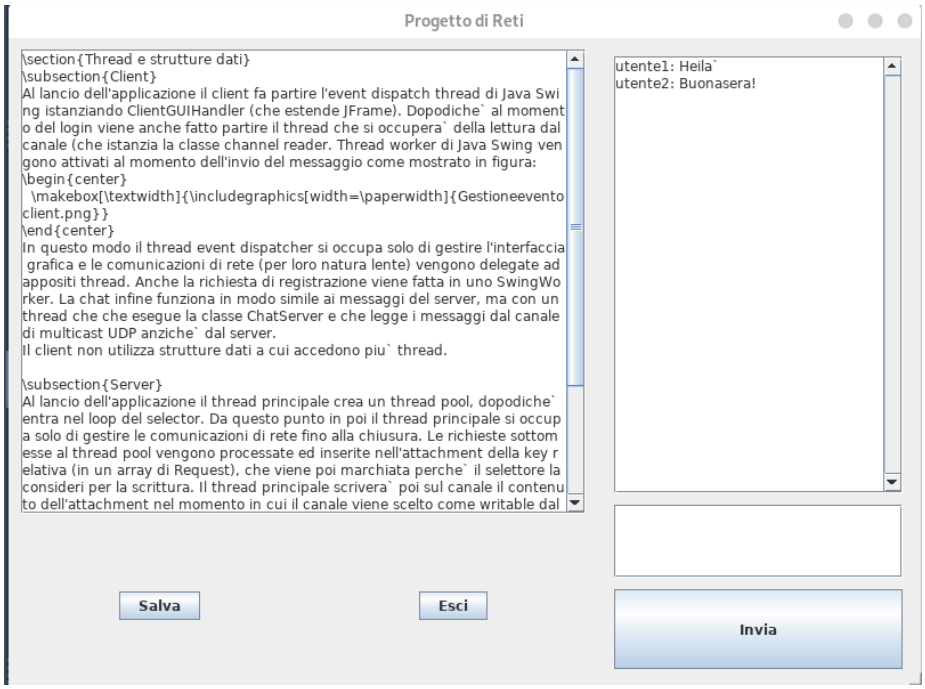
4.1 Test

La cartella tests contiene alcuni test delle funzionalità del server scritti in JUnit ed utilizzati durante lo sviluppo dell'applicazione. Permettere di eseguirli da riga di comando avrebbe richiesto di includere i file di JUnit ed eseguire ogni test a mano, percui si è preferito evitare, ma è stato comunque incluso il sorgente. I test provano le funzionalità del server che non hanno a che fare con la comunicazione di rete (Es. gestione dei login, dei file, etc.)

4.2 Programma

Essendo un'interfaccia grafica, il grosso dei comandi è implementato dai bottoni. Quando è necessario un input (testo o numero) verrà aperta una schermata di popup. Di seguito un esempio della schermata del menu e di modifica di una sezione.




Progetto di Reti
\section{Thread e strutture dati}
\subsection{Client}
Al lancio dell'applicazione il client fa partire l'event dispatch thread di Java Swi
ng istanziando ClientGUIHandler (che estende JFrame). Dopodiche` al moment
o del login viene anche fatto partire il thread che si occupera` della lettura dal
canale (che istanzia la classe channel reader. Thread worker di Java Swing ven
gono attivati al momento dell'invio del messaggio come mostrato in figura:
\begin{center}
\makebox[\textwidth]{\includegraphics[width=\paperwidth]{Gestioneevento
client.png}}
\end{center}
In questo modo il thread event dispatcher si occupa solo di gestire l'interfaccia
grafica e le comunicazioni di rete (per loro natura lente) vengono delegate ad
appositi thread. Anche la richiesta di registrazione viene fatta in uno SwingWo
rker. La chat infine funziona in modo simile ai messaggi del server, ma con un
thread che che esegue la classe ChatServer e che legge i messaggi dal canale
di multicast UDP anziche` dal server.
Il client non utilizza strutture dati a cui accedono piu` thread.
\subsection{Server}
Al lancio dell'applicazione il thread principale crea un thread pool, dopodiche`
entra nel loop del selector. Da questo punto in poi il thread principale si occup
a solo di gestire le comunicazioni di rete fino alla chiusura. Le richieste sottom
esse al thread pool vengono processate ed inserite nell'attachment della key r
elativa (in un array di Request), che viene poi marchiata perche` il selettore la
consideri per la scrittura. Il thread principale scrivera` poi sul canale il contenu
to dell'attachment nel momento in cui il canale viene scelto come writable dal
utente1: Heila`
utente2: Buonasera!
Salva
Esci
Invia