



UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

RELAZIONE DEL PROGETTO - LABORATORIO DI RETI

TURING: disTribUted collaboRative edItiNG

Jacopo Massa

543870

Luglio 2019

Indice

| | | |
|----------|--|-----------|
| 1 | Introduzione | 2 |
| 2 | Descrizione generale | 2 |
| 2.1 | Server | 2 |
| 2.1.1 | Inizializzazione | 2 |
| 2.1.2 | Gestione dei client | 2 |
| 2.1.3 | Chiusura della connessione | 2 |
| 2.2 | Client | 3 |
| 2.2.1 | Inizializzazione | 3 |
| 2.2.2 | Thread e concorrenza | 3 |
| 2.3 | Memorizzazione dei documenti | 4 |
| 2.3.1 | Memorizzazione | 4 |
| 3 | Interfaccia grafica | 5 |
| 3.1 | Server Panel | 5 |
| 3.2 | Login Panel | 6 |
| 3.3 | Turing Panel | 6 |
| 3.4 | Create, Edit, Show e Invite Panels | 7 |
| 4 | Strutture Dati | 8 |
| 4.1 | Client | 8 |
| 4.1.1 | InviteTask | 8 |
| 4.1.2 | ChatTask | 8 |
| 4.2 | Server | 8 |
| 4.2.1 | FileInfo | 8 |
| 4.2.2 | UserInfo | 9 |
| 4.2.3 | RegisteredUsers | 9 |
| 4.3 | Strutture comuni | 9 |
| 4.3.1 | Message | 9 |
| 4.3.2 | Operation | 10 |
| 5 | Protocollo di comunicazione | 10 |
| 5.1 | Richieste | 10 |
| 5.2 | Chat | 11 |
| 6 | Informazioni aggiuntive | 11 |
| 7 | Istruzioni per l'esecuzione | 12 |

1 Introduzione

TURING offre un servizio per l'editing collaborativo di documenti. Registrandosi a tale servizio, un utente può creare, modificare e visualizzare documenti di testo (salvati e gestiti in sezioni).

È stata utilizzata un'architettura CLIENT-SERVER per implementare tale servizio.

Il **server** si occupa della gestione dei documenti e degli utenti registrati; è privo di interfaccia grafica, ma offre una finestra in cui viene mostrato un log degli eventi dovuti alle richieste dei client.

Il **client** fornisce un'interfaccia grafica semplice e intuitiva che permette di inviare richieste al server, ricevere risposte da esso e restare in contatto (tramite la chat) con gli altri utenti, in fase di editing.

2 Descrizione generale

2.1 Server

Il server single thread, utilizza *Selector* e *Channel* per gestire le richieste e le risposte da e verso i client.

2.1.1 Inizializzazione

Come prima cosa il server inizializza alcune strutture dati:

- Una collezione per gli *utenti registrati* a TURING;
- Una collezione per mantenere traccia degli *indirizzi multicast* assegnati alle chat dei documenti;
- Una collezione per i *file caricati* dagli utenti;
- Un registry per fornire il *servizio di registrazione* a nuovi utenti.
- Il **ServerSocketChannel** su cui accettare le richieste dei client.

2.1.2 Gestione dei client

Il server successivamente entra in un loop, nel quale:

key.idAcceptable() Registra nuove connessione da parte dei client;

key.isReadable() Riceve richieste dai client, soddisfacendole se possibile;

key.isWritable() Comunica ad un client l'esito della sua richiesta;

2.1.3 Chiusura della connessione

Si è scelto di interrompere una connessione con un client solo nei seguenti casi:

- Non viene letto nulla dal socket TCP registrato per un client;
- Il client richiede il logout dal servizio TURING.
- Avviene una *IOException*.

2.2 Client

Il client multithread offre, tramite una interfaccia grafica sviluppata usando la libreria JSwing, la possibilità di gestire la comunicazione con il server.

2.2.1 Inizializzazione

- Imposta uno **ShutdownHook**, ovvero una *funzione di cleanup* chiamata al termine dell'esecuzione del processo client.
Tale funzione è stata aggiunta per evitare che in fase di disconnessione accidentale del client, il server mantenga uno stato inconsistente del client e/o dei files;
- Inizializza una collezione che conterrà gli **inviti pendenti** ricevuti mentre si era disconnessi;
- Apre un **clientSocketChannel** per la *trasmissione delle richieste* e dei rispettivi esiti;
- Apre un **inviteSocketChannel** per la *ricezione di inviti* da parte di altri utenti.

2.2.2 Thread e concorrenza

Il client è un processo formato da 3 thread:

mainThread: il thread principale che fornisce l'interfaccia grafica, effettua le richieste al server e ne ascolta le risposte;

inviteThread: il thread che ascolta la ricezione di nuovi inviti, comunicandoli all'utente tramite interfaccia (quindi al *mainThread*).

chatThread: il thread che (in fase di editing di un documento) gestisce la chat con altri utenti che stanno editando lo stesso documento.

L' *inviteThread* e il *chatThread* vengono avviati dal *mainThread* al momento del login, e vengono interrotti da esso al momento del logout. L'unica risorsa condivisa è la finestra della chat, in cui i vari thread scrivono comunicazioni da far visualizzare all'utente (inviti, messaggi di altri utenti...).

Non è stato necessario implementare meccanismi di gestione della concorrenza, dato che Java fornisce nativamente l' **EVT**, ovvero **Event Dispatching Thread**, cioè un Thread che processa gli eventi delle primitive grafiche, occupandosi lui stesso degli accessi concorrenti ad ogni componente grafico.

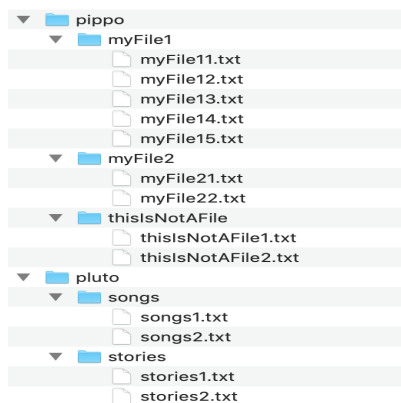
2.3 Memorizzazione dei documenti

2.3.1 Memorizzazione

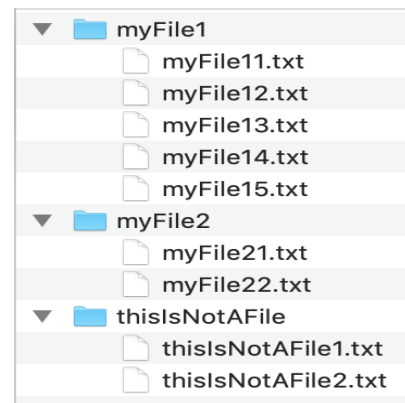
Client e server usano la stessa tecnica per memorizzare i files, ovvero:

- ogni documento è identificato da una directory e salvato in sezioni;
- ogni sezione è un file, che si trova all'interno della directory nominata come il documento.

In più, il server crea una cartella per ogni utente. Ciò permette di avere documenti con lo stesso nome, ma di proprietà di utenti diversi.



(a) File nel Server



(b) File nel Client

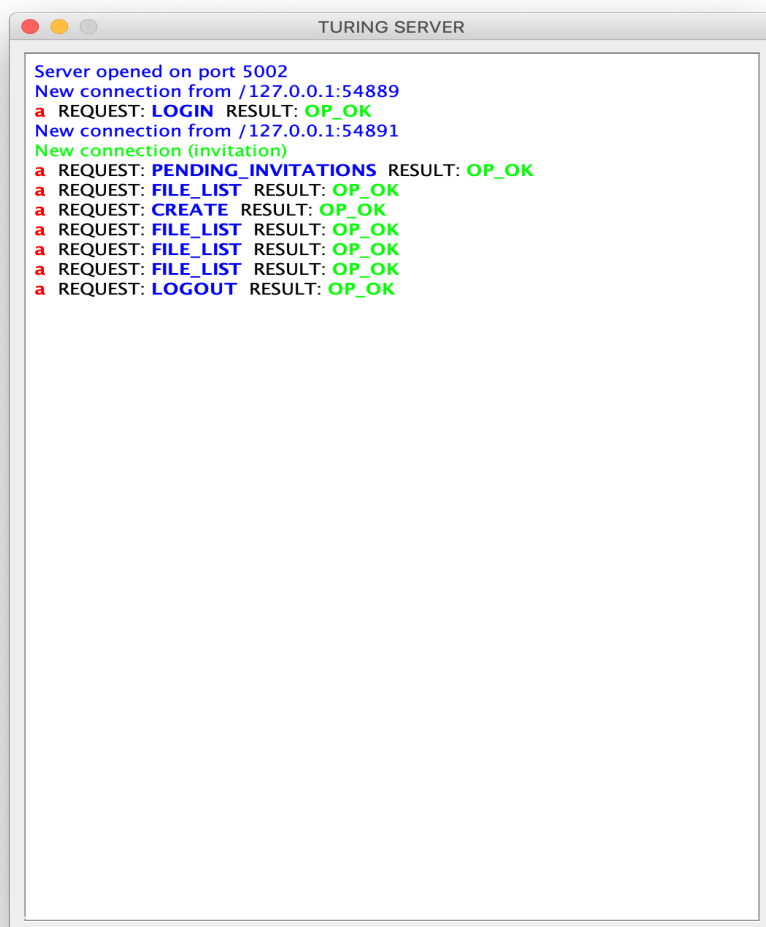
In tutto il progetto l'accesso ai file e il trasferimento tramite le socket avviene usando funzioni della libreria NIO.

3 Interfaccia grafica

Ogni finestra visualizzata è l'istanza di un oggetto **MyFrame**, che estende la classe *JFrame* della libreria JSwing. In base al tipo di finestra, essa contiene un diverso pannello, sviluppato estendendo la classe *JPanel*, e implementando l'interfaccia *ActionListener* per la gestione degli eventi del mouse e della tastiera.

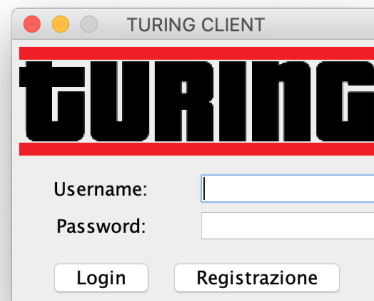
3.1 Server Panel

È l'unica finestra visualizzata dal server. Contiene una lista degli eventi occorsi in conseguenza alle richieste dei client.



3.2 Login Panel

Permette al client di effettuare la **registrazione** e il **login** al servizio turing.



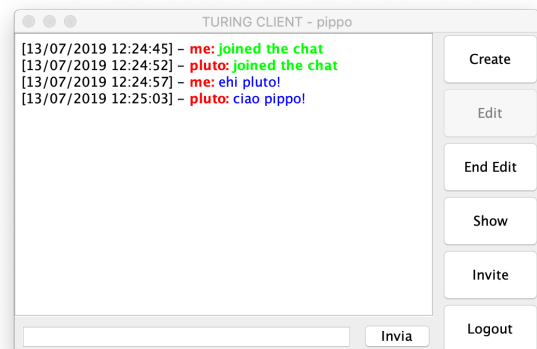
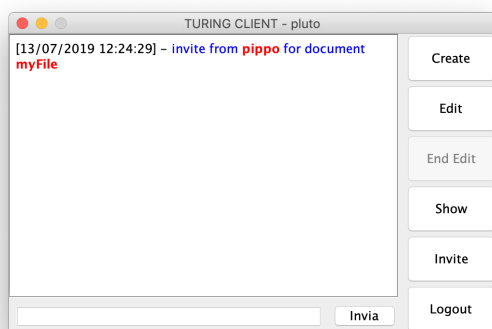
3.3 Turing Panel

Contiene i bottoni per le **principali richieste** da effettuare al server. Ogni bottone apre un Panel diverso (vedi successivi), tranne:

End Edit Serve a far terminare la fase di editing e notificare ciò al server;

Logout Effettua la disconnessione dal servizio TURING, mandando la richiesta al server e riportando l'interfaccia al Login Panel.

Inoltre permette al client di visualizzare e partecipare alla **chat** in fase di editing di un documento.



3.4 Create, Edit, Show e Invite Panels

Questi pannelli sono tutti molto simili.

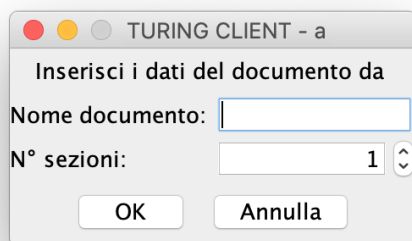
Il loro nome spiega la richiesta che mandano al server, in caso venga premuto il tasto "OK" al loro interno.

Create Panel Richiede la creazione di un file, inserendo nome e numero di sezioni totali.

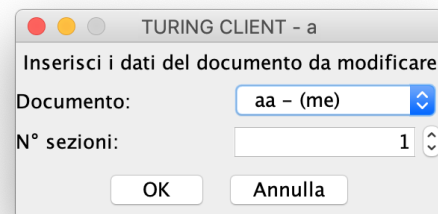
Edit Panel Richiede la modifica di un file, indicando quale sezione si vuole editare.

Show Panel Richiede la visualizzazione di un file intero (SHOW ALL) o di una specifica sezione.

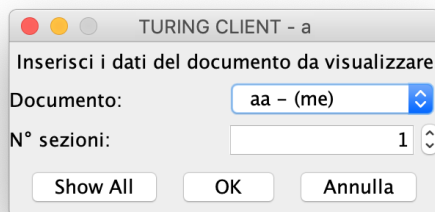
Invite Panel Permette di invitare un utente, specificandone il nome, alla modifica di uno dei propri files.



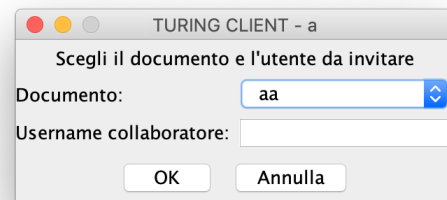
(a) Create Panel



(b) Edit Panel



(c) Show Panel



(d) Invite Panel

Per notificare il successo o l'errore di un'operazione, vengono usati dei *JOptionPane*, ovvero finestre di dialogo personalizzabili in base al tipo di messaggio che si vuole mostrare.

4 Strutture Dati

4.1 Client

4.1.1 InviteTask

Rappresenta l'oggetto target eseguito dal Thread listener degli inviti.

Parametri

socketChannel la socket TCP su cui ascoltare la ricezione degli inviti.

4.1.2 ChatTask

Rappresenta l'oggetto target eseguito dal Thread manager della chat in fase di editing.

Parametri

address indirizzo su cui aprire la socket Multicast della chat;
(= *ADDRESS* in Utils)

port numero di porta su cui aprire la socket Multicast della chat;
(= *MULTICAST_PORT* in Utils)

4.2 Server

4.2.1 FileInfo

Rappresenta le informazioni utili alla gestione di un documento (chiamato anche file) da parte del server TURING (eccetto il filename, salvato a parte in una HashMap istanziata dal server).

Parametri

owner creatore e proprietario del file;

nsections numero di sezioni totali del file;

sections array di booleani; indica lo stato delle sezioni (*TRUE* = in editing, *FALSE* altrimenti)

counterEditors numero di utenti che stanno editando contemporaneamente; quindi indica anche il numero di sezioni attualmente in fase di editing;

address indirizzo della chat multicast associato al file

4.2.2 UserInfo

Rappresenta le informazioni utili riguardanti un utente (eccetto l'username, salvato nella struttura `RegisteredUsers`)

Parametri

password password dell'utente;

online stato dell'utente (*TRUE* = online, *FALSE* = offline)

files array di booleani; lista dei nomi dei documenti a cui l'utente ha accesso.

pendingInvitation lista di `Message`; rappresenta gli inviti ricevuti da un utente mentre era offline

inviteSocketChannel riferimento al socket TCP usato per ricevere gli inviti in tempo reale

editingFilename nome del documento che l'utente sta editando
(= *stringa vuota*, altrimenti)

editingSection numero della sezione del documento che l'utente sta editando
(= *0*, altrimenti)

4.2.3 RegisteredUsers

È una collezione che contiene tutti gli utenti registrati a TURING. Non ha parametri da inizializzare.

Consiste in una `ConcurrentHashMap` del tipo `<username, UserInfo>`. La scelta è ricaduta su una struttura concorrente, poiché un'istanza di `RegisteredUsers` è usata nel server come stub del servizio di registrazione tramite RMI.

Quindi, dato che viene acceduta da più client contemporaneamente, è necessario un meccanismo che gestisca la concorrenza (ovvero la `ConcurrentHashMap`).

4.3 Strutture comuni

4.3.1 Message

Rappresenta gli inviti di collaborazione all'editing o i messaggi mandati sulla chat tra gli utenti che editano lo stesso documento.

Parametri

sender mittente dell'invito / messaggio;

body nome del file per il quale si sta mandando l'invito / corpo del messaggio;

date data e ora dell'invio dell'invito / messaggio

4.3.2 Operation

Rappresenta una richiesta che il client manda al server; contiene il codice della richiesta da effettuare e altri parametri utili.

Parametri

code codice della richiesta;

username username dell'utente che effettua la richiesta;

password password dell'utente che effettua la richiesta;

filename nome del file che si vuole editare/visualizzare o per cui si richiede una collaborazione

owner proprietario del file 'filename';

section numero di sezione del file 'filename';

In base alla richiesta che si effettua, alcuni campi potrebbero avere valore NULL (tranne *code* che è sempre presente) ma ciò non influisce sulla serializzazione che l'oggetto riceve per poter essere spedito sul socket tra client e server.

Il codice, l'username e la password sono campi obbligatori. Se non presenti, l'esito della richiesta sarà fallimentare (= *OP_FAIL*).

5 Protocollo di comunicazione

Il servizio di registrazione viene implementato tramite RMI, ed è l'unico tipo di comunicazione che non sfrutta le socket TCP/UDP. Il server crea un registro sulla porta *REGISTRATION_PORT* (in Utils).

Infatti, per il resto del progetto, sono state usate:

socket TCP nell'invio e ricezione di richieste tra client e server;

socket UDP (o meglio le **MulticastSocket UDP**) per la comunicazione tra i vari client che condividono la chat per uno stesso documento.

5.1 Richieste

Sulle socket viaggiano oggetti di tipo Operation e Message serializzati. Ogni client in fase di login, apre due socket TCP (entrambe sulla porta = *CLIENT_PORT* in Utils):

- una socket per le richieste (e i dati ad esse associati);
- una socket per ricevere gli inviti.

Tramite la seconda socket, il client invia al server una richiesta con codice = *SET_INVITATION_SOCKET* e il proprio username; il server così capisce che la socket da cui ha ricevuto tale richiesta verrà associata al client 'username', e usata per spedirgli gli inviti di altri utenti.

Il client invia sempre all'interno dell'oggetto Operation:

- username
- password
- codice della richiesta

Dopo aver eseguito la richiesta di un utente, inviandogli sezioni e/o altre info ove necessario, il server manda un codice che identifica l'esito della richiesta, sempre come ultimo messaggio.

Di seguito si riporta una tabella che mostra quali altre informazioni vengono spedite dal client / server, in base alla richiesta effettuata:

| RICHIESTA | Client | Server |
|-----------------------|---------------------------------|--------------------------|
| PENDING_INVITATIONS | | pending invitations list |
| <i>CREATE</i> | filename, section number | |
| <i>EDIT</i> | filename, section number, owner | multicast address |
| <i>SHOW, SHOW_ALL</i> | filename, section number | |
| <i>INVITE</i> | filename, collaborator name | |
| SECTION_RECEIVE | filename, section number, owner | section |
| FILE_LIST | | accessible files list |
| <i>END_EDIT</i> | filename, section number, owner | |

5.2 Chat

Per quanto riguarda la chat, essa avviene tramite una socket UDP multicast, non coinvolgendo per nulla il server. Esso infatti, fornisce insieme ai file, anche l'indirizzo di multicast associato al file stesso. Il client userà questo indirizzo per collegarsi alla chat e poter comunicare, spedendo anche qui degli oggetti Message serializzati.

È importante sottolineare il riuso degli indirizzi di multicast:

nell'assegnazione di un indirizzo ad un file da parte del server, esso controlla prima che ce ne sia uno già generato e non utilizzato. Se tutti quelli generati sono in uso per altre chat, allora ne genererà uno nuovo.

6 Informazioni aggiuntive

Consultare il file *OpCode.java* per maggiori informazioni riguardo:

- codici delle operazioni principali
- codici dell'esito delle richieste (successo/errore)

Non ci si è soffermati molto nella descrizione dell'interfaccia grafica; se interessati si invita a leggere il codice, interamente reperibile all'interno del *package GUI*

Nel file *Utils.java* invece, come specificato anche spesso in questa relazione, si trovano alcune delle costanti usate nel progetto (e modificabili a piacere), più alcune funzioni di utility.

7 Istruzioni per l'esecuzione

Sono stati creati due file eseguibili (*Server.jar* e *Client.jar*) all'interno della home directory del progetto. Nel caso si vogliano avviare più client, copiare il file *Client.jar* tante volte quanti sono i client voluti.

Si spera che apprezziate l'interfaccia grafica, aggiunta per rendere più semplice e simpatica tale applicazione.