Università di Pisa, CdL in Informatica, A.A. 2018-19

Relazione del Progetto di Laboratorio di Reti

TURING: implementazione di uno strumento per l'editing collaborativo di documenti

Giovanni Marco Fenu

Sommario

1. Introduzione

Il progetto di fine corso di Laboratorio di Reti richiedeva di realizzare TURING (disTribUted collaboRative edItiNG), un insieme di servizi minimale per l'editing collaborativo di documenti con architettura client-server, che utilizzasse i vari meccanismi di comunicazione visti durante le lezioni. Questa relazione è volta a spiegare l'architettura generale della mia soluzione, nonché le scelte progettuali più importanti.

1.1. Struttura del codice

Il codice sorgente del programma è stato diviso in tre package:

1. ***SocialTuring***. Realizza la logica dell'editor Turing dichiarando classicatori quali User, Document e metodi per la registrazione di utenti, e l'aggiunta di documenti.

2. ***Server.*** Realizza la parte del sistema che memorizza lo stato dell'editor Turing e offre ai client canali di comunicazione su cui inviare richieste di login, creazione di documenti, condivisine di documenti ecc. Il sotto-package Server.gui implementa una semplice finestra che facilita l'avvio del server.

3. ***Client.*** Realizza il lato client, fornendo una classe omonima i cui metodi permettono a un utente di interagire col server. Anche qui e presente un sotto-package di interfaccia grafica.

I tre package verranno nel dettaglio illustrati nelle successive sezioni 2, 3 e 4. Segue in questa sezione una breve descrizione dei modi con cui il server e i client comunicano.

1.2. Protocollo di comunicazione

Tutti i messaggi spediti dal client al server tramite TCP hanno un'intestazione di un byte che indica il tipo di richiesta. I valori possibili sono dati dalle costanti simboliche definite in Server.RequestTypes e riportate nella seguente tabella:

LOGIN	LOGOUT	CREATE_DOCUMENT
START_EDIT_DOCUMENT	END_EDIT_DOCUMENT	SHOW_DOCUMENT
SHOW_SECTION	SHARE_DOCUMENT	LIST_DOCUMENT

La parte di un messaggio è variabile in base al tipo di richiesta.

Per quando riguarda UDP, è stato utilizzato per implementare la Chat-Multicast, che si avvia ogni qual volta, si inizia a editare un documento.

1.3. Interfacce remote

L'insieme dei metodi remoti che realizzano le funzionalità di registrazione di un utente e notifica degli inviti e gestione degli eventi, sono definiti nelle interfacce RemoteRegistrationClient e RemoteClientCallback. La prima, implementata dal server, e offre l'operazione di registrazione di un client, e registrazione di una callBack, per la notifica di invito a collaborare a un documento. La seconda interfaccia implementa la notifica degli inviti con il metodo notifyShare<Sting notifica>.

Ho utilizzato una callBack per la gestione degli inviti, per la semplicità di gestione dell'invio e ricezione di notifica;

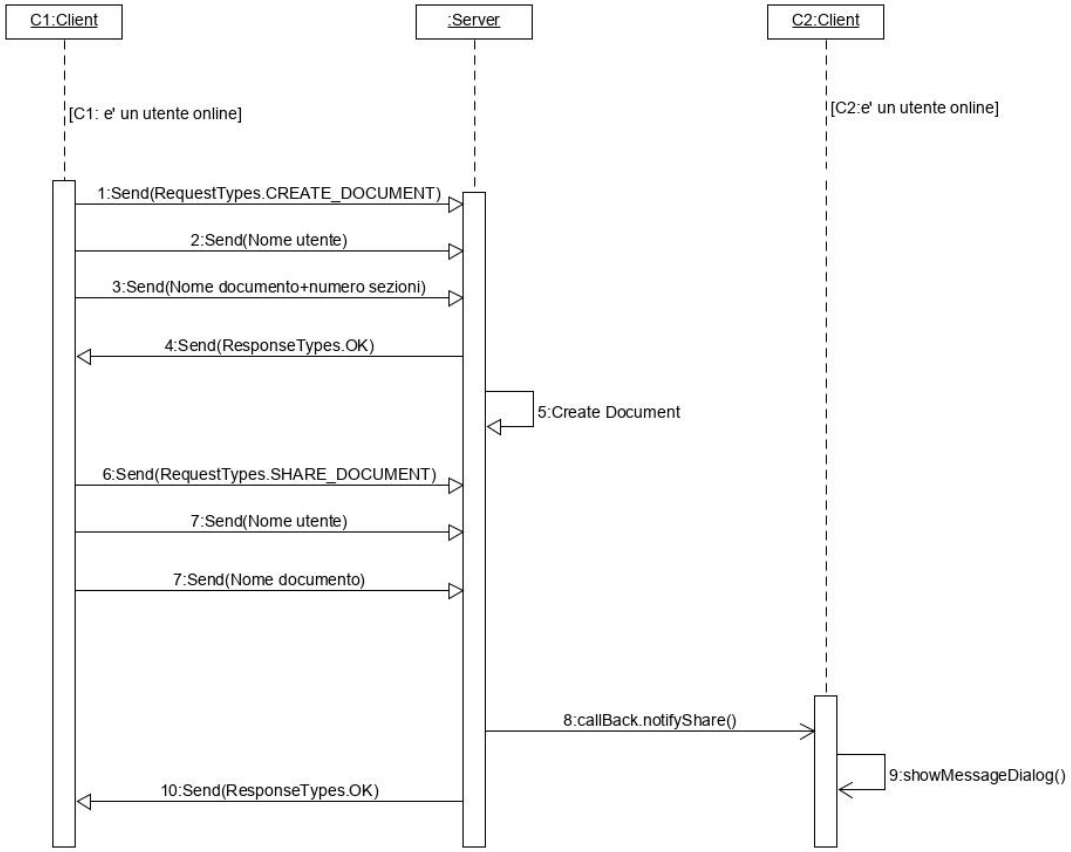

Figura 1 : Le interazioni fra due client e il server durante la creazione e condivisione di un file, con 2 utenti online. Il Server crea e memorizza il documento

2. Turing

2.1. Descrizione delle classi

UsersNetwork offre metodi che manipolano la rete di utenti, garantendo allo stesso tempo il rispetto dei vincoli di univocità dei nomi degli utenti. In ***UsersNetwork*** è presente un HashMap<String, User> per mantenere, gli utenti registrati al servizio, e per poter recuperare un User, identificandolo con il suo nome.

DocumentNetwork offre metodi che manipolano i documenti presenti in Turing, garantendo allo stesso tempo l'univocità dei documenti presenti. In ***DocumentNetwork*** è presente un Hash-Map<String, Document> per mantenere, i documenti presenti in Turing, e per poter recuperare un Document, identificandolo con il suo path.

IpMulticastChat offre un metodo per creare/recuperare un Ip-Multicast. Utilizza una List, per mantenere gli Ip-Multicast da riutilizzare.

2.2. Concorrenza e sincronizzazione

UsersNetwork, DocumentNetwork e IpMulticastChat. Ognuna di queste classi utilizza una propria ReadWriteLock in modo tale che più thread possano leggere la risorsa concorrentemente, ma soltanto uno vi può scrivere.

3.Server

3.1. Paradigma non-blocking-I/O

Il server adotta il modello non-blocking-I/O, con un thread per connessione. Ho preferito questa tecnica rispetto a un server single-thread con blocking I/O per i seguenti motivi:

- tutte le interazioni tra client e server sono di brevissima durata, quindi i thread non rimangono bloccati a lungo sulle operazioni di lettura/scrittura; di conseguenza, usando un thread pool l'overhead di spazio e tempo dovuto alla creazione e la gestione di tanti thread è basso;
- per sfruttare l'architettura multicore della macchina su cui viene eseguito il server;
- pochi threads possono gestire centinaia o migliaia di connessioni con client diversi;

Per semplicità le operazioni sono atomiche, leggo tutti i dati in una volta sola, in modo da non salvare e ripristinare lo stato ad ogni iterazione.

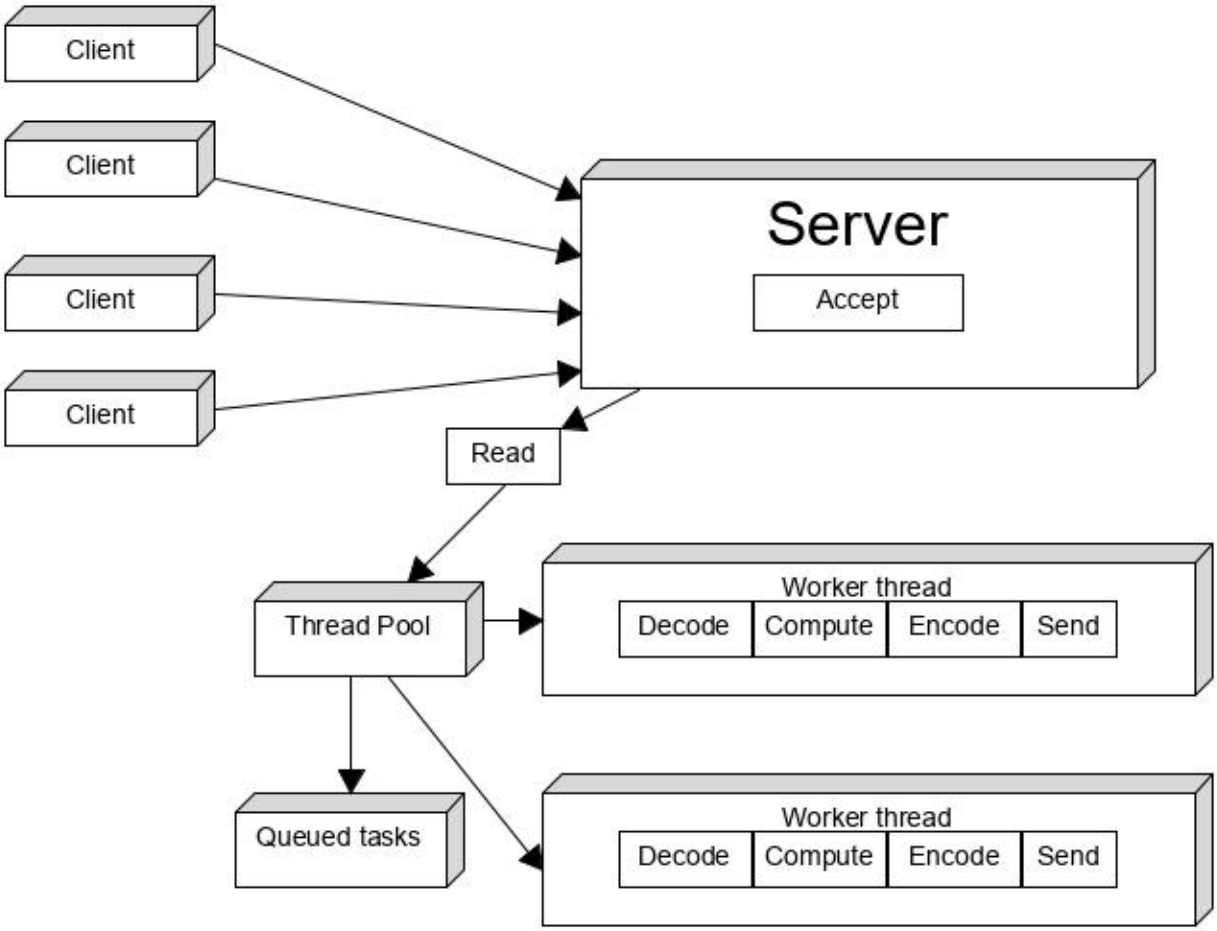

Figura 2: Struttura generale del Server

3.2. Descrizione delle classi

Le classi della componente Server sono le seguenti:

AttachChannel:

- Crea un oggetto da collegare alla key, relativa a una specifica connessione con il Client. Viene creata al momento dell'accettazione di una connessione.
- Al momento del login di un client, viene associato un nome utente.
- Contiene uno "***state***", una variabile bolleana, che indica se si sta leggendo o scrivendo sul canale, relativo alla key in questione; utilizza metodi syncronized, per get() e set().
- Contiene anche due buffer, per leggere e scrivere sul canale.

Server:

- Crea una cartella per contenere i documenti relativi ai client;
- Accetta le connessioni da parte dei Client;
- Ad ogni client viene associato un oggetto, AttachChannel() al momento del login;
- Ogni qual volta ci sia un dato da leggere sulla key del client, e la chiave non sia in lettura con un altro thread, sottometto un nuovo ***ServerTask*** a un thread-pool. ***ServerTask.Run()***, legge l'intestazione del pacchetto e lo smista, andando a eseguire uno dei metodi privati della classe(login(), logout(), createDocument()…);

3.3. Gestione dei File

Al momento dell'avvio del Server, viene creata una cartella Document, nello spazio di lavoro di avvio del Server. Ogni qual volta, un client autenticato invia una richiesta di creazione di un documento, viene creata una cartella con il nome del documento ricevuto, al suo interno ci saranno i file "1.txt, 2.txt, …, n.txt", dove n indica il numero di sezione ricevute da parte del client. I nomi dei file sono univoci, anche per client diversi.

3.4. Concorrenza e sincronizzazione

Ogni singola richiesta da parte di un client viene eseguita dal server su un thread separato e accede alle strutture dati di Turing in modo concorrente con le altre richieste. La sincronizzazione degli accessi su UsersNetwork e DocumentNetwork è stata già discussa nella sezione 2.2. Per l'editing dei file in modo concorrente, viene utilizzata una struttura sincronizzata Map<Utente, pathFile>, che mappa i file che si stanno editando.

4.Client

4.1. Descrizione delle classi

Come per il server, ho isolato l'interfaccia grafica dalla logica del programma. È presente una classe ***Client***, che offre tutte le operazioni col server, separando la loro implementazione.

Le classi sono:

TcpConnectionFactory: Costruisce una connessione col server, con hostname e porta fissati. (Quelli del server)

ComunicationManager: Gestisce la connessione di un singolo utente con un Turing server. In particolare, si occupa della registrazione e del login di un utente, nonché della gestione dei file: creazione, edit di un documento, lista di documenti, etc. Un oggetto Client, chiama le funzioni di questa classe.

RmiConnection: Si occupa di creare il riferimento all'oggetto remoto effettivamente attivo su server Turing, inoltre si occupa di creare e registrare sul server la callback, per le notifiche degli inviti.

ClientCallbackImpl: implementa l'interfaccia remota ***RemoteClientCallback.*** Contiene il metodo notifyShare, metodo chiamato dal Server, per notificare un invito quando on-line, oppure al primo login del client.

ThreadNotify: Legge le notifiche relative agli inviti.

ThreadUDP: Si mette in ascolto sulla porta:8888 per ricevere i messaggi della chat-multicast. Viene attivato solo, se un utente sta editando un documento.

4.2. Concorrenza e sincronizzazione

Per la gestione degli inviti in modo concorrente viene utilizzata una List<String>, a cui ci si accede mediante lock.

4.3. Gestione dei file

Ogni qual volta si richiede di editare un file, il file viene salvato col nome di "Edit.txt";
Ogni qual volta si richiede di visualizzare un file, il file viene salvato col nome di "NomeFile.txt";
Ogni qual volta si richiede di visualizzare una sezione, il file viene salvato col nome di "NomeFile-Numero di Sezione.txt";

Tutti i file verranno salvati in una cartella, col nome dell'utente.

5.Manuale utente

Per lanciare le applicazioni server o client è sufficiente fare doppio click sul rispettivo file con estensione .jar.
L'applicazione server è composta da una sola finestra che mostra un'area con i messaggi di log e due pulsanti, uno per l'avvio e l'atro per l'interruzione della componente server: dopo che è stato premuto il pulsante Start, i client possono cominciare a connettersi e a utilizzare i servizi offerti dal server.

L'applicazione client è composta da tre finestre: quella di login, home e chat. Nella finestra di login è possibile registrarsi e connettersi alla rete sociale.

La finestra home (v. Figura 3):

- sulla destra un'area di testo, che mostra la lista di documenti che a cui un utente può collaborare, una volta premuto il tasto "*List Document*"

- sulla destra sei bottoni:

    - "*Create Document*", realizza la creazione di un documento. Dopo aver cliccato il relativo bottone, appare una finestra di dialogo (v. Figura 4), per inserire nome file e numero di sezioni;

    - "*Edit Document*", realizza la richiesta di edit di un documento. Dopo aver cliccato il relativo bottone, appare una finestra di dialogo, per inserire nome file e sezione; una volta scaricato il file si apre in automatico il programma predefinito per la lettura di file con estensione "*.txt". Disabilita tutti gli altri bottoni, ad eccezione di quest'ultimo, che cambia nome in *"End Edit"* e si apre la finestra di chat. Dopo aver cliccato sul bottone *"End Edit"*, la finestra di chat si chiude, e tutti i bottoni vengono riabilitati.

    - "*Show Section*" realizza la richiesta di visualizzazione di un documento. Dopo aver cliccato il relativo bottone, appare una finestra di dialogo, per inserire nome file e sezione; una volta scaricato il file si apre in automatico il programma predefinito per la lettura di file con estensione "*.txt;

    - "*Show Document*" realizza la richiesta di edit di un documento. Dopo aver cliccato il relativo bottone, appare una showConfirmDialog, per inserire nome file; una volta scaricato il file si apre in automatico il programma predefinito per la lettura di file con estensione "*.txt;

    - "*Share Document*" realizza la richiesta di condivisone di un documento. Dopo aver cliccato il relativo bottone, appare una showConfirmDialog, per inserire nome file e nome utente invitato;

- "*List Document*" realizza la richiesta di lista di un documento, che l'utente e autorizzato ad accedere e modificare. Dopo aver cliccato il relativo bottone, la lista viene mostrata sull'apposita area di testo;

La finestra di chat (v. Figura 5), si attiva ogni qual volta l'utente edita un documento. Contiene due aree di testo per ricevere e scrivere i messaggi da inviare, e un bottone per inviare i messaggi.

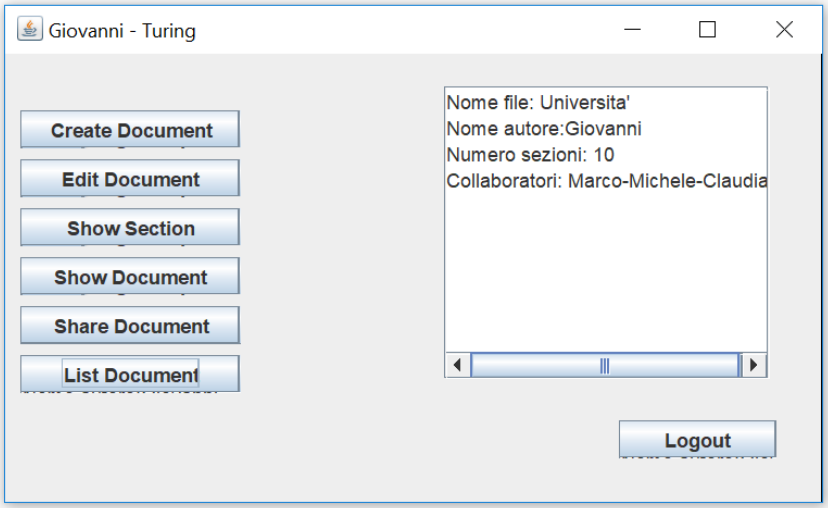

Figura 3: Finestra home, relativo a un utente di nome Giovanni.

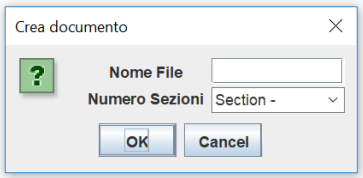

Figura 4: Finestra di dialogo, che appare, quando si clicca il bottone "Create Document".

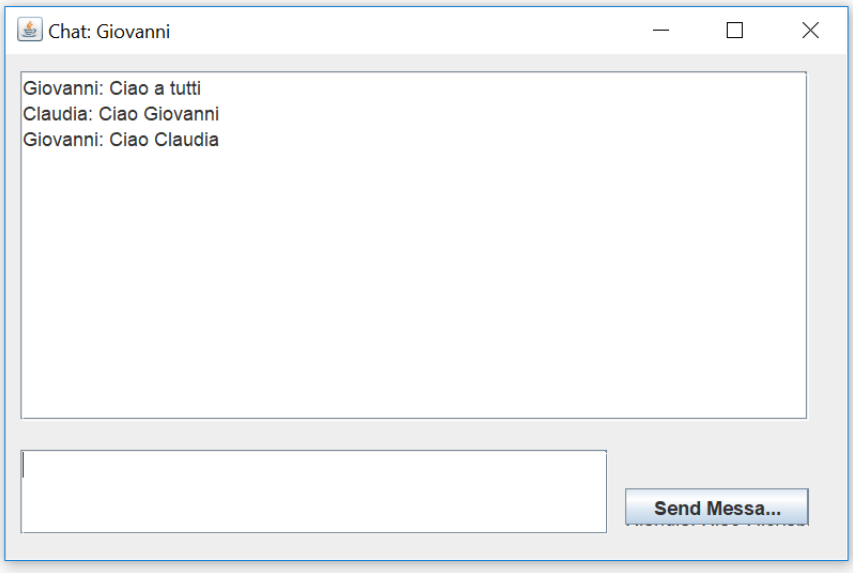

Figura 5: Finestra di chat, fra due utenti