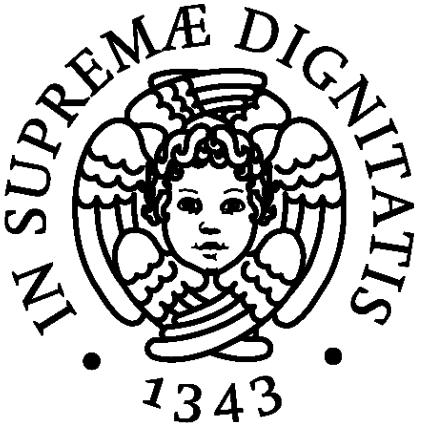


UNIVERSITÀ DI PISA

Anno Accademico 2018/2019

Relazione Progetto TURING

Laboratorio Di Reti B

https://github.com/PieroBook/Turing

Docente Corso

Federica Paganelli

Studente

Pietro Libro

1 Architettura del sistema

Il sistema software è stato realizzato usando il linguaggio Java, con architettura Client-Server.

1.1 Server

La parte server è implementata senza interfaccia grafica (GUI) pertanto le informazioni circa lo stato del server vengono presentate attraverso command line interface (CLI).
Il sistema conserva i dati in directory distinte per utente, sotto forma di file enumerati in base alle sezioni se quest'ultime sono presenti.
Tutte le informazioni sono conservate dal server alla terminazione dello stesso sotto forma di file JSON (DATA/Has hMap.json). Difatti all'avvio il server verifica la presenza del suddetto file per caricare le informazioni nelle rispettive strutture dati, procede poi all'inizializzazione di tutti gli oggetti transient.
Se il file non è presente tutte le strutture dati vengono inizializzate come se il sistema fosse OOB.
Successivamente procede creando e inizializzando la connessione RMI con cui espone il metodo di registrazione utente, le strutture per la gestione degli indirizzi multicast (Max 254 indirizzi contemporaneamente), il ServerSocketChannel e relativo selettore con cui svolge multiplexing delle connessioni client in entrata, ed infine il cachedThreadPool che effettua i task necessari al funzionamento.

1.1.1 Registrazione Utente

L'utente può registrarsi attraverso interfaccia RMI offerta dal server. Quest'ultimo infatti espone sulla porta 1099 (RegistryPort standard RMI) un interfaccia sotto il nome "Server-Turing" che dispone del metodo di richiesta registrazione. In base all'esito della registrazione la risposta opportuna viene comunicata al client via booleano.

1.1.2 Gestione richieste e risposte

Il multiplexing viene effettuato tramite istanza della classe Selector, distinguendo i 3 stati in cui si trova il channel attraverso il SelectionKey.

1. In fase di connessione (ACCEPT) il server accetta il client e viene creata una lock afferente al suo SocketChannel;
2. In fase di lettura (READABLE) il server crea un ReadTask per la connessione in oggetto che legge la richiesta del client e la accoda in una mappa di richieste e lo manda in esecuzione sul CachedThreadPool;
3. In fase di scrittura (WRITABLE) il server recupera la richiesta per il client in oggetto e procede ad evadere la richiesta attraverso un WriteTask mandato in esecuzione sul CachedThreadPool.
   Se il channel è ancora attivo alla terminazione delle operazioni, la selectionkey viene settata readable, in modo da accettare nuove richieste.

Sia il WriteTask che il ReadTask operano acquisendo la mutua esclusione sul socket da cui proviene la richiesta.

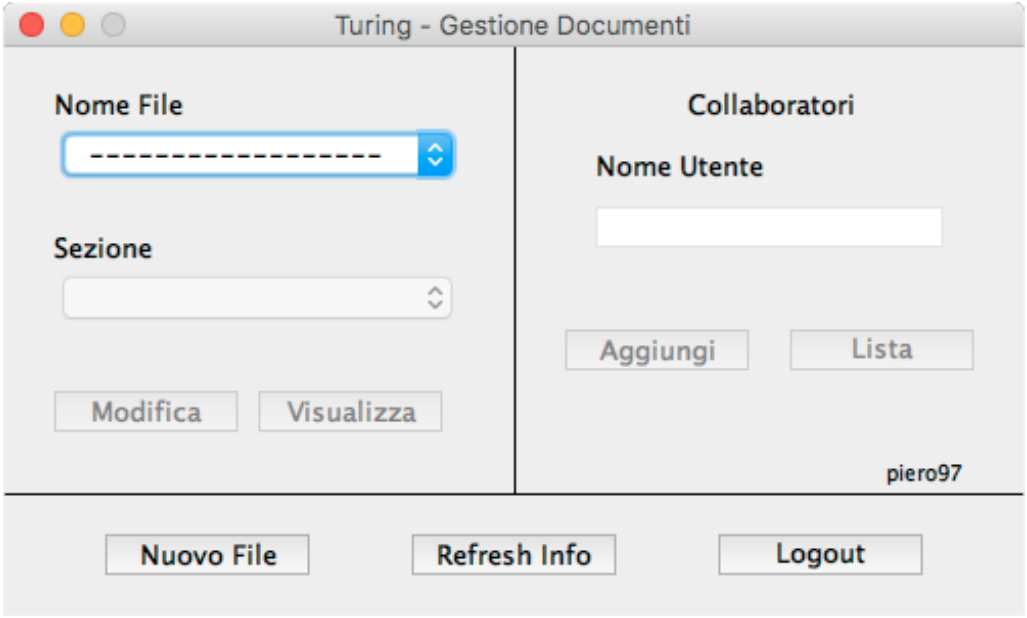
1.1.3 Terminazione Server

È possibile interrompere in qualsiasi momento l'esecuzione dell'istanza server attraverso la combinazione CTRL+C. L'interruzione del processo comporta l'esecuzione di uno ShutdownHook che si occupa di lasciare il server in uno stato consistente senza perdita di informazioni. Nello specifico conserva lo stato nel file JSON, esegue shutdown del CachedThreadPool e chiude tutti i channel prima di terminare definitivamente l'esecuzione.

1.2 Client

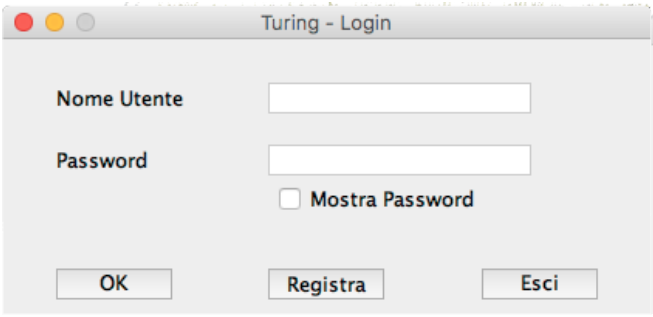
Il client presenta un'interfaccia grafica per l'interazione da parte dell'utente, realizzata con tecnologia Java AWT e Java Swing in minima parte.
Sono stati realizzati diversi Frame per assolvere i diversi compiti del client:

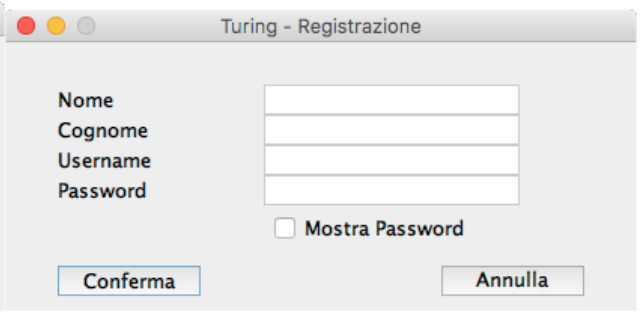
- **Frame Turing**: Frame principale per l'interfacciamento dell'utente con il sistema diviso in zone di distinzione operativa.



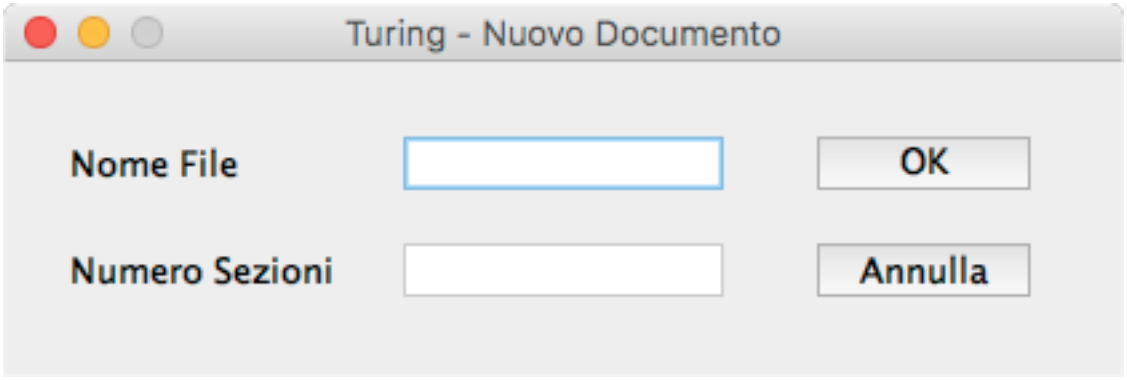
La zona sinistra permette di scegliere i documenti a disposizione dell'utente per modifica e visualizzazione della sezione o per quest'ultima operazione, di tutte le sezioni. Terminato lo scaricamento del documento richiesto in caso di visualizzazione di una sezione in modifica da parte di un altro utente viene visualizzato un box informativo.
La zona a destra consente, una volta selezionato un documento di aggiungere un utente ai collaboratori, e di visualizzarne la lista.

- **Frame Login**: Frame di Login che consente di accedere al frame di Registrazione o di effettuare il login con credenziali.
- **Frame Registrazione**: Frame di Registrazione che consente di registrare un nuovo utente al sistema tramite metodo recuperato via RMI Registry.

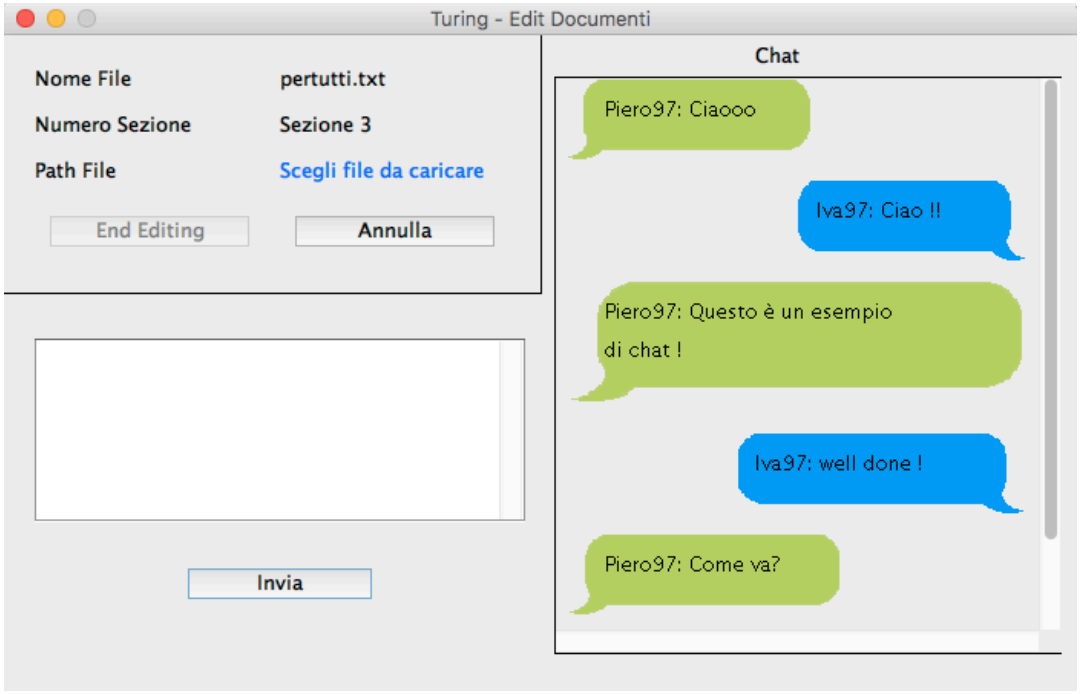




- **Frame Nuovo Documento**: Frame per la creazione di un nuovo documento e relative sezioni. Due utenti possono possedere documenti diversi con lo stesso nome.



- **Frame Edit**: Frame per la fase di modifica della sezione di un documento con relativa Chat afferente al documento. Una volta finito la modifica e richiesto l'end-edit il sistema verifica il MD5 hash del file e se diverso dal precedente procede al re-upload della sezione al server. Per realizzare quest'ultima funzionalità è stata preferita la libreria esterna Apache Commons Codec.



La chat è UDP Multicast, con indirizzi assegnati al documento alla prima richiesta di edit, da parte del server, e rilasciati quando il documento non è più in edit in nessuna sua sezione.
(Nota: La modifica deve avvenire mediante editor di testi esterno a TURING)

1.3 Scelte progettuali

Lato server è stato scelto di realizzare il multiplexing delle connessioni unito ad un multithreading incentrato sui 'work' di ricezione della richiesta ed elaborazione-invio della risposta, separati. Questo risulta essere un trade-off efficiente e largamente utilizzato, specialmente se il thread pool in questione è un CachedThreadPool, che ci da un livello di elasticità non indifferente, può infatti espandersi potenzialmente all'infinito ma soprattutto quando un thread termina il suo 'task' resta disponibile ad assolvere un altro 'task' per 1 minuto. In questo modo viene limitata la distruzione e creazione di thread.

Per la memorizzazione dello stato del Server, e per il trasferimento delle richieste e risposte via TCP, è stato scelto di convertire tutte le strutture dati in formato Json, usando la libreria Gson sviluppata da Google. La libreria è disponibile nella cartella lib del progetto ma anche via Maven e/o Gradle.
È stata preferita rispetto ad altre librerie che assolvono lo stesso compito in quanto molto potente ed immediata nell'utilizzo.

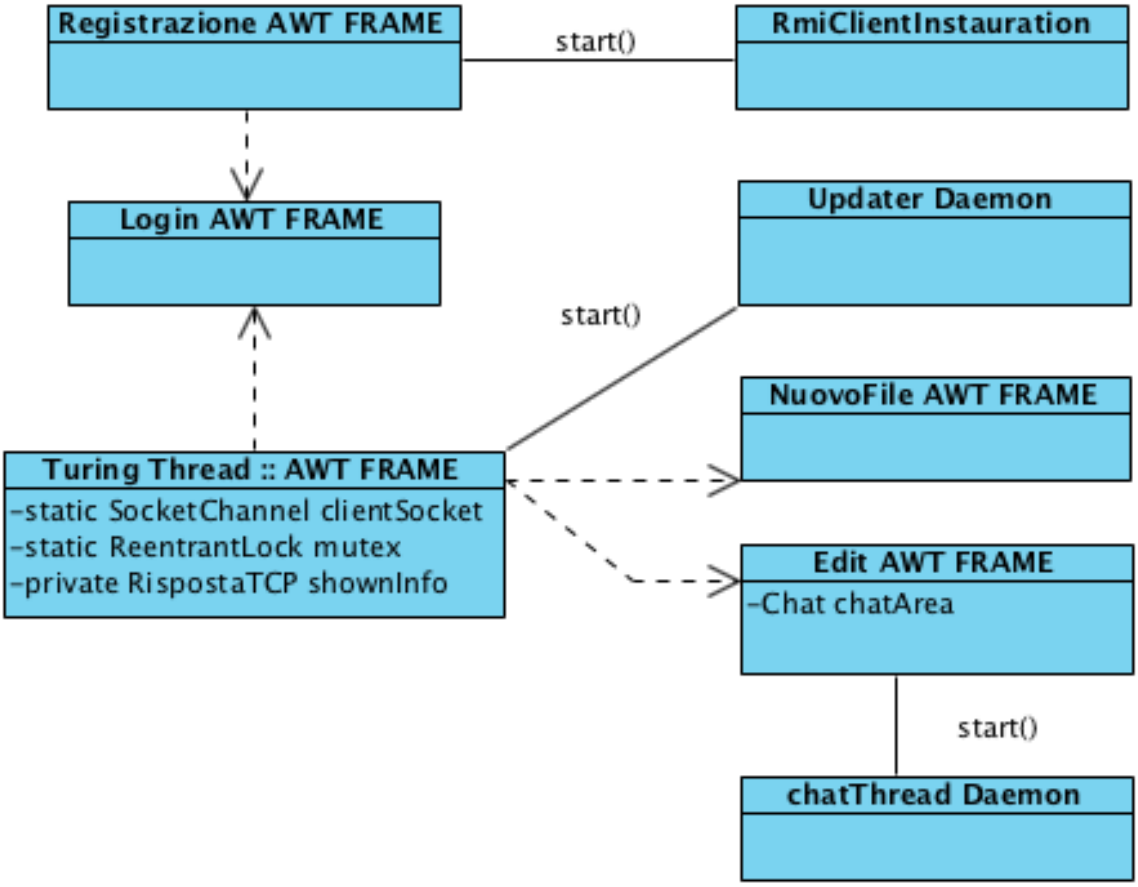
Rispetto alla specifica è stata introdotta una funzionalità di auto-update delle informazioni mostrate in GUI. È comunque possibile richiedere l'update delle informazioni anche manualmente.
Successivamente all'autenticazione avvenuta con successo, viene avviato un thread daemon che ogni 1,5 sec verifica se è presente la necessità di aggiornare le info visualizzate e quindi procede all'invio della relativa richiesta, altrimenti procede a rilevare la presenza di un'eventuale notifica ricevuta dal server.

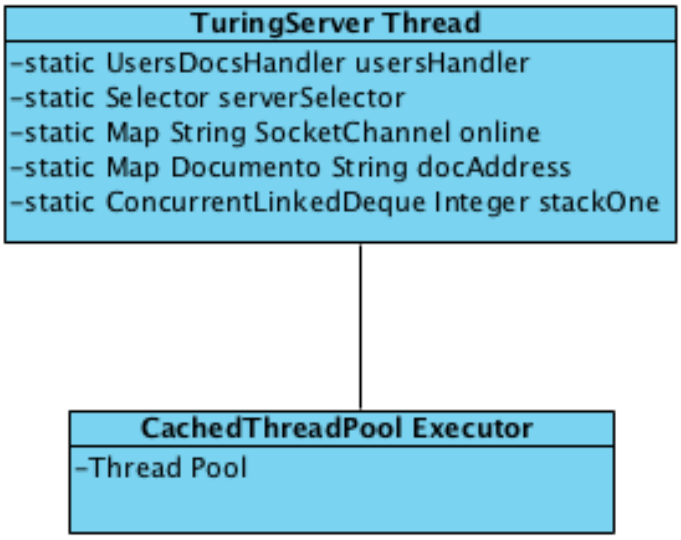
Sulla parte grafica lato client è stato preferito AWT a Swing in quanto il primo utilizza esclusivamente il sistema grafico nativo del sistemo operativo in cui è in esecuzione il codice. Data la necessità di una parte grafica minimale è stato ritenuto sufficiente l'utilizzo di questa tecnologia.

La concorrenza dei thread, sia sul client che sul server viene gestita dove necessario con ReentrantLock che garantiscono la mutua esclusione della risorsa a cui afferiscono. In altri casi si è preferito fare affidamento a classi sviluppate appositamente per la gestione di dati in modo concorrente, ad esempio ConcurrentHashMap e synchronizedList.

2 Schema Thread e Strutture Dati

Nella seguente rappresentazione, sono stati inseriti anche alcuni Thread inerenti la gestione della grafica lato client, e solo le strutture dati fondamentali.





3 Descrizione delle classi

Breve descrizione delle classi realizzate e necessarie per l'implementazione proposta del Client Turing.

- TURING: Classe che implementa il frame AWT lato client, che orchestra tutte le funzionalità del client.
- RichiestaTCP: Classe che implementa la struttura dati per la creazione e invio delle richieste;
- RispostaTCP: Classe che implementa la struttura dati per la creazione e invio delle risposte;
- EditFrame: Classe che implementa il frame AWT di modifica della sezione e visualizzazione chat;
- Chat: Classe che implementa il pannello chat del frame di editing della sezione di un documento;
- MuticastChat: Classe che implementa il task di ricezione messaggi e il metodo di invio messaggi;
- ImageComponent: Componente estetico che implementa un messaggio ricevuto nella chat;
- NuovoFile: Classe che implementa il frame AWT per la creazione di un nuovo file;
- Login: Classe che implementa il frame AWT per il login dell'utente;
- Registrazione: Classe che implementa il frame AWT per la registrazione;
- RMIClientInstauration: Classe che implementa il task di instaurazione connessione RMI lato client;
- InterfaceUser: Interfaccia per la registrazione via RMI;

Breve descrizione delle classi realizzate e necessarie per l'implementazione proposta del Server Turing.

- TuringServer: Classe che implementa il thread principale del server;
- Documento: Classe che descrive un documento;
- Utente: Classe che concretizza un utente rispetto al sistema;
- UserDocsHandler: Classe per la gestione degli utenti registrati e delle associazioni dei documenti agli utenti;
- Utility: Classe di supporto alle funzionalità del server;
- WriteTask: Classe che implementa il task di risposta ad una richiesta ovvero elaborazione ed invio.
- ReadTask: Classe che implementa il task di ricezione richiesta da parte del client;

4 Istruzioni per la compilazione e l'esecuzione

Essendo necessarie due librerie esterne, Gson.jar e ApacheCommonsCodec.jar il seguente comando assolve lo scopo di compilare facendo rifermento alle suddette librerie.

Dopo essersi posizionati nella directory principale di Turing:

```
javac -cp lib/gson-2.8.5.jar:lib/commons-codec-1.12.jar: src/*
mv src/*.class .
```

On Windows:

```
javac -cp lib/gson-2.8.5.jar;lib/commons-codec-1.12.jar; src/*
cd src
move *.class ../
```

Dopo la compilazione i seguenti comandi per mandare in esecuzione Client e Server

```
java -cp lib/gson-2.8.5.jar:lib/commons-codec-1.12.jar: Turing  &
java -cp lib/gson-2.8.5.jar:lib/commons-codec-1.12.jar: TuringServer
```

On Windows:

```
START /B java -cp lib/gson-2.8.5.jar;lib/commons-codec-1.12.jar; Turing
java -cp lib/gson-2.8.5.jar:lib/commons-codec-1.12.jar: TuringServer
```

In alternativa su MacOS o Linux è possibile usare il file bash howToCompileAndRun.sh, che procede a compilare ed avviare un'istanza server e due istanze client.

Lo script è situato nella root directory del progetto.

```
chmod 775 howToCompileAndRun.sh
./howToCompileAndRun.sh
```