Università di Pisa

Dipartimento di Informatica

Relazione del progetto `TURING`

RCL a.a. 18-19

Autore: Michele Zoncheddu

Maggio 2019

Indice

1 Scelte progettuali **2**
- 1.1 Architettura del server 2
- 1.2 Comunicazione client-server 2
- 1.3 Architettura del client 3

2 Gestione della concorrenza **5**
- 2.1 `userManager` 5
- 2.2 `documentManager` 5
- 2.3 `addressManager` 5

1 Scelte progettuali

Di seguito verranno illustrate le principali scelte progettuali effettuate durante la realizzazione del progetto.

1.1 Architettura del server

L'architettura del server è stata scelta tra due soluzioni, che sono state trattate più in dettaglio durante il Laboratorio di Reti di Calcolatori:

1. multithread sincrona con I/O bloccante (`Sockets` di `Java IO`);
2. monothread sincrona con I/O non bloccante (`Selectors` di `Java NIO`).

Le due soluzioni hanno pregi e difetti complementari, ma il trade-off principale è tra **velocità** e **scalabilità**. Ai fini di questo progetto didattico è stata ritenuta più importante la reattività, garantita (sotto carichi non eccessivi) dalla prima soluzione.

Possiamo dividere il server in due livelli: **interfaccia** e **core**.

1. A livello di interfaccia si trovano:
   (a) Il socket TCP del server
   (b) L'API del servizio di registrazione utente
   (c) Il servizio RMI di notifiche push.
2. A livello core si trovano:
   (a) Il thread pool di client handlers
   (b) I manager di utenti, documenti e indirizzi IP multicast.

Per ogni client che si connette, il server riserva un thread, che si occuperà di gestire la connessione TCP per l'intero tempo di vita del client.

1.2 Comunicazione client-server

Per la comunicazione client-server è stata adottata una tecnica a scambio di messaggi sincrona. I messaggi sono in formato JSON, e vengono inviati attraverso una connessione TCP. La scelta del formato JSON al posto degli oggetti Java serializzati può permettere in un futuro momento di scrivere client per `TURING` in qualsiasi linguaggio di programmazione[1].

[1] A patto di modificare leggermente la parte di registrazione di un nuovo utente, che è stata richiesta esplicitamente in Java RMI.

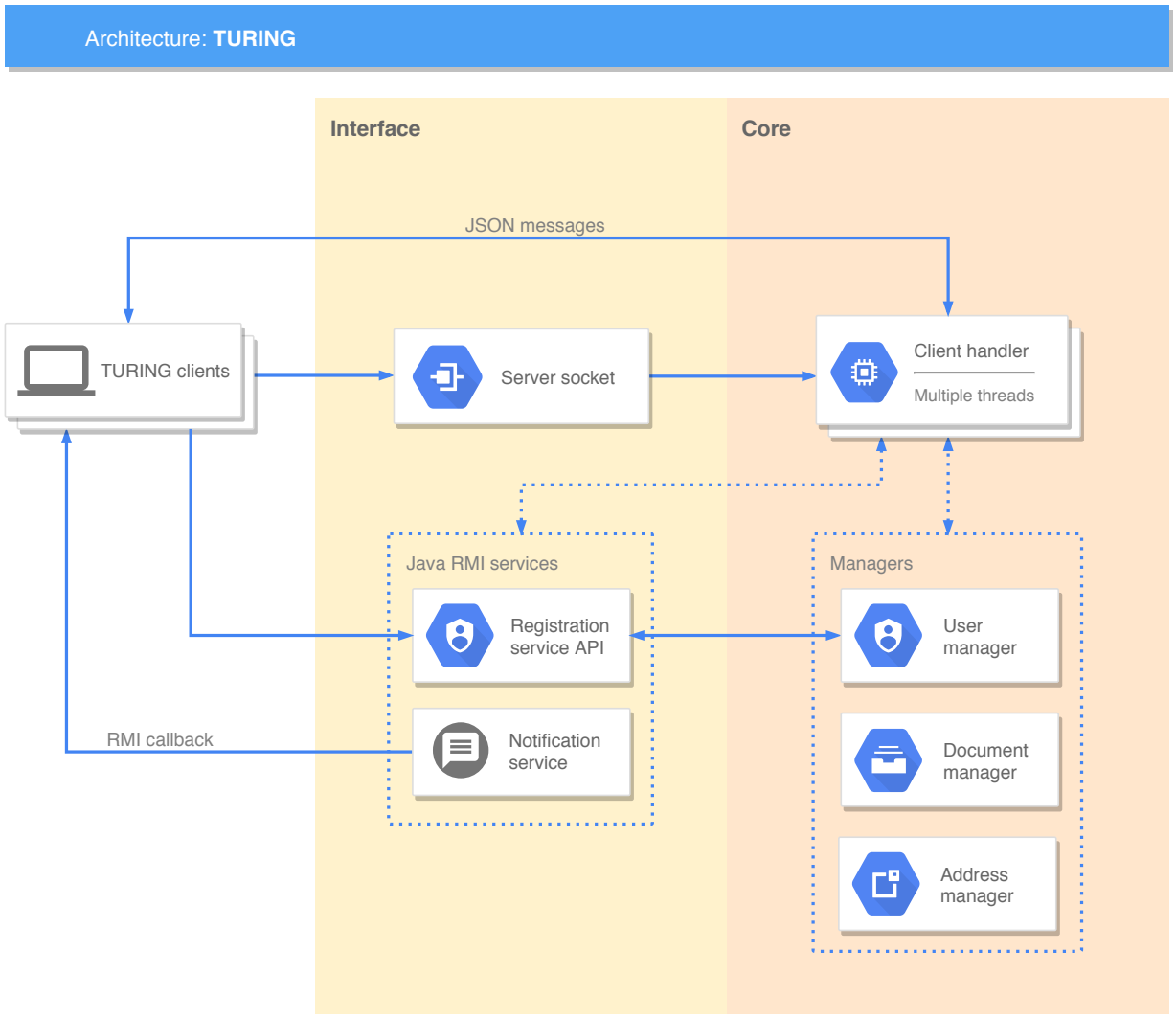

Figura 1: Architettura di `TURING` e interazioni tra le varie componenti.

1.3 Architettura del client

L'architettura del client è più semplice: effettua richieste in formato JSON e resta in attesa di una risposta. Il thread grafico è aggiornato all'occorrenza dal metodo *invokeLater*. L'interfaccia grafica si adatta automaticamente alle dimensioni dello schermo, ed è stato cercato di renderla il più coerente, gradevole e intuitiva possibile.

Dalla schemata di registrazione e login si può accedere all'area personale di `TURING`, dove ogni utente può creare nuovi documenti, invitare altri utenti alla modifica, e in una tabella dinamica vede i documenti che può modificare, con la lista delle sezioni e informazioni aggiuntive, come il nome del creatore del documento (indispensabile nel caso di documenti omonimi) e un'icona che indica se il documento è stato condiviso o no. La tabella si aggiorna in tempo reale quando si viene invitati alla modifica di un nuovo documento (con apposito messaggio di notifica), ma è anche possibile forzare un aggiornamento manuale tramite il tasto "refresh".

Dopo che una sezione di un documento è stata selezionata per la modifica, viene visualizzata la schermata di lavoro, nella quale è possibile modificare il testo precedentemente salvato, chattare con gli utenti che stanno modificando lo stesso documento, salvare le modifiche o ignorarle. I documenti sono codificati nello standard `UTF-8`.

Il formato dei messaggi è indicato nel file `message_format.txt`, assieme ad un esempio di messaggio che richiede al server la creazione di un documento.

2 Gestione della concorrenza

I thread sono gestiti da un **cached thread pool**, scelto per la sua elasticità nella gestione delle risorse; sono inoltre indipendenti ed isolati, e per ogni richiesta esterna devono utilizzare uno dei seguenti manager:

- `userManager`: gestisce concorrentemente gli utenti registrati;
- `documentManager`: gestisce concorrentemente i documenti salvati;
- `addressManager`: gestisce concorrentemente gli indirizzi IP multicast da assegnare alle chat dei gruppi di lavoro.

2.1 `userManager`

È il nucleo del servizio di autenticazione di `TURING`, ed è implementato da una tabella hash concorrente (chiave: username).

Viene utilizzato dal server come manager locale per il login e per i controlli di sicurezza delle operazioni; inoltre viene utilizzato dai client come manager remoto per la registrazione, invocando i metodi esposti da una API.

2.2 `documentManager`

Implementato da una tabella hash non concorrente[2] (chiave: username concatenato con il nome del documento), gestisce la creazione, la modifica e l'insieme degli utenti autorizzati alla modifica.

2.3 `addressManager`

Per assegnare e rilasciare su richiesta gli indirizzi multicast per le chat dei documenti, ho utilizzato un *TreeSet* con un comparatore apposito per indirizzi IP, in quanto la classe *InetAddress* non è nativamente comparabile in Java.

Viene riservato un indirizzo da 239.0.0.0 a 239.255.255.255[3] non appena un utente inizia a modificare una sezione di un documento, e liberato quando l'ultimo utente termina la modifica del documento.

[2] In questo caso la corretta gestione della concorrenza non è garantita da una struttura dati concorrente, ma dalle locks delle sezioni, in quanto hanno una granularità più fine.

[3] Organization-Local Scope. Fonte: IANA.