TURING

Relazione del progetto

Francesco Pirrò - Matr. 544539

Indice

1 Introduzione

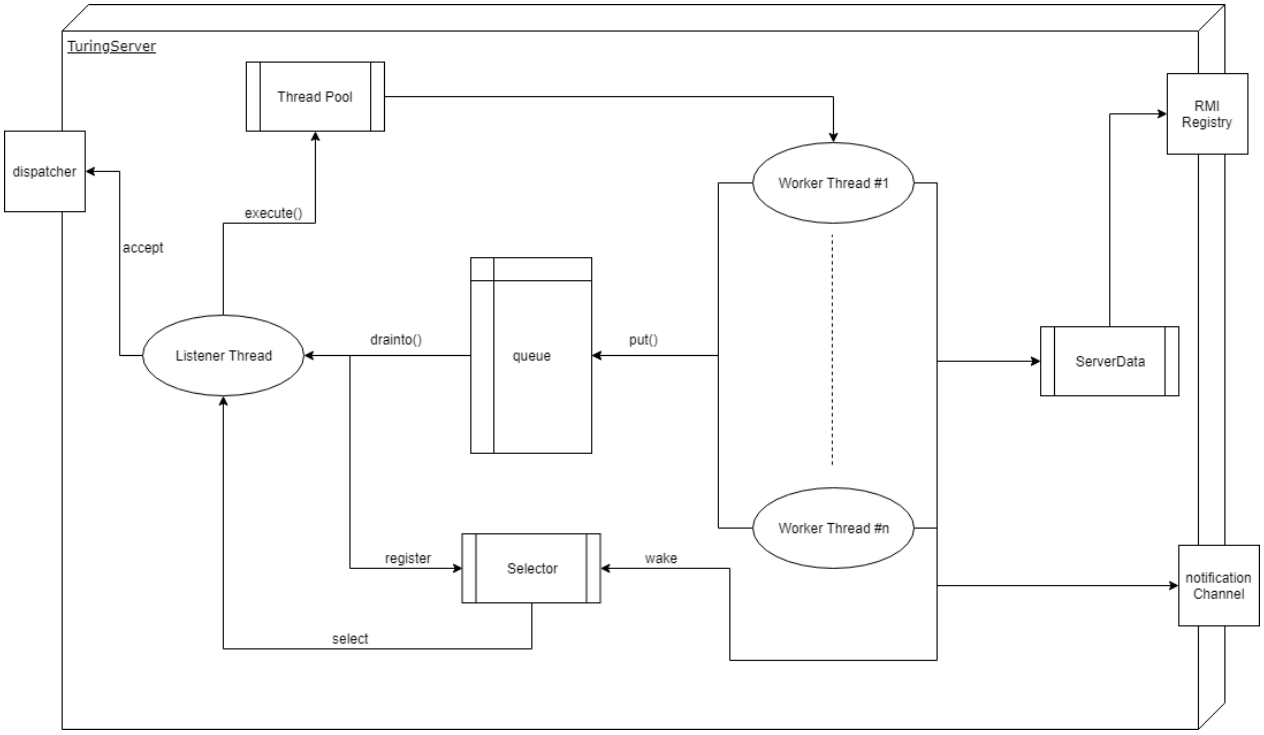
Lo scopo del progetto è la realizzazione di uno strumento per l'editing collaborativo di documenti. Un utente, una volta registrato al servizio ha la possibilità di creare nuovi documenti, invitare altri utenti all'editing, mostrare l'intero documento o mostrare solo una singola sezione. Più utenti che stanno modificando sezioni diverse dello stesso documento hanno la possibilità di comunicare tra di loro tramite l'utilizzo di una chat. Il server conserva il contenuto dei documenti, tiene traccia degli utenti registrati e di quello che stanno modificando, sta in attesa di richieste da parte del client e si occupa di inviare le opportune risposte. Il client fornisce una interfaccia grafica all'utente per poter facilmente inviare richieste al server, visualizzare le risposte e comunicare con gli altri utenti mentre si sta modificando una sezione.

2 Architettura Generale

Il sistema TURING è composto da un server unico a cui sono connessi uno o più client. I client comunicano con il server tramite una connessione TCP, e ogni richiesta dei client è gestita all'intero di un threadpool nel server. Il server può anche comunicare alcune notifiche al client tramite messaggi spediti con protocollo UDP. Per poter registrare un nuovo utente nel servizio, il client può eseguire una "Remote Method Invocation" sul server.

2.1 TuringServer

Il server è composto da un thread listener e un pool di worker thread fissato, come specificato nel file di configurazione. Tutti i socket per le connessioni sono impostati in modalità non-blocking, il thread listener quindi utilizza un selector per determinare quali sono pronti per una operazione di "I/O": un solo socket detto "dispatcher" è registrato per le operazioni di accept, mentre tutti gli altri per le read. Quando il dispatcher è pronto vuol dire che un client sta cercando di collegarsi al server: il server quindi accetta la connessione e registra il nuovo socket sul selector; quelli che sono pronti per una operazione di read invece vengono de registrati dal selector e un thread che legge la richiesta e la esegue viene aggiunto nel thread pool. Il socket viene de registrato per evitare che più thread, con il compito di eseguire la richiesta dello stesso client, vengano generati solo perché la richiesta ancora non è stata letta: in questo modo esiste al più uno e un solo thread che si occupa della comunicazione con un singolo client. Quando il thread finisce l'operazione comunica al listener che il socket può' eseguire una nuova richiesta utilizzando una coda, il listener prima di procedere con una select ri-registra tutti i socket contenuti nella coda per le operazioni di read. La disconnessione del client ha come effetto sul selector che il socket risulterà pronto per una operazione di read, ma la read di per sé restituirà 0 byte: questo effetto si può' utilizzare per capire quando una disconnessione è avvenuta e chiudere il socket (eventualmente fare anche il log-out di un utente).



L'operazione di registrazione al servizio è implementata tramite una RMI invocata dal client: il server esporta un oggetto istanza di una classe che implementa RemoteUserTable e che quindi fornisce il metodo che può' essere invocato dal client. Il Registro RMI è aperto su una porta definita dal file di configurazione.

Eventuali notifiche che il server deve mandare al client sono spediti tramite UDP al client, la porta usata dal client per la comunicazione UDP è fornita al momento del login insieme a username e password.

Al momento dell'apertura il server provvederà alla pulizia della cartella specificata nel file di configurazione e che dovrebbe essere usata per conservare le sezioni dei file degli utenti: una conferma verrà chiesta all'utente per assicurarsi che il server non cancelli inaspettatamente dati importanti sulla macchina in cui è in esecuzione.

2.1.1 Strutture Dati

Il server possiede una serie di variabili statiche che contengono vari valori letti dal file di configurazione, quali le porte da utilizzare per alcuni servizi, il numero di thread del pool e il range di indirizzi multicast da generare. Un'istanza della classe Properties detta "config" contiene i valori di configurazione letti dal file, mentre un'altra istanza chiamata defaultConfig contiene una serie di valori di default. Un'istanza di ServerData si occupa dunque di conservare tutte le informazioni necessarie al server circa i documenti presenti e gli utenti registrati: tutte le operazioni richieste dai client hanno, in qualche modo, sempre a che fare con dati contenuti all'intero di ServerData. Una BlockingQueue viene condivisa tra listener e thread worker per poter comunicare quei socket che hanno appena terminato l'esecuzione di una richiesta, e che quindi necessitano di essere riaggiunti nel selector per poter richiederne un'altra o anche semplicemente per segnalare la disconnessione. Un ThreadPoolExecutor gestisce i vari worker thread: in questo caso il numero di thread è fissato, e tale numero è deciso dal file di configurazione, altrimenti ha un valore di default di 8 thread. Un SocketChannel detto dispatcher, aperto su una porta specifica, fornisce un punto di connessione per i client ma che necessitano comunque di una accettazione esplicita della connessione da parte del server: questo avviene correttamente grazie al selector che notifica quando un'operazione di accept è disponibile. Un DatagramChannel è anch'esso aperto per poter comunicare via UDP con il client. I thread worker sono implementati da ServerExecutor che utilizzano il socket ottenuto dal selector per poter recuperare la richiesta e comunicare la risposta corrispondente. Un ServerExecutor esegue una sola richiesta e poi termina (possibilmente verrà sostituito da un altro serverExecutor che si starà occupando della richiesta di un altro client)

2.1.2 Threads e Concorrenza

Il Numero di Thread presenti nel server in un qualsiasi momento sarà al più 1+numThreads: un thread listener e numThreads thread worker. Come già spiegato in precedenza, i thread worker riescono a comunicare con il listener tramite una BlockingQueue (la cui implementazione è thread-safe). Il set del selector non è thread-safe: per questo si evita che siano i worker stessi ad effettuare l'operazione di register e si lascia il compito al thread listener, che raggiungerà tutti i socket che deve prima di effettuare una nuova select. Tutti i thread worker condividono lo stesso ServerData, la cui implementazione thread-safe verrà discussa successivamente.

2.2 TuringClient

Il programma Client prende come argomento l'indirizzo del server a cui deve connettersi mentre la porta TCP del server, la porta per i servizi RMI e il path della cartella in cui salvare i file ricevuti dal server sono passati da un file di configurazione. Il valore di default per la porta RMI e la porta TCP sono i valori di default definiti nel server. Una volta che la connessione con il server è stata stabilita correttamente, il client lancia l'interfaccia grafica con cui l'utente può inviare le richieste al server tramite TextBox e pulsanti. Tutte

le richieste che vengono effettuate si appoggiano su un RequestExecutor che fornisce per ogni richiesta un metodo con gli argomenti di cui necessita e si occupa di comunicare la richiesta al server, una volta ricevuta la risposta viene restituita sotto forma di MessageBuffer. Il MainForm, che si occupa di gestire l'interfaccia grafica, utilizza per ogni operazione lo stesso RequestExecutor. Il client apre una connessione UDP su cui si mette in ascolto per ricevere eventuale notifiche da parte del server, ed eventualmente può aprire una seconda connessione UDP che sfrutta un indirizzo multicast fornito dal server per comunicare con altri client che stanno modificando sezioni diverse dello stesso documento.

2.2.1 Strutture Dati

Il client non possiede particolari strutture dati, considerando che molti dei dati delle richieste sono conservate nei componenti dell'interfaccia grafica.

2.2.2 Threads e Concorrenza

Il client, se si trascurano i thread per la gestione del loop degli eventi e dell'interfaccia grafica, ha un thread particolare il cui compito è quello di restare in attesa di eventuali notifiche da parte del Server su un DatagramChannel. Il canale è in modalità blocking, quindi il thread rimarrà in attesa finché un messaggio di notifica sarà arrivato dal server che a quel punto sarà mostrato tramite una finestra di dialogo. Questo thread viene attivato quando si apre per la prima volta la MainForm e rimane attivo finché la MainForm non sarà chiusa. Un altro thread che il Client può' generare è il thread ChatRoom, che rimane in ascolto di eventuali messaggi di altri utenti su un DatagramChannel collegato ad un gruppo di multicast. Il thread ChatRoom è generato solo quando il Client entra nella form di editing di un documento: una richiesta è mandata al server che risponde con l'indirizzo di multicast e la porta della corrente sessione di chat legata al documento. Il thread è interrotto quando si esce dalla form di editing visto che esaurito il suo scopo.

2.2.3 Interfaccia Grafica

L'interfaccia grafica del client è composta da 3 finestre principali alle quali si posso aggiungere svariate finestre di dialogo per comunicare gli esiti delle operazioni richieste al server. La prima finestra che compare è la form di log in che contiene due TextBox in cui inserire nome utente e password e due bottoni per effettuare il log in o la registrazione al servizio. Se il log in avviene con successo si passa alla finestra principale, la MainForm che contiene 2 TextBox (una che contiene le informazioni dei documenti a cui l'utente corrente può' accedere, l'altra un log degli esiti ricevuti dal server), un bottone per aggiornare le informazioni sui documenti, un bottone per effettuare il log out dell'utente e, cosa più importante, un riquadro a schede dal quale si possono scegliere le varie richieste da mandare al server: La richiesta di creazione di un documento, l'invito di un utente, il download di un documento intero, il download di una sezione intera e la richiesta di editing di un documento intero. Se una richiesta di editing di un documento ha avuto esito positivo, viene aperta una form di editing in cui viene specificato il path del file che si sta modificando al momento e una chatbox per poter comunicare con gli altri utenti. Ad ogni richiesta una finestra di dialogo che ne mostra l'esito viene generata, a meno che non si tratti di una richiesta di editing, che in caso di esito positivo porta direttamente alla sua form.

3 Comunicazione Client-Server

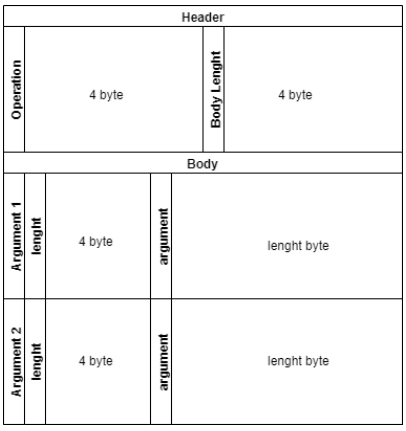
Come è stato già descritto nell'architettura generale, client e server comunicano tra di loro sfruttando 3 vie: le richieste con le relative risposte vengono scambiate utilizzando una connessione TCP, la richiesta di registrazione al servizio viene effettuata tramite una Remote Method Invocation mentre eventuali notifiche che il server vuole mandare al client vengono spedite tramite UDP.

3.1 TCP

Tutte le richieste del client, con l'eccezione della registrazione, vengono comunicate al server tramite una connessione TCP in quanto questa garantisce affidabilità. Nel Server i socket che si occupano di gestire la comunicazione con i client sono in modalità non-blocking, il che permette che possano essere utilizzati con una selector e generare un thread solo quando effettivamente una richiesta è arrivata. Per il client non c'è necessità di avere un socket in modalità non-blocking in quanto deve gestire una comunicazione con un solo server, e senza la risposta di esso non può avanzare. Ha quindi senso che i socket dei client siano bloccanti, mentre quelli dei server no. Il client può' inoltre chiedere una sola operazione alla volta per evitare che più risposte di richieste diversi si confondano tra di loro e il client non riesca più a capire quale esito corrisponda a quale richiesta.

3.1.1 Formato del Messaggio

Client e Server si scambiano messaggi che utilizzano lo stesso formato sia per le richieste che per gli esiti. Il messaggio è composto da un header di 8 byte e di un corpo di dimensione variabile. I primi 4 byte codificano un intero che corrisponde al tipo di richiesta/esito ricevuto, i restanti 4 byte codificano la dimensione del corpo del messaggio. Il corpo del messaggio può' però avere più argomenti di dimensione variabile: ogni argomento è composto da 4 byte che ne codificano la lunghezza e il suo contenuto. Vengono letti argomenti fino a quando l'intero corpo del messaggio è esaurito.



3.1.2 Descrizione Operazioni

L'operazione di login è codificata dall'intero 1 e ha 3 argomenti: nome utente, password e numero di porta UDP da utilizzare per la comunicazione delle notifiche. Il messaggio di risposta a questa operazione è senza corpo in quanto ne comunica solo l'esito positivo o negativo

L'operazione di creazione di un documento è codificata dall'intero 2 e ha 2 argomenti: il nome del documento da creare e il numero di sezioni in cui il documento deve essere diviso. Il messaggio di risposta a questa operazione è senza corpo in quanto ne comunica solo l'esito positivo o negativo

L'operazione di download è codificata dall'intero 3 e può avere 1 o 2 argomenti: se ha un solo argomento l'utente vuole il file intero e l'argomento ne specifica il nome; se sono due si vuole solo una sezione il primo argomento specifica il nome e il secondo la sezione. Se l'esito è positivo il corpo ha due argomenti: nel primo si trova il file richiesto (completo o singola sezione) e nel secondo una stringa che comunica al client quali utenti e quali sezioni sono eventualmente in fase di editing. In caso di fallimento il corpo ha 0 argomenti.

L'operazione di richiesta della lista di documenti è codificata dall'intero 4 e non ha argomenti. Quando il server riceve una richiesta del genere risponde con un messaggio che contiene un solo argomento: in questo argomento è contenuta una stringa che descrive tutti i documenti a cui l'utente in questione ha accesso. Se l'utente in questione non ha accesso a nessun documento, una stringa che comunica questo dato verrà spedita.

L'operazione di invito di un utente è codificata dall'intero 5 e ha 2 argomenti: nel primo è contenuto il nome del documento e nel secondo il nome dell'utente da invitare. Il messaggio di risposta è senza corpo e contiene solo l'esito positivo o negativo.

L'operazione di richiesta di editing è codificata dall'intero 6 e ha 2 argomenti: nel primo è specificato il nome del documento e nel secondo la sezione da modificare. Se la sezione è disponibile e l'esito positivo il messaggio di risposta conterrà il file da modificare come unico argomento, altrimenti il corpo sarà vuoto.

L'operazione di richiesta della chat room è codificata dall'intero 7 e ha un solo argomento: il nome del documento. Se l'esito è positivo il messaggio contiene 3 argomenti: nel primo si trova l'indirizzo di multicast utilizzato dalla chat room, nel secondo la porta utilizzata da tutti i client per la chat e nella terza il nickname che bisogna utilizzare per identificarsi nella chat.

L'operazione di richiesta di fine editing è codificata dall'intero 7 e ha 3 argomenti: il nome del file, la sua sezione e il contenuto del file aggiornato. Una volta verificata la validità della richiesta un semplice messaggio di esito positivo o negativo (senza argomenti) sarà spedito al client.

In caso il server dovesse ricevere una richiesta con un numero sbagliato di argomenti, o un identificatore di operazione invalido sarà restituito al client un errore INVALID REQUEST codificato con -15

3.2 RMI

Il server, al momento della sua apertura, mette a disposizione un registro RMI, a quale il client può accedere per effettuare una invocazione remota di un metodo e inserire un nuovo utente. La struttura dati che implementa RemoteUserTable è legata al registro tramite la keyword "USERTABLE-TURING": questa keyword verrà poi utilizzata dal client per ottenere il suo stub su cui effettuare la chiamata remota. RemoteUserTable

fornisce solo il metodo per poter registrare un nuovo utente, quindi nessun altro metodo può essere invocato dal client da remoto.

3.3 UDP

Quando un utente A invita un utente B a modificare un documento, il server ha bisogno di comunicare in qualche modo all'utente B questo evento: utilizzare la stessa connessione TCP utilizzata per la comunicazione delle richieste avrebbe comportato una complicazione sul lato client. Il client controlla la connessione con il server solo quando sta attendendo una risposta, quindi la notifica non sarebbe arrivata fino al momento in cui il client non avesse deciso di effettuare un'operazione. Si è deciso quindi di usare una comunicazione dedicata per questo genere di eventi: il server manda al client una stringa in cui è espresso il contenuto della notifica. Per questa comunicazione è stato utilizzato il protocollo UDP in quanto i messaggi scambiati sono brevi, pochi e molto distanti tra di loro. La poca affidabilità del protocollo UDP non è un problema in quanto la mancata notifica di un evento non va a influire sull'esito dell'evento stesso: il messaggio potrebbe non arrivare ma l'utente B rimane invitato.

4 Comunicazione tra Client

I client possono inoltre comunicare tra di loro tramite gruppi multicast. Ad ogni documento attualmente in modifica, il server attribuisce un indirizzo multicast che può essere utilizzato dagli utenti editor per comunicare tra di loro. L'indirizzo è generato dal server in maniera randomizza in un range specifico e tutti gli utenti che stanno modificando sezioni diverse dello stesso file possono mettersi in ascolto tutti sullo stesso gruppo e scambiarsi messaggi. Il DatagramChannel che si occupa di questa comunicazione è obbligato ad utilizzare socket che utilizzano indirizzi IPv4, perché gli indirizzi generati sono indirizzi IPv4. Il Time to Live dei messaggi spediti è impostato ad 1 in modo che questi pacchetti non escano dalla rete locale. La dimensione massima di un messaggio è 1024 byte.

5 Packages

5.1 ChatRoom

Il package ChatRoom contiene classi che permettono sia al Client che al Server di implementare correttamente la chat multicast

5.1.1 ChatOrganizer

ChatOrganizer è una classe utilizzata dal Server che permette di generare indirizzi IPv4 in un range definito alla creazione, inoltre tiene traccia di quali indirizzi sono attualmente in uso e quali no. Un indirizzo in uso può diventare libero se viene invocato correttamente il metodo CloseRoom. Gli indirizzi sono generati come

numeri interi, e successivamente da numeri interi trasformati in stringhe che contengono l'indirizzo in dotted notation. ChatOrganizer è thread-safe

5.1.2 ChatRoom

ChatRoom è una classe utilizzata dal Client e che implementa il thread che riceve eventuali messaggi ricevuti dal gruppo multicast e li inserisce in una TextBox di cui ne possiede il riferimento. ChatRoom inoltre fornisce un metodo sendMessage che permette di inviare un messaggio al gruppo.

5.2 ClientGui

Il package ClientGui contiene le classi utilizzate per implementare l'interfaccia grafica del Client. Si va dagli handler dei bottoni alle form.

5.2.1 ButtonHandler

ButtonHandler implementa l'interfaccia ActionListener che gli permette di implementare gli handler dei bottoni utilizzati per le varie richieste al server. Il tipo di richiesta effettuata viene deciso dal valore di Operation, specificato al momento della creazione. Le operazioni di comunicazioni con il server sono effettuate tramite un'istanza di RequestExecutor

5.2.2 EditorForm

EditorForm implementa la finestra di dialogo utilizzata durante le operazioni di editing di un documento: Questa form fornisce la ChatBox per comunicare con gli altri utenti e un pulsante per mandare una richiesta di terminazione dell'editing. La form ricorda, inoltre, all'utente il path del file che si sta modificando al momento. Alla sua apertura EditorForm si occupa di richiedere al server l'indirizzo di multicast e di avviare il thread ChatRoom; che verrà interrotto alla sua chiusura. Se si tenta di chiudere la form senza che si sia prima fatto un end edit, l'applicazione verrà chiusa.

5.2.3 EnterListener

EnterListener implementa un banalissimo KeyListener che preme un pulsante specificato al momento della creazione quando registra che è stato premuto il pulsante invio. Questa classe è utilizzata per permettere all'utente di inviare una richiesta anche solo premendo invio, senza dover premere il pulsante con il mouse.

5.2.4 LogForm

LogForm implementa la finestra di dialogo utilizzata durante le fasi di Login: Questa form permette sia di fare un log in normale, che di registrare un nuovo utente al servizio. Se si effettua una sign up con successo,

la form invierà automaticamente una richiesta di log in in modo che il client sia subito utilizzabile. Se si cercherà di chiudere la finestra senza che prima ci sia stato un log in di successo, l'applicazione verrà chiusa.

5.2.5 MainForm

MainForm implementa la finestra principale, nella quale sono contenute le richieste da fare al server e varie informazioni sui documenti modificabili e un log degli esiti delle operazioni effettuate. Se un log out viene richiesto, una richiesta viene mandata al server e una nuova finestra di log in viene aperta. Un EditorForm è aperto ogni qual volta l'utente richiede un edit. Al suo avvio il MainForm attiva anche un thread NotificationListener, per poter catturare eventuali notifiche da parte del server. Questo thread è interrotto quando la MainForm è chiusa

5.2.6 NotificationListener

NotificationListener implementa un thread che sta in attesa di notifiche da parte del server, e ne mostra il contenuto tramite una finestra di dialogo

5.2.7 RequestExecutor

RequestExecutor implementa una classe che si occupa di prendere gli argomenti corretti di una operazione, di preparare il messaggio da spedire al server e di restituire l'esito sotto forma di MessageBuffer. Il RequestExecutor è unico per tutto il client, infatti ogni metodo utilizza la keyword synchronized così che l'accesso alla socket avvenga in mutua esclusione, e più messaggi non si confondano tra di loro.

5.2.8 ResultDialog

ResultDialog implementa una finestra di dialogo che mostra un messaggio con un pulsante "OK" che permette di chiuderla. una ResultDialog può essere configurata in modo che alla sua chiusura, anche la finestra padre che l'ha generata venga chiusa o che l'intera applicazione venga chiusa: queste opzioni sono molto comode per gestire errori durante la comunicazione e il passaggio da una form ad un'altra

5.2.9 SelectOperation

SelectOperation implementa il blocco a schede che contiene le varie richieste che si possono inviare al server. Ogni scheda contiene delle textBox per inserire gli argomenti e un pulsante per inviare la richiesta desiderata.

5.3 Message

Il package Message è utilizzato sia dal server che dal client, e si occupa dell'implementazione del formato dei messaggi, del loro invio e della loro ricezione

5.3.1 MessageBuffer

MessageBuffer implementa il formato dei messaggi descritto precedentemente: un MessageBuffer può essere ottenuto creandone uno nuovo passando come argomento un tipo Operation (che codifica i primi 4 byte dell'header) e un numero variabile di argomenti passati come array di byte; oppure tramite il metodo statico readMessage che permette di leggere un messaggio da un socket. Un MessageBuffer può essere poi spedito usando il metodo send() oppure si possono ottenere gli argomenti utilizzando un getArgs().

5.3.2 Operation

Operation è un tipo Enumeratore che permette di esprime facilmente le varie codifiche degli header dei MessageBuffer. Esiste un metodo statico per ottenere l'Operation corretto in base all'intero e un metodo statico per ottenere una Stringa che descrive l'Operation indicato come argomento.

5.4 RemoteUserTable

Il package RemoteUserTable contiene l'omonima interfaccia utilizzata dal client e dal server per implementare il servizio RMI

5.5 ServerData

Il package ServerData contiene tutte le strutture dati utilizzate dal server per tenere traccia delle informazioni fondamentali per una corretta esecuzione del servizio.

5.5.1 Document

Document implementa una struttura dati thread-safe che contiene tutte le informazioni fondamentali di un documento quali il nome dello stesso, il nome del suo creatore, il numero di sezioni, la dimensione massima di ogni sezione, il path di ogni sezione, gli utenti che possono modificare il documento e quali lo stanno facendo in questo momento. La classe inoltre fornisce un set minimo di metodi per poter eseguire operazioni come il download, l'invito e le richieste di editing e stop editing.

5.5.2 DocumentTable

DocumentTable implementa una struttura dati thread-safe che, tramite una ConcurrentHashMap, conserva le informazioni di tutti i documenti associando al loro nome la corretta istanza di Document. Il nome contenuto come chiave nella hash map non è lo stesso contenuto nell'istanza di Document, in quanto per poter differenziare documenti con nomi identici ma creatori diversi l'identificatore univoco dei documenti è dato dal "nome del creatore"/"nome del documento". La classe inoltre fornisce un set minimo di metodi per poter eseguire operazioni come la creazione di un nuovo documento, il download, l'invito e le richieste di editing e stop editing.

5.5.3 ServerData

ServerData implementa una struttura dati che conserva al suo interno tutte le informazioni fondamentali per una corretta esecuzione del server: oltre a DocumentTable e UserTable contiene una map di Socket e Username per rappresentare gli utenti connessi, e un'istanza di ChatOrganizer per poter generare gli indirizzi multicast.

5.5.4 ServerExecutor

ServerExecutor implementa un thread che legge un MessageBuffer da un socket, esegue la corretta operazione in base alla richiesta e prepara e spedisce il risultato. Una volta che la richiesta è stata soddisfatta ServerExecutor comunica al thread listener di riaggiungere il socket al selector tramite una coda, e infine sveglia il selector per assicurarsi che il suo set venga aggiornato il prima possibile

5.5.5 User

User implementa una struttura dati thread-safe che contiene tutte le informazioni fondamentali di un utente quali il suo nickname, la sua password, i documenti che può modificare, il numero di notifiche non ancora ricevute,il documento che sta modificando attualmente e il suo indirizzo socket per inviare le notifiche. La classe inoltre fornisce un set minimo di metodi per poter eseguire operazioni come il login, l'invio di notifiche pendenti, l'aggiunta di notifiche e il logout.

5.5.6 UserTable

UserTable implementa una struttura dati thread-safe che, tramite una ConcurrentHashMap, conserva le informazioni di tutti gli utenti associando al loro nickname la corretta istanza di User. La classe inoltre fornisce un set minimo di metodi per poter eseguire operazioni come il login, l'invio di notifiche pendenti, l'aggiunta di notifiche e il logout e implementa l'interfaccia RemoteUserTable per poter creare nuovi utenti

6 Gestione File

6.1 Server

Il server conserva tutti i file in una cartella specificata tramite il file di configurazione. All'avvio del server questa cartella viene ripulita di qualsiasi contenuto, per evitare che vi siano file rimasti da qualche precedente esecuzione del server. Un avviso è comunque mandato all'utente per avvisarlo dell'imminente cancellazione ed annullare in caso qualche file importante stia per essere cancellato. La stessa notazione utilizzata per definire univocamente i file, è utilizzata per i path delle sezioni: la prima cartella è quella con il nome del creatore, la seconda è quella con il nome del documento. Nella seconda cartella si trovano un numero di file pari al numero di sezioni: ogni sezione è salvata in un file particolare, per semplificare l'accesso ad una sezione casuale di un documento. Quando un intero file o una singola sezione è richiesta da un utente, il contenuto è inserito in un

ByteBuffer. Il ByteBuffer ha come dimensione massima Integer.MAXVALUE byte, la dimensione massima di un documento deve quindi essere limitata per evitare buffer overflow; la dimensione massima di un intero documento è poi divisa tra le sue sezioni, ed ogni sezione avrà il suo massimo. Se si cerca di configurare un massimo superiore ad una certa soglia, il server utilizzerà un valore predefinito, che è di circa 1,9 GB.

6.2 Client

Il client salva i file in una cartella specificata tramite un file di configurazione in maniera del tutto analoga a quella del server; tuttavia quando il client viene avviato non cerca di pulire la cartella in modo da permettere all'utente di poter accedere ai documenti scaricati anche una volta disconnesso dal server, e di non preoccuparsi di perderli quando riaprirà il client ed in caso di conflitto il server sostituirà la vecchia versione con la nuova appena scaricata.

7 Testing

Client e Server sono stati inizialmente testati in ambiente Linux e su una stessa macchina, per assicurasi che funzionassero a dovere su casi molto semplici. Successivamente è stato testato con il server su una macchina Linux e i client su macchine Windows nella rete locale: anche in questo caso tutte le operazioni venivano svolte correttamente. Infine il client è analizzato il comportamento del client nel caso in cui vengano richieste operazioni non valide.