Simple Social

Leonardo Cariaggi

503140

June 6, 2016

Contents

1 Introduzione

Simple Social è una mini rete sociale che permette agli utenti che ne fanno parte di condividere messaggi e rimanere aggiornati sulle ultime notizie pubblicate dagli altri membri.

2 Architettura del sistema

2.1 Componenti del sistema

Il sistema consiste principalmente di due macro elementi, che sono il Social Server e i Social Client. Tra gli utenti possono esistere diversi tipi di relazioni:

- **Amicizia:** due membri possono stabilire tra loro un legame di amicizia. L'utente che vuole stabilire tale legame può inviare una richiesta di amicizia ad un altro utente, che può accettare o rifiutare.
- **Follower:** un utente A può decidere di diventare follower dell'utente B soltanto se esiste già un legame di amicizia tra A e B. Ogni volta che un utente decide di pubblicare un contenuto, tutti i suoi follower riceveranno una copia di esso.

L'architettura generale è di tipo client-server: ogni messaggio che un utente ha intenzione di condividere viene spedito al server che, in base alla rete di amicizie e agli interessi registrati dagli altri utenti, decide come inoltrarlo.

Il Social Server gestisce la maggior parte delle informazioni che compongono Simple Social. Esso tiene traccia di tutti gli utenti registrati (e del loro stato, Online/Offline) e di tutti i legami che intercorrono tra di essi, sia quelli di amicizia che quelli di following. Come già accennato gestisce i messaggi generati dagli utenti ma non prevede la loro memorizzazione.

Il Social Client gestisce l'interazione con l'utente, memorizza i contenuti generati da altri utenti e partecipa ad un gruppo di multicast per notificare al server l'attività o l'inattività dell'utente.

2.2 Interfaccia utente

Simple Social, tramite un menu, offre all'utente la possibilità di scegliere di eseguire una tra le diverse operazioni alla volta. L'interfaccia è a riga di comando.

2.3 Comunicazioni tra utenti e server

La maggior parte delle comunicazioni avviene tramite TCP. Ogni operazione messa a disposizione degli utenti prevede infatti l'apertura di una connessione TCP con il server. Tutte le funzionalità sono state implementate sfruttando i meccanismi di I/O offerti dal package **java.io** di Java.

Fanno eccezione la gestione dei messaggi di keepalive (implementata tramite UDP) e la notifica di nuovi contenuti da parte del server (che utilizza java RMI).

Piuttosto che attivare un singolo thread che gestisce tutte le operazioni è stato definito un thread separato per ogni tipo di operazione (sia dalla parte del

client che da quella del server). Ad ogni funzionalità, dunque, corrisponde una diversa connessione TCP.

È stato scelto di utilizzare i meccanismi di blocking I/O (offerti dal package `java.io`) unitamente al pattern definito nella sezione 3.1 per incapsulare ogni funzionalità in un gruppo di 3 thread e far sì che ad ogni operazione fosse dedicata una diversa connessione TCP.

3 Thread attivati

3.1 Implementazione delle funzionalità

L'implementazione di ogni funzionalità messa a disposizione degli utenti segue il medesimo schema:

- Social Client attiva il thread `<nomeFunzionalità>Thread` e stabilisce una connessione con il server su una precisa porta, diversa per ogni tipo di operazione. Dopo lo scambio di alcuni messaggi (dipendenti dall'operazione) la connessione viene chiusa.
- Social Server, al suo avvio, avvia il thread `Server<nomeFunzionalità>Thread`, che si mette in attesa di connessioni sulla porta specifica dell'operazione. Ogni volta che riceve una richiesta di connessione, questo thread attiva un altro thread, `<nomeFunzionalità>Executor`, a cui viene passato l'oggetto `Socket` che rappresenta la connessione appena stabilita. Fatto ciò, egli si rimette in attesa di altre connessioni.
- Il thread `<nomeFunzionalità>Executor` è colui che si occupa dell'interazione vera e propria con il corrispettivo thread di Social Client. Esso è dunque il responsabile della modifica della struttura dell'intera rete.

L'interazione tra questi gruppi di thread può essere schematizzata in questo modo:

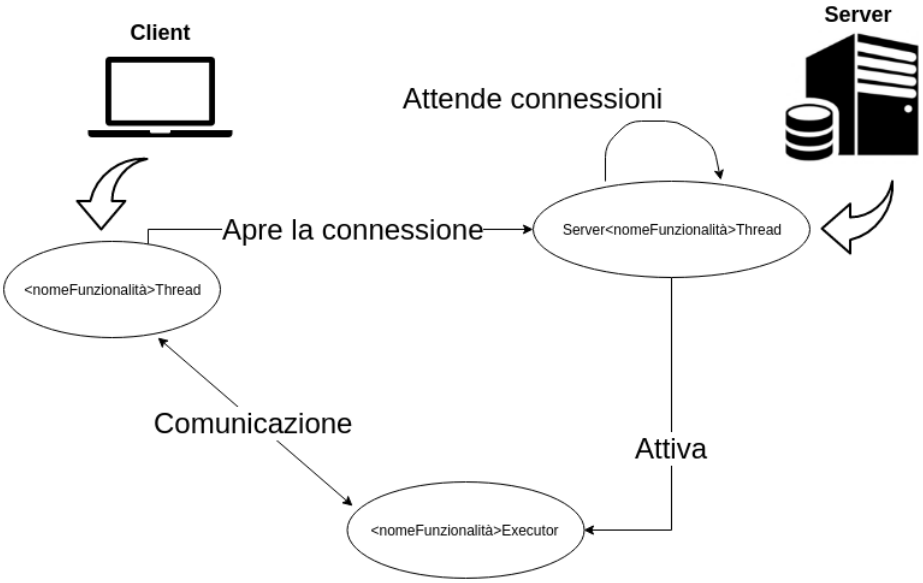

Figura 3.1: Struttura generale dei thread che implementano le funzionalità offerte all'utente.

3.2 Gestione della concorrenza

Sono molte le occasioni in cui è opportuno coordinare l'accesso dei vari thread alle strutture condivise. Ogni volta che un utente si registra, ogni volta che un utente accede ed ogni volta che un utente richiede una nuova amicizia, accetta una richiesta di amicizia, consulta la lista degli iscritti c'è bisogno di apportare delle modifiche ad alcune strutture memorizzate nel server. In particolare sia la lista degli utenti che la rete di amicizie sono soggette a continui cambiamenti:

- **Lista degli utenti:** la lista degli utenti iscritti a Simple Social è modellata tramite un `ArrayList<User>`. Un oggetto di tipo `UsersManager` racchiude tale lista e tutte le operazioni attuabili su di essa. Ogni metodo che opera sulla lista degli utenti è dichiarato `synchronized`: ciò evita che diversi thread possano accedere concorrentemente alla lista.
- **Rete di amicizie:** un'amicizia in Simple Social non è altro che un oggetto di tipo `EstabilishedFriendship`: esso comprende due variabili di istanza di tipo `User`, che rappresentano i due utenti che hanno stretto un legame di amicizia. La rete di amicizie è rappresentata quindi da un `ArrayList<EstabilishedFriendship>`: anche in questo caso c'è un oggetto che include la lista e incapsula tutte le operazioni eseguibili, il `FriendshipManager`. Di nuovo, i metodi che comprendono la modifica della lista sono dichiarati `synchronized`.
- **Amicizie in sospeso:** `FriendshipManager` si occupa anche delle richieste in attesa di conferma generate dagli utenti. La concorrenza è gestita allo stesso modo, ma in questo caso un amicizia in sospeso è modellata tramite un oggetto di tipo `PendingFriendship`, che contiene il nome del richiedente e dell'utente a cui è destinata la richiesta (variabili di tipo `String`).

3.3 I listener thread

Oltre a quelli che riguardano direttamente le implementazioni delle funzionalità offerte all'utente ci sono altri thread che è opportuno nominare: i thread listener.

- **Friendship Listener:** quando al server viene domandato di inoltrare una richiesta di amicizia ad un utente, esso deve assicurarsi il destinatario sia online affinché possa ricevere e memorizzare la richiesta. Per inviare tale richiesta all'utente, però, il server deve conoscere il socket address di quest'ultimo. Per ovviare a questo problema, dopo la fase di login viene attivato un thread che comunica al server l'indirizzo IP e la porta su cui l'utente vuole ricevere richieste di amicizia. Questo processo segue esattamente la stesso metodo adottato per l'implementazione delle funzionalità principali (descritto nella sezione 3.1). In pratica, il thread `FriendshipListenerThread` invia le informazioni di reperibilità al server e subito dopo si mette in attesa di connessioni. Dalla parte del server invece è `FriendshipListenerExecutor` che si incarica di inoltrare le richieste di amicizia.

- **Online Listener:** questi thread seguono anch'essi il già citato schema (sezione 3.1) ed il modo con cui operano è equivalente a quello dei Friendship Listener. La differenza sta nel fatto che adesso al server viene fornito un socket address per far sì che, ogni volta che la lista degli utenti online dev'essere aggiornata in seguito ad una richiesta, l'utente risponda all'appello e venga mantenuto nella lista degli utenti attivi.

3.4 I thread di keep-alive

Dopo la fase di login, Social Client attiva `KeepAliveThread`, un thread che aderisce e resta in ascolto di messaggi di keep-alive su un gruppo di multicast. `ServerKeepAliveThread` è un thread del server che, ogni 10 secondi, invia un messaggio di keep-alive su un gruppo di multicast e aspetta risposte da parte degli utenti nell'arco di 10 secondi. Gli utenti da cui non arriva nessuna risposta sono considerati inattivi e vengono dunque rimossi dalla lista degli utenti online.

4 Classi

4.1 Vista strutturale d'uso

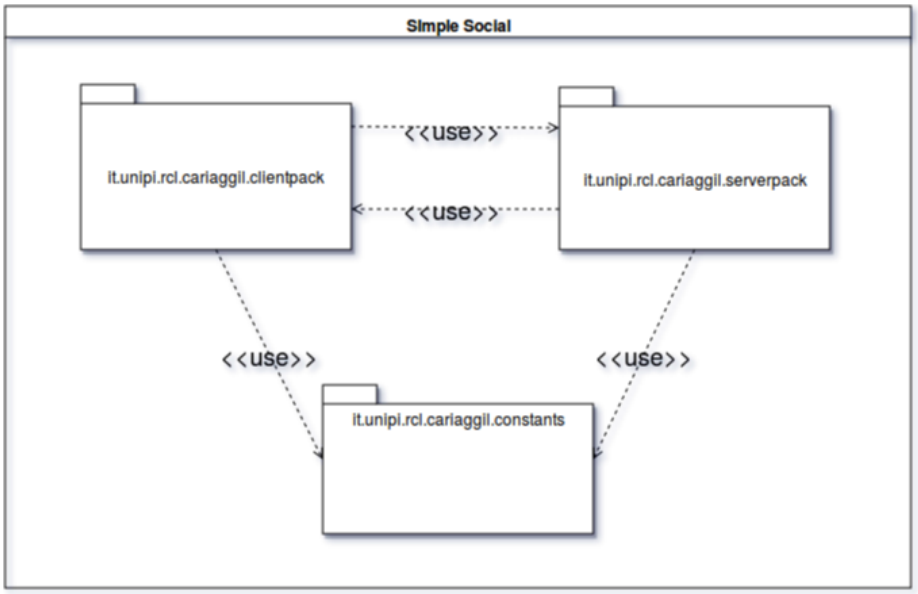

Figura 4.1: relazioni d'uso tra i package

4.2 Il package it.unipi.rcl.cariaggil.clientpack

Le classi definite in questo package riguardano tutto ciò che rappresenta le operazioni eseguibili dall'utente.

- `ClientService.java`: Classe che implementa `IClientService`, interfaccia che rappresenta le operazioni invocabili dal server tramite il meccanismo di Remote Method Invocation (RMI).

- `ClientUtils.java`: Classe che incapsula i metodi utilizzati dal Social Client per salvare su disco le richieste di amicizia non confermate e i contenuti non ancora visualizzati al momento del logout.
- `FindUsersThread.java`: Thread che si occupa di richiedere al server la lista degli utenti iscritti a Simple Social.
- `FriendListThread.java`: Thread che recupera la lista degli amici di un utente.
- `FriendshipConfirmThread.java`: Thread che viene attivato quando l'utente richiede l'operazione "Conferma nuova amicizia".
- `FriendshipListenerThread.java`: Thread che resta in ascolto di connessioni da parte del server quando quest'ultimo deve inoltrare una richiesta di amicizia.
- `FriendshipRequestThread.java`: Thread che viene attivato quando l'utente richiede l'operazione "Nuova amicizia".
- `IClientService.java`: Interfaccia che rappresenta le operazioni invocabili dal server tramite RMI (notifca dei nuovi contenuti).
- `KeepAliveThread.java`: Thread che risponde ai messaggi di keep-alive inviati dal server.
- `LoginThread.java`: Gestisce l'operazione di login da parte del client.
- `LogoutThread.java` Gestisce l'operazione di logout da parte del client.
- `OnlineListenerThread.java`: "Risponde" all'appello del server (non invia nessun messaggio), che aggiorna la lista degli utenti online ogniqualvolta gli viene richiesta un'operazione.
- `PublishContentThread.java`: Questo thread viene attivato quando viene richiesta l'operazione "Pubblica contenuto".
- `RegistrationThread.java`: Gestisce l'operazione di registrazione da parte del client.
- `SocialClient.java`: Si tratta del thread principale. Esso si occupa di interagire con l'utente fornendo un menu contenente la lista delle operazioni eseguibili. Tutti i thread sopra citati vengono attivati da esso.

4.3 Il package it.unipi.rcl.cariaggil.constants

- `Constants.java`: Classe che contiene tutti i valori costanti utilizzati in Simple Social.

4.4 Il package it.unipi.rcl.cariaggil.serverpack

- `EstabilishedFriendship.java`: Oggetto che testimonia il fatto che due utenti abbiano stretto amicizia. Contiene le informazioni dei due utenti in questione.
- `FindUsersExecutor.java, ServerFindUsersThread.java`: Coppia di thread che risponde alla richiesta "Cerca utenti" seguendo lo schema della sezione 3.1
- `FriendListExecutor.java, ServerFriendListThread.java`: Coppia di thread che risponde alla richiesta "Lista amici" seguendo lo schema della sezione 3.1
- `FriendshipConfirmExecutor.java, ServerFriendshipConfirmThread.java`: Coppia di thread che risponde alla richiesta "Conferma amicizie" seguendo lo schema della sezione 3.1
- `FriendshipListenerExecutor.java, ServerFriendshipListenerThread.java`: Coppia di thread che si occupa di memorizzare il socket address su cui un utente intende ricevere richieste di amicizia.
- `FriendshipManager.java`: Classe che racchiude la rete di amicizie stabilite in Simple Social e tutte le operazioni eseguibili su di essa.
- `FriendshipRequestExecutor.java, ServerFriendshipRequestThread.java`: Coppia di thread che risponde alla richiesta "Nuova amicizia" seguendo lo schema della sezione 3.1. Esso modifca la lista delle amicizie in sospeso.
- `IServerService.java`: Interfaccia che espone i metodi invocabili dall'utente tramite RMI (registrazione callback e registrazione interessi).
- `LoginExecutor.java`: Finalizza la fase di login richiesta dagli utenti. Modifica la lista degli utenti.
- `LogoutExecutor.java`: Finalizza la fase di logout richiesta dagli utenti. Modifica la lista degli utenti.
- `OnlineListenerExecutor.java, ServerOnlineListenerThread.java`: Coppia di thread che si occupa di memorizzare il socket address su cui un utente intende essere reperibile al momento dell'aggiornamento degli utenti online.
- `PendingFriendship.java`: Un oggetto di questo tipo rappresenta il fatto che un utente A abbia richiesto di essere amico di un utente B. Le richieste di amicizia in sospeso sono eliminate quando vengono confermate.
- `PublishContentExecutor.java, ServerPublishContentThread.java`: Coppia di thread che esegue la richiesta "Pubblica contenuto" seguendo lo schema della sezione 3.1.
- `RegistrationExecutor.java, ServerRegistrationThread.java`: Coppia di thread che esegue la richiesta "Registrazione" seguendo lo schema della sezione 3.1.

- `ServerService.java`: Classe che implementa l'interfaccia `IServerService`.
- `SocialServer.java`: Thread principale del server. Al suo avvio attiva tutti i thread che offrono i servizi messi a disposizione. Esso salva anche perodicamente lo stato della rete su disco e cerca di ricaricarlo al suo avvio.
- `User.java`: Oggetto che contiene tutte le informazioni di un utente iscritto a Simple Social.
- `UsersManager.java`: Classe che incorpora la lista di tutti gli utenti iscritti al social network e tutti metodi per operare su di essa. La concorrenza è gestita tramite la dichiarazione di metodi `synchronized`.
- `Utils.java`: Classe di utilità che contiene metodi per salvare su disco e ripristinare lo stato dell'intera rete.