Progetto Programmazione Concorrente e Distribuita

Nicola Magnabosco

31 dicembre 2013

1 Descrizione Progetto

Il progetto realizzato permette lo scambio e la condivisione di risorse, simulando un sistema P2P, dove lo scambio effettivo avviene tra i client connessi al sistema.

2 Progettazione

Il sistema è formato da: *server* e *client*. Ogni client è connesso ad un server e i server sono tutti collegati da loro. Questo permette a un client di ottenere una risorsa che non possiede, che può essere posseduta da un client connesso allo stesso server o un client connesso ad un altro server del sistema.

3 Distribuzione

Le comunicazioni remote utilizzano il modello RMI. I server del sistema rappresentano oggetti remoti che si registrano nel RMIRegistry, mentre i client si registrano nei vari server. Solamente i client possiedono risorse; i server quindi non conoscono come sia implementata una risorsa, ma si riferiscono ad essa solamente tramite il nome e il numero delle parti da cui è formata.

3.1 Server

L'interfaccia `Server` estende Remote e rende disponibili vari metodi per la comunicazione con i client.

L'oggetto remoto `ServerImpl` implementa l'interfaccia Server e rappresenta i Server del sistema.

Ogni server, al momento della creazione e connessione, si registra nell'**RmiRegistry**; successivamente, controlla i server già connessi al sistema (attraverso il metodo `refreshList()` e li aggiunge alla propria lista di server attivi.

Al momento della disconnessione, il server effettua l'**unbind** dall'RMIRegistry e comunica agli altri server la sua disconnessione.

3.1.1 Concorrenza

- Per ogni richiesta di **connessione** da parte dei client, il server avvia un **thread** `AddClient`, il quale aggiunge il client alla lista di client attivi.
- Per ogni richiesta di disconnessione il server procede nel seguente modo:

  - avvia un thread `ControllaRegistroRisorse` che cancella dal **Registro Risorse** eventuali risorse possedute solo dal client che ha richiesto la disconnessione, così da mantenere aggiornata la lista delle risorse effettivamente disponibili;
  - avvia un thread `DisconnectClient` che procede alla disconnessione del client e alla sua rimozione dal registro dei client connessi.

- Ogni qualvolta un client aggiunge una risorsa, questo provvede ad informare il server che una nuova risorsa è disponibile. In particolare si avvia un thread `AggiungiRisorsaRegistro` che aggiunge la risorsa qualora il server non la possedesse già;

- Per ogni richiesta di download, il server offre una lista di client che possiedono la risorsa:
  - il client che richiede la risorsa invoca il metodo `clientConRisorsa` sul server al quale è connesso;
  - il server avvia un thread `GetClientWithResource` che popola la lista di client che possiedono la risorsa. Il server ricerca prima nella propria lista risorse e successivamente anche negli altri server del sistema, ritornando tutti i client con la risorsa connessi al sistema.

3.1.2 Classe Registro

Questa classe rappresenta il registro delle risorse disponibili possedute dai client a lui connessi. Per ogni risorsa tiene conto anche di quanti client la posseggono.

3.1.3 ClientGUI

È l'interfaccia grafica del server. Mostra quali sono i server connessi al sistema e quali sono i client a lui connessi.

3.2 Client

L'interfaccia `Client` estende Remote e rende disponibili vari metodi per la comunicazione tra i client e i server del sistema. `ClientImpl` implementa l'interfaccia Client definiendo l'oggetto client del sistema.

Ogni client ha una lista di risorse possedute e può scaricarne altre ricercandole attraverso il server a cui è connesso.

3.2.1 Concorrenza

Un client può effettuare un download alla volta. Può però scaricare concorrentemente più parti di risorsa da client diversi. Ogni volta che si ricerca una risorsa, una volta ottenuta la lista di client che la possiedono si procede al download. Il numero di download paralleli dipende dal numero di server che possiedono la risorsa, dalla capacità di download del client e dal numero di parti della risorsa.

Download il metodo `download` calcola quanti download paralleli sono possibili avviare, calcolando il minimo tra (*client disponibili con risorsa, capacità, numero parti risorsa*). Ogni download parallelo è rappresentato da un thread `DownloadPart`. Il metodo download crea quindi un thread per ogni parte, prestando attenzione ai seguenti punti:

- non posso scaricare concorrentemente più parti da un client. Quando un client è occupato nell'upload di una parte, viene inserito in una lista `ClientAttivi` e finchè rimarrà in quella lista, non potrà essere utilizzato per scaricare altre parti.
- se ho ancora parti da scaricare, ma non posso lanciare altri thread download paralleli, il metodo `download` si mette in wait() su un oggetto `Object download`.
- quando un thread finisce di scaricare una parte, libera il client occupato per l'upload e risveglia il metodo `download` che era in attesa di avviare il thraed di download
- ll download viene simulato con un tempo di attesa **configurabile** nel corpo del costruttore di `ServerImpl`(di default = 5000)
- lo scaricamento di una parte viene **simulato** attraverso vettore `Vector<Boolean>`, inizializzato a false e di dimension pari al numero di parti della risorsa. Al thread download della risorsa i-esima, viene passata la cella i-esima che verra messa a `true` per simulare il download della parte
- una volta completato il download delle parti (tutti i thread download sono terminati), il metodo `download` controlla che esse siano state correttamente scaricate, scorrendo il vettore `Vector<Boolean>`, controllando che tutte le celle siano state messe a true
- nel caso in cui, durante il download, tutti i client con la risorsa cercata si siano disconnessi, il download della parte terminerà, informando l'utente del mancato scaricamento

4 Risorsa

L'interfaccia `Risorsa`, estende Remote viene implementata da `RisorsaImpl` che rappresenta l'oggetto Risorsa che viene scambiato tra i vari client. Ogni risorsa è formata dal `nome` e dal `numero delle parti` da cui è formata.