

# RELAZIONE PROGETTO PCD

A.A. 13/14 - ENRICO ROTUNDO, 1008052

Settembre 2014

Il presente documento rappresenta la relazione del progetto per l'esame "Programazione Concorrente & Distribuita". Di seguito verranno illustrate e motivate le scelte progettuali effettuate, in particolare riguardo all'architettura e alla gestione della concorrenza e distribuzione.

## 1 ARCHITETTURA

L'architettura implementata aderisce al pattern architetturale MVC<sup>1</sup>, si compone dei seguenti packages:

- **controller:** contiene le interfacce e le classi dei controller di Client e Server con relative classi di "servizio" come i thread demoni.
- **model:** rappresenta lo stato dell'applicazione. Le classi del model rendono disponibili dei metodi che generano i dati da visualizzare nell'interfaccia tramite oggetti di tipo `AbstractListModel`. Ogni modifica allo stato dei dati notifica l'accaduto alla view.
- **view:** contiene le classi dell'interfaccia grafica di Client e Server. Riceve le notifiche di modifica dal model e aggiorna l'interfaccia.
- **starter:** sono le classi che si occupano di creare gli oggetti Client e Server.

Tale architettura si ben presta allo sviluppo di applicazioni con interfaccia grafica, separa nettamente le responsabilità delle classi consentendo una rapida comprensione del funzionamento e dell'architettura.

### 1.1 Classi

#### 1.1.1 *controller.client*

L'interfaccia remota è **controller.client.ClientInterface**. Implementazione dell'interfaccia è invece **controller.client.Client**, essa rappresenta un oggetto Client con i riferimenti a interfaccia grafica e modello dati. Il costruttore riceve dal **ClientStarter** il nome del server al quale connettersi che verrà ricercato tra quelli disponibili in **rmi://localhost/Server/**. Il client possiede un **controller.client.ConnectionChecker** che testa lo stato della connessione al server e aggiorna l'interfaccia in caso di disconnessione. Il client possiede un **controller.client.DownloadScheduler** che viene avviato ogni volta che viene richiesto il download di una risorsa. Questa classe mantiene una lista dei client possessori della risorsa da scaricare, si occupa di lanciare lo scaricamento delle singole parti con il

<sup>1</sup> <http://en.wikipedia.org/wiki/Model-view-controller>

livello di concorrenza richiesto dalla specifica. Il download di una singola parte viene affidato ad un **controller.client.PartDownloader**.

#### 1.1.2 *controller.server*

L'interfaccia remota del server è **controller.server.ServerInterface**. L'implementazione dell'interfaccia è **controller.server.Server**, essa rappresenta un oggetto Server con riferimenti a interfaccia grafica e modello dati. Il costruttore riceve dal **Server-Starter** il nome che il server deve avere. Il server viene registrato all'indirizzo **rmi://localhost/Server/** su cui rende pubblicamente disponibili sia i metodi remoti per la connessione/disonessione di client, sia i metodi remoti *getResourceOwners(String)* e *getLocalResourceOwners(String)*. Il primo viene invocato in prima battuta dal client C richiedente la risorsa, il server di appartenenza S chiama quindi *getLocalResourceOwners(String)* per ottenere le risorse dei client a lui registrati e, per ogni server registrato nella rete S', chiama *getResourceOwners(String)* che risponderà con le risorse dei client registrati presso S'. In questo modo viene generato al bisogno l'elenco dei client possessori della risorsa richiesta. Questa soluzione è ottimale in quanto efficiente (evita il *polling*<sup>2</sup> che richiederebbe ripetutamente lock su risorse remote del client) e consistente (assicurata dai client con lock sulle risorse disponibili, dai server con lock sulla lista costantemente aggiornata dei client connessi (**controller.server.ClientChecker**)). Ogni oggetto Server gode di una lista di server contattabili mantenuta da **controller.server.ServerChecker**.

#### 1.1.3 *model.client*

Il Client avrà un model di tipo **model.client.ClientResources** che rende disponibili metodi per interfacciarsi alle risorse disponibili ed in download. La classe **model.share.Resource** rappresenta la singola risorsa che possiedono i client.

#### 1.1.4 *model.server*

Il Server avrà un oggetto di tipo **model.server.ConnectedClients** e uno di tipo **model.server.ConnectedServers** che rappresentano, per quel Server, lo stato dei client e server connessi.

#### 1.1.5 *view*

La classe **view.AbstractBasicFrame** implementa un area di log sincronizzata. I metodi astratti sono lasciati alle diverse implementazioni tra client (**view.ClientFrame**) e server (**view.ServerFrame**) con pannelli e componenti diverse.

## 2 CONCORRENZA

### 2.1 Scaricamento risorsa

Il client scarica al più una risorsa alla volta grazie al controllo effettuato in *performSearch()* di Client controller. La fornitura concorrente di parti di risorse possedute è consistente grazie ai lock sui dati a livello del server model. Il download di una parte è simulato dalla una *sleep* nel metodo client remoto *download(string)* per un tempo pari a **Client.UPLOAD\_TIME**. Il client richiedente mantiene una lista di client dai quali sta scaricando in modo da poter controllare di scaricare contemporaneamente al più da un client.

<sup>2</sup> [http://en.wikipedia.org/wiki/Polling\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Polling_(computer_science))

## 2.2 Comunicazioni

Le richieste di connessione, disconnessione e ricerca possessori risorse sono tutte protette da sincronizzazione, le prime due sull'oggetto *clientsMonitor* mentre l'ultima su *serversMonitor* a livello di controller, sono ulteriormente sincronizzate a livello model sui rispettivi Vector.

## 3 DISTRIBUZIONE

Le parti di risorsa sono scambiate esclusivamente tra client, i server non detengono risorse e non fanno da tramite durante gli scambi; questo è facilmente verificabile chiudendo i server durante uno scambio parti, si noterà che lo scambio proseguirà indisturbato. Le comunicazioni server-client e server-server riguardano solamente riferimenti remoti a oggetti client e/o server, in base alle necessità. I riferimenti remoti ai client non vengono inseriti in registri pubblici, questo accade solamente per i server. La robustezza viene garantita dalla gestione dei fallimenti delle entità del sistema che si presentano sotto forma di eccezioni, tali occorrenze vengono stampate nell'area di log della gui. I messaggi degli eventi lanciati dai thread demoni vengono generalmente inviati a *System.out*.

## 4 PROCEDURA PER L'UTILIZZO DELL'APPLICATIVO

Di seguito vengono elencati i passi per poter avviare l'intero applicativo ed un esempio d'uso. Per avviare il programma eseguire sempre dalla root la seguente sequenza di comandi da terminale:

1. **make clean**
2. **make**
3. **make stop**
4. **make start**

A questo punto l'utente visualizzerà le interfacce dei client e dei server. I client, all'avvio si connettono automaticamente al server prescelto, se questo non dovesse essere disponibile verrà visualizzato un messaggio; sarà possibile riconnettersi tramite l'apposito bottone. Per scaricare una risorsa è necessario e sufficiente compilare il campo testuale sito nell'interfaccia del client e cliccare con il mouse sul bottone di ricerca. Il campo testuale è di tipo *JFormattedTextField* ed è configurato per accettare stringhe di lunghezza 3 validate secondo la maschera "U #" di tipo *MaskFormatter*<sup>3</sup>, U equivale a [A-Z], # equivale a [0-9]. Il controller client controlla che il client sia connesso, che non ci sia una risorsa in download, che la risorsa non sia in possesso al client, che vi sia almeno un solo possessore della risorsa connesso al network. Se tutti questi controlli restituiscono esito positivo viene avviato il thread che gestisce il download della risorsa, è possibile monitorare l'andamento del download tramite la lista sulla destra. Una volta completato il download la risorsa viene aggiunta tra quelle disponibili. Se durante un download tutti i client possessori dovessero risultare irraggiungibili, dopo un paio di secondi verrà annullato il download.

<sup>3</sup> <http://docs.oracle.com/javase/7/docs/api/javax/swing/text/MaskFormatter.html>