Relazione Progetto PCD - Parte 3

Giacomo Manzoli 1049820

25 gennaio 2015

Indice

1 Introduzione

Questa parte del progetto consiste nel trasformare la soluzione concorrente ideata nella seconda parte in una soluzione distribuita su un'architettura di tipo client-server. In particolare viene richiesto che:

- La soluzione utilizzi Java RMI;
- Il client si occupi della logica di input/output;
- Il server si occupi della logica di risoluzione;
- Il client non pubblichi oggetti nel suo registro RMI;
- La soluzione sia robusta nei confronti degli errori di comunicazione.

2 Logica client-server

La comunicazione tra il client e il server viene gestita nel seguente modo:

1. Il client carica i pezzi del puzzle all'interno di una `PuzzleElementList` e invoca il metodo `makePuzzle(PuzzleElementList)` sull'oggetto reso pubblico dal server;
2. Il server riceve la `PuzzleElementList` e la usa per costruire e risolvere il `Puzzle`, una volta fatto ciò, ritorna al client un oggetto `Puzzle` contenente il puzzle risolto;
3. Una volta ricevuto l'oggetto il client riprende l'esecuzione e stampa sul file la soluzione.

Per rendere possibile questa soluzione è stato necessario rendere serializzabili tutte le classi che rappresentano i vari componenti del puzzle. Per maggiori informazioni si rimanda alla sezione 4.2.

3 Robustezza della soluzione

Viene richiesto che la soluzione sia robusta e in grado di gestire i vari errori che possono sorgere dalla distribuzione di un programma.
Le seguenti sottosezioni contengono la descrizione di come vengono gestite tutte le situazioni critiche individuate.

3.1 Il client tenta di connettersi ad un server spento/inesistente

In questo caso viene stampato sulla console del client il messaggio:

Errore di comunicazione con il server.

E sul file di output viene prodotta la stampa di un puzzle senza soluzione[1];

[1] Un puzzle senza soluzione, quando viene stampato, provoca la stampa di: "Soluzione non trovata"

3.2 Il client tenta di recuperare un remote object che non è nel registro del server

In questo caso viene stampato sulla console del client il messaggio:

Indirizzo del server errato.

E sul file di output viene prodotta la stampa di un puzzle senza soluzione;

3.3 Il client viene terminato mentre il server risolve il puzzle

In questo caso, l'esecuzione del server continua regolarmente senza produrre problemi. Il puzzle risolto non sarà ritornato al client e l'istanza presente sul server verrà distrutta dal `GarbageCollector`;

3.4 Il server viene terminato mentre un client aspetta la risposta

Questa eccezione viene gestita dal client, come se stesse cercando di connettersi ad un server spento;

3.5 Il server non riesce a risolvere il puzzle

Se il server non riesce a risolvere il puzzle, viene inviata al client un'eccezione di tipo `BadPuzzleException`, in questo modo il client può riprendere la sua esecuzione e stampare sulla console un messaggio d'errore. Anche in questo caso il file prodotto conterrà la stampa di un puzzle senza soluzione. Maggiori informazioni riguardo questo caso si possono trovare nella sezione 4.3.

4 Cambiamenti rispetto alla seconda parte

I cambiamenti necessari per rendere distribuita la versione precedente sono stati minimali. Inoltre, sono stati aggiunti nuovi controlli allo scopo di rendere la soluzione più robusta.

4.1 PuzzleMaker / PuzzleMakerImp

Il server espone, sul registro RMI, un oggetto di tipo `PuzzleMakerImp`, il client può usare questo oggetto per invocare sul server il metodo `makePuzzle(PuzzleElementList)` che, data una lista di pezzi di un puzzle, costruisce un oggetto di tipo `Puzzle` e lo risolve.
Per poter invocare il metodo, il client utilizza l'interfaccia `PuzzleMaker`.
Al fine di garantire il corretto funzionamento del programma, l'interfaccia `PuzzleMaker` estende l'interfaccia `java.rmi.Remote` e la classe `PuzzleMakerImp` estende la classe `java.rmi.UnicastRemoteObject`.

4.2 Serializable

Per rendere possibile il passaggio di parametri tramite serializzazione le classi che rappresentano gli elementi del puzzle ora implementano anche l'interfaccia `Serializable`.
In questo modo, quando avviene il passaggio di parametri tra il client e il server (e viceversa), viene fatta la copia degli oggetti, evitando così la creazione di ulteriori riferimenti remoti che complicherebbero inutilmente la soluzione.
Le classi che implementano `Serializable` sono:

- `PuzzleItem`;
- `PuzzleRow`;
- `PuzzleBlock`;
- `Puzzle`;
- `PuzzleElementList`;
- `BadPuzzleException` (tratta più avanti).

4.3 Gestione puzzle incompleti

Per evitare che il client rimanga in attesa qualora il puzzle che ha inviato al server non ha soluzione è stato necessario introdurre dei meccanismi di controllo che, controllando lo stato dei `SolverThread`, stabiliscano se il puzzle corrente ha soluzione.
In particolare, il thread del server sul quale viene eseguito il metodo remoto, controlla ogni 2500[2] millisecondi che i `SolverThread` siano attivi e nel caso che tutti i solver siano andati in `wait`, interrompe l'algoritmo risolutivo.
Infatti, un `SolverThread`, si mette in `wait` solo se non riesce ad avanzare nella risoluzione del puzzle perché gli manca un pezzo, di conseguenza se tutti i `SolverThread` sono in `wait` vuol dire che manca almeno un pezzo.
Questa logica di funzionamento è stata implementata dentro il costruttore della classe `Puzzle`, il quale avvia in automatico il processo di risoluzione.

4.3.1 BadPuzzleException

Quando il server non riesce a risolvere il puzzle perché mancano dei pezzi, invia al client una `BadPuzzleException`, in questo modo, quando il client riceve un'eccezione di questo tipo può terminare fornendo il messaggio d'errore corretto.

4.4 Modifiche alla classe SolverThread

Per rendere possibile la gestione di puzzle con un pezzo mancante è stato necessario aggiungere del codice al metodo `run()` in modo che quando un thread di questo tipo non riesce a trovare due pezzi da attaccare vada in sospensione.
Questa sospensione è corretta perché, per come è strutturato il programma, un `SolverThread`

[2]Si è cercato di prendere un tempo non troppo basso per evitare di fare il test troppo frequentemente quando si sta risolvendo un puzzle complesso.

non riesce a trovare due pezzi che si combinano solamente quando mancano dei pezzi, perché la risoluzione del puzzle viene avviata quando si hanno a disposizione tutti i pezzi possibili[3].

4.5 Caricamento dei pezzi

La classe `LoaderThread` è stata rimossa, in quanto non ha più senso tenere il caricamento del puzzle parallelo alla risoluzione.
La logica di caricamento dei pezzi del puzzle è stata spostata dentro `PuzzleElementList`.

[3]Nella seconda parte del progetto questa affermazione non era vera, in quanto la risoluzione del puzzle avveniva concorrentemente al caricamento dei pezzi