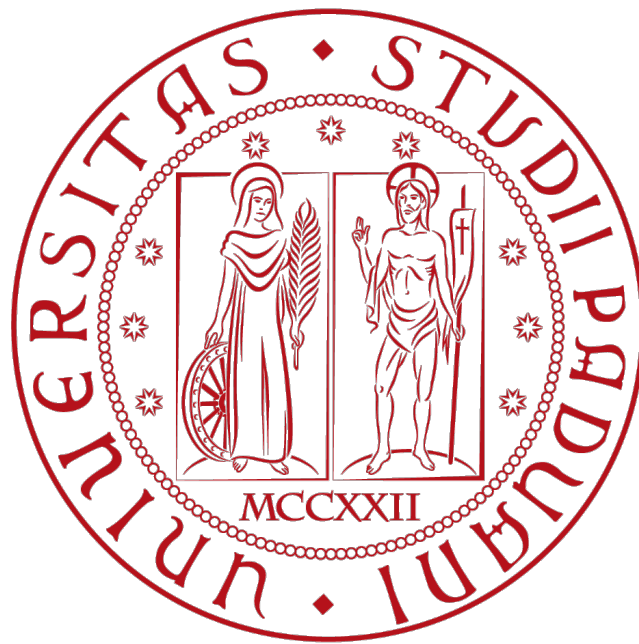


Programmazione Concorrente e Distribuita

Terza Parte

Suierica Bogdan Ionut 1008089

June 11, 2015



1 Cambiamenti

In questa sezione verranno descritti i cambiamenti apportati al codice per permettere al programma di essere distribuito tra un server e un client. In particolare la classe *Puzzle* contiene anche una copia dell'insieme non ordinato di *Tile* che rappresentano il puzzle. La classe *PuzzleToSolve*, in seguito al cambiamento della classe *Puzzle*, implementa l'interfaccia *SolverAlgorithm* con il suo metodo *solve* e si preoccupa solamente della risoluzione del puzzle. Il metodo *solve* dell'interfaccia *SolverAlgorithm* è stato modificato e adesso accetta un parametro di tipo *Puzzle* e ritorna un *Puzzle*.

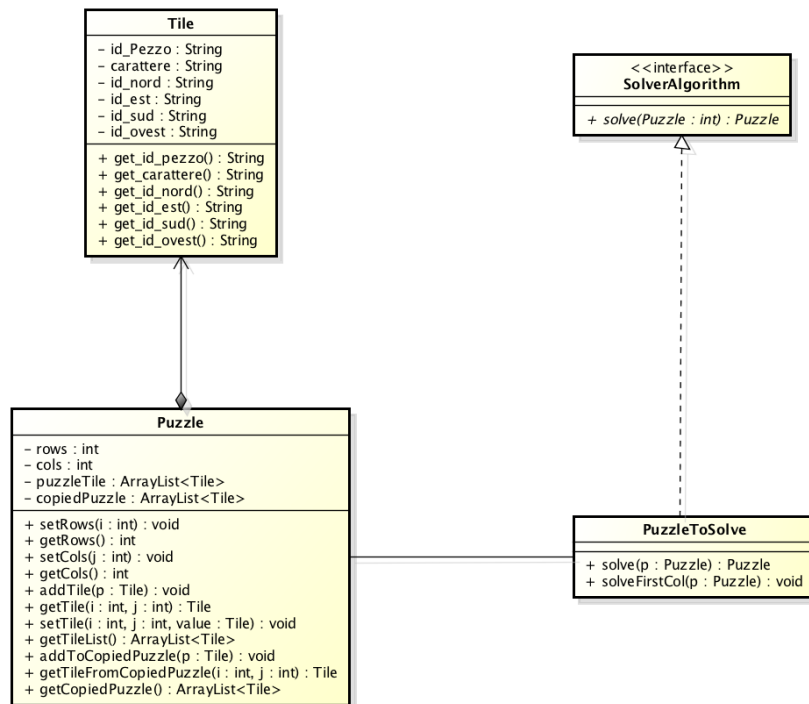


Figure 1:

Come richiesto dalla specifica il client si occupa della gestione dei file di input e di output.

Il programma è stato suddiviso in 3 parti. Sono stati quindi individuati 3 package:

- **Server:** il package contiene le classi:
 - **PuzzleToSolve:** la classe definisce l'oggetto remoto implementando l'interfaccia *SolverAlgorithm* con il suo metodo *solve* (che si occupa di risolvere il puzzle) ed estendendo la classe *UnicastRemo-*

teObject. La classe contiene un costruttore con corpo vuoto che può sollevare un'eccezione di tipo *RemoteException*;

- **PuzzleSolverServer:** è la classe che definisce il metodo *main*. La classe crea l'istanza dell'oggetto remoto *PuzzleToSolve* e gli associa un nome. Attraverso il metodo *rebind* lo registra nell'RMI registry.

- **Client:** il package contiene le classi:

- **IOReader:** è la classe che si occupa della gestione dei file in input;
- **IOWriter:** è la classe che si occupa della gestione dei file in output;
- **PuzzleSolverClient:** è la classe che contiene il metodo *main*. Il programma client interroga il registro RMI utilizzando il metodo statico *lookup* della classe *Naming*, il quale restituisce un riferimento di tipo *Remote* all'oggetto cercato. Infine invoca il metodo remoto *solve* facendo un downcast al tipo *SolverAlgorithm*.

- **Shared:** il package contiene le classi in comune del server e del client. Le classi *Tile* e *Puzzle* estendono l'interfaccia *Serializable* in quanto gli oggetti vengono passati come parametri e come valori restituiti dal metodo *solve*:

- **Tile:** rappresenta il singolo pezzo del puzzle;
- **Puzzle:** rappresenta il puzzle;
- **SolverAlgorithm:** rappresenta l'interfaccia remota. L'interfaccia estende l'interfaccia *Remote*. È presente il metodo *solve*, che ha il compito di risolvere il puzzle. Il metodo può sollevare un'eccezione di tipo *RemoteException*.

2 Logica di comunicazione client-server

Per la comunicazione tra un server e un client il programma adotta la libreria RMI il cui scopo è di rendere trasparente al programmatore quasi tutti i dettagli della comunicazione su rete. Essa consente infatti di invocare un metodo di un oggetto remoto.

Per implementare il meccanismo RMI sono richiesti i seguenti punti:

- Definizione dell'interfaccia remota *SolverAlgorithm*;
- Definizione dell'oggetto remoto *PuzzleToSolve*;
- Creazione del server con il compito di creare un'istanza dell'oggetto remoto e registrare il riferimento associandogli una stringa identificativa. La registrazione avviene invocando il metodo statico *rebind* della classe *java.rmi.Naming*;
- Creazione del client che ha il compito di interrogare il registro RMI utilizzando il metodo statico *lookup* della classe *java.rmi.Naming*, il quale restituisce un riferimento di tipo *Remote* all'oggetto cercato.

La classe *Puzzle* e *Tile* estendono l'interfaccia *Serializable*. Viene trasmessa quindi una copia.

3 Compilazione

Dalla root principale è possibile avviare il comando per la compilazione di tutti i file attraverso l'istruzione **make**. Se si vuole lanciare il programma si devono utilizzare i seguenti script bash:

- *sh puzzlesolverserver.sh [nome_del_server]* : questo script riceve in input il nome del server. Con il lancio verrà eseguito il comando per avviare il registro rmi e il main del server;
- *sh puzzlesolverclient.sh [input_file][output_file][nome_del_server]* : questo script riceve in input 3 parametri, file in input, file in output e il nome del server. Con il lancio verrà eseguito il main del client.

Prima di procedere con il lancio degli script bash si deve aver compilato.