Peer-Review 2: UML e NETWORKING

Domenico Parlato, Andrea Santaniello, Juan Marco Patagoc, Eliseo Paoletti

Gruppo CG53

Il presente documento descrive l'implementazione del networking utilizzando Java RMI.

## **Architettura del Server RMI**

### Inizializzazione e Registrazione:

- Il **`ServerRMI`** viene avviato e si registra nel registro RMI disponibile nella “*Def\_Port*=”1234”.

- Il server crea un'istanza di **`GameController`**, che è esportata per le chiamate remote.

**// Specific methods for game operations**

void turnCard(String p, int handPosition, boolean side) throws RemoteException;

int chooseCellPosition(String p, int positionArray) throws RemoteException;

int chooseCard(String p, int positionCard) throws RemoteException;

void playCard(String p, int positionCardToPlay, int positionArray) throws CardPlacementException, RemoteException;

void drawCard(String p, DRAWTYPE drawType) throws RemoteException;

**// Additional methods for managing client chat and connection**

void registerClient(ClientCallback client) throws RemoteException;

void sendMessage(String message) throws RemoteException;

void performGameAction(String action) throws RemoteException;

* **PerformGameAction** raccoglie tutti i metodi del turno del giocatore: la scelta della carta nel mazzo, la scelta del lato, la posizione dove giocarlo e il pescaggio.

### Gestione dei Client:

- Il server accetta le registrazioni dei client tramite il metodo **“registerClient()”**. Ogni client connesso è aggiunto a una lista di client attivi

- Gestisce le disconnessioni in modo da rimuovere i client inattivi dalla lista

- Viene implementato un ping (metodo del GameControllerInterface) che permette l’individuazione di improvvise disconnessioni (ogni x secondi viene inviato un segnale di controllo da client e server

### Elaborazione delle Richieste:

- Il server riceve input da parte dei client e aggiorna lo stato del gioco di conseguenza. Viene utilizzata la sincronizzazione per mantenere coerenza tra client e server in fase di propagazione delle modifiche apportate al gioco.

## **Architettura del Client RMI**

### Connessione al Server:

- Il client utilizza **“LocateRegistry.getRegistry(host)”** per ottenere il riferimento al registro RMI del server.

- Recupera lo stub del **“GameController”** dal registro e invoca metodi remoti.

### Interfaccia Utente e Interazione:

- Gli utenti interagiscono con l'applicazione tramite una GUI /TUI, scelta all’avvio del gioco

- L’interfaccia viene aggiornata in seguito all’esecuzione di metodi da parte del server, propagati in seguito ai client tramite protocollo RMI

### F.A.: Comunicazione tra Giocatori - Funzionalità di Chat:

- Viene Implementata una chat tra utenti, dove il metodo **sendMessage()** del server propaga i messaggi a tutti i client registrati.

-La chat prevede due classi: Chat e **Messages**

Immagine che contiene testo, Software multimediale, Software per la grafica, schermata

Descrizione generata automaticamente

### Gestione delle Eccezioni

- Il client e il server implementano una robusta gestione delle eccezioni per affrontare errori di rete, interruzioni e dati errati.

- Vengono implementati diversi blocchi di try catch, che incapsulano le operazioni di rete e lanciano le dovute Exception in caso di errori. Un esempio è il metodo del client **“connect()”**, che prova ad instaurare una connessione tra client e server, dove ciò fallisce (Excepion e) si aspetta un numero determinato di secondi **(SecondsBetweenRetry)** per poi riprovare. Viene impostato un numero massimo di retry, al termine del quale il programma termina con un messaggio d’errore.