Documentazione protocollo di rete

Progetto di Ingegneria del Software GC-19

Si è deciso di implementare sia la comunicazione Socket che quella RMI.

La comunicazione avviene nel seguente modo:

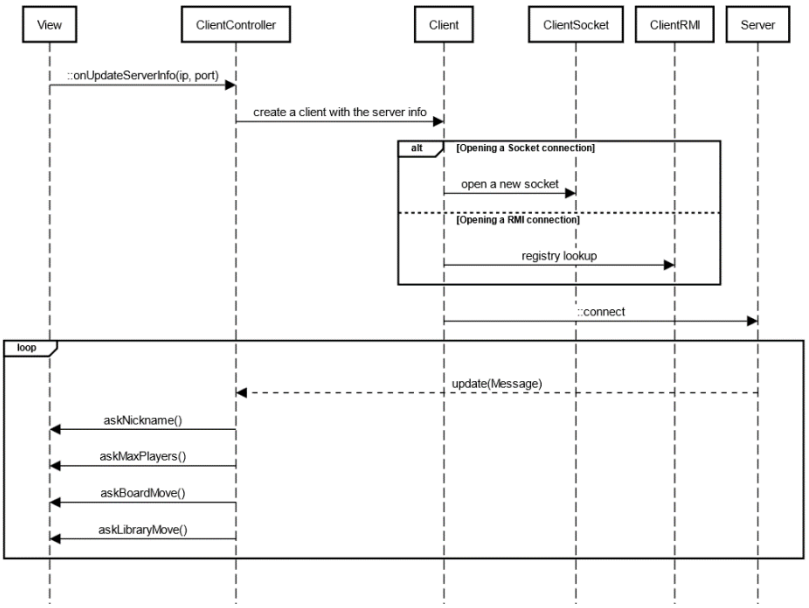

Figura 1: Connessione di un client al server

L'utente interagisce con la View (CLI o GUI), che aggiornerà il clientController sull'azione appena eseguita. Il clientController si prende carico della creazione del messaggio che verrà inviato sulla rete tramite il protocollo scelto dall'utente all'avvio dell'applicativo.

Tramite la connessione socket, il messaggio è inoltrato all'oggetto clientSocket, che infine lo invia in rete tramite l'apposito output stream. Il server riceverà il messaggio sul proprio output stream. Mediante la connessione RMI, invece, si effettua il lookup del registro sulla rete e a seguito della registrazione al registro, si potranno invocare direttamente i metodi remoti del server.

Al fine di seguire le regole di buona progettazione del software, si è deciso di nascondere il più possibile lo strato di rete sia lato client, sia lato server. Lato client, infatti, il ClientController non è a conoscenza del tipo di connessione utilizzata, visto che contiene solo un riferimento a un oggetto di tipo Client non specializzato. Lato server, si è fatto lo stesso nascondendo i vari tipi di client sotto l'oggetto ClientHandler.

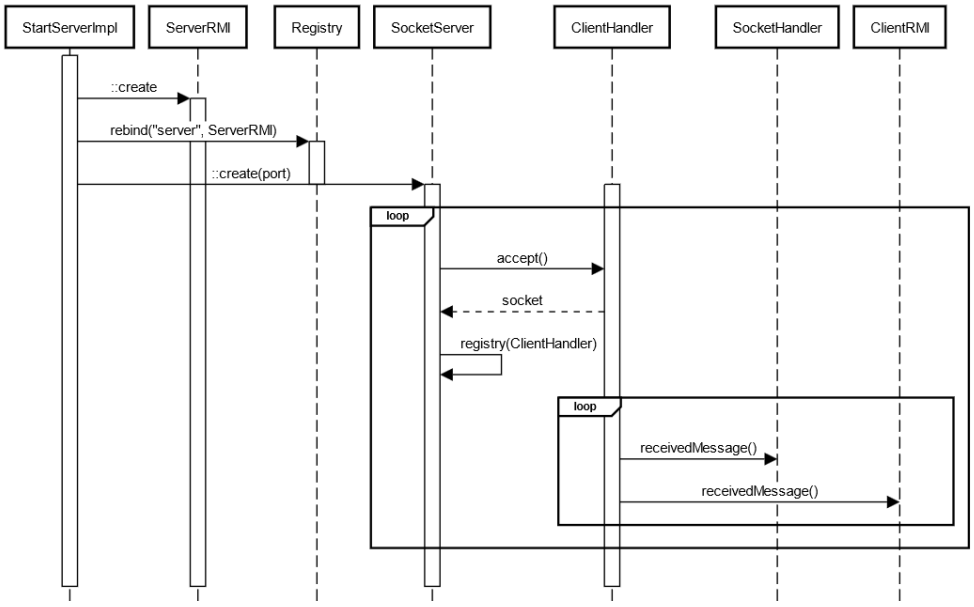

Figura 2: Connessione a basso livello

Il main del server è contenuto in StartServerImpl che crea a sua volta sia il server RMI, sia il server socket, infine crea il Controller di gioco. Entrambi i server accettano i client praticamente allo stesso modo. Il server RMI crea il registro e lo mette a disposizione dei client. Il server socket si mette in ascolto dei tentativi di connessione.

Alla connessione di client, viene creato un ClientHandler che verrà aggiunto alla lista rappresentante i client connessi. Il server poi utilizzerà i ClientHandler per gestire la connessione con i vari client che possono usare protocolli differenti.

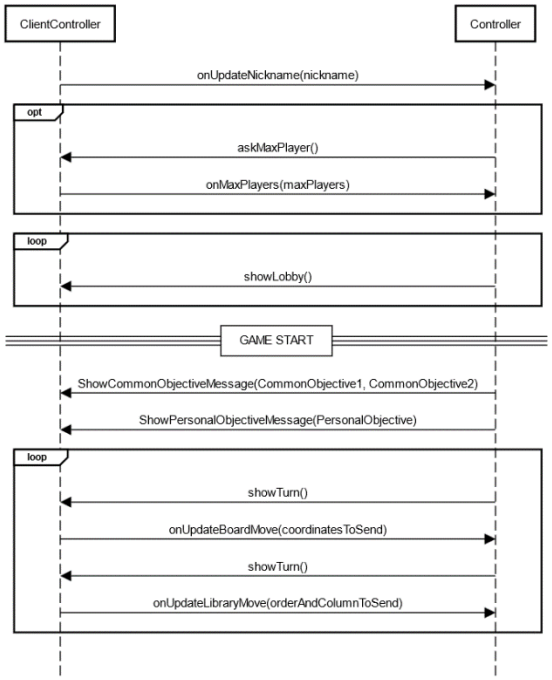

Figura 3: Comunicazione ad alto livello durante il gioco

Ad alto livello si è cercato di mantenere lo strato di rete nascosto, difatti il Controller di gioco aggiorna una vista fittizia che a sua volta inoltra i dati come messaggi alla classe StartServerImpl. Tale classe poi si occuperà del trasporto, come descritto in precedenza.

Alla base della comunicazione ci sono i messaggi, rappresentati dalla classe astratta “Message” che avrà diversi figli specializzati nel trasporto di differenti informazioni. Tra i messaggi principali si possono trovare:

- **ShowTurnMessage**: mostra il turno attuale ai giocatori e contiene la tavola di gioco, la libreria e gli oggetti in mano del giocatore nel turno, i punti degli obiettivi comuni acquisiti dal giocatore nel turno e i punti ancora disponibili dagli obiettivi comuni.
- **ShowCommonObjectivesMessage**: viene inviato ad ogni giocatore all’inizio della partita e contiene gli obiettivi comuni. Gli obiettivi verranno salvati anche lato client.
- **ShowPersonalObjective**: viene inviato ad ogni giocatore all’inizio della partita e contiene l’obiettivo personale. Anche questo obiettivo viene salvato lato client.
- **PickObjectMessage**: viene inviato dal Client quando l’utente ha inserito le coordinate degli oggetti che vuole prendere dalla tavola di gioco.
- **PutObjectInLibraryMessage**: viene inviato dal Client quando l’utente vuole inserire gli oggetti che ha in mano (ObjectsInHand) nella libreria in una colonna specifica.
- **MaxPlayersMessage**: viene inviato dal Client quando si vuole inserire il numero di giocatori per la partita.
- **UserInfoForLoginMessage**: viene inviato dal Client quando quest’ultimo vuole aggiungersi a una partita.

I messaggi “Show” attiveranno, tramite il ClientController, la view per mostrare il turno o gli obiettivi comuni e personali. In questo modo si possono facilmente aggiungere tipi di messaggi nel caso in cui si dovessero fare azioni aggiuntive.