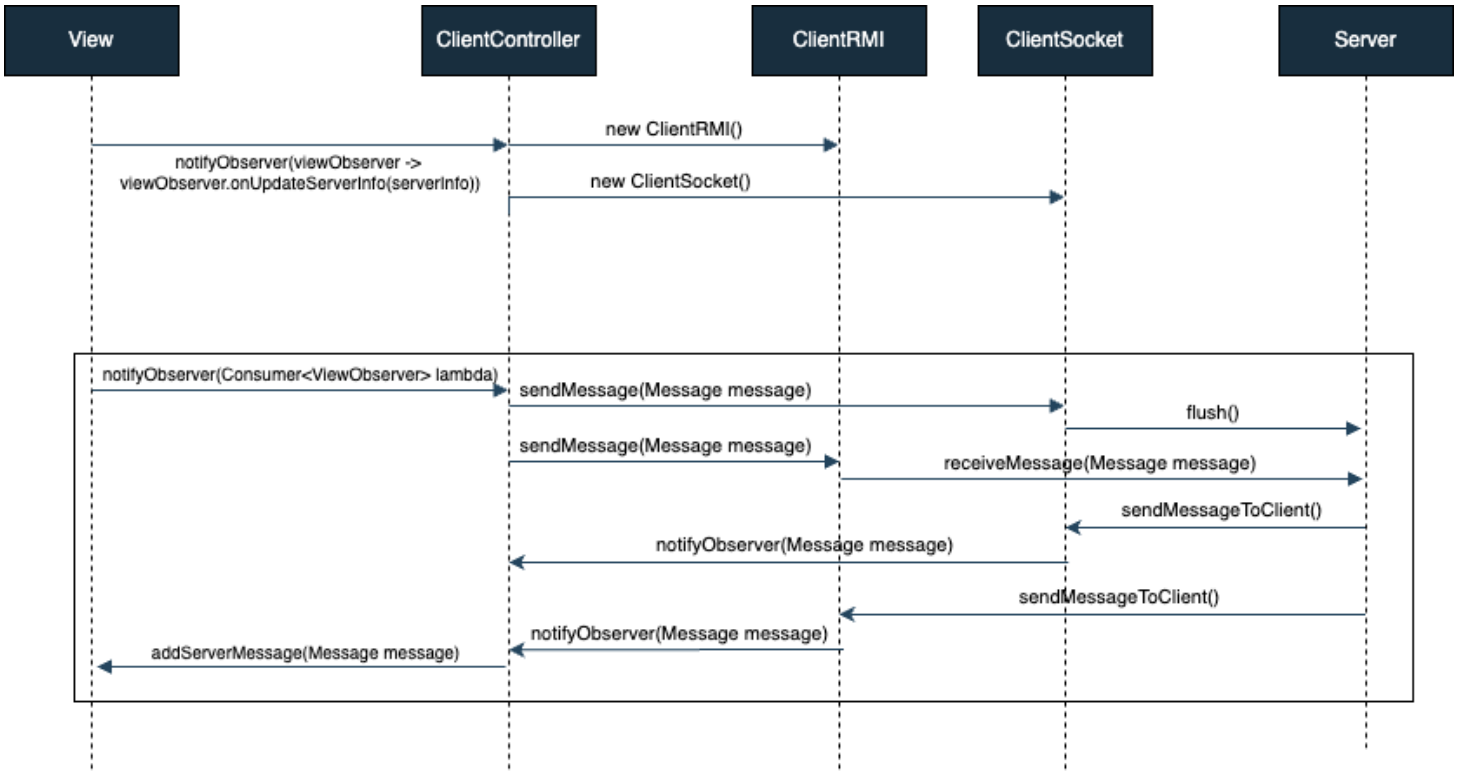
Documentazione Protocollo di Rete

Si è deciso di implementare sia la comunicazione Socket che quella RMI.Ecco il sequence diagram che descrive la comunicazione:



Ricostruendo la sequenza con cui avviene la comunicazione, possiamo partire dalla classe Cli, che estende l'interfaccia ViewObservable. I messaggi da scambiare in rete vengono inviati dalla Cli(o GUI) al ClientController, che estende ViewObserver. Quindi il meccanismo è il seguente: ViewObserver "ascolta" i cambiamenti del ClientController. Cli comunica con ClientController tramite notifyObserver(Consumer<ViewObserver> lambda), che prende quindi in ingresso una funzione che richiede un ViewObserver come parametro.

A questo punto tramite onUpdateServerInfo il ClientController crea un nuovo oggetto ClientSocket o ClientRMI in base alla scelta fatta dall'utente(se connettersi tramite l'uno o l'altro) e si aggiunge come observer del clent creato(ClientController ha quindi un riferimento client all'oggetto creato, Socket o RMI).

Poi quindi ClientController invia messaggi passati da Cli o GUI(tramite notifyObserver) al ClientRMI o ClientSocket mediante il metodo sendMessage del client creato. I messaggi tornano quindi indietro mediante notifyObserver del ClientSocket o ClientRMI, di cui ClientController è observer.

Il ServerRMI comunica al ClientRMI tramite la sendMessageToClient(String clientId, Message message), che crea un ClientHandler e invia il messaggio al ClientRMI tramite handler.receivedMessage(message), che aggiungerà il messaggio alla ArrayList di messaggi sul ClientRMI.

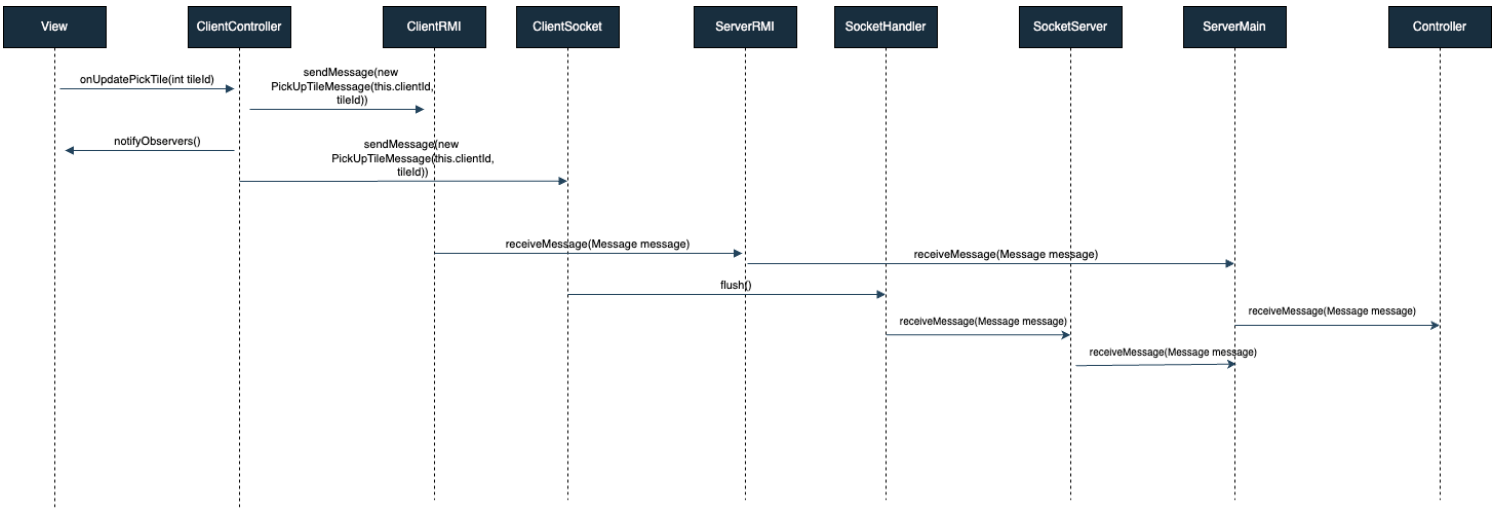
I messaggi tornano poi al ClientRMI tramite il metodo receivedMessage, che lo aggiunge alla ArrayList messages. Poi il messaggio verrà processato dal thread in esecuzione che quando "rileva

un nuovo messaggio" eseguirà notifyObserver(messages.remove(0)), che comunicherà quindi con il ClientController.

Analogamente il SocketServer tramite receiveMessage riceve i messaggi e li processa, "rispondendo" al client tramite sendMessageToClient che crea un SocketHandler e invia al client il messaggio tramite handler.receivedMessage(message).

I messaggi del server vanno poi dal ClientController alla Cli o GUI tramite addServerMessage.

esempio di comunicazione con il messaggio PickedTileMessage



Una volta che il controller riceve il messaggio chiama su di esso il metodo process. All'interno di process viene richiamato sul controller lo specifico metodo che deve eseguire(handlePickUpTileMessage), in questo modo siamo riusciti a sfruttare il polimorfismo evitando di implementare nel controller uno switch sul nome dei messaggi.

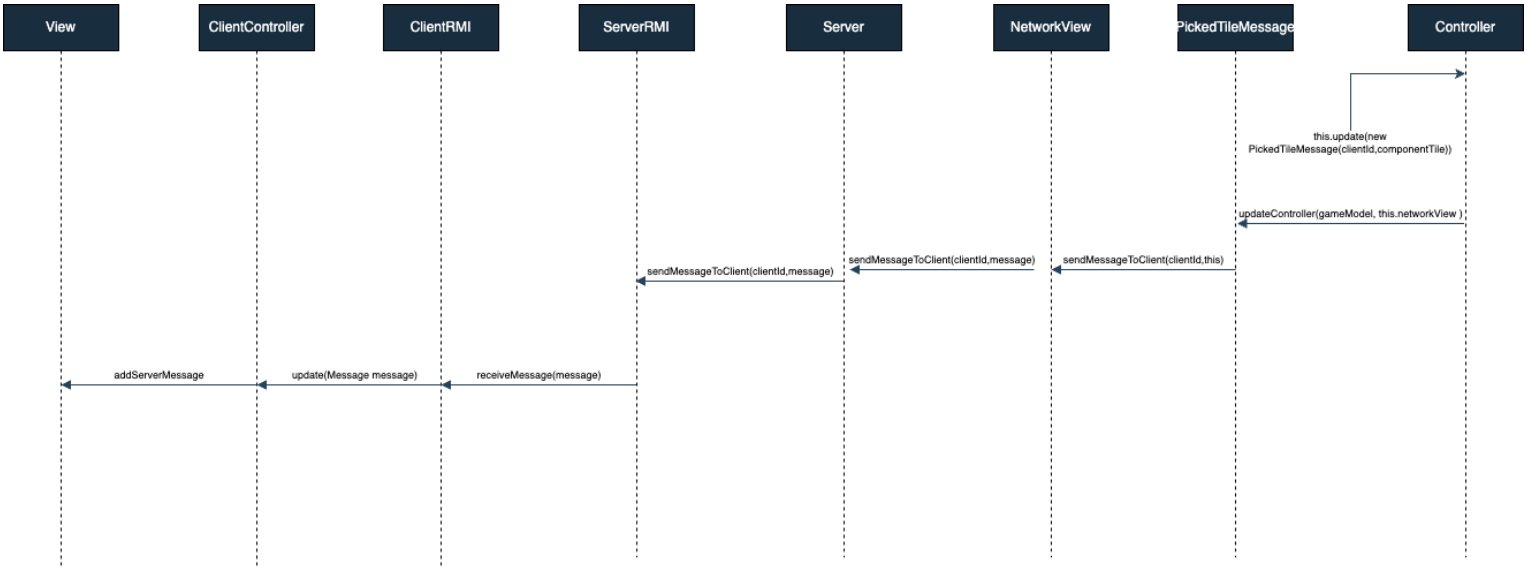
Una volta che viene chiamato il metodo addServerMessage questo viene implementato sia dalla TUI che dalla GUI inserendo il messaggio arrivato dal server in una coda di messaggi.

La coda viene poi risolta da un Thread che legge e rimuove in continuo il messaggio dalla coda, decidendo anche qui in maniera polimorfica che metodo chiamare sulla TUI/GUI.
Una volta chiamato il sendMessageToClient sul ServerMain qui viene deciso in base al clientId del sender del messaggio, a chi mandare il messaggio: se il client è connesso tramite TCP (clientId inizia con TCP_[...]) allora viene chiamato il sendMessageToClient sul SocketServer; se il client è connesso tramite RMI (clientId inizia con RMI_[...]) allora viene chiamato il sendMessageToClient sul ServerRMI.

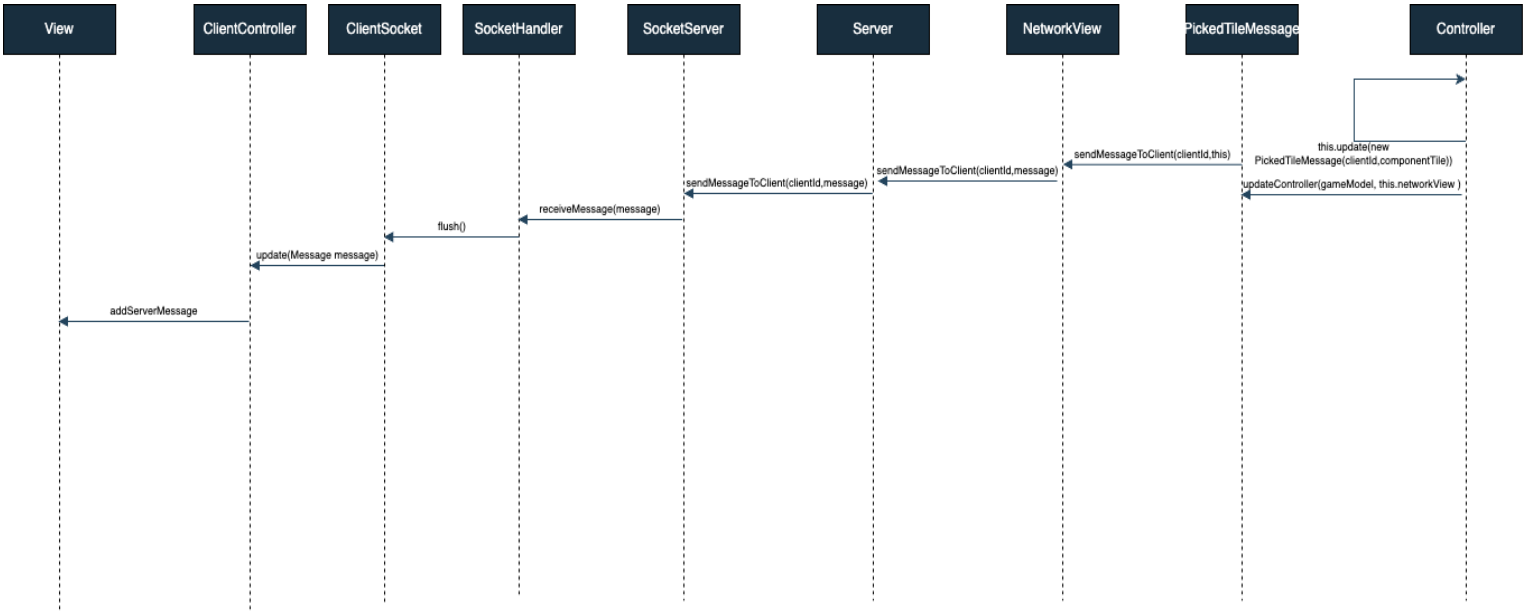
La comunicazione avviene in modo analogo con tutti gli altri messaggi.

Risposta del Server con invio del messaggio al client connesso tramite RMI:

Documentazione Protocollo di Rete



Risposta del Server con invio del messaggio al client connesso tramite Socket:



Ecco una lista di tutti i messaggi che vengono scambiati in rete:

- *AbandonedShipChoiceMessage*
- *AbandonedStationChoiceMessage*
- *ActivateCannonMessage*
- *ActivateEngineMessage*
- *ActivateShieldMessage*
- *AddedPlayerMessage*
- *AskFlightTypeMessage*
- *AskMaxPlayerAndFlightTypeMessage*
- *AskNicknameMessage*
- *AskSelectGameMessage*
- *CardsPileMessage*
- *ChangeTileDirectionMessage*

Documentazione Protocollo di Rete

- *ChooseFlightTypeMessage*
- *ClearPageMessage*
- *CombatZoneChoiceMessage*
- *CreateGameMessage*
- *DefeatedPiratesMessage*
- *DiceRolledMessage*
- *DrawnCardMessage*
- *EndTurnMessage*
- *Fase1ChoiceMessage*
- *Fase2ChoiceMessage*
- *Fase3ChoiceMessage*
- *FinishedBuildingMessage*
- *GainGoodMessage*
- *GameClosedMessage*
- *GameListMessage*
- *GenericErrorMessage*
- *GenericErrorMessage2.java*
- *GenericMessage*
- *GenericModelChangeMessage*
- *JoinGameMessage*
- *MaxPlayersForGameMessage*
- *Message*
- *MeteorHitMessage*
- *MeteorSwarmChoiceMessage*
- *NextFaseMessage*
- *NextPlayerMessage*
- *NextRollMessage*
- *NextTurnMessage*
- *NicknameAcceptedMessage.java*
- *OpenSpaceChoiceMessage*
- *PickedCardPileMessage*
- *PickedTileMessage*
- *PickUpCardMessage*
- *PickUpCardPileMessage*
- *PickUpTileMessage*
- *PingMessage*
- *PiratesChoiceMessage*
- *PlanetChoiceMessage*
- *PutCardPileBackMessage*
- *PutTileBackMessage*
- *PutTileInShipMessage*
- *RemoveBatteryMessage*
- *RemoveFigureMessage*
- *RemoveGoodMessage*
- *RemoveTileMessage*
- *RepairingShipMessage*
- *ReturnAllShipBoardsMessage*
- *RollDiceMessage*

- *ServerDisconnectedMessage*
- *ShowAllShipBoardsMessage*
- *ShowLobbyMessage*
- *ShowShipBoardMessage*
- *ShowTilesDeckMessage*
- *SlaversChoiceMessage*
- *SmugglersChoiceMessage*
- *TimerMessage*
- *UsernameMessage*