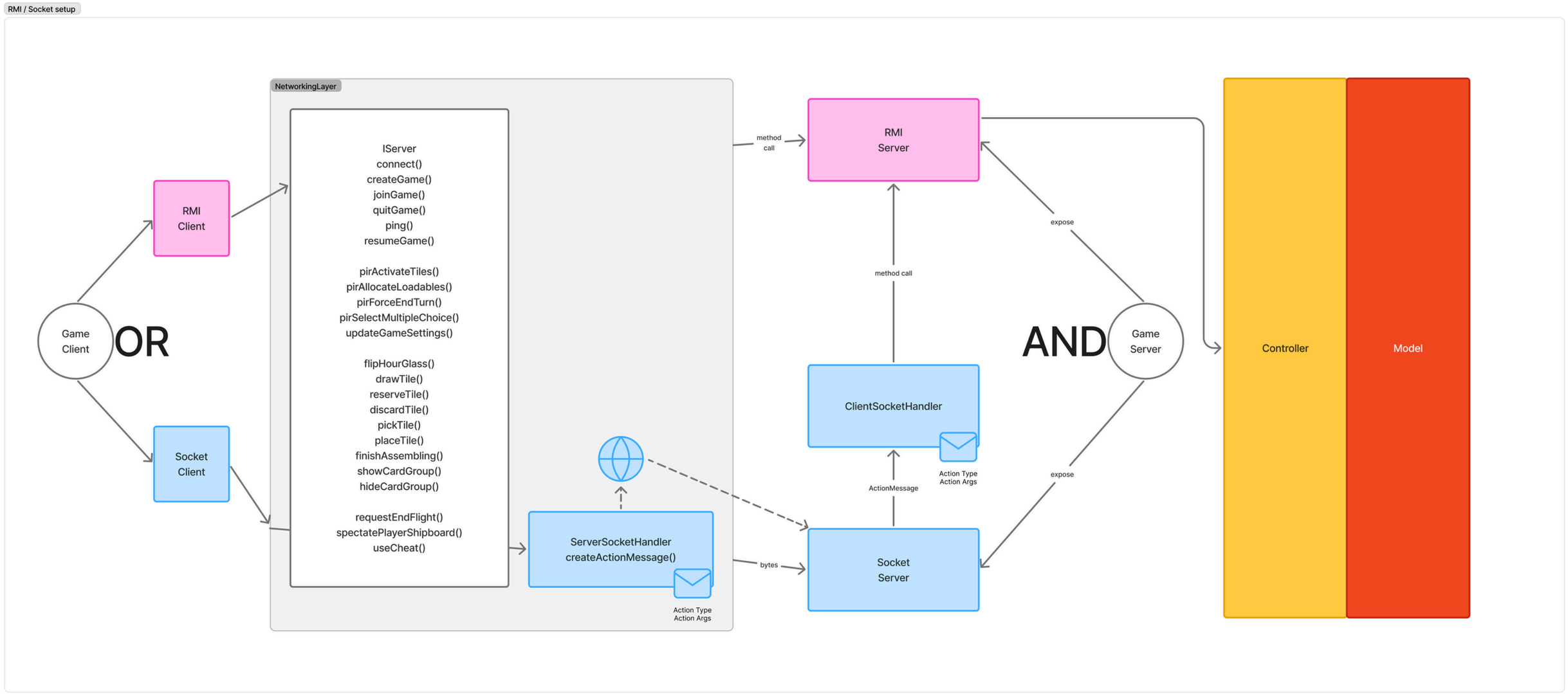
RMI + Socket Setup



RMI + Socket Setup

Il Client può scegliere che tipo di protocollo usare per comunicare con il server.
Nel caso dell'**RMI**, vengono esposti dei metodi **Remoti** sull'oggetto remoto IServer, che il client può chiamare. L'invio del messaggio è gestito dalla libreria di RMI stessa.

Nel caso della **socket**, il client ha accesso ad un'implementazione diversa dell'interfaccia IServer. Questa, chiamata **ServerSocketHandler**, crea, serializza, e manda al server dei messaggi appositi contenenti il tipo di comando richiesto e gli argomenti.

```
@Override  2 usages
public void joinGame(IClient client, UUID gameId, String username, MainCabinTile.Color desiredColor) {
    SocketMessage mess = SocketMessage.joinGameMessage(gameId, username, desiredColor);
    sendSocketMessage(mess);
}
```

RMI + Socket Setup

```
try{
    byte[] decodedMessage = Base64.getDecoder().decode(line);
    message = SocketMessage.deserialize(decodedMessage);
}catch(ClassNotFoundException | IOException e){
    System.err.println("Could not deserialize message: " + line);
    e.printStackTrace();
```

```
switch (message.getType()) {
    case PING -> getServer().ping( client: this);
    case JOIN_GAME -> getServer().joinGame(
            client: this,
            (UUID) message.getArgs().getFirst(),
            (String) message.getArgs().get(1),
            (MainCabinTile.Color) message.getArgs().get(2)
    );
    case CREATE_GAME -> getServer().createGame(
            client: this,
            (String) message.getArgs().getFirst(),
            (MainCabinTile.Color) message.getArgs().get(1)
    );
    case UPDATE_SETTINGS -> getServer().updateGameSettings(
```

Il Server riceve sulla socket il messaggio, lo deserializza e lo legge, capendo quale metodo deve essere chiamato e con quali argomenti.

Viene quindi fatto accesso al riferimento dell'RMI, e viene riusato il metodo sull'RMIServer corrispettivo, per poter riciclare il codice della logica Controller in modo pulito.

Sequence Diagram

Accesso al gioco

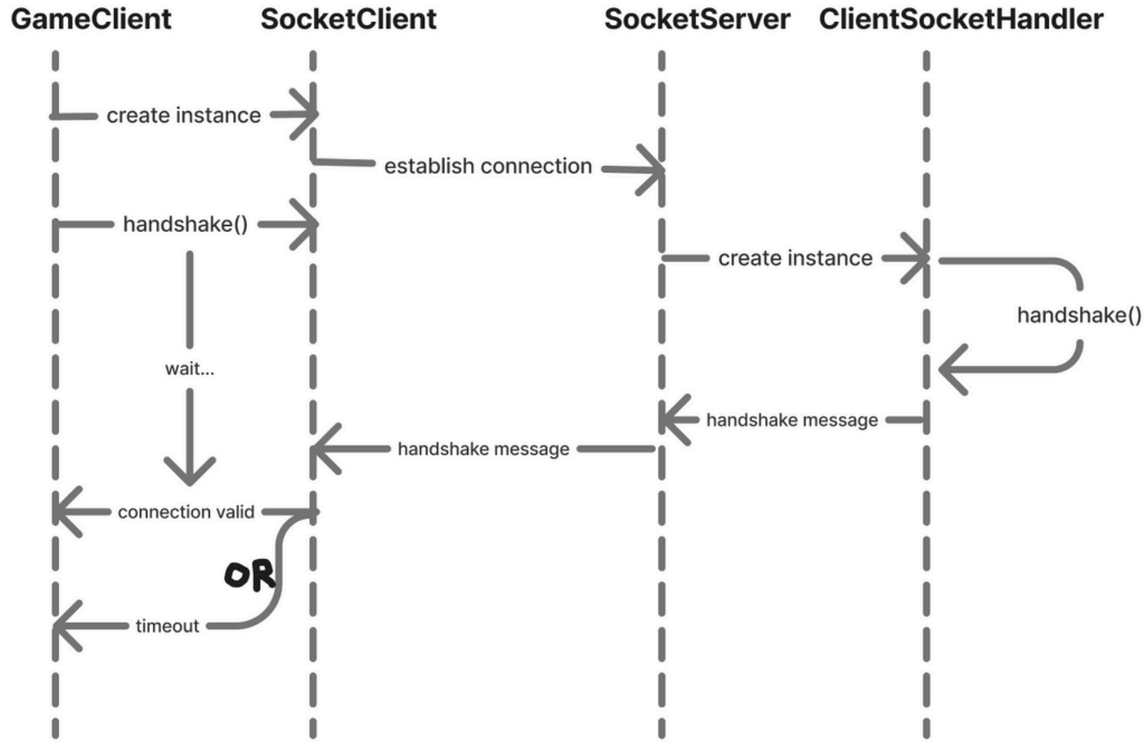
Socket Handshake

Necessaria per assicurarsi che il client stia usando una connessione di tipo **SOCKET** sulla porta del server corretta, e non su quella per **l'RMI**.

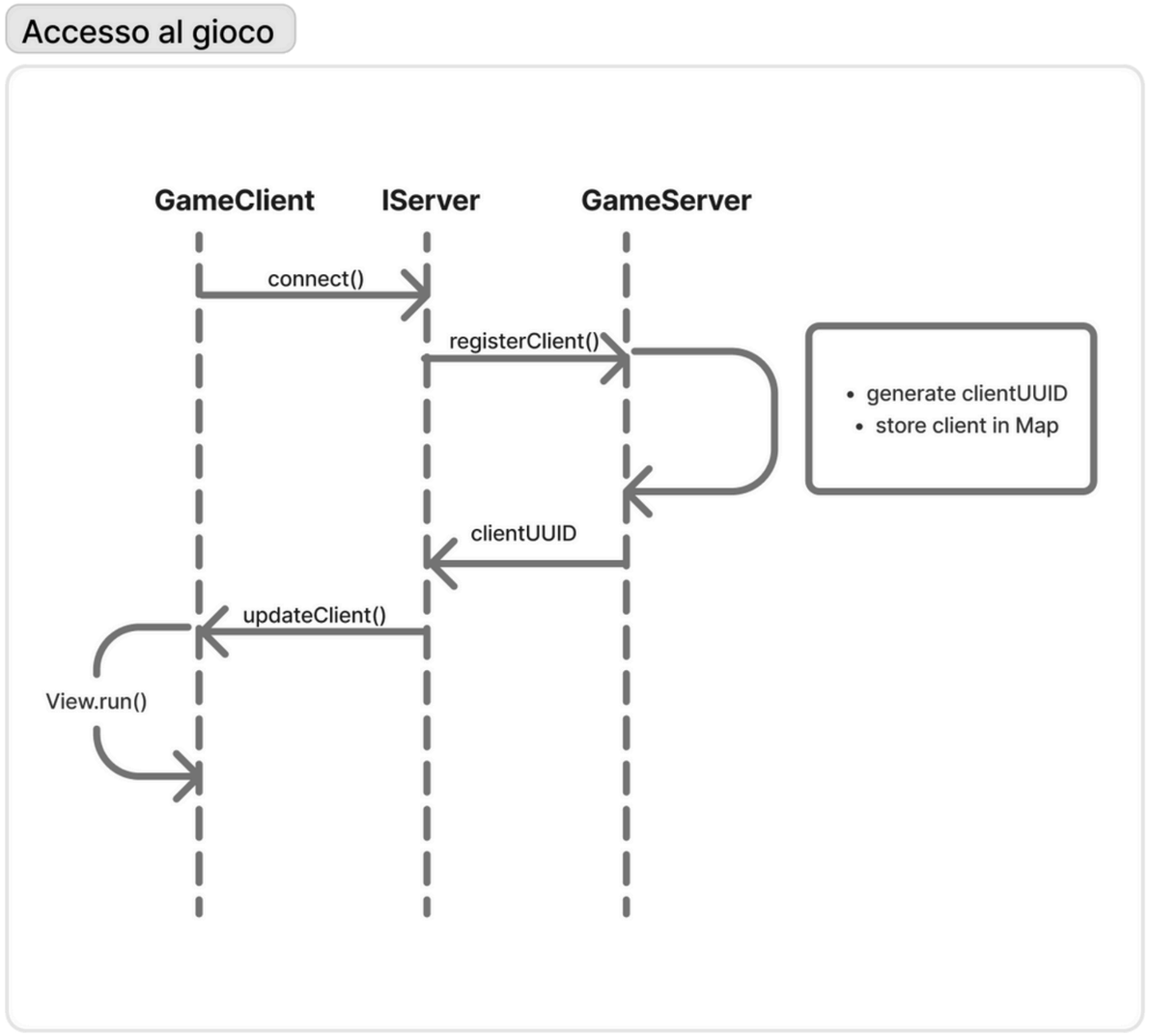
Appena il canale di comunicazione si apre, il client aspetta un messaggio specifico dal server, che viene emesso **SOLTANTO** se è stata usata la porta per la comunicazione **SOCKET**.

Se il messaggio non arriva entro un timeout, il client chiude la connessione e termina l'esecuzione.

Handshake Socket



Accesso al Gioco



Quando viene rilevata una nuova connessione, il Server genera un UUID casuale e lo assegna a questa.

L’oggetto della connessione viene inserito in una mappa, e viene inviato un update al client con le informazioni attuali (es lista di partite attive, etc)

Sequence Diagram

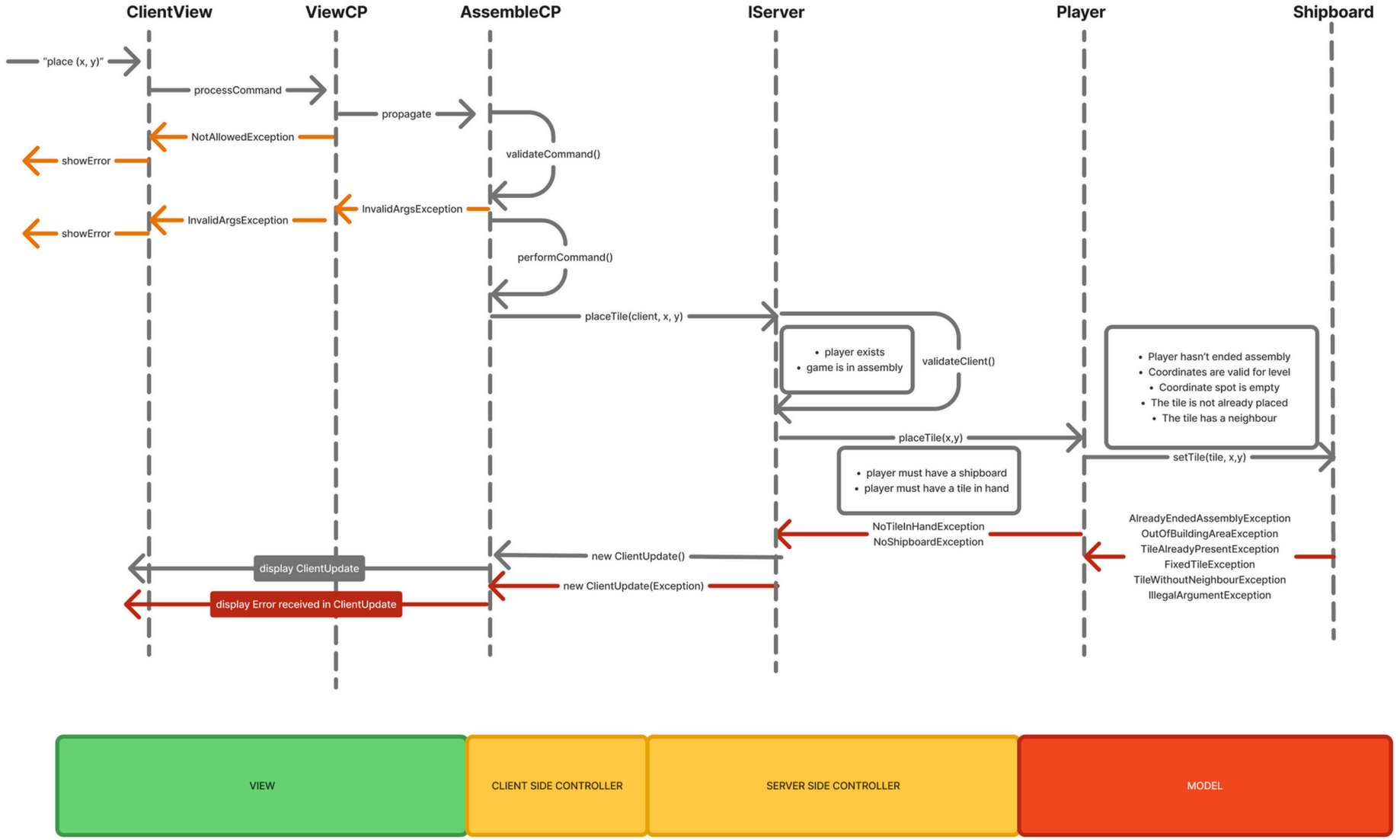
Agganciare un Componente

Agganciare un componente

Il client rileva, verifica il comando inserito dall'utente e fa dei controlli preliminari **CLIENT SIDE**.

Se tutto ha successo manda al server il comando, che ripete gli stessi controlli **SERVER SIDE**, ed applica al model la trasformazione.

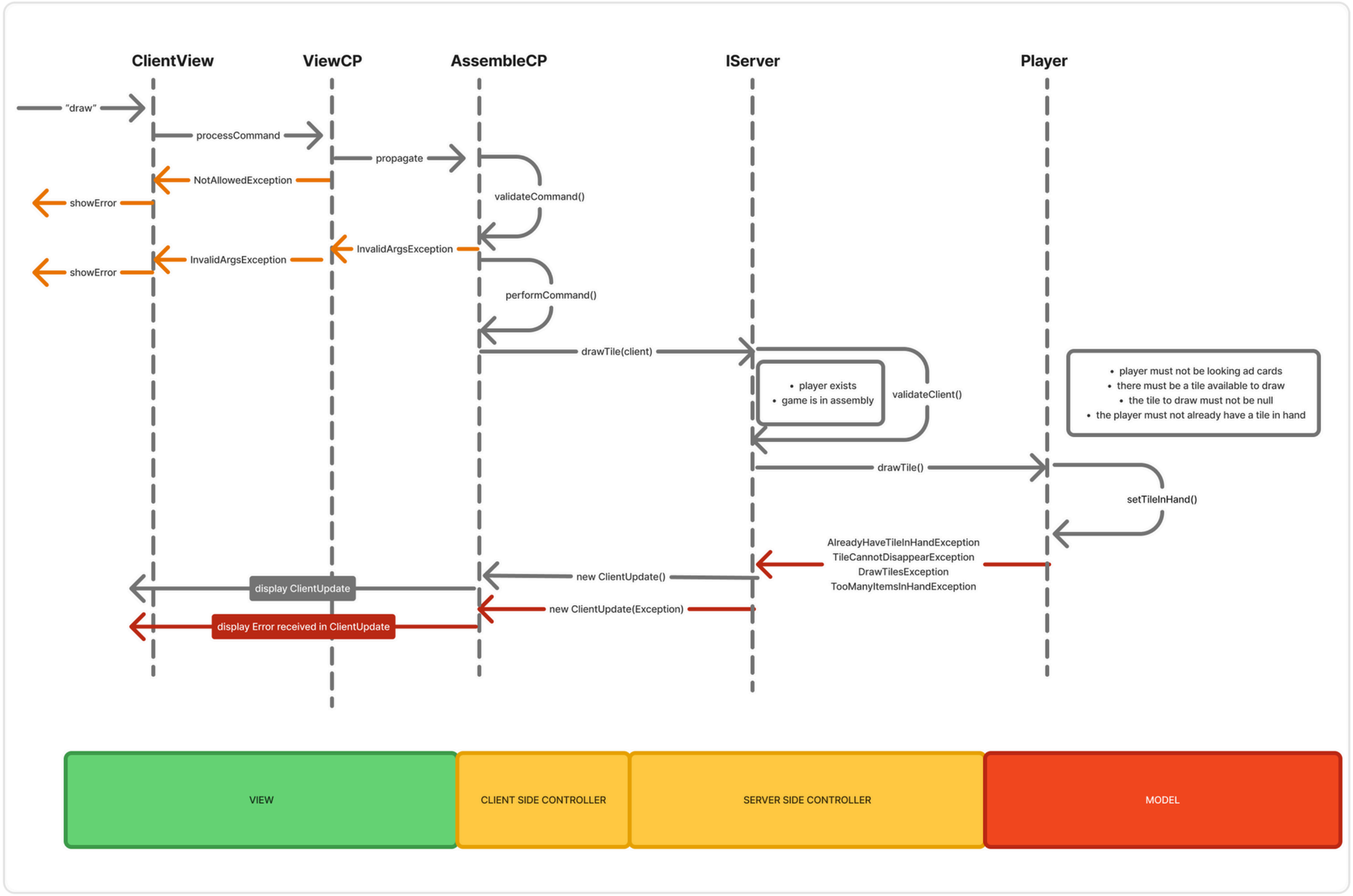
Viene infine generato ed inviato un client update con il nuovo stato.



Sequence Diagram

Pescare una Carta Componente

Pescare Una Carta Componente



Il client rileva, verifica il comando inserito dall'utente e fa dei controlli preliminari **CLIENT SIDE**.

Se tutto ha successo manda al server il comando, che ripete gli stessi controlli **SERVER SIDE**, ed applica al model la trasformazione.

Viene infine generato ed inviato un client update con il nuovo stato.