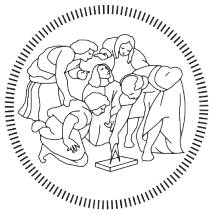
UML Peer Review

Gruppo GC36:
Andrea Tarabotto, Nicola Tummolo,
Gabriel Voss, Francesco Zuliani

6 Maggio 2024



POLITECNICO
MILANO 1863

Indice

1 Introduzione

Per gestire la parte di rete nel pattern MVC si è optato per una struttura come in fig.1. Il server ed il client scambiano messaggi attraverso un'interfaccia di rete che può essere divisa in due direzioni (*GO* e *COME*). Esse poi comunicano come da pattern MVC con controller e view della macchina su cui sono ospitati.

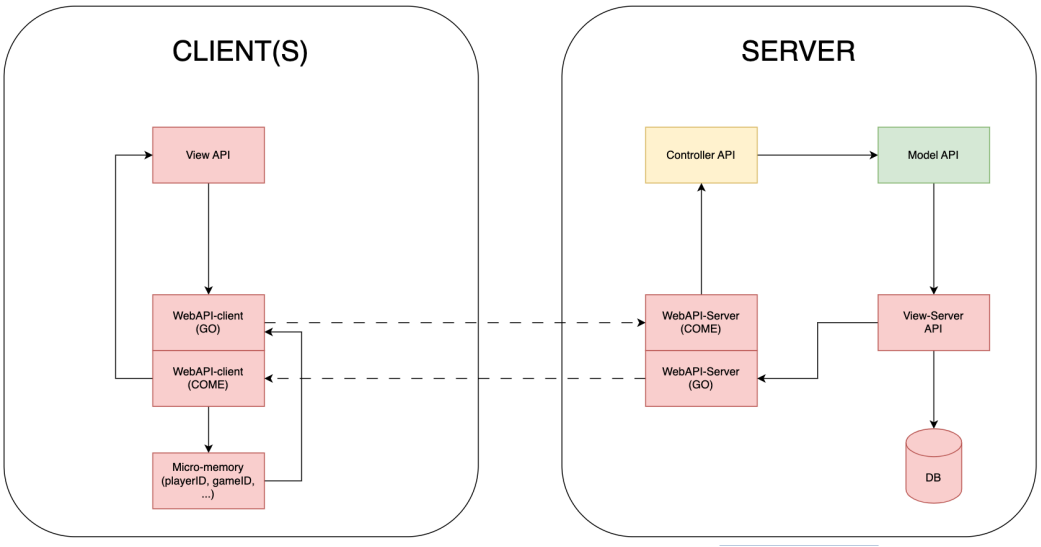


Figura 1: ciao

L'idea generale che ha guidato lo sviluppo è stata quella di instaurare una comunicazione attraverso dei messaggi (di tipo *Message*). I messaggi sono gestiti in una coda specifica il cui accesso deve essere sincronizzato.

2 Lobby

La struttura della lobby risulta particolarmente semplice, una classe Lobby svolge il ruolo di *controller*, che gestisce le strutture dati delle stanze (*Room*) e della lista di messaggi in ingresso (*LobbyMessageQueue*).

2.1 Managers

RMI_Manager e *SocketManager* sono due classi deputate all'ascolto di nuove connessioni. Il loro unico compito è accettare nuovi giocatori e creare nuovi handler personali.

2.2 Handlers

RMI_handler e *Socket_handler* sono gli handler *client-specific*: dalla parte del server ogni handler è associato ad uno specifico client, questo semplificherà notevolmente lo svolgimento delle procedure di invio e ricezione dei messaggi. Dalla parte del client, ovviamente l'handler sarà singolo e di un solo tipo (RMI o Socket).

3 ServerAPI (COME GO)

Nel momento in cui viene instanziata una nuova partita, ad ogni controller viene associata una struttra di rete di questo tipo. Le due classi sono state progettate per separare l'interfaccia di uscita da quella di ingresso, in questo modo il controller non deve comunicare e gestire i vari handler (che tra l'altro devono poter essere di tipologia differente, ovvero RMI o Socket). Il controller infatti si interfaccerà solamente con l'interfaccia COME, da cui estrarrà un messaggio (dalla lista dei messaggi arrivati) e, dopo un controllo che il messaggio sia del tipo atteso, eseguirà il metodo execute del messaggio passsando in argomento se stesso. L'interfaccia GO del server sarà invece dedicata alla view, che potrà comunicare (in broadcast o al singolo player) il messaggio che desidera.
Una struttura analoga è implementata anche nel client.

4 Protocollo di inizio della connessione

Il protocollo, come anticipato, si basa sullo scambio di messaggi. Per ogni connessione all'inizio sarà necessario passare i riferimenti della connessione remota dai "Manager" ad una nuova istanza delle classi *Handler*. Questo problema è affrontato diversamente a seconda che la connessione sia via RMI o Socket.

4.1 RMI

A seguito dell'instaurarsi della connessione tra *Manager* (che si trova sul Server) e *ServerHandler* (che si trova sul client), verrà mandato da parte del Client un messaggio del tipo *NewRMI_ConnectionMessage* che contiene tutti i riferimenti all'oggetto remoto di cui avrà bisogno il server per aprire la connessione dalla sua parte. A seguito della creazione dell'Handler, viene inviato un *WelcomeMessage* da parte del Server che al suo interno ha le generalità per instanziare una nuova connessione RMI verso l'Handler appena creato. Questo permettere di passare da una struttura "a triangolo" formata da client, server e manager, ad un collegamento diretto tra client e server, permettendo di bypassare il manager.

4.2 Socket

Questo in socket non è necessario, in quanto il client invia direttamente il welcome message al server. Il Server Manager in questo caso si mette in ascolto di nuove connessioni, ricevuta una connessione viene instanziato il nuovo handler e gli viene passato il Socket, viene quindi fatto partire un thread in ascolto. Notiamo che non è necessario scambiare messaggi che si passino le generalità come in RMI perchè è sufficiente passarsi l'oggetto Socket.

4.3 Parte Comune

Una parte comune alle due connessioni è lo scambio dei messaggi che permettono di gestire l'inizio della partita: in entrambi i casi un *WelcomeMessage* viene mandato dal Server verso il Client con all'interno (oltre ai dati relativi ad RMI se necessario) una lista di String con all'intero i nomi delle stanze a cui è possibile accedere. Si ha poi una risposta da parte del Client con un messaggio in cui compaiono nickname e nome della partita a cui si vuole accedere (nel caso si voglia creare una nuova stanza andrà specificato anche il numero dei giocatori). Il server risponde infine con un messaggio di ACK al cui interno specifica il nome del giocatore destinatario.

5 Note Finali

La concorrenza è garantita proprio grazie all'uso delle liste dei messaggi in arrivo: un thread si occupa di continuare a leggere ed elaborare i messaggi presenti in coda (qualora ve ne siano), mentre altri thread (auto-generati nel caso di RMI o generati alla generazione dell'handler nel caso di Socket) accodano nuovi messaggi quando avviene una comunicazione. Sono, inoltre, stati esclusi i messaggi dall'UML per una questione di compattezza.