Peer Review II: Network Analysis

Puccia Niccolò, Pozzato Luca, Romano Stefano, Saso Edoardo

May 10, 2024

Abstract

La presente peer review si propone di evidenziare punti di forza, criticità e spunti di miglioramento emersi in seguito all'analisi del diagramma UML fornitoci prestando particolare attenzione agli aspetti relativi alla progettazione della rete. Le osservazioni di seguito elencate mirano a condurre il gruppo revisionato verso una progettazione efficiente e coerente ma potrebbero risentire di assunzioni fatte dal gruppo revisore in casi in cui la sola analisi della sintassi UML o delle intenzioni implementative risultasse poco chiara o, a tratti, equivoca.
In quanto gruppo 17 ci rendiamo disponibili per qualsiasi tipo di chiarimento o precisazione in merito a quanto di seguito riportato.

1 Punti di forza

Dall'analisi del modello proposto sono emersi molteplici aspetti positivi che dimostrano una buona comprensione dei principi di design del software. Alcuni di questi hanno condotto a un ripensamento delle strutture del nostro codice favorendo un'occasione di dibattito e apprendimento interno al gruppo. Di seguito vengono riportate le principali:

1.1 Utilizzo del command pattern

La scelta implementativa del Command Pattern permette una maggiore modularità e disaccoppiamento tra i componenti del progetto aumentando di conseguenza anche l'estensibilità. Nel caso in esame permette al server di essere agnostico riguardo modalità di connesione dei vari client.

1.2 Compartimentazione logica delle fasi del gioco

La scelta di separare il modello del gioco in una fase di connessione e una di normale proseguimento della partita si rivela essere a nostro avviso pienamente rispondente alle esigenze di rete oltre a semplificare concettualmente la rappresentazione dei protocolli di comunicazione. Occorre tuttavia tenere a

1.3 Utilizzo del paradigma observer/observable

Con l'implementazione del design pattern gli osservatori possono essere notificati in modo asincrono quando avviene un cambiamento di stato, consentendo un'architettura reattiva e responsiva

1.4 Sguardo al futuro

Abbiamo avuto modo di notare come il modello tenga in conto dalla fase iniziale delle funzionalità avanzate che il gruppo ha scelto di implementare. Considerare da principio l'inserimento degli elementi necessari al funzionamento delle versioni avanzate del gioco permette infatti di rispariamare del tempo altrimenti impiegato nella modifica di porzioni di codice precedentemente funzionante ma incompatibili con le funzionalità aggiuntive.
La scelta di introdurre ad esempio un gestore di partite multiple o l'introduzione di un meccanismo di invio di ping al fine di testare le disconnessioni dei client dimostrano la capacità di visione di lungo periodo del gruppo revisionato.

2 Criticità e possibili miglioramenti

Durante l'approfondita analisi del sequence diagram relativo al progetto, emergono diverse criticità che richiedono un'attenta revisione e correzione.

2.1 Chiarezza espositiva

In particolare, la modalità di rappresentazione delle scelte di connessione risulta ambigua e poco chiara, lasciando incertezze sul momento esatto in cui avviene la selezione della connessione stessa. La nota relativa all'erroneo nome "RMI setup", non chiarisce appieno la situazione, poiché fa riferimento anche a una fase di serializzazione del nickname, un'operazione non inclusa nella connessione RMI. Una migliore chiarezza potrebbe essere ottenuta raffigurando in modo più esplicito la sequenza di scelte, magari iniziando con la selezione dell'interfaccia utente, seguita dalla scelta del tipo di connessione e infine dalla visualizzazione della schermata principale.

2.2 Gestioni errori

Inoltre, un'altra imprecisione nel sequence diagram è rappresentata dalla mancanza di gestione degli errori nel processo di selezione del nickname. Dopo che il cliente ha effettuato la sua scelta, inviando il nickname al server, non è chiara la modalità con cui il cliente riceverà una conferma di successo o un eventuale messaggio di errore. Si suggerisce quindi di integrare nel diagramma una rappresentazione delle possibili risposte del server, al fine di garantire una comunicazione chiara ed efficace tra client e server.
Un ulteriore aspetto che potrebbe risultare critico riguarda l'interfacciamento tra il client RMI e il SocketClientHandler durante il flusso di RMI GameFlow.

La documentazione relativa alla classe SocketClientHandler risulta poco chiara nell'illustrare in modo esaustivo il suo ruolo all'interno del contesto RMI, rendendo difficile la comprensione del suo funzionamento. Si consiglia pertanto di migliorare la chiarezza nella denominazione delle classi coinvolte, ad esempio sostituendo il nome "SocketClientHandler" con "RmiClientHandler", in modo da garantire una maggiore coerenza e comprensibilità nel contesto dell'interazione tra client RMI e gestore degli eventi di rete.

2.3 Asincronia/ Sincronia delle connessioni

Dalla rappresentazione dei protocolli forniti non risulta immediatamente chiara la scelta di rendere asincorno RMI oppure sincronizzare socket per garantire lo stesso funzionamento indipendentemente dalla connessione scelta. Seppur l'intenzione del gruppo è derivabile dall'osservazione attenta del modello tale scelta potrebbe essere resa più esplicita.

2.4 Observer/observable

Per quanto riguarda l'UML, la presenza di numerosi listeners, uno per componente, potrebbe rendere il diagramma confusionario e di difficile interpretazione. Un'alternativa valida potrebbe l'adozione di un approccio basato su eventi che aggiornano un modello semplificato. In questo scenario, ogni evento ricevuto dal client verrebbe elaborato da un modello semplificato, il quale propagherebbe le modifiche alle views interessate. Questo approccio non solo garantirebbe una maggiore chiarezza e coerenza nel design complessivo del sistema, ma semplificherebbe anche la gestione degli eventi e dei componenti, rendendo il diagramma più intuitivo e facilmente comprensibile.

3 Confronto tra architetture

Nell'ambito delle scelte riguardanti la rete i due progetti si rivelano simili sotto molteplici aspetti tra cui la gestione separata delle fasi di handshake e gioco e la volontà di rendere trasparente al meccanismo di gestione dei comandi il tipo di connesione tramite cui essi transitano in rete.
Analogamente alla soluzione da voi proposta anche il nostro gruppo ha scelto di implementare il command pattern per eliminare la necessità del server di conoscere i dettagli dell'implementazione dei metodi del client e della sua modalità di connessione, ma interagisce solo con un'interfaccia comune (il comando).
Riteniamo che il command pattern sia utile per gestire code di operazioni da eseguire in modo asincrono o in un momento successivo. I comandi possono essere inseriti in una coda e processati in ordine quando è disponibile la capacità di eseguirli.
Inoltre abbiamo scelto di non configurare esplicitamente le view come osservatori del server ma di implementare una copia semplificata del model lato client (da ora in avanti minimodel).

Esso contiene tutte le informazioni necessarie alle view per visualizzare i componenti del gioco e per costruire i messaggi da far pervenire al controller tramite la rete.
Dunque alla ricezione della richiesta dell'utente di effettuare un'azione il model verifica la legittimità della mossa e invia indietro il delta dei cambiamenti da visualizzare aggiornando il minimodel. Nella creazione dell'evento di modifica del model da parte della view sarà proprio qil minimodel a rappresentare la collezione di oggetti dalla quale prelevare quelli funzionali alla chiamata dei giusti metodi del controller.
I due approcci, per quanto diversi da un punto di vista di design del progetto, sono funzionalmente equivalenti e assumono significato nel contesto all'interno del quale sono stati pensati.