Peer-Review 2: PROTOCOLLO DI RETE

Simone Boscain, Andrea Croci, Giovanni De Marco, Nicolò Di Carlo

Gruppo AM40

Valutazione del diagramma UML delle classi del gruppo AM49.

Lati positivi

Il protocollo di comunicazione del gruppo AM49, anche ad una rapida analisi, risulta molto chiaro ed intuitivo che copre i punti giusti ed essenziali al funzionamento della rete. Grazie anche al file di accompagnamento inviato in allegato ai sequence diagram si evince molto bene il funzionamento di, innanzitutto, tutti i metodi di rete legati al login correlati alla creazione delle Room e, infine, dei metodi di gioco racchiusi dentro GameAction e GameUpdate che ben si adattano al paradigma di programmazione ad eventi MVC.
Risulta molto ordinato anche il meccanismo delle Room e di gestione delle eccezioni che fornisce un'ottima organizzazione al Server per ospitare partite multiple. Inoltre è molto apprezzabile il meccanismo di ping pong utilizzato per testare se la connessione è caduta oppure no, con conseguente diagramma per illustrare la riconnessione del client in caso di errore di rete che ci fa intuire la vostra scelta nell'implementazione della FA "Resilienza alle disconnessioni".

Lati negativi

L'architettura rappresenta una visione molto ad alto livello dal lato client che lascia per scontato la correttezza degli input dell'utente per l'associazione del metodo RMI. Con questa affermazione si intende dire che se un utente dovesse posizionare una carta e fosse, quindi, nel gameState PlaceCardState, ma decidesse per qualunque motivo di mettere un input non valido oppure, per errore, mettesse un input tipico della draw cosa succederebbe? A intuito, il gameState e il currentPlayer passato tramite un update alla fine della fase precedente potrebbero essere usati come checker di contesto, però non viene accennato sul grafico o sul file di accompagnamento un modo per gestire la correttezza dell'input. Non è un vero e proprio lato negativo, visto che questo non concerne direttamente la comunicazione rete via RMI, ma soltanto il meccanismo di scelta per l'utilizzo di un metodo rispetto che ad un altro a seguito di un input.
Un meccanismo che pone un gran dubbio è quello di utilizzare GameAction già da client con RMI, a seguito delle dichiarazioni fatte dai tutor e dai professori in laboratorio. La vostra scelta di design non sarebbe come mandare un messaggio TCP (GameAction) e fare parsing attraverso un cast del dato in una GameAction che verrà eseguita nel Server? Il funzionamento per RMI è il medesimo, tramite invocazione di un metodo remoto RMI che fa "quasi tutto" e si specializza tramite dinamismo a runtime a causa del polimorfismo dell'azione passata tramite parametro. Ciò non sarebbe TCP, mascherato sotto forma di RMI?
Passando ad aspetti più tecnici, visto l'utilizzo prevalente della logica RMI per rappresentare i sequence diagram, si nota la mancanza dei tipici proxy, quindi dei paradigmatici RMiRegistry, Stub e Skeleton. Siamo coscienti che ad alto livello, quale può essere un sequence diagram, questo possa risultare un dettaglio abbastanza insignificante al fine di garantire una migliore visione di insieme. Si fa presente, però, che a livello implementativo, vista la vostra volontà di realizzare anche un canale di comunicazione tramite Socket TCP, risulterà più modulare e funzionale creare interfacce separate e adeguate al canale di comunicazione che verrà scelto dall'utente, altrimenti si rischierà di rendere la classe server estremamente pesante.

Ciò è dovuto anche dal fatto che il paradigma di programmazione ad eventi MVC risulta leggermente alterato dal classico funzionamento e si suppone che sia a causa di un passaggio più diretto di informazioni da Client a Model e viceversa. Ciò si rispecchia nel sequence diagram dell'esecuzione di una GameAction: solitamente la VirtualView del Server emula la View del Client, la quale, ricevendo degli input con lo scopo di modificare i dati e lo stato del Model, notifica il Controller, che funge da Listener e viene avvisato delle modifiche da eseguire e, una volta apportate, il nuovo stato del Model viene notificato alla View (se viene utilizzata un canale di rete, si passa dalla VirtualView e poi alla View del Client). Nel vostro caso, il Server, che riceve gli input del Client tramite invocazione di metodo remota, agisce sulla Room modificando direttamente lo stato del Model e una volta fatto ciò viene notificata la VirtualView di quello che è successo e a sua volta vengono avvisati i client in broadcast. A questo punto, si nota un netto distaccamento dal paradigma di programmazione ad eventi MVC che si traduce in un utilizzo spaccato della Virtual View tradizionale che lascia la logica separata in due parti: Server funge da azionatore oppure pseudo-poster (visto l'utilizzo di chiamata a metodo diretta di RMI), per il Controller che si presume possa essere Room e VirtualView che funge da Listener per il Model, una volta notificato, avvisa i Client in qualche modo. Non è chiaro come VirtualView possa notificare i Client via rete, visto che Server è il principale punto di entrata per la comunicazione. Quindi, senza chiarimenti, la VirtualView, presente nel vostro sequence diagram, non riesce a fare intuire come possa "quadrare" il funzionamento complessivo di update da parte del server. Finché si sceglie di utilizzare solo RMI la scelta funziona e rimane limitata, però, integrando TCP, Server rischia di diventare davvero molto ingombrante visto che racchiuderà i metodi di interfaccia RMI, le funzionalità di TCP con parsing del messaggio e funge da "meccanismo di partenza" lato server per le azioni da effettuare sul modello.

Confronto tra le architetture

Il gruppo in analisi ha un'idea di fondo generale del protocollo di rete abbastanza simile alla nostra, le Room sono sostanzialmente i nostri NetworkParty con il fine medesimo di gestire al meglio partite multiple e altre classi che svolgono funzioni quasi identiche hanno soltanto nomi diversi. Graficamente, rispetto al nostro protocollo di rete, quello del gruppo in analisi risulta molto più pulito e intuitivo, soprattutto con il meccanismo multicast per l'update di informazioni da parte del Server, quindi sarebbe un'ottima modifica prendere spunto da questo gruppo la modalità di rappresentazione degli "attori" sui sequence diagram per gestire la comunicazione con più Client. Infine, potremmo prendere sicuramente spunto dai sequence diagram inerenti al controllo della connessione, quindi, il meccanismo di ping pong per vedere se un Client si è disconnesso o è ancora collegato e il conseguente meccanismo di riconnessione per l'implementazione della FA "Resilienza alle disconnessioni".