Peer-Review 2: Network

Berardinelli, Genovese, Grandi, Haddou
Gruppo AM21

May 7, 2024

Peer review del protocollo di rete del gruppo AMXX

Contents

1 Lati positivi

1.1 Chiarezza del diagramma presentato

Il diagramma presentato è tutto sommato abbastanza chiaro e gestisce la quasi totalità delle meccaniche di gioco.

1.2 Getter per i game disponibili

Abbiamo apprezzato l'inclusione della funzionalità che consente di visualizzare i giochi a cui il giocatore può partecipare (tramite `getGameIds()`), poiché semplifica notevolmente il processo di accesso a una partita

2 Lati negativi

Raccogliamo qui una serie di appunti, nella speranza che risultino d'aiuto al gruppo in sede di valutazione del progetto.

2.1 Note sulla rappresentazione del diagramma

2.1.A Sequence diagram connessione TCP Il sequence diagram UML per la connessione TCP presenta talvolta delle invocazioni di metodi fra gli attori `Client` e `Server`. La rappresentazione è errata poichè TCP non permette la *remote method invocation*. Sarebbe opportuno mostrare il messaggio che viene inviato da client a server tramite *socket*, modellando il *controller* come un altro attore per evidenziarne – fra il server e questo – l'invocazione dei metodi, le return effettuate o le eccezioni tirate al server e gli specifici messaggi che vengono inviati da quest'ultimo al client nei vari casi.

2.1.B Sequence diagram connessione RMI Dualmente a come scritto in **2.1.A**, il sequence diagram UML per la connessione RMI presenta talvolta dei messaggi. Poichè il protocollo RMI non contempla listener di messaggi, al posto di messaggi di conferma o errore, dovrebbero essere riportate le return dei metodi o le eccezioni tirate dai metodi del controller.

2.1.C Loop, flow alternativi e opzionali

1. Non sono indicate nel diagramma quali parti della sequenza si ripetano e fino a quali condizioni.

2. Talvolta sono indicate sullo stesso messaggio situazioni alternative e mutualmente esclusive che probabilmente sono gestite da ognuno degli attori in modo differente.

3. Se fosse meglio dettagliata, alcune parti della sequenza di messaggi/invocazioni di metodi sarebbero da intendersi come opzionali.

Lo standard del sequence diagram UML mette a disposizione blocchi grafici per situazioni iterative, alternative mutualmente esclusive e condizionali. Si potrebbero inserire nel diagramma per documentare in modo più preciso le singole situazioni.

2.1.D Distinzione fra messaggi sincroni e asincroni I messaggi e le chiamate sincrone o asincrone utilizzano nei diagrammi lo stesso puntatore. Lo standard UML ne prevede due diversi per i due tipi di messaggio/chiamata.

2.1.E Situazioni non modellate

- Il messaggio di `placeCard` non contempla un messaggio di errore. Abbiamo ipotizzato la validazione potesse avvenire lato client ma ci è sembrato uno scenario poco auspicabile, in cui sarebbe facile modificare un client in modo tale da invalidare lo stato di gioco nel server.

- Non è contemplata una fase di scelta della faccia della starter card da piazzare sul tabellone di gioco. Abbiamo ipotizzato potesse avvenire con una `placeCard` ma non ci sarebbe nessun modo, che sia descritto, per garantire lato server che questa sia piazzata per prima.

- Non sono modellate le situazioni in cui il client deve scegliere degli oggetti fra un pool di possibili, come ad esempio le carte obiettivo. Come viene comunicato al client quali fra le due deve scegliere? Inoltre dopo la `drawCard()` non sembra venga comunicato al client che carta viene pescata, in quanto viene inviato solo una conferma/errore.

2.1.F Nits

- La presenza dello `stub` all'interno del diagramma dedicato alla versione RMI del protocollo ci sembra ridondante. Vengono utilizzati gli stessi messaggi (vedere **2.1.B**) sia a sinistra che a destra di questo.
- Dato che la scelta di una `SecretObjectiveCard` avviene fra un paio di carte, sarebbe meglio modellizzarla con un *boolean* chiamato ad esempio `first` .

2.2 Livello di dettaglio del diagramma

Talvolta il diagramma non è particolarmente meticoloso nel dettagliare le interazioni fra gli attori coinvolti. Sottolineamo che lo scopo della documentazione è quello di redigere minuziosamente il protocollo prima di implementarlo, pensando a tutti gli edge case del caso e come vengono gestiti in modo isolato l'uno dall'altro.

2.2.A Conferme e discriminazione degli errori Se come gruppo pensiamo che per le conferme non siano necessarie particolari descrizioni, crediamo che gli errori potrebbero essere meglio differenziati per garantire una esperienza migliore lato client e per rendere possibile una gestione differenziata di questi con flussi alternativi o opzionali. Non è chiaro infatti come avvenga la gestione di un errore da parte del client. Aiuterebbe descriverlo più minuziosamente come anticipato in **2.1.C**

2.2.B Contenuto dei messaggi Riteniamo che gioverebbe alla documentazione del protocollo allegare un class diagram UML dello schema dei messaggi che vengono inviati.

2.2.C Rappresentazione di un unico attore client Il diagramma presentato contempla la comunicazione con un client solo alla volta. Si potrebbe caratterizzare meglio la differenza fra un client che sta giocando e uno che non sta giocando aggiungendoli al diagramma come due attori diversi, `Client` e `Playing client`, per specificare a quali fra questi due vengono inviati i messaggi che aggiornano le *view* e quali sono le interazioni tipo di un client che non sta giocando.

2.3 Perplessità

- A cosa si riferisce `playerAccessToken`?
- Non vi sono messaggi che notifichino il primo giocatore che il gioco è iniziato.
- Non vi sono messaggi specifici per notificare i client che il gioco è terminato.
- In che contesto viene lanciata `leaveGame()`? Abbiamo pensato fosse un metodo previsto per la funzionalità avanzata della persistenza dei giocatori, ma se il client dovesse disconnettersi la *leaveGame()* non verrebbe mai inviata al server.
- Che tipo di messaggio di errore lancia la `drawCard()`? Si tratta di un'eccezione quando il deck è vuoto? E' questa o il messaggio di broadcast a fine turno a notificare il player dell'ultimo round da giocare?

2.3.A Implementazione dei messaggi Non è specificato nella documentazione come vengano realizzati i messaggi: se con oggetti serializzati, se con testo in formato `JSON` o `XML`, che ci sentiamo di suggerire, o se come sembra dal diagramma se si tratti di semplici stringhe senza una struttura rigida.

2.4 Criticità

2.4.A Aggiornamenti delle view tramite stringhe Ci sentiamo, poichè questo non è specificato, di suggerire un formato più robusto e di cui sia più facile il parsing di una semplice stringa per i messaggi che aggiornano le view inoltrati da `broadcastPlayerMessage()`. Si potrebbe inoltre pensare di dividere i messaggi di aggiornamento per renderli più essenziali e rendere le modifiche indipendenti l'una dall'altra.

3 Confronto fra architetture

3.1 Granularità del processo della join Game

Come gruppo, con un approccio diverso dal gruppo AM30, abbiamo scelto di rendere il processo di entrata nel gioco più granulare, prevendendo una `Lobby view` nella quale vengono scambiati una serie di messaggi per settare in modo sequenziale tutti gli attributi del gioco, appunto gestendo ogni richiesta di set nel *player building* process in modo più granulare, cambiando poi a fine di questo processo la view in quella del game.

3.2 Atomicità dell'aggiornamento delle view

Abbiamo scelto un approccio più granulare anche nell'aggiornamento delle view di tutti i client. Viene inviato un messaggio che contiene un oggetto serializzato di tipo diverso che riporta solo gli attributi essenziali per ogni evento che modifica la view, dal piazzamento della carta, pesca di una carta, cambiamento dello status di gioco o del player corrente, delle playerboard e aggiornamento degli score dei player.

4 Appendice

Alleghiamo in appendice il diagramma presentato dal gruppo AM30.

Figure 1: Sequence diagram presentato dal gruppo AM30: TCP

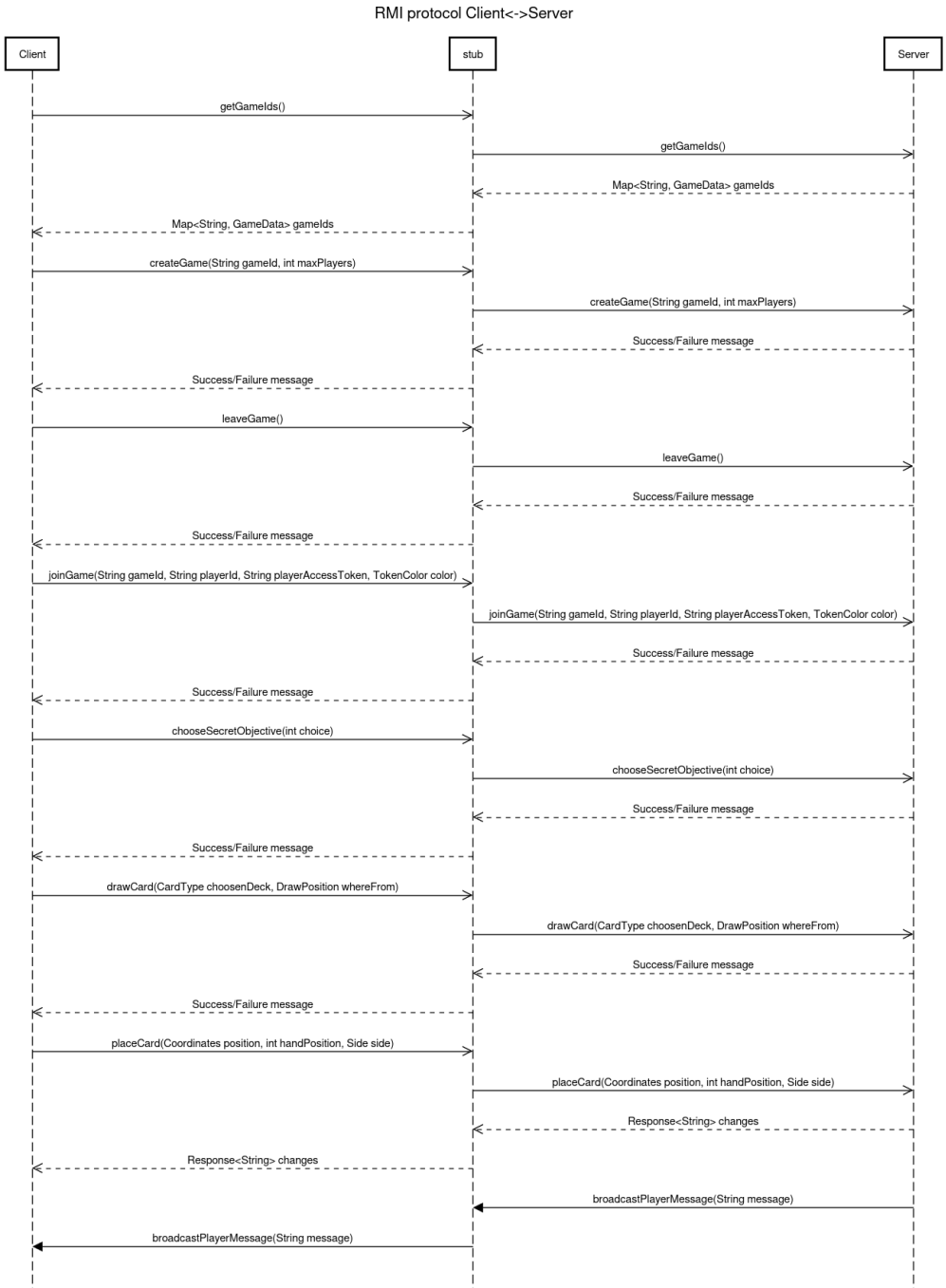

Figure 2: Sequence diagram presentato dal gruppo AM30: RMI