PCD ASS03 (2)

Federico Bravetti
federico.bravetti2@studio.unibo.it

Tommaso Patriti
tommaso.patriti@studio.unibo.it

Alex Testa
alex.testa@studio.unibo.it

I. SUDOKU MOM

A. Tecnologie

Per l'implementazione del progetto è stato utilizzato RabbitMQ. Questa scelta offre diversi vantaggi, tra cui l'utilizzo di una coda immutabile e l'uso di un meccanismo di acknowledgment (ack) per garantire che i messaggi siano stati ricevuti e processati correttamente.

B. Caratteristiche Operative

Per la gestione di utenti e partite sono state effettuate le seguenti scelte:

- Se un utente entra in una griglia non iniziata, resta in attesa che qualcuno la crei;
- Se viene creata una griglia con un id già esistente andrà a sovrascrivere quella esistente;
- Se si partecipa ad una griglia più volte con lo stesso username si avranno varie viste per lo stesso utente.

C. Implementazione

Come illustrato nel diagramma in Fig. 2, all'utente viene richiesto di inserire uno username, l'ID di una griglia, e di specificare se desidera iniziare una nuova partita o partecipare a una esistente. Indipendentemente dalla scelta se non esiste già, viene creato un exchange relativo alla griglia al quale viene associata una coda relativa al client. La coda inizialmente ascolta solo il topic `UPDATE_GRID`. Poi:

- Nel caso di una nuova partita, viene generata una nuova griglia e mandato un messaggio di `UPDATE_GRID`
- Nel caso di partecipazione ad una partita viene mandato un messaggio di `NEW_USER_JOINED`

Alla ricezione del primo messaggio di `UPDATE_GRID` viene inizializzata la griglia e ci si mette in ascolto del topic `NEW_USER_JOINED`. Alla ricezione del messaggio `NEW_USER_JOINED` viene inviato un messaggio di `UPDATE_GRID` con la griglia corrente. Successivamente, ogni volta che viene effettuata una selezione o inserito un numero la griglia aggiornata, se corretta, viene inviata tramite un messaggio di `UPDATE_GRID`. La griglia per essere inviata all'exchange viene prima convertita in formato JSON (marshalling), che viene poi riconvertito in una nuova griglia dai client che la ricevono (unmarshalling).

Ogni client (anche chi propone la modifica) riceve l'aggiornamento della griglia esclusivamente attraverso i messaggi di `UPDATE_GRID` del broker, questo garantisce la consistenza del sistema. Dato che la griglia è utilizzata in modo atomico senza problemi di check-and-act e modificata da un solo thread alla volta, per gestire problemi di concorrenza è stato necessario solamente definirla *volatile*.

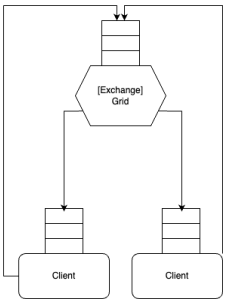

Figure 1. Architettura SudokuMOM

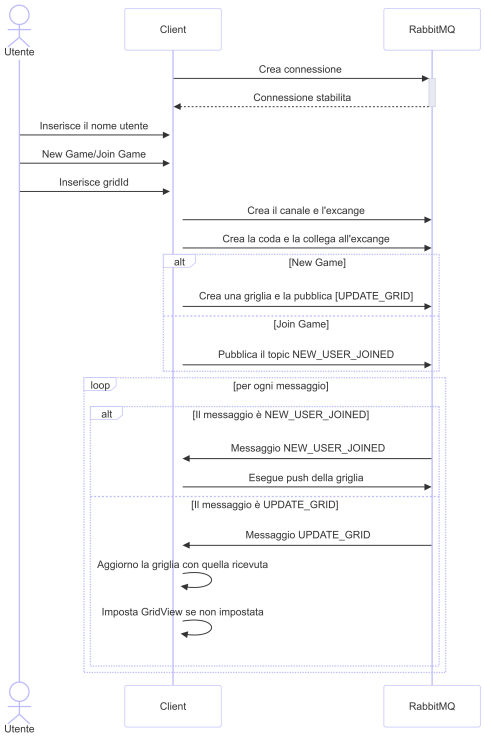

Figure 2. Diagramma di sequenza

D. Vantaggi e Svantaggi

- **Vantaggi:**
  - *Solidità e affidabilità:* Ogni client invia la griglia aggiornata ai nuovi utenti che si uniscono, garantendo che una copia consistente della griglia venga ricevuta e utilizzata da tutti.
  - *Semplicità dell'implementazione:* La logica per gestire lo scambio di messaggi è semplice da implementare, poiché tutti i client si comportano allo stesso modo e non c'è bisogno di un coordinamento complesso.
- **Svantaggi:**
  - *Ridondanza nei messaggi:* Ogni volta che un nuovo utente si unisce, tutti i client inviano la propria versione della griglia, generando un'eccessiva ridondanza nei messaggi. Questo può causare un inutile carico sulla rete e aumentare la latenza.

II. Sudoku RMI

A. Tecnologie

Per l'implementazione del progetto è stato utilizzato *JavaRMI*. Il punto forte di questa tecnologia è la quasi trasparenza che si ha rispetto ad una classica modellazione ad oggetti. Le uniche differenze presenti infatti sono dovute a caratteristiche intrinseche di una architettura distribuita.

B. Caratteristiche Operative

Per la gestione di utenti e partite sono state effettuate le seguenti scelte:

- Per partecipare ad una partita questa deve essere prima stata creata;
- Le partite sono identificate da un SudokuId, quindi per riutilizzare un SudokuId la partita precedente deve essere stata abbandonata da tutti i partecipanti;
- I client sono identificati dallo stesso username dell'utente, quindi all'interno di una partita non ci possono essere partecipanti con lo stesso username.

C. Architettura

L'architettura astratta, riportata in Fig. 3, nonostante risulti banale è rappresentativa dell'intero sistema. Infatti, se non si considera l'overhead dovuto alla trasposizione di questa architettura concorrente in una distribuita, una volta recuperati i riferimenti agli oggetti remoti (Stub) la struttura che si ottiene è assimilabile a questa.

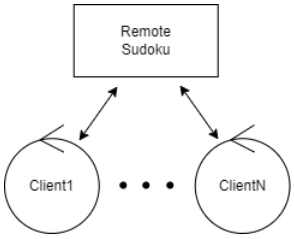

Figure 3. Architettura astratta SudokuRMI

Inoltre si può notare come la soluzione scelta sia centralizzata, infatti la griglia relativa ad una partita è salvata all'interno di un RemoteSudoku che gestisce in modo consistente i vari aggiornamenti. L'architettura reale, riportata in Fig 4, prevede l'interazione tra almeno tre processi distinti: un *RegistrationService*, un *RMIRegistry* e uno o più *Client*.

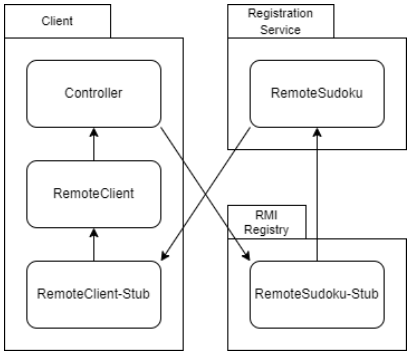

Figure 4. Architettura SudokuRMI

In particolare:

- Nel Client è definita la logica di interazione con i RemoteSudoku ed è esposto un metodo per la ricezione di aggiornamenti;
- Il servizio RegistrationService permette di disaccoppiare i Client dai sudoku che generano così da preservare l'accesso alle partite anche nel caso in cui il Client host si disconnette (termina);
- L'RMIRegistry è di supporto all'interazione tra i Client e gli oggetti remoti poiché raggruppa i vari Stub (RemoteSudoku-Stub e RegistrationService-Stub) e rende semplice recuperarli ed utilizzarli. Il Client-Stub, essendo legato al contesto di una partita, viene direttamente inviato al relativo RemoteSudoku.

Per evitare la presenza di entry orfane nell'RMIRegistry (e.g. a causa di un crash) su di esso si effettuano aggiunte sempre attraverso un'operazione di *Rebind*. Per questo motivo la gestione dell'unicità dei SudokuId è stata delegata al RegistrationService.

D. Comportamento del sistema

Per analizzare nel dettaglio il funzionamento del programma, sono stati riportati di seguito due diagrammi. Nel primo, riportato in Fig. 5, è descritto il

comportamento in fase di creazione/partecipazione ad una partita. In queste fasi si possono notare le principali differenze introdotte dall'utilizzo di JavaRMI, ossia le dinamiche di *Bind* e *Lookup*.

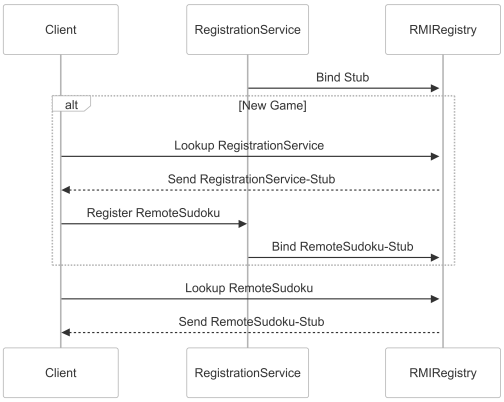

Figure 5. Bind e Lookup degli oggetti remoti

Osservando ora il secondo diagramma in Fig. 6 si può notare come una volta registrati e recuperati gli Stub dall'RMIRegistry, l'esecuzione proceda in maniera del tutto trasparente.

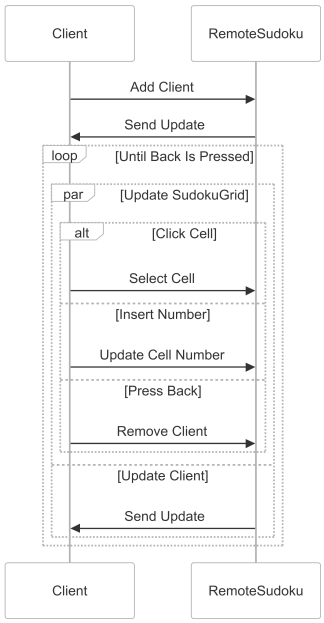

Figure 6. Evoluzione di una partita

Anche le considerazioni legate alla concorrenza sono agnostiche rispetto all'utilizzo di JavaRMI, infatti essendo RemoteSudoku e RegistrationService tecnicamente oggetti soggetti ad accessi concorrenti sono implementati come dei monitor. L'unica differenza che rimane è la possibilità che vengano lanciate delle *RemoteException*.