Prova Finale (Progetto) di Ingegneria del Software - A.A. 2022/23
Sintesi del documento di specifica del protocollo di comunicazione
Peer review n. 2

Andrea Caravano Biagio Cancelliere Alessandro Cavallo Allegra Chiavacci

22 aprile 2023

Indice

1 Socket

L'architettura prevede l'uso di un eseguibile Client e di uno Server.

Sono inoltre state formulate delle entità puramente descrittive, al fine di serializzare e deserializzare strutture parziali, semplificate per il solo client.

Messaggi di stato

I messaggi di stato rappresentano la forma degli scambi relativi all'evoluzione delle fasi del gioco, delle sue mosse e della loro formalizzazione e verifica di correttezza generale.

Vengono di seguito riportati **alcuni** messaggi significativi.

1.1 Accesso al gioco

1.1.1 Nickname

Scambio del nickname che il client vuole adottare nella partita.

Richiesta

```
{
  "status": "Request_Nickname"
}
```

Risposta

```
{
  "status": "Response_Nickname",
  "message": "Player3"
}
```

Dove Player3 rappresenta il nickname che il giocatore vuole adottare all'interno della partita.

1.2 Gestione del turno

1.2.1 Non è il tuo turno

Il client non è il giocatore corrente.
Esso attende l'evoluzione del gioco e viene informato dei cambiamenti in atto, in seguito alle mosse degli altri giocatori.

Formalizzazione

```
{
  "status": "NotYourTurn"
}
```

1.2.2 Mossa

Messaggio di formalizzazione di una mossa del giocatore corrente.

È il tuo turno Il giocatore corrente riceve l'indicazione di poter procedere con una richiesta di mossa, a cui il server risponde.

```
{
  "status": "YourTurn"
}
```

Richiesta

```
{
  "status": "Move_Request",
  "message": "{
                "column": 1,
                "x": [
                        3,
                        3,
                        3
                    ],
                "y": [
                        3,
                        4,
                        2
                     ]
              }"
}
```

Dove sono indicate (in forma di lista) le singole/coppie/triplette di coordinate facenti parte della mossa corrente.
Nell'esempio, si sta indicando la volontà di prendere dalla plancia di gioco le carte oggetto nella riga 3, colonne 2, 3, 4 e di inserirle nella propria libreria alla colonna 1.
Si noti che l'ordine delle coordinate (che in questo caso differiscono solo per la coordinata y) è vincolante, relativamente all'inserimento nella libreria del giocatore.

1.3 Carattere generale

1.3.1 Invio della libreria personale

Viene allegata al messaggio la serializzazione JSON della libreria personale del giocatore.
Il client fa uso di un modello ridotto utile alla sola visualizzazione a video.

Formalizzazione

```
{
  "status": "SendingShelf",
  "message": "{
  "cards": [
    [
      [
        {
          "type": "CATS",
          "image": 2
        }
      ],
      [
        {
          "type": "BOOKS",
          "image": 3
        }
      ], ...
    ]
  ]
  }"
}
```

Dove sono state mostrate due celle di una libreria personale esemplificativa.

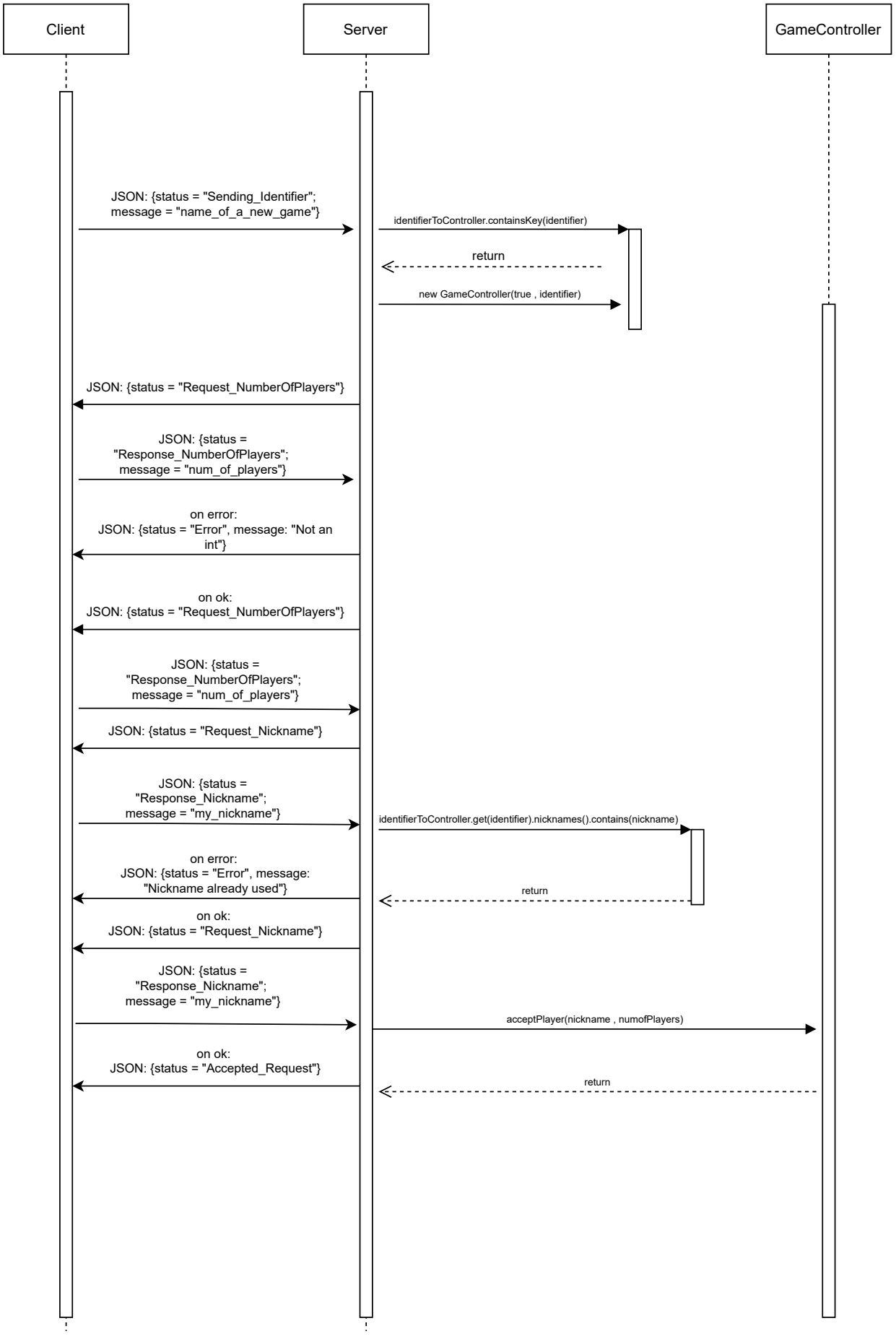
1.4 Note conclusive

Si noti che per i **motivi di sintesi** legati alla peer review, sono stati riportati **solamente alcuni** messaggi di stato esemplificativi della struttura generale.
L'uso di strutture semplificate, lato client, obbliga a dover completare la struttura di informazioni che per sua natura il client non deve conoscere, quali i riferimenti alla plancia di gioco, alla libreria e alle carte di gioco.
Ciò avviene attraverso l'uso di classi `Serializers`, che hanno il compito di identificare tali parametri e completarli con i riferimenti che il server fornisce, ottenendo la struttura completa.
Ad esempio: in una mossa, il client comunica al server solo le coordinate da cui vuole prendere un gruppo di carte oggetto e la colonna in cui vuole posizionarle all'interno della propria libreria.
Il server, per completare tale mossa, dovrà essere a conoscenza dei riferimenti di plancia di gioco e libreria, che verranno completati all'atto della deserializzazione.

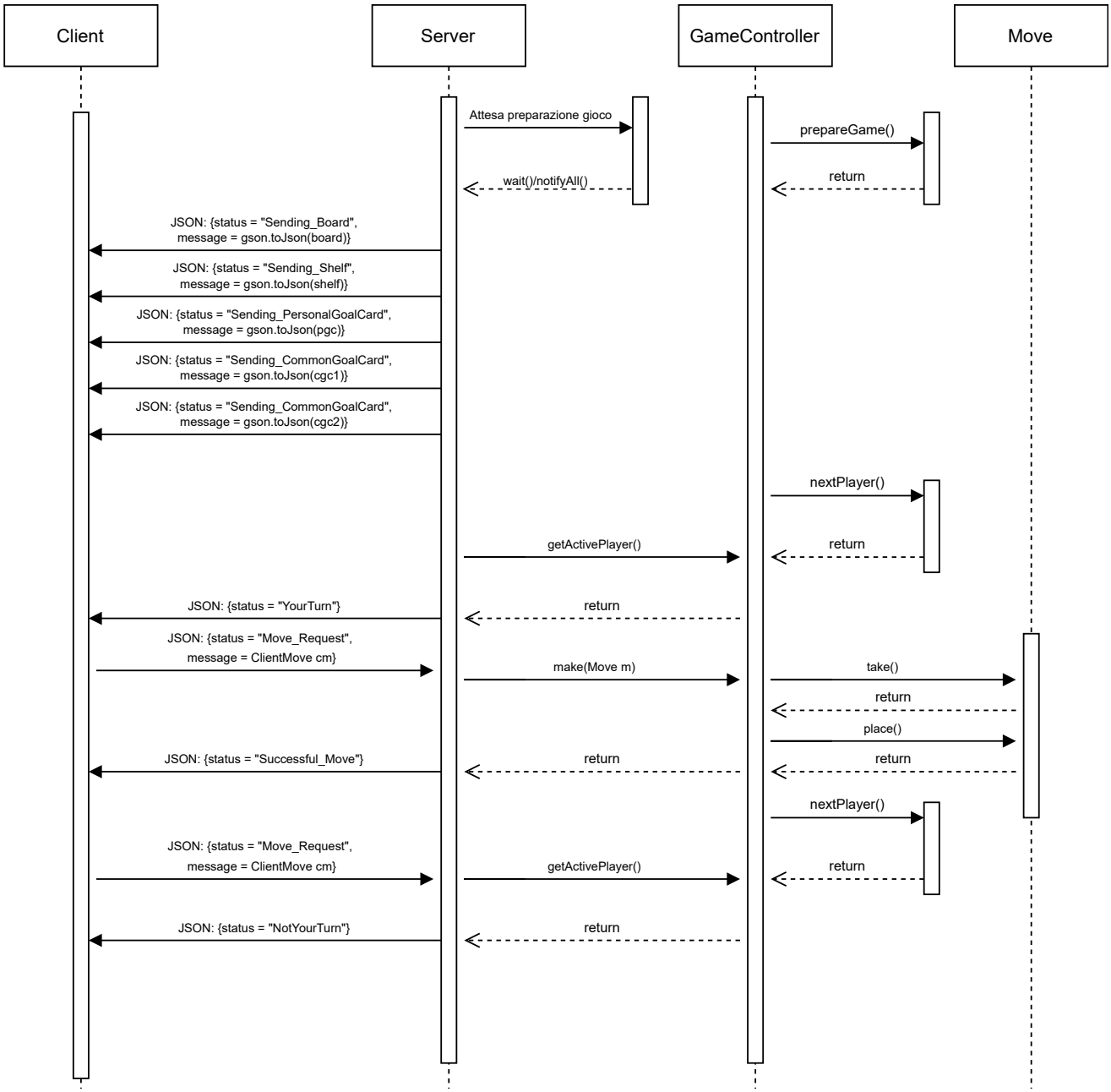
I tipi di dato `Optional` inoltre, essendo di recente introduzione, hanno difficoltà ad essere correttamente serializzati (anche in RMI).
Anche questo problema è risolto da classi appositamente costruite di `Serializers` e `Deserializers`.

1.5 Diagrammi di sequenza

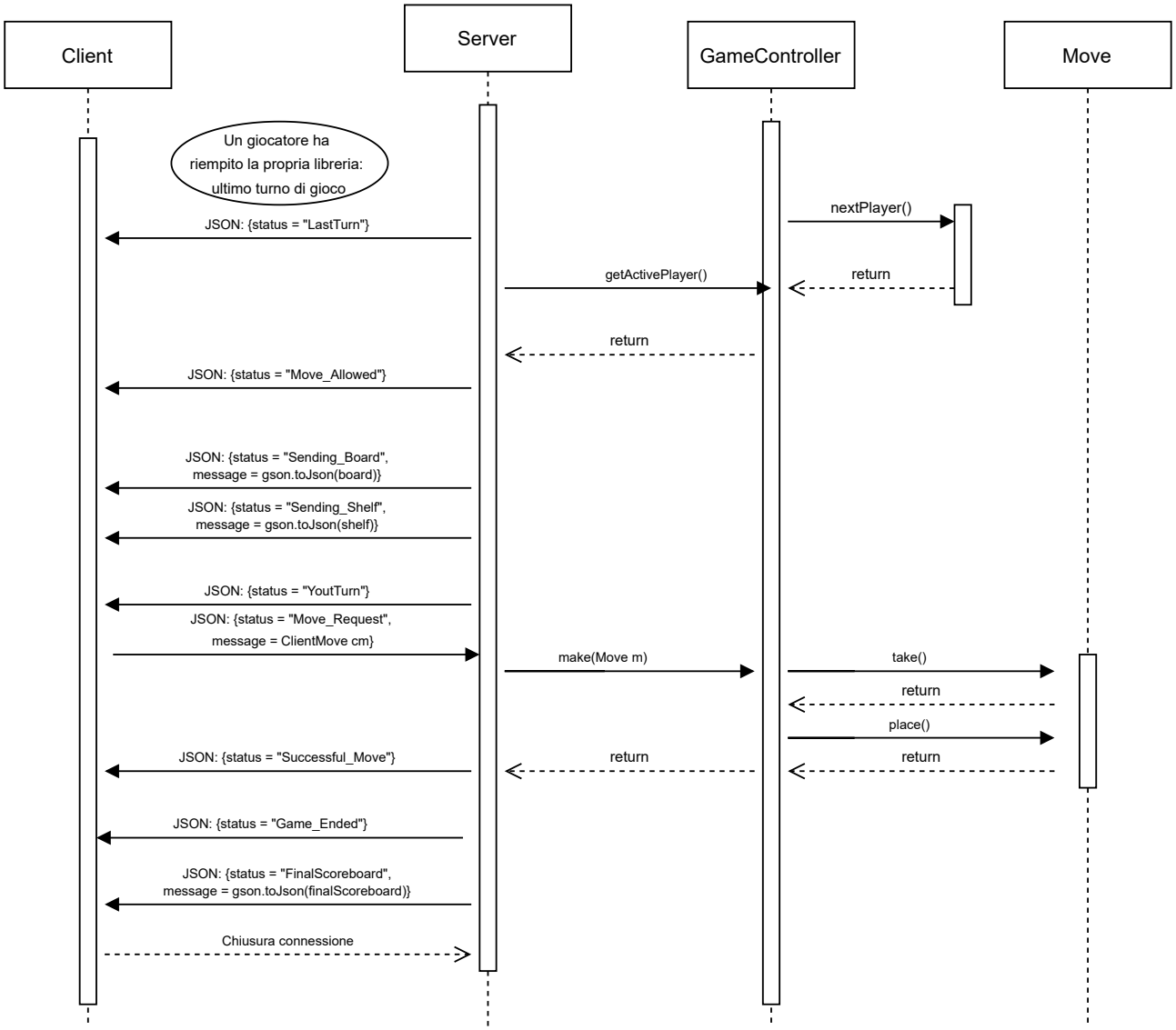
1.5.1 Accesso, selezione e ammissione al gioco



1.5.2 Preparazione e svolgimento del turno



1.5.3 Conclusione del gioco e formulazione della classifica finale



2 RMI

La struttura predisposta per RMI adotta numerose interfacce, attraverso cui il client fa uso dei metodi preposti alla gestione del gioco sul server.

Sono previste classi di `callback`, intermediari attraverso cui il client fornisce al server il proprio riferimento remoto, in modo che possa interagirvi, fornendogli i riferimenti alle interfacce che ha il diritto di utilizzare.

Sono inoltre previsti "classi di stato", che raccolgono i riferimenti a tali oggetti remoti e li mettono a disposizione del client.

2.1 Interfacce

Di seguito, **alcune** delle interfacce che il client ha a disposizione per controllare l'andamento del gioco sul server (esso implementa, naturalmente, controlli di validità).
Tali interfacce dovranno, idealmente, comparire in ogni implementazione del client in RMI, che non ha dunque bisogno di conoscere i dettagli implementativi del modello.

2.1.1 Plancia di gioco

```java
package it.polimi.ingsw;

import java.rmi.Remote;

public interface BoardInterface extends Remote {
    /**
     * Maximum X dimension of the board
     */
    public static final int BOARD_DIM_X = 9;
    /**
     * Maximum Y dimension of the board
     */
    public static final int BOARD_DIM_Y = 9;

    /**
     * Checks if a space is usable given coordinates
     *
     * @param x coordinate
     * @param y coordinate
     * @return usability of that space
     * @throws Exception if parameters are trying to go out of the board
     */
    public boolean isSpaceUsable(int x, int y) throws Exception;

    /**
     * Returns Object Card's type ordinal, given a Board Space
     *
     * @param x coordinate
     * @param y coordinate
     * @return the card's ordinal
     * @throws Exception if parameters are trying to go out of the board
     */
    public int getCardOrdinalFromSpace(int x, int y) throws Exception;

    /**
     * Returns Object Card's image path, given a Board Space
```

```
37          *
38          * @param x coordinate
39          * @param y coordinate
40          * @return the card's image path
41          * @throws Exception if parameters are trying to go out of the board
42          */
43         public String getCardImageFromSpace(int x, int y) throws Exception;
44     }
```

2.1.2 Stato del client

Aggregato: contiene i riferimenti alle interfacce cui il client ha diritto di accesso durante il gioco.
Si noti che il client ha accesso alle sole interfacce e non conosce i dettagli di implementazione del modello.

Per favorire la leggibilità, si omette la JavaDoc di alcuni metodi (la cui funzione è ritenuta chiara dal contesto).

```
1   package it.polimi.ingsw;
2
3   import java.rmi.Remote;
4   import java.rmi.RemoteException;
5   import java.util.List;
6
7   public interface ClientStatusInterface extends Remote {
8       public Status getStatus() throws RemoteException;
9
10      public boolean setStatus(Status status) throws RemoteException;
11
12      public String getIdentifier() throws RemoteException;
13
14      public void setIdentifier(String identifier) throws RemoteException;
15
16      public String getNickname() throws RemoteException;
17
18      public void setNickname(String nickname) throws RemoteException;
19
20      /**
21       * Aggregate setter for Game initial parameters
22       *
23       * @param board Game's board
24       * @param shelf Player's shelf
25       * @param mi    Move Intermediate for the client
26       */
27      public void setGameParameters(BoardInterface board, ShelfInterface shelf,
        ↪  MoveIntermediateInterface mi) throws RemoteException;
28
29      public BoardInterface getBoard() throws RemoteException;
30
31      public ShelfInterface getShelf() throws RemoteException;
32
33      public PersonalGoalCardInterface getPersonalGoalCard() throws RemoteException;
34
35      public List<CommonGoalCardInterface> getCommonGoalCard() throws RemoteException;
36
```

```
37      public void setCards(PersonalGoalCardInterface pgCard,
        ↪  List<CommonGoalCardInterface> cgCard) throws RemoteException;
38
39      public MoveIntermediateInterface getMoveIntermediate() throws RemoteException;
40
41      public String getCurrentPlayer() throws RemoteException;
42
43      public void setCurrentPlayer(String currentPlayer) throws RemoteException;
44
45      public ScoreBoardInterface getScoreBoard() throws RemoteException;
46
47      public void setScoreBoard(ScoreBoardInterface sci) throws RemoteException;
48  }
```

2.1.3 Intermediario dello stato

Interfaccia di `callback` per il server.
Il client fornisce al server un riferimento alla propria implementazione del gestore di stato, che si occupa di far evolvere il client.
Il server usa tale riferimento per comunicare l'evoluzione al client ed invitarlo a svolgere le azioni di gestione delle fasi di gioco.

```
1   package it.polimi.ingsw;
2
3   import java.rmi.Remote;
4   import java.rmi.RemoteException;
5
6   public interface StatusIntermediateInterface extends Remote {
7       /**
8        * Setter for Client Status Object
9        * It makes the server manage Game's evolution
10       *
11       * @param csi Interface for the Client Status
12       * @throws RemoteException      related to RMI
13       * @throws InterruptedException related to Thread management
14       */
15      public void setIntermediate(ClientStatusInterface csi) throws RemoteException,
        ↪  InterruptedException;
16  }
```

2.1.4 RMI Controller

Livello di astrazione al controller: fornisce al client i metodi necessari per la gestione delle fasi iniziali (ammissione e creazione nuove partite).
Si rende necessario per la gestione di partite multiple.
Sono implementati stringenti controlli di integrità relativamente all'accesso al Controller e al Modello sottostanti.

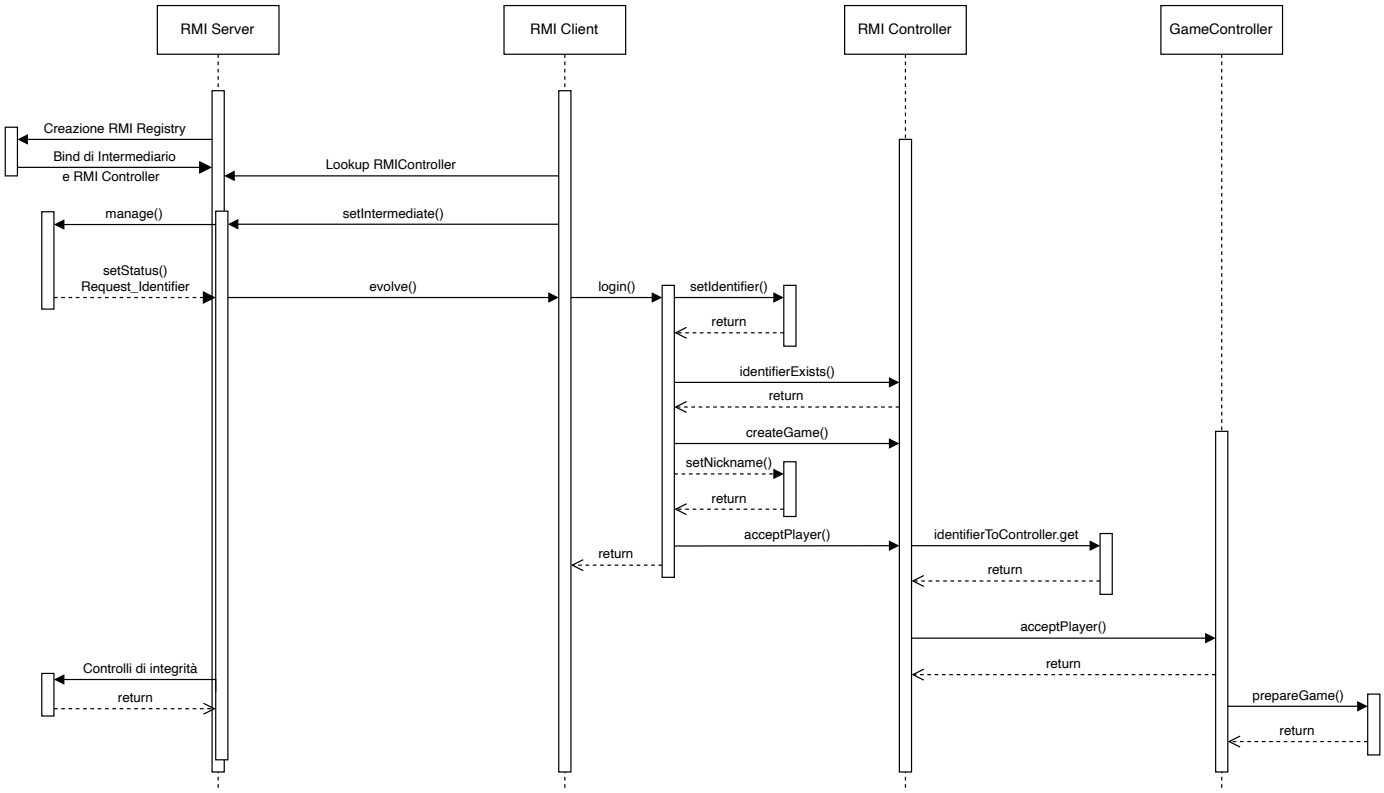
```java
package it.polimi.ingsw;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMIControllerInterface extends Remote {
    /**
     * @param identifier Game's identifier
     * @return whether the identifier exists
     */
    public boolean identifierExists(String identifier) throws RemoteException;

    /**
     * @param identifier Game's identifier
     * @param nickname   Player's nickname
     * @return whether the nickname exists in the given Game's identifier
     */
    public boolean nicknameExists(String identifier, String nickname) throws
    ↪  RemoteException;

    /**
     * Creates a game using the corresponding identifier
     *
     * @param identifier Game's identifier
     * @return true if the Game have been correctly created
     */
    public boolean createGame(String identifier) throws RemoteException;

    /**
     * Admission phase
     *
     * @param identifier Game's identifier
     * @param nickname   Player's nickname
     * @param maxPlayers Maximum number of players for the game
     *                   used only if the first player is being accepted to the game.
     *                   It is ignored otherwise.
     * @return true if the admission process have been correctly completed
     * @throws Exception related to Model management
     */
    public boolean acceptPlayer(String identifier, String nickname, int maxPlayers)
    ↪  throws Exception;
}
```

2.1.5 Note conclusive

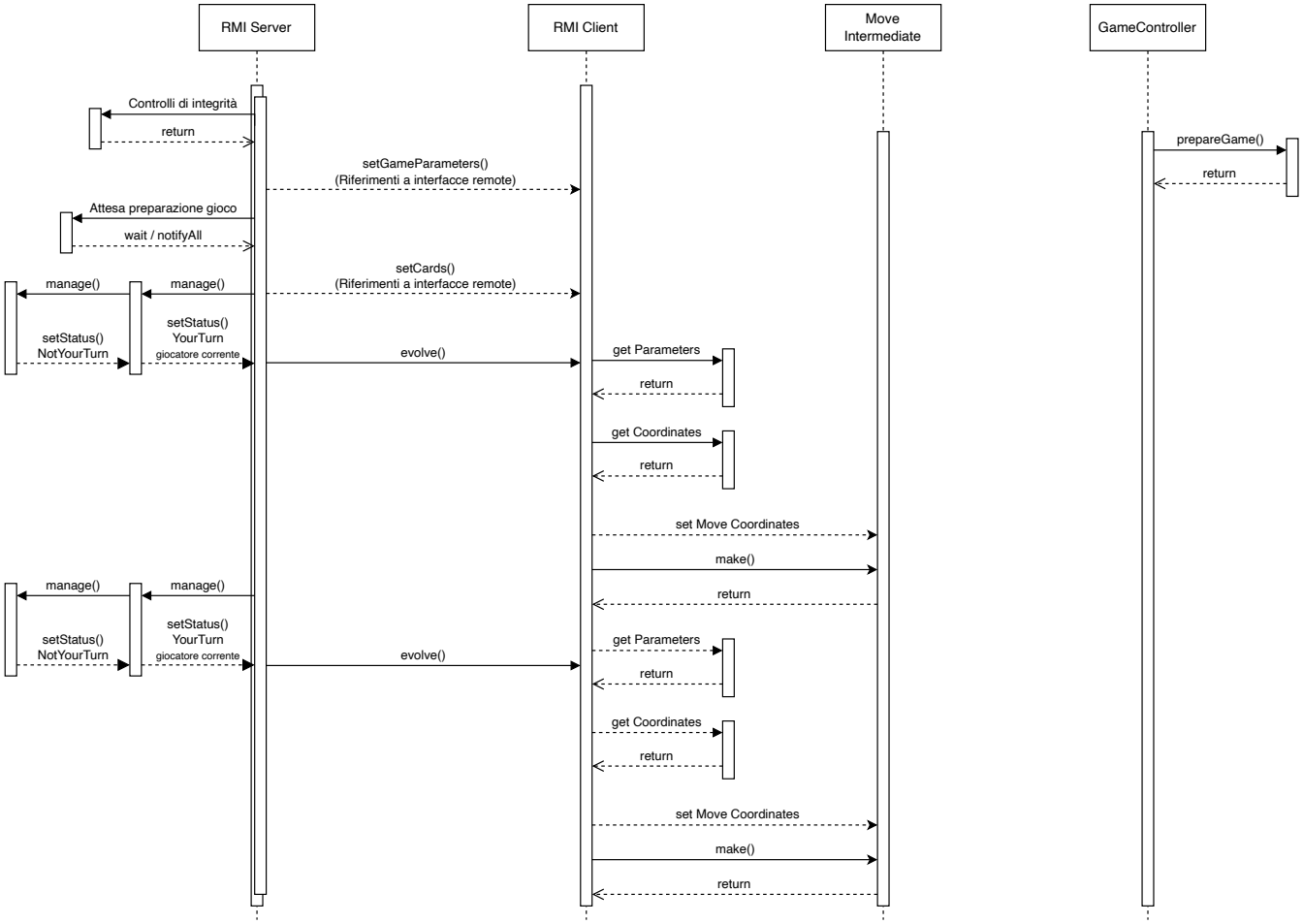
Si noti che anche per RMI, sono stati riportati **solamente alcune** interfacce significative, ad esemplificazione della struttura generale.

2.2 Diagrammi di sequenza

2.2.1 Accesso, selezione e ammissione al gioco



2.2.2 Due mosse di esempio



2.2.3 Conclusione del gioco e formulazione della classifica finale