Prova Finale (Progetto) di Ingegneria del Software - A.A. 2022/23
Documento di specifica del protocollo di comunicazione

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.

Nota bene: al fine di facilitarne la lettura, gli scambi di messaggi presentati nel seguito sono scritti su più righe. In fase di implementazione, questi vengono inviati su una sola riga, in formato JSON.

Sono definiti i messaggi seguenti.

1.1 Accesso al gioco

1.1.1 Identificativo del gioco

Scambio dell'identificativo del gioco di cui il client desidera entrare a far parte.

Formalizzazione

```
{
  "status": "Sending_Identifier",
  "message": "Game's name"
}
```

Dove Game's name rappresenta il nome del gioco di cui il client desidera di entrare a far parte.

1.1.2 Numero di giocatori

Scambio del numero di giocatori di una nuova partita. Il messaggio è per sua natura opzionale.

Richiesta

```
{
  "status": "Request_NumberOfPlayers"
}
```

Risposta

```
{
  "status": "Response_NumberOfPlayers",
  "message": "4"
}
```

1.1.3 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.

1.1.4 Richiesta negata

La richiesta di ammissione alla partita non è stata accettata. Rivedere l'integrità dei parametri trasmessi.

Formalizzazione

```
{
  "status": "Denied_Request",
  "message": "Nickname is already in use"
}
```

Dove è stata motivata la negata accettazione.

1.1.5 Richiesta accettata

La richiesta di ammissione alla partita è stata accettata.

Formalizzazione

```
{
  "status": "Accepted_Request"
}
```

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.

1.2.3 Mossa andata a buon fine

La mossa richiesta è andata a buon fine.

Formalizzazione

```
{
  "status": "SuccessfulMove"
}
```

1.2.4 Mossa invalida

La mossa richiesta era invalida e non è stata eseguita.

Formalizzazione

```
{
  "status": "FailedMove"
}
```

1.2.5 Ultimo turno

Messaggio broadcast inviato a tutti i client, una volta completata una mossa.
Il turno corrente viene ultimato, chi ha già eseguito la propria mossa ha concluso la partita.

Formalizzazione

```
{
  "status": "LastTurn"
}
```

1.2.6 Mossa permessa

A seguire il messaggio di ultimo turno, il server informa il giocatore sulla possibilità di dover concludere l'ultimo turno con un'altra mossa o meno.
Il messaggio sarà seguito da un ordinario scambio caratterizzante un turno di gioco.

Formalizzazione

```
{
  "status": "MoveAllowed"
}
```

1.2.7 Mossa non permessa

A seguire il messaggio di ultimo turno, il server informa il giocatore sulla possibilità di dover concludere l'ultimo turno con un'altra mossa o meno.
Il giocatore si trovava dunque in una posizione precedente al giocatore che ha ottenuto la carta di fine gioco.

Formalizzazione

```
{
  "status": "MoveNotAllowed"
}
```

1.2.8 La partita è finita

Alla conclusione dell'ultimo turno, il messaggio informa i client sulla fine del gioco.

Formalizzazione

```
{
  "status": "GameEnded"
}
```

1.2.9 Conclusione forzata

Un giocatore si è disconnesso, i client vengono informati di dover concludere forzatamente la partita.

Formalizzazione

```
{
  "status": "ForcedGameEnd"
}
```

1.3 Carattere generale

1.3.1 Fine di sequenza

Messaggio che identifica la fine di una sequenza di linee di lunghezza arbitraria.

Formalizzazione

```
{
  "status": "EOS"
}
```

1.3.2 Invio della plancia di gioco

Viene allegata al messaggio la serializzazione JSON della plancia di gioco.
Il client fa uso di un modello ridotto utile alla sola visualizzazione a video.

Formalizzazione

```
{
  "status": "SendingBoard",
  "message": "{
  "spaces": [
    [
      {
        "x": 0,
        "y": 0,
        "usable": false,
        "card": [
          null
        ],
        "dots": [
          null
        ]
      },
        ...
        {
        "x": 2,
        "y": 3,
        "usable": true,
        "card": [
          {
            "type": "GAMES",
            "image": 2
          }
        ],
        "dots": [
          null
        ]
      },
        ...
    }
  ]
  }"
}
```

Dove sono stati mostrati due spazi di una plancia di gioco esemplificativa.

1.3.3 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.

1.3.4 Invio della carta obiettivo personale

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

Formalizzazione

```
{
  "status": "SendingPersonalGoalCard",
  "message": "{
  "type": 1,
  "pattern": [
    [
      [
        "PLANTS"
      ],
      [
        null
      ],
      [
        "FRAMES"
      ],
      [
        null
```

```
      ],
      [
        null
      ]
    ],
    ...
  ]
  }"
}
```

Dove è stata utilizzata come esempio la Carta obiettivo personale n. 1 (in riferimento all'ordinamento delle risorse fornite dal produttore).

1.3.5 Specifica della carta obiettivo comune

Viene allegata al messaggio la specifica della prima/seconda carta obiettivo comune.
Il client fa uso di un modello ridotto utile alla sola visualizzazione a video.

Formalizzazione

```
{
  "status": "SendingCommonGoalCardSpecification",
  "message": "{
  "type": 5,
  "description": "Three columns each formed by 6 cards of maximum three different
  ↪  types. One column can show a different combination of another column."
  }"
}
```

Dove è stata utilizzata come esempio la Carta obiettivo comune n. 5 (in riferimento all'ordinamento delle risorse fornite dal produttore).

1.3.6 Invio della classifica finale

Viene allegata al messaggio la serializzazione JSON della classifica finale.
Il client fa uso di un modello ridotto utile alla sola visualizzazione a video.

Formalizzazione

```
{
  "status": "FinalScoreboard",
  "message": "{
  "scoreBoard": {
    "Player 4": 40,
    "Player 3": 30,
    "Player 2": 20,
    "Player 1": 10
  }
  }"
}
```

1.3.7 Stato non valido

Stato di default.

Formalizzazione

```
{
  "status": "InvalidStatus"
}
```

1.3.8 Errore

Errore, tipicamente generato da un'eccezione.

Formalizzazione

```
{
  "status": "Error",
  "message": "Reason of the error"
}
```

Dove Reason of the error è la ragione dell'errore, comunicata dal server sottoforma di messaggio testuale.

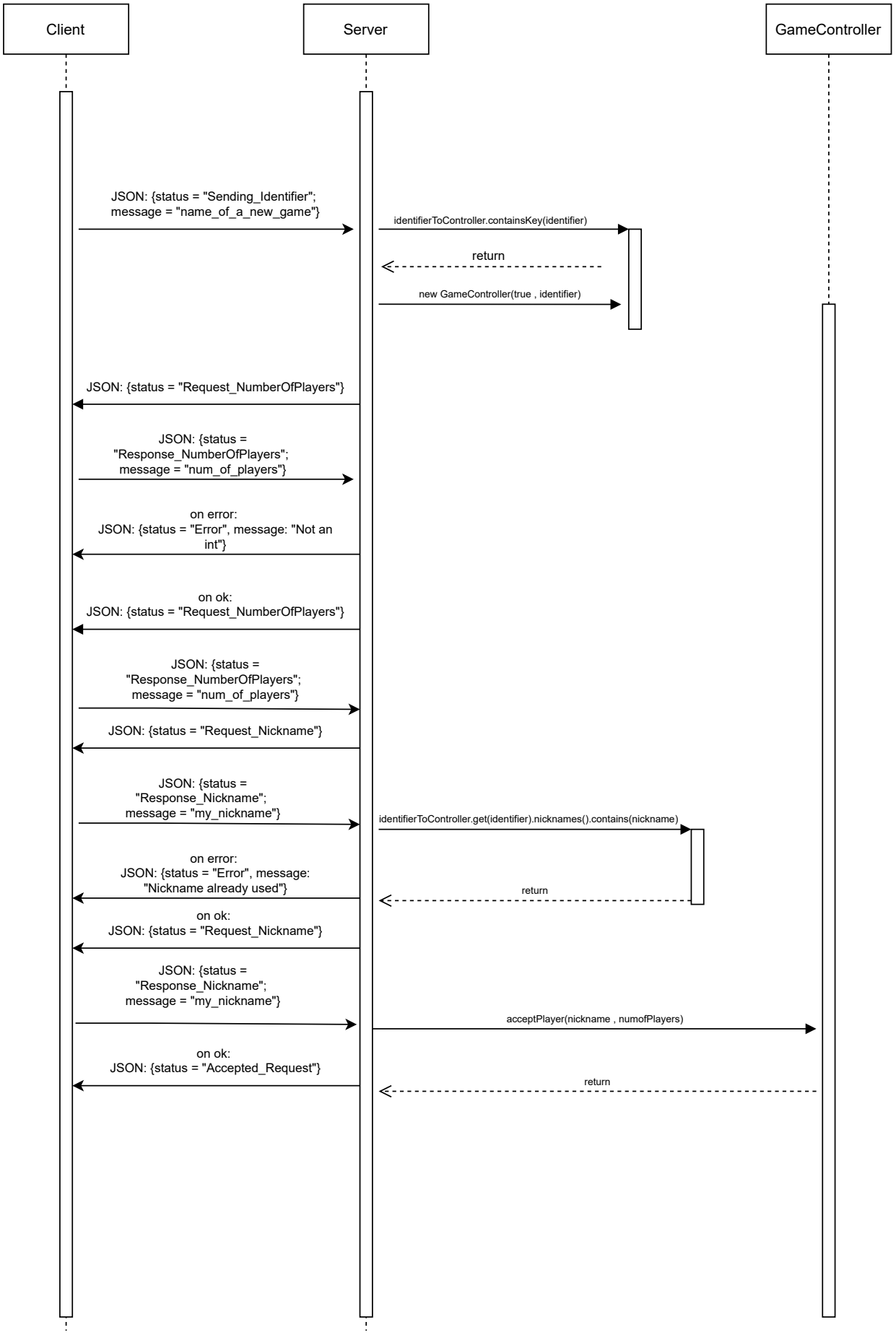
1.4 Note conclusive

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.

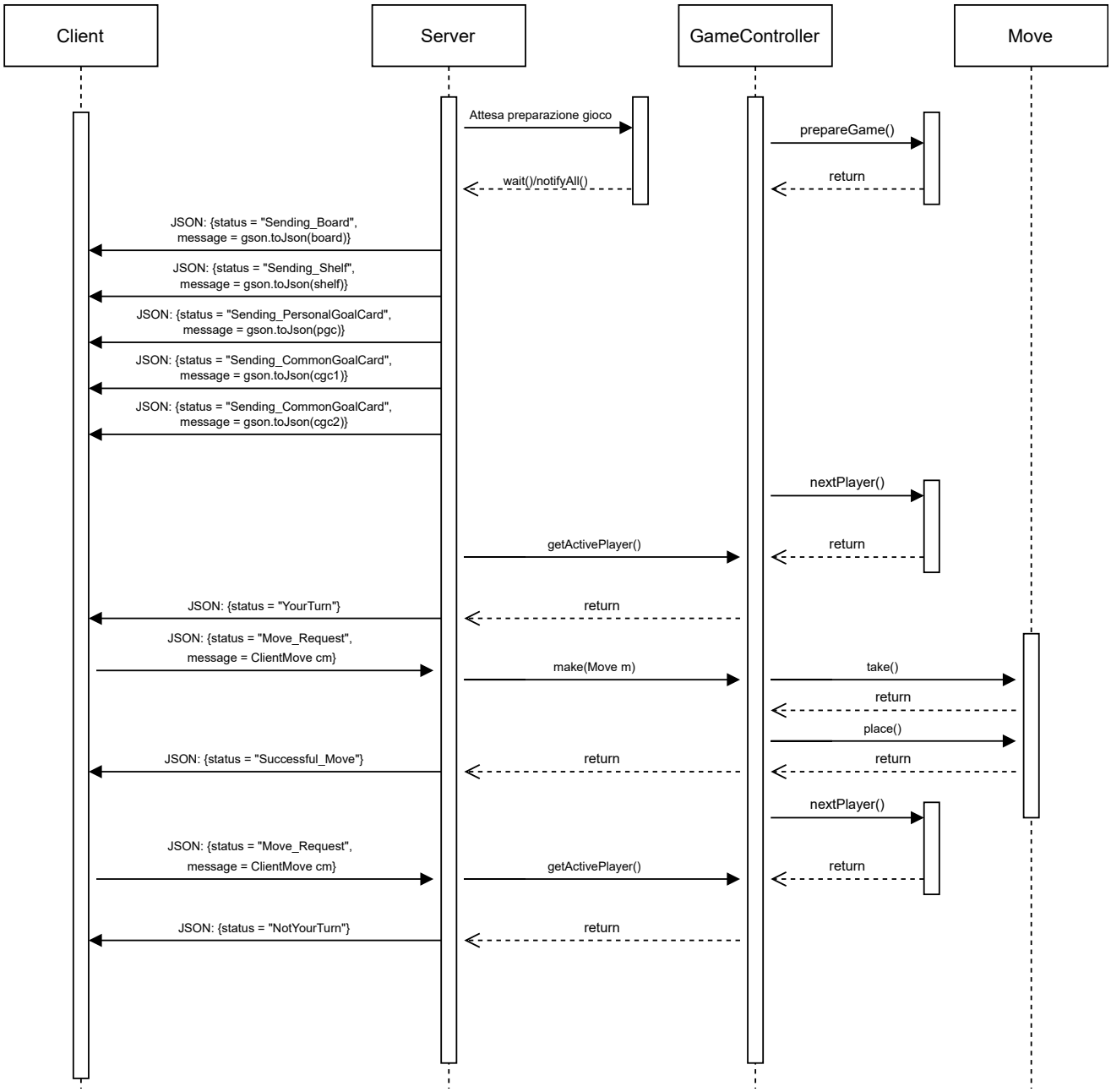
I tipi di dato `Optional` inoltre, essendo di recente introduzione, hanno difficoltà ad essere correttamente serializzati (anche in RMI).
Anche questo problema è risolto da classi appositamente costruite di `Serializers` e `Deserializers`.

1.5 Diagrammi di sequenza

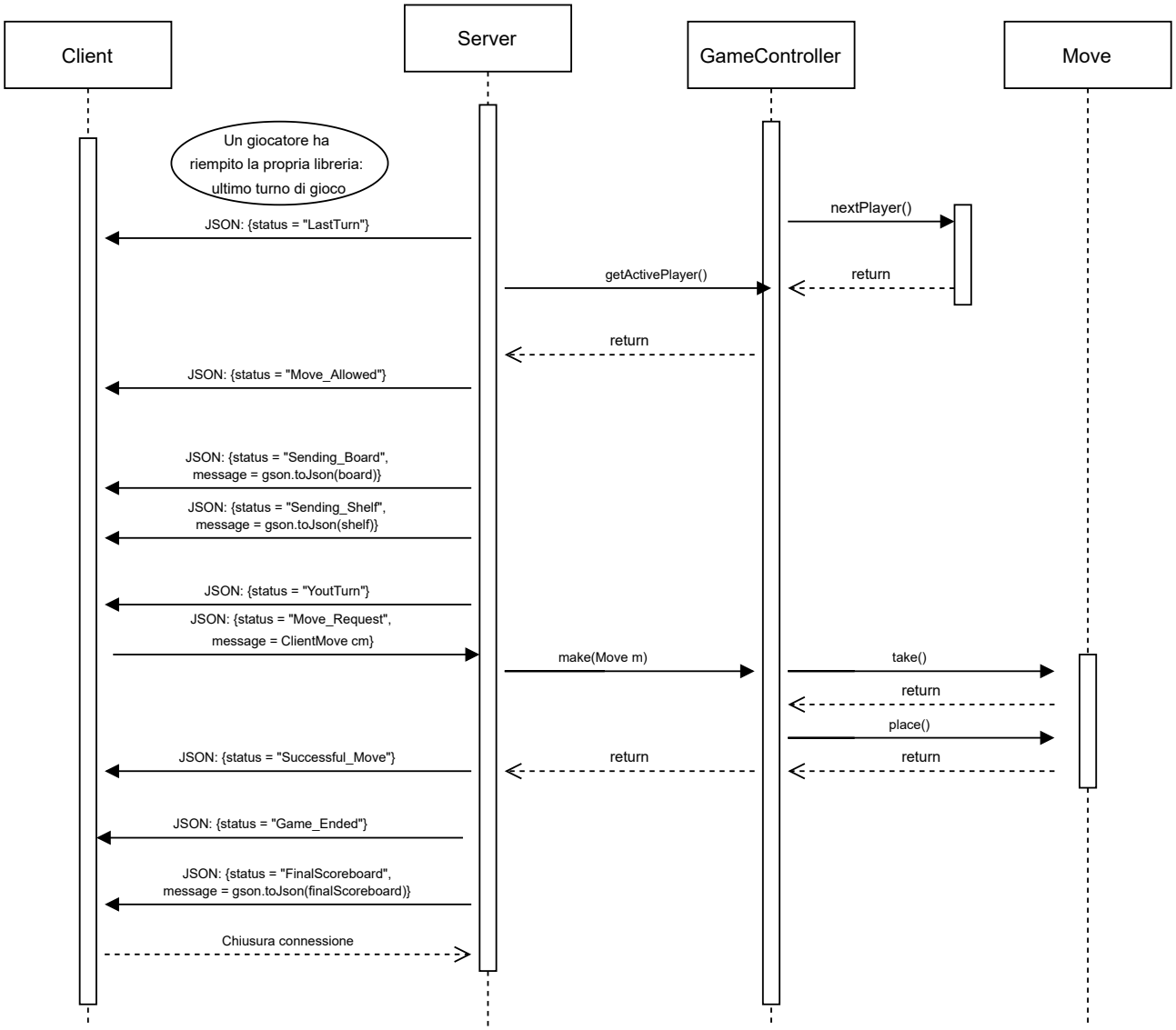
1.5.1 Accesso, selezione e ammissione al gioco



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, le 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 Libreria personale

```java
package it.polimi.ingsw;

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

public interface ShelfInterface extends Remote {
    /**
     * Maximum X dimension of the shelf
     */
    public static final int SHELF_DIM_X = 6;
    /**
     * Maximum Y dimension of the shelf
     */
    public static final int SHELF_DIM_Y = 5;

    /**
     * Returns Object Card's type ordinal in position (x, y) if valid (else -1)
     *
     * @param x coordinate
     * @param y coordinate
     * @return requested Object Card's type ordinal
     * @throws Exception if coordinates are invalid
     */
    public int getCardOrdinal(int x, int y) throws Exception;

    /**
     * Returns Object Card's image path in position (x, y) if valid (else null)
     *
     * @param x coordinate
     * @param y coordinate
     * @return the card's image path
     * @throws Exception if parameters are trying to go out of the board
     */
    public String getCardImage(int x, int y) throws Exception;

    /**
     * @return true if the Shelf is full
     */
    public boolean isFull() throws RemoteException;
}
```

2.1.3 Carta oggetto

```java
package it.polimi.ingsw;

import java.rmi.Remote;

public interface ObjectCardInterface extends Remote {
    /**
     * This method associates an int to a letter referred to an ObjectCardType
     *
     * @param c char read from file
     * @return an int
     * @throws Exception when char doesn't match any first letter of the
     ↪  ObjectCardTypes
     */
    public static int getOrdinal(char c) throws Exception {
        switch (c) {
            case 'C' -> {
                return 0;
            }
            case 'B' -> {
                return 1;
            }
            case 'G' -> {
                return 2;
            }
            case 'F' -> {
                return 3;
            }
            case 'T' -> {
                return 4;
            }
            case 'P' -> {
                return 5;
            }
        }
        throw new Exception();
    }

    /**
     * Returns a char corresponding to the ordinal of a given card type.
     *
     * @param ordinal position in ObjectCardType enumeration
     * @return the corresponding char
     * @throws Exception if ordinal given does not correspond to an ObjectCardType
     */
    public static char getChar(int ordinal) throws Exception {
        switch (ordinal) {
            case 0 -> {
                return 'C';
            }
            case 1 -> {
                return 'B';
            }
```

```
52              case 2 -> {
53                  return 'G';
54              }
55              case 3 -> {
56                  return 'F';
57              }
58              case 4 -> {
59                  return 'T';
60              }
61              case 5 -> {
62                  return 'P';
63              }
64          }
65          throw new Exception();
66      }
67  }
```

2.1.4 Carta obiettivo personale

```java
package it.polimi.ingsw;

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

public interface PersonalGoalCardInterface extends Remote {
    /**
     * Limit of Personal Goal Cards.
     */
    public static final int LIMIT = 12;

    /**
     * Getter method for Personal Goal Card
     *
     * @return type of the card
     */
    public int getType() throws RemoteException;

    /**
     * getter method
     *
     * @param x coordinate
     * @param y coordinate
     * @return ordinal of ObjectCardType or -1 if the card is empty
     * @throws Exception if coordinates are invalid
     */
    public int getOrdinal(int x, int y) throws Exception;
}
```

2.1.5 Carta obiettivo comune

```java
package it.polimi.ingsw;

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

public interface CommonGoalCardInterface extends Remote {
    /**
     * Limit of Common Goal Cards.
     */
    public static final int LIMIT = 12;

    /**
     * This method returns the textual description of the pattern that the player has to achieve, in order
     * to gain points.
     *
     * @return String: textual description of the pattern
     */
    public String getDescription() throws RemoteException;
}
```

2.1.6 Intermediario delle mosse

Per ogni client, esso conosce i riferimenti al suo Controller di gioco e alla relativa plancia e carte obiettivo comune e personale, nonché alla libreria personale del giocatore.

```java
package it.polimi.ingsw;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface MoveIntermediateInterface extends Remote {
    /**
     * Move parameters
     *
     * @param x      list of coordinates (x)
     * @param y      list of coordinates (y)
     * @param column in the Player's shelf
     */
    public void setParameters(List<Integer> x, List<Integer> y, int column) throws
    ↪  RemoteException;

    /**
     * Move's creation and confirmation
     *
     * @return validity of the move
     * @throws Exception related to Shelf and Board management
     */
    public boolean make() throws Exception;

    /**
     * @return validity of the last move
     */
    public boolean isLastMoveValidated() throws RemoteException;
}
```

2.1.7 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).

```java
package it.polimi.ingsw;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface ClientStatusInterface extends Remote {
    public Status getStatus() throws RemoteException;

    public boolean setStatus(Status status) throws RemoteException;

    public String getIdentifier() throws RemoteException;

    public void setIdentifier(String identifier) throws RemoteException;

    public String getNickname() throws RemoteException;

    public void setNickname(String nickname) throws RemoteException;

    /**
     * Aggregate setter for Game initial parameters
     *
     * @param board Game's board
     * @param shelf Player's shelf
     * @param mi    Move Intermediate for the client
     */
    public void setGameParameters(BoardInterface board, ShelfInterface shelf,
    ↪  MoveIntermediateInterface mi) throws RemoteException;

    public BoardInterface getBoard() throws RemoteException;

    public ShelfInterface getShelf() throws RemoteException;

    public PersonalGoalCardInterface getPersonalGoalCard() throws RemoteException;

    public List<CommonGoalCardInterface> getCommonGoalCard() throws RemoteException;

    public void setCards(PersonalGoalCardInterface pgCard,
    ↪  List<CommonGoalCardInterface> cgCard) throws RemoteException;

    public MoveIntermediateInterface getMoveIntermediate() throws RemoteException;

    public String getCurrentPlayer() throws RemoteException;

    public void setCurrentPlayer(String currentPlayer) throws RemoteException;

    public ScoreBoardInterface getScoreBoard() throws RemoteException;
```

```
46
47      public void setScoreBoard(ScoreBoardInterface sci) throws RemoteException;
48  }
```

2.1.8 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.

```java
package it.polimi.ingsw;

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

public interface StatusIntermediateInterface extends Remote {
    /**
     * Setter for Client Status Object
     * It makes the server manage Game's evolution
     *
     * @param csi Interface for the Client Status
     * @throws RemoteException      related to RMI
     * @throws InterruptedException related to Thread management
     */
    public void setIntermediate(ClientStatusInterface csi) throws RemoteException,
    ↪  InterruptedException;
}
```

2.1.9 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;
}
```

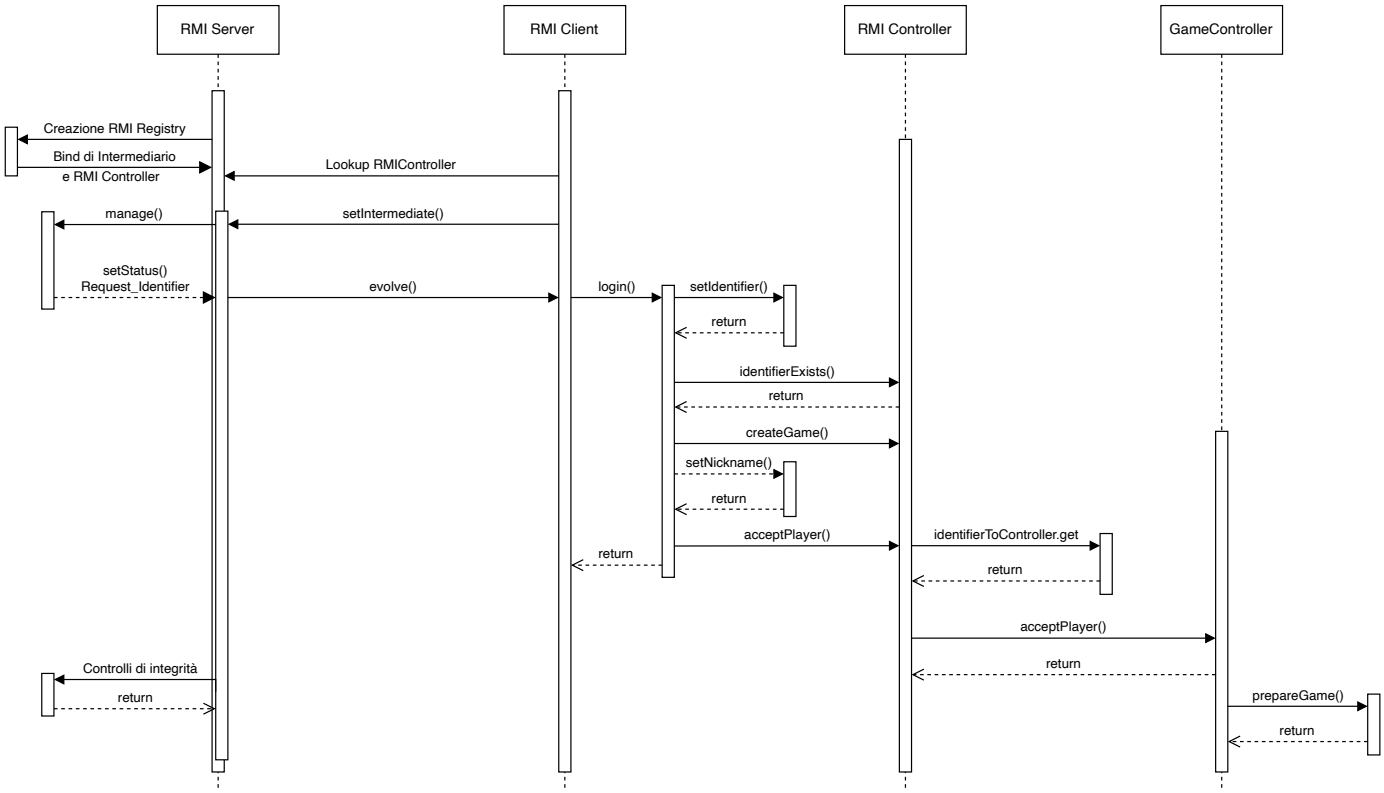
2.1.10 Classifica finale

Mappa ordinata che associa i giocatori (con il loro nickname) al loro punteggio.

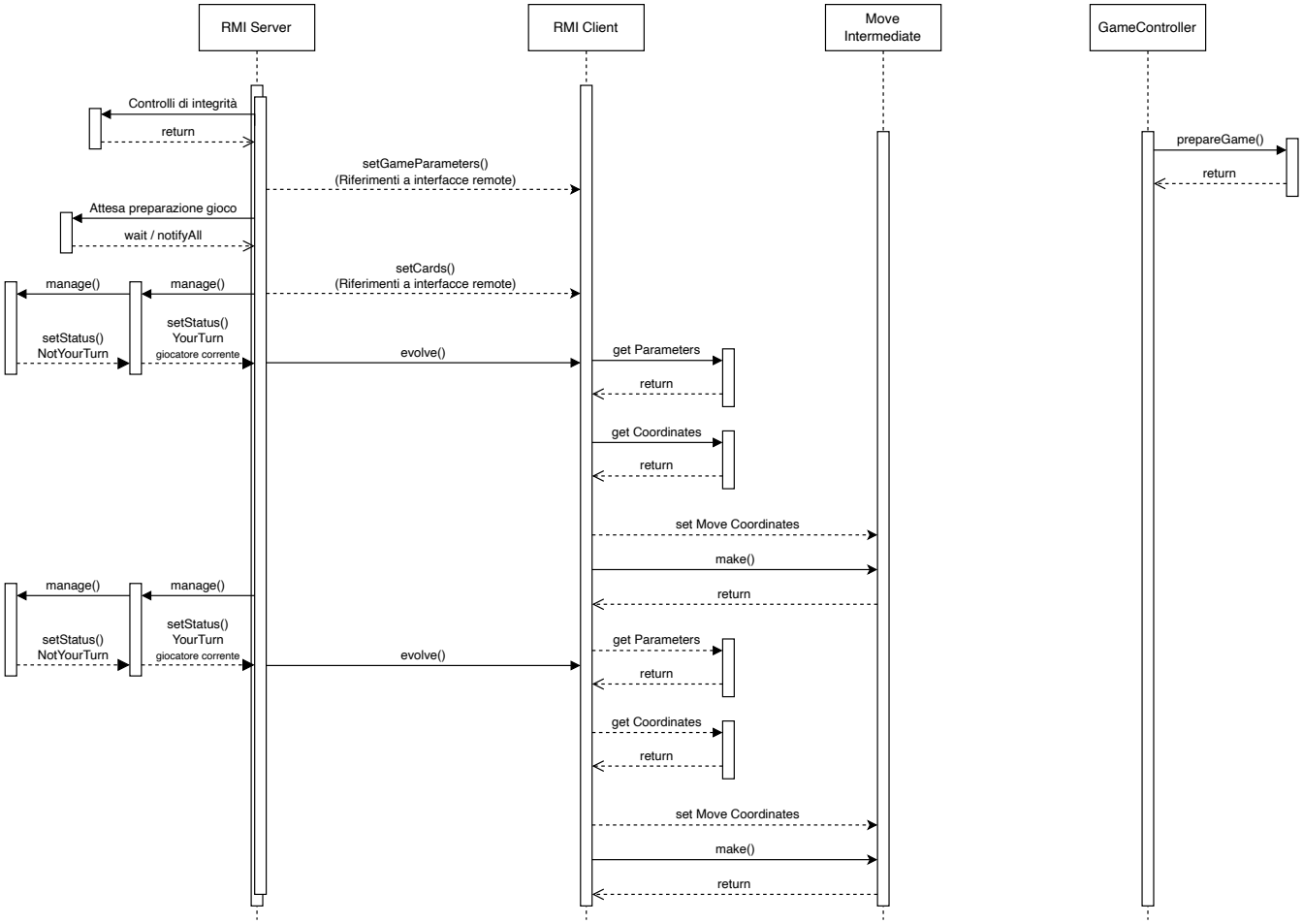
```java
package it.polimi.ingsw;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Map;

public interface ScoreBoardInterface extends Remote {
    /**
     * Getter for the final Scoreboard sorted Map
     *
     * @return final Scoreboard
     */
    public Map<String, Integer> getScoreBoard() throws RemoteException;
}
```

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