Communication Protocol

Group ID: AM13 - A.Y. 2020/21

This document's aim is to provide a clear overview of Maestri del Rinascimento project's communication protocol. Most relevant and crucial exchanges are going to be highlighted here. All the messages follow the JSON format shown below:

```
{
    "messageID" : "TAG",
    "payload" : "{message content for the receiver}"
}
```

The `messageID` gives information about the type of event that has arisen in the sender, the `payload` field adds useful information for the receiver in order for it to react correctly.

When an event occurs inside the ClientView, commands will be created based on the messageID (the effective type of the event), each specific command knows how to serialize the content of the payload so that it can encapsulate everything inside a MessageEnvelope which will be sent, through the outputStream, to the Server.

Once the server receives the MessageEnvelope, through a MessageAllocator it maps the messageID to the corresponding Java Message Class so that it can be deserialized correctly into the message object, which will contain all the necessary information for the Model's public methods' calls.

Table of Contents

Initialization

1. Player registration

The connection phase is executed upon the players' insertion of their IP address and port. They will contact the server on the default port (which can be configured through the config.json file), that has the purpose of listening for new incoming players and directing them to a new port that will be dinamically assigned to them for the whole game duration. This expedient has been thought as a way to allow the server to host multiple games at the same time.

The client app contains some logic for the correct game initialization. Players have to insert their nickname and choose the type of game they wish to play (*singleplayer mode* or *multiplayer mode*) after the server checks if the player was already part of a game, otherwise the player is reconnected to that game. The initial registration is as follows:

```
// Message: client -> server
{
    "messageID" : "REGISTER_SINGLE",
    "payload" : {
    "nickname" : "Gatto"
    }
}

// Message: server -> client
{
    "messageID" : "SERVER_STATE",
    "payload" : {
        "accepted" : true,
        "starting" : true,
        "wasInGame" : false
    }
}
```

If the player wasn't already part of a game, he will be given the choice of SinglePlayer or MultiPlayer game. For the MultiPlayer game, more choices have to be made:

- **(A)** Create a new multiplayer game and specify how large they wish the lobby to be;
- **(B)** Join an already existing game on hold for other players.

Case (A):

```
// Message: client -> server
{
    "messageID" : "REGISTER_MULTI",
    "payload" : {
        "nickname" : "Gatto",
        "nPlayers" : 4
    }
}
```

Players will wait until the the lobby is full. The lobby creator will be the first player once the game starts.

Case (B):

```
// Message: client -> server
{
    "messageID" : "REQUEST_LOBBY",
    "payload" : {}
}

// Message: server -> client
{
    "messageID" : "SHOW_LOBBY",
    "payload" : {
        {
            "idLobby" : 1,
            "nPlayersWaiting": 2,
            "totPlayers": 4
        },

         ...

        {
            "idLobby": n,
            "nPlayersWaiting": 1,
            "totPlayers": 3
        }
    }
}

// Message: client -> server
{
    "messageID" : "REGISTER_MULTI",
    "payload" : {
        "nickname" : "Gatto",
        "idLobby" : 1,
    }
}
```

Where `idLobby` indicates which lobby the player wishes to join among those available sent by server.

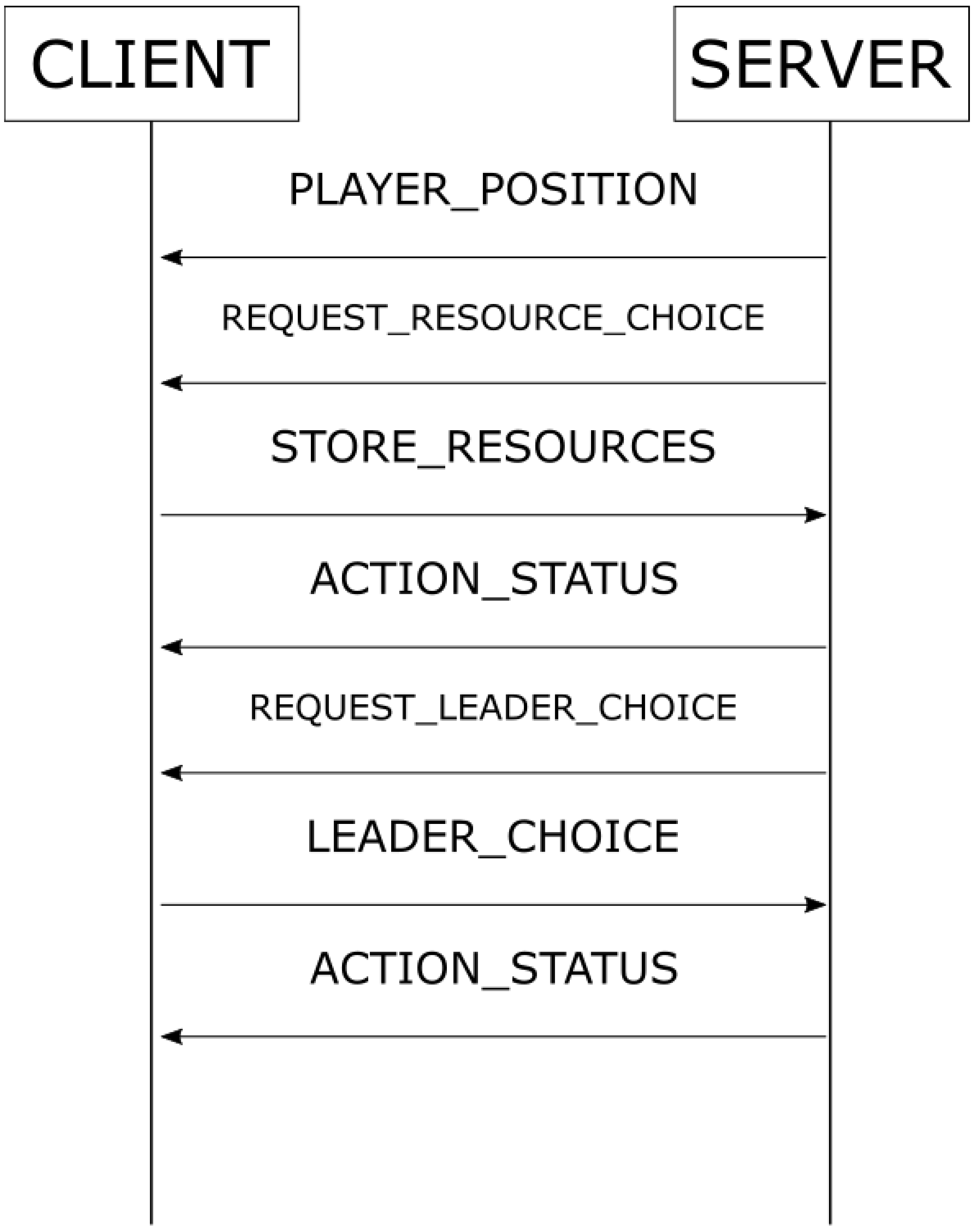
Server response for accepting the player:

```
// Message: server -> client
{
    "messageID" : "SERVER_STATE",
    "payload" : "{
        "accepted" : true,
        "starting" : false
    }"
}
```

Mind that, in case **(B)**, `accepted` could be false if someone else simultaneously filled the last free spot. In that case, the player should choose another lobby.

Once a lobby is full, the server sends a `SERVER_STATE` message to all clients waiting for that lobby, stating `"starting" : true`.

2. Game initiation



There are a few things to do upon game initiation:

- Update each player's view with their initial position on the faith track based on their turn order, with messages like the following:

```
// Message: server -> client
{
    "messageID" : "PLAYER_POSITION",
    "payload" : {
        "player" : 1,
        "boardCell" : 2
    }
}
```

- Request players' preferred resources based on their turn order, with messages like the following:

```
// Message: server -> client
{
    "messageID" : "REQUEST_RESOURCE_CHOICE",
    "payload" : {
        "quantity": 2
    }
}

// Message: client -> server
{
    "messageID" : "STORE_RESOURCES",
    "payload" : {
        {
            "resource": "SHIELD",
            "position": "SmallWarehouse"
        },
        {
            "resource": "COIN",
            "position": "MediumW"
        }
    }
}

// Message: server -> client
{
    "messageID" : "ACTION_STATUS",
    "payload" : {
        "accepted": true
    }
}
```

Every value and faithPoint assignment follows the ruleset in the table below:

Player	Resources of your choosing	Faith Points
1st	0	0
2nd	1	0
3rd	1	1
4th	2	1

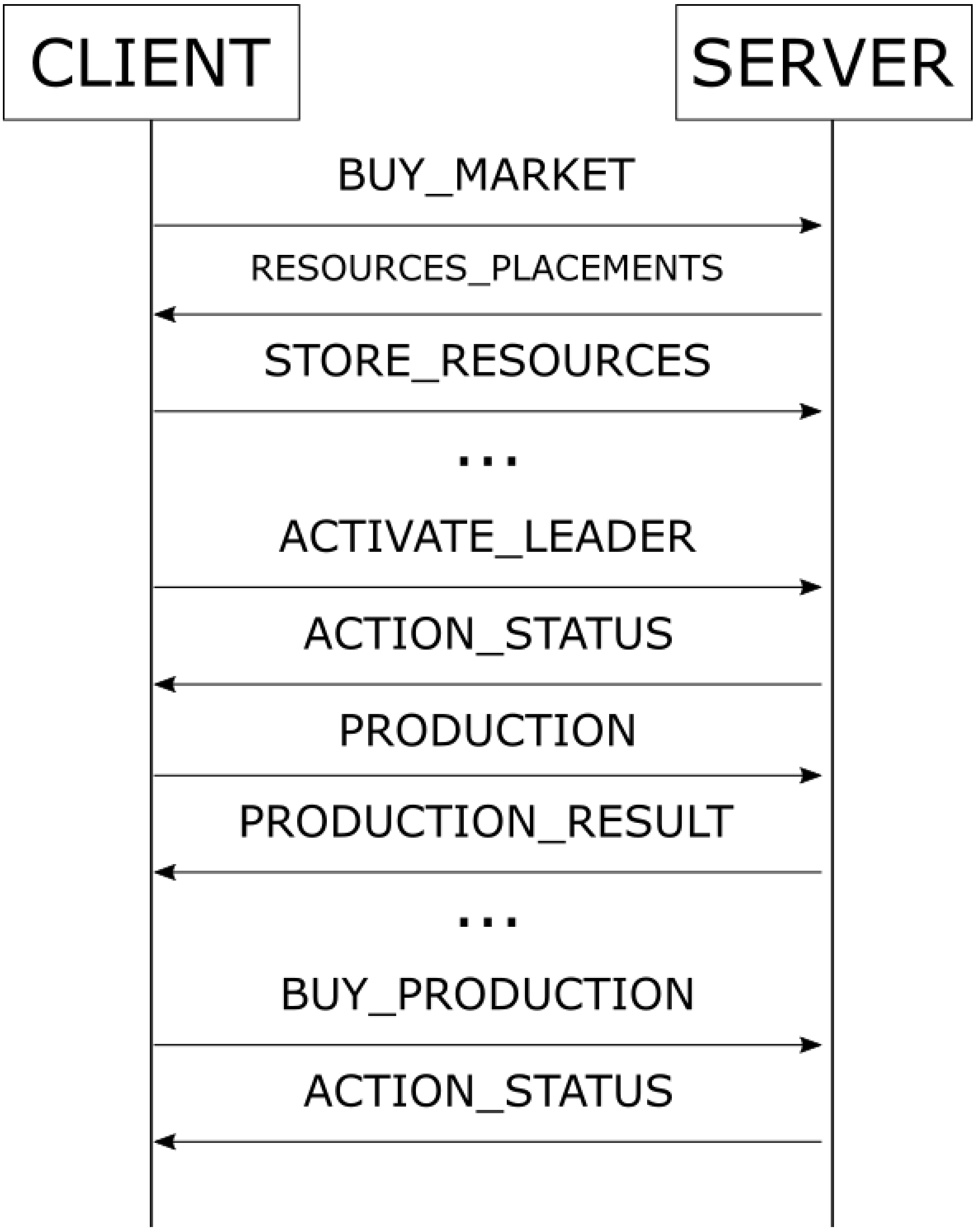
- Each player needs to choose 2 out of their 4 given LeaderCards, the messages will look as such:

```
// Message: server -> client
{
    "messageID" : "REQUEST_LEADER_CHOICE",
    "payload" : {
        "leaderCards" : [3, 4, 6, 8]
    }
}

// Message: client -> server
{
    "messageID" : "LEADER_CHOICE",
    "payload" : {
        "leaderCardsToKeep" : "[4, 8]"
    }
}

// Message: server -> client
{
    "messageID" : "ACTION_STATUS",
    "payload" : {
        "accepted": true
    }
}
```

Mid-game messages



3. Server to Client Messages

3.1 Client View Update

The following messages are sent to each player in the game, because everyone has to see the changes in the model caused by other players as well:

Current player notification

```
{
    "messageID" : “CURRENT_PLAYER",
    "payload" : {
        "player" : 1
    }
}
```

This incapsulate the player's id that has to play.

Progression onto the *Faith Track*

```
{
    "messageID" : “PLAYER_POSITION",
    "payload" : {
        "player" : 1,
        "boardCell" : 17
    }
}
```

- `player` specifies which player has gone forward onto the *Faith Track*. The id refers to the player's turn. This message can be caused by the current player's production phase
  if some faith points are generated, or from someone else discarding resources;
- `boardCell` is the updated position on the player's board.

Purchase from Market

```
{
    "messageID" : "MARKET_UPDATE",
    "payload" : {
        "dimension" : "column/row",
        "index" : 2,
        "changes" : ["BLANK", "FAITH", "SHIELD"],
        "extra" : "COIN"
    }
}
```

- `dimension` and `index` are the coordinates on the market board where some changes have been applied because somobody has bought resources from there.
- new setup of market (only changes notification).

Purchase of Production Cards

```
{
    "messageID" : "AVAILABLE_PRODUCTION_CARDS",
    "payload" : {
        {
            "deck" : 1,
            "cardID" : 3
        },

        ...

        {
            "deck" : 12,
            "cardID" : 3
        }
    }
}
```

The payload displays all 12 decks containing the currently available production cards. A player can buy only the card on top of all the others belonging to the same deck, so the view can just show the 12 available ones. If a deck is empty, then `"cardID" : 0`.

- `deck` indicates the deck among the (initial) 12 available;
- `cardID` is the ID of the card to show on that deck.

End game

```
{
    "messageID" : "END_GAME",
    "payload" : {
        "winner" : 1
    }
}
```

In a singleplayer game, if Lorenzo wins, then `"winner" : 0`.

3.2 Requests for current player turn

Production

```
{
    "messageID" : "PRODUCTION_RESULT",
    "payload" : {
        "state" : boolean,
        "outcome" : {"SHIELD", "STONE", "SHIELD"}
    }
}
```

- `state` signals the success of the production asked by the player;
- `outcome` is the actual production that will be put in the player's lootchest; if state is negative, then this will be empty.

Resources placement

```
{
    "messageID" : "RESOURCES_PLACEMENTS",
    "payload" : {}
}
```

The server notifies the players that they have to place the resources that they previously acquired, for example, from the market.

4. Client to Server Messages

Leader card activation

```
{
  "messageID" : "ACTIVATE_LEADER",
  "payload" : {
       "leaderCard" : id
  }
}
```

Production

```
{
    "messageID" : "PRODUCTION",
    "payload" : {
        "productionCards" : [id1, id2...],
        "fromWarehouse" : {"STONE"},
        "fromLootchest" : {"SHIELD", "STONE"},
        "fromExtraStorage1" : {"SERVANT"},
        "fromExtraStorage2" : {},
        "boostAbilityCards" : [id1, id2],
        "outputLeader" : [out1, out2],
        "basicProduction" : boolean,
        "outputBasic" : "COIN"
    }
}
```

- `productionCards` contains the cards that the user wants to use for production.
- `fromWarehouse` indicates the resources that should be taken from the warehouse.
- `fromLootchest` indicates the resources that should be taken from the lootchest.
- `fromExtraStorage1` and `fromExtraStorage2` indicate the resource that should be taken from the extra storage supplied by an active leader card of StorageAbility.
- `boostAbilityCards` contains the leader cards of type BoostAbility that will be used during production.
- `outputLeader` contains all the BoostAbility leaderCards outputs of player's choice as a list of resources.
- `basicProduction` indicates if the user would like to use the standard production given by the game board.

- `outputBasic` indicates the type of Resource the user wants to receive from basicProduction.

Buy from Market

```
{
    "messageID" : "BUY_MARKET",
    "payload" : {
        "info" : {
            "dimension" : "row",
            "index" : 2
        },
        {
        "leaders" : [
            {
                "marbleLeader" : id,
                "quantity" : 1
            },
            {
                "marbleLeader" : id,
                "quantity" : 2
            }
        ]
    }
}
```

`leaders` contains the information regarding active MarbleAbility leader cards
which the player would like to use upon the specified quantity of
blank marbles collected from the market.

Buy Production Cards

```
{
    "messageID" : "BUY_PRODUCTION",
    "payload" : {
        "prodCards" : [id1, id2, id3],
        "fromWarehouse" : ["SHIELD"],
        "fromLootchest" : ["COIN", "COIN"],
        "fromExtraStorage1" : [],
        "fromExtraStorage2" : [],
        "discountAbility" : [id1, id2]
    }
}
```

- `prodCards` contains the cards that the user wants to buy.
- `fromWarehouse` indicates the resources that should be taken from the warehouse.
- `fromLootchest` indicates the resources that should be taken from the lootchest.
- `fromExtraStorage1` and `fromExtraStorage2` indicate the resource that should be taken from the extra storage supplied by an active leader card of StorageAbility.

- `discountAbility` indicates the leader cards of Discount type that the player wants to use during this phase.