**NETWORK PROTOCOL**

**Networking architecture requisites:**

* A ServerApp listening and accepting inbound connections. Accepted connections will be transferred to a newly instantiated ClientHandler network object.
* Many ClientHandlers, one for each connection/client.
* Many Clients capable of talking with the ClientHandlers.

**Opening connection:**

A user, interacting with the client, can either:

* Ask to **create** a new Lobby (specifying his nickname, and the lobby size)
* Ask to **join** an existing Lobby (specifying his *preferred* nickname, and the lobby number)
* Ask to **rejoin** a lobby, in case he got disconnected (specifying his previous nickname and the lobby number)

All of these 3 initial actions are done by:

* Opening a connection with the server <ip,port>
* Sending a opening message:
  + MSG\_CREATE\_LOBBY ( nickname, lobbySize )
  + MSG\_JOIN\_LOBBY ( nickname, lobbyNumber )
  + MSG\_REJOIN\_LOBBY ( nickname, lobbyNumber )

The ClientHandler, who just received one of these 3 above messages, performs some checks, and gives a response to the Client:

* If everything gone well, sends an OK message.
  + MSG\_OK\_CREATE ( lobbyNumber )
  + MSG\_OK\_JOIN ( assigned Nickname )
  + MSG\_OK\_REJOIN ( assigned Nickname )
* If something bad happens, a MSG\_Error is sent, and the Client goes back into MainMenu.

If an OK message was sent from the server, the Client enters the GamePhase and waits for a MSG\_UPD\_Full message. This message is able to completely setup/restore his Simplified Model.

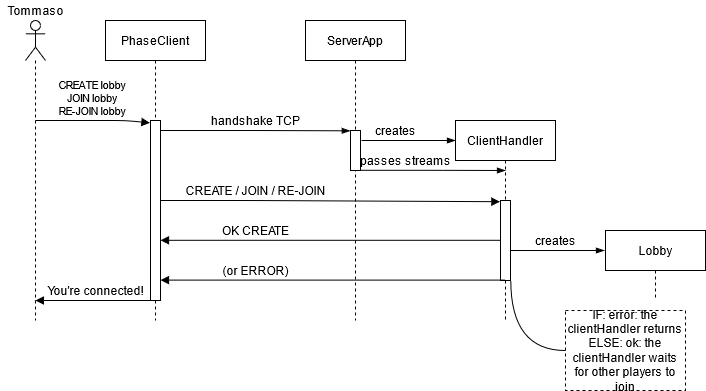


Fig. 1 – enstablishing connection

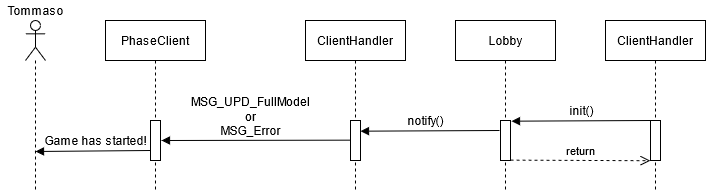


Fig. 2 – Lobby/game initialization(). Some other ClientHandler invoked the Lobby.init(), which prompts the other waiting ClientHandlers to transmit a MSG\_UPD\_Full message or an Error message.

**GamePhase – action forwarding**

In a standard game phase (suppose the user is able to make an action, because it is its turn), the player is able to create and send a MSG\_ACTION. The messages could be:

* MSG\_INIT\_ChooseLeaderCards ( list of LeaderCards )
* MSG\_INIT\_ChooseResource ( chosen Resource )
* MSG\_ACTION\_ACTIVATE\_LEADERCARD ( number of the LeaderCard )
* MSG\_ACTION\_DISCARD\_LEADERCARD ( number of the LeaderCard )
* MSG\_ACTION\_BUY\_DEVELOPMENT\_CARD ( )
* MSG\_ACTION\_CHOOSE\_DEVELOPMENT\_CARD ( chosen Card, chosen Slot )
* MSG\_ACTION\_GET\_MARKET\_RESOURCES ( row/col, number )
* MSG\_ACTION\_MARKET\_CHOICE ( choice )
* MSG\_ACTION\_CHANGE\_DEPOT\_CONFIG ( depot configuration )
* MSG\_ACTION\_ENDTURN

The behavior of the actions is described in another Document. We’ll focus on the messages generated and used in the networking layer.

Note that the actions are **Boolean methods**. They may return either true or false. We’ll discuss its usage in a moment.

The user sent a MSG\_ACTION. The ClientHandler received it, passed it to the Lobby, and the Lobby passed it to the Common Filter of the Controller. The controller modifies the model and then returns.

A number of messages are generated from the model and captured by the ClientHandlers, which are observing the Model Objects:

* MSG\_UPD\_DevDeck
* MSG\_UPD\_FaithTrack
* MSG\_UPD\_Game
* MSG\_UPD\_Market
* MSG\_UPD\_Player
  + MSG\_UPD\_DevSlot
  + MSG\_UPD\_Extradepot
  + MSG\_UPD\_Strongbox
  + MSG\_UPD\_WarehouseDepot
* MSG\_UPD\_DevCardsVendor
* MSG\_UPD\_LeaderBoard
* MSG\_UPD\_LeaderCardsObject
* MSG\_UPD\_MarketHelper
* MSG\_UPD\_ResourceObject

All these messages are directly forwarded to the Clients, using the update( message ) method of the ClientHandlers.

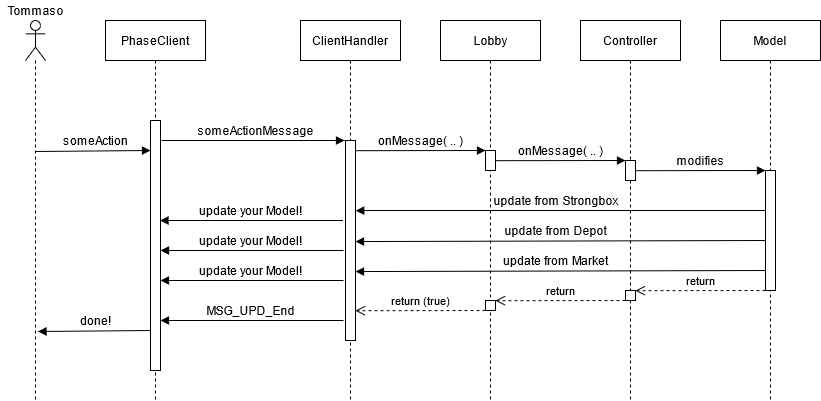


Fig. 3 – example of client-server standard interaction

The network Protocol defined requires that **after many update messages**, a MSG\_UPD\_End is sent. The Clients, upon receival of this message, are able to “close” the updating phase. The Clients are able to update their status, and may become the current Player.

On the other hand, there are situations in which a MSG\_UPD\_End is not required.

* A MSG\_ERROR is not followed by a MSG\_UPD\_End
* A MSG\_NOTIFICATION is not required to be followed by a MSG\_UPD\_End. It may be sent inside a updating cycle, or outside one. (check disconnection document)
* A MSG\_UPD\_LeaderBoard is not followed by anything
* Initial messages (such as CREATE LOBBY) are not updating messages, so they are not followed by an MSG\_UPD\_End.

The ClientHandler knows when to send this last MSG\_UPD\_End message, because the Controller tells him so. Every action of the actionMessage follows a standard and can either return true or false:

* True: add the MSG\_UPD\_End
* False: don’t

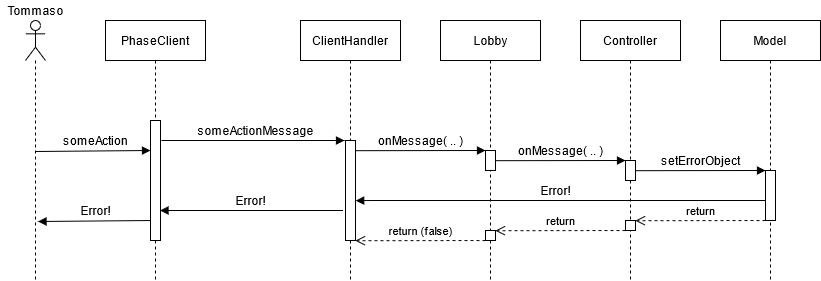


Fig. 4 – Error example

**Closing Phase**

The last message being sent by the server is a MSG\_UPD\_Leaderboard.

The ClientHandler, upon generating that message, understands that the game is over and begins the procedure to eliminate the lobby, the ClientHandlers, the connections.

The Client, upon receiving that message, terminates his connection to the server.

**What if – a player is not the current Player?**

if a player is not the current Player his Client is unable to generate Action messages. It can only receive update messages from the server.

The ClientHandler is still listening for his Client Messages. If he receives a message from the Client he handles, but his player is not the currentPlayer, the packet is simply dropped.

**What if – someone wanted to implement a game chat?**

With this network architecture it is very easy:

* We just need a new type of message ( like, MSG\_CHAT ( string text ) )
* The Clients needs a new routine that generate this message, in a manner very similar to other messages
* The ClientHandler is still listening to his Client. He can receive the message and call a service method proponed by the Controller, using the messageHelper.
* The messageHelper generates a MSG\_NOTIFICATION message containing the custom text.
* Every other Client can receive this message without anything disrupting going on.