Ingegneria Del Software - Prof. Alessandro Margara
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Santorini XML Communication Protocol

Piersilvio De Bartolomeis
Marco Di Gennaro
Alessandro Di Maio
(Gruppo AM10)

July 3, 2020

Contents

1. Introduction

In this document, we present the client-server communication protocol used in our implementation of Santorini game. Our choice was to develop our own XML protocol for this project. This choice is due to the desire to abstract the protocol from the language with which Santorini is developing and the willingness to test, during development, every single package sent or received.
The decision to take the path of a thin client approach has led, in addition to client request and server response messages, to design messages from server to client to start every stage of the game.
As required our server is a multi-client server, so each client is managed by a dedicated thread. We've also implemented the advanced feature "multi-game". To do this we designed a system based on lobbies, where a single lobby allows us to manage a single match. When a user connects to the server he is addressed by the lobbies manager to the first free lobby.

As for the messages designed, our protocol is based on four different messages:

- (Client Side) toSendRequest: the client sends this message to request something (like login, move, build, etc.)
- (Server Side) toSendAnswer: the server sends this message back to the client that made the request, it can be either accepted or rejected
- (Server Side) updateMsg: the server sends this message back to all the other clients connected
- (Server Side) toDoMsg: the server sends this message to the client notifying him of the action required from his side

The next two chapters explain deeper how we use the previous messages and how, in every step of the game, the communication protocol works.

2. Ping

During the implementation of our communication protocol it was chosen to implement a ping mechanism between client and server and vice versa.

2.1 Server side

On the server side we implemented a ping message that the server sends to the client with the frequency of 1.5 seconds. The server socket has a timeout of 3 seconds, beyond which, if it does not receive messages, it disconnects the client. This timeout will, in case of correct operation, reset by a ping message from the client which works in the same way as the one just described.

2.2 Client side

On the client side we implemented a ping message that the client sends to the server with the frequency of 1.5 seconds. The client socket has a timeout of 3 seconds, beyond which, if it does not receive messages, it disconnects from the server. This timeout will, in case of correct operation, reset by a ping message from the server which works in the same way as the one just described.

3. Setup

3.1 Login

During the login phase the Server asks the client to login (using a toDoMsg message), the client sends a login request (which includes the client's username) and the server accepts or rejects it based on its content, alerting the client with an answer (that contains, if the request has been accepted, the other users' username there are into the same lobby). At last, the server notifies all the other clients that a new client has logged in if the request is successful. If the request has been accepted, the server randomizes the workers' color to assign to the new user.

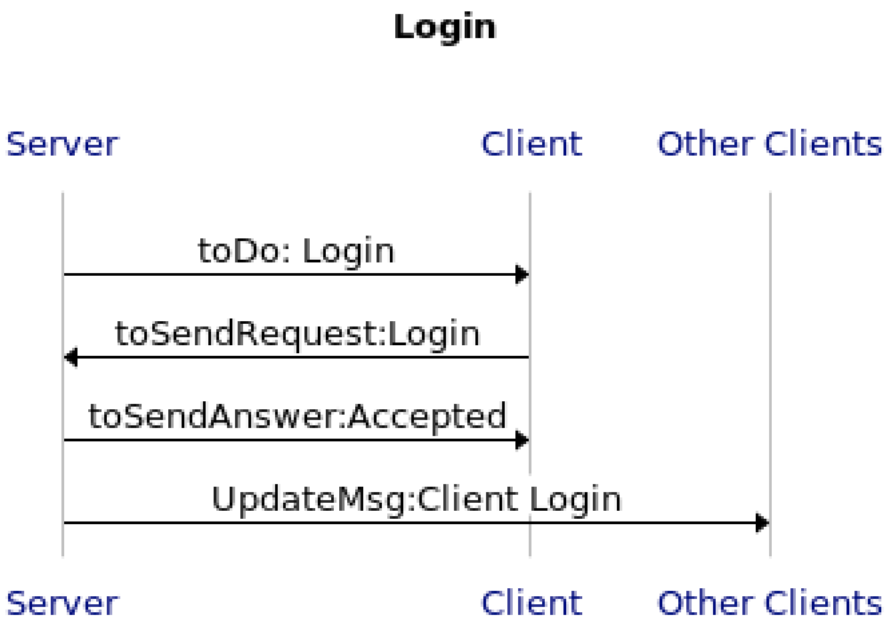

Figure 3.1: Login Sequence Diagram

3.2 Start Game

Once there are at least two clients connected, the server asks the lobby creator (the first connected to this) if he wants to start the game. In this case, the creator can ignore the message and wait for a third client to connect. Once a third client has connected the match will automatically start. As soon as the game starts (whether the start is due to a creator's request, or whether it started automatically) a update warns all connected users that the game has started.

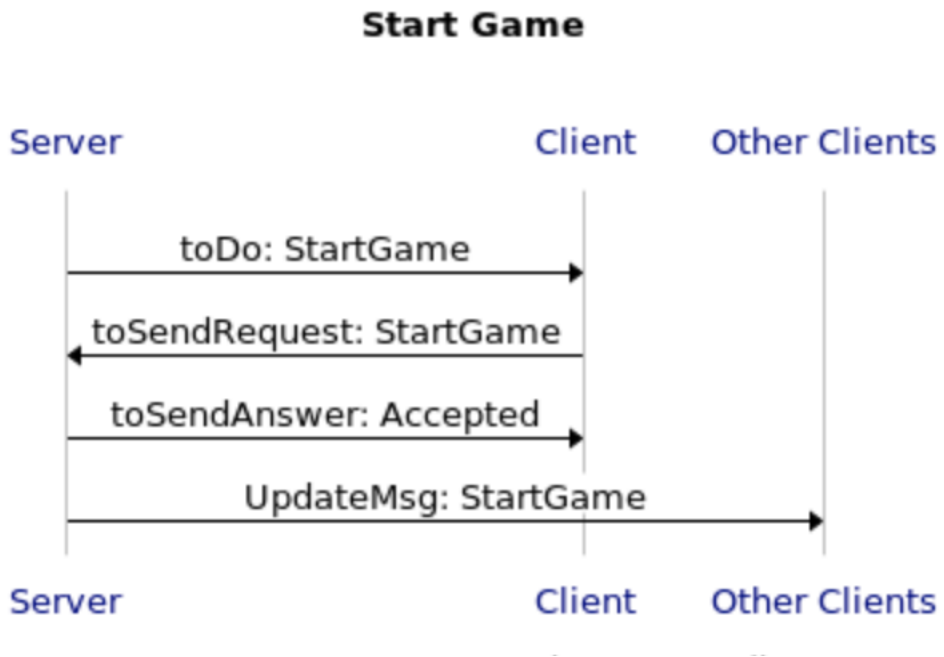

Figure 3.2: Start Game Sequence Diagram

3.3 Create Gods

When the game starts, the controller sets the Challenger with a random function. Then, the server asks the challenger to choose three gods for the match. At the same time it sends all the other clients a wait message, specifying why they have to wait and who is currently performing the specific action). The challenger sends a request containing the three gods and the server answers with an accepted/rejected response and at last, all the other clients are updated with the chosen gods if the request is successful.

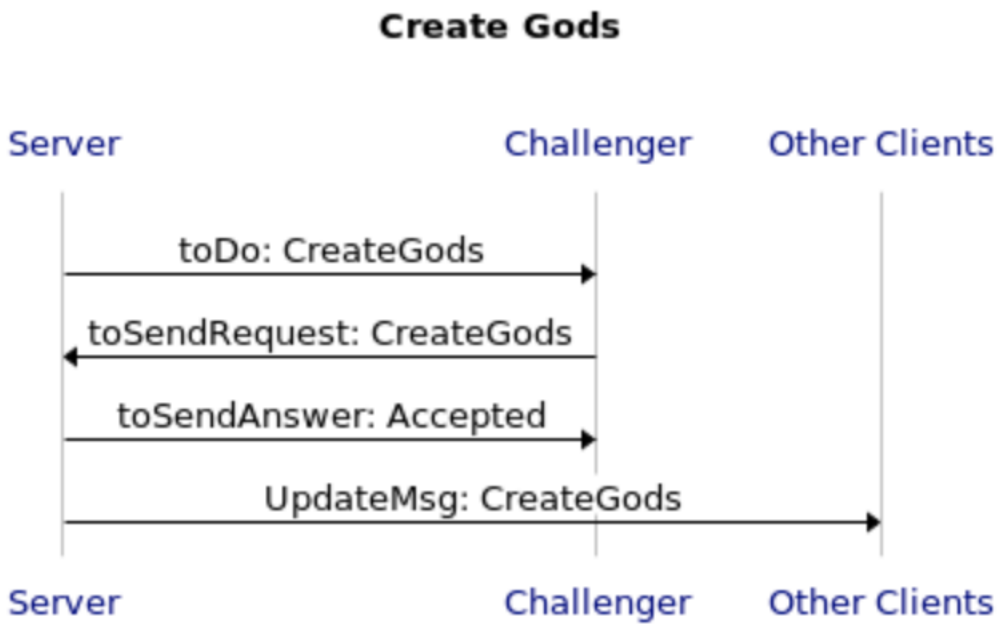

Figure 3.3: Create Gods Sequence Diagram

3.4 Gods Choice

Once the Challenger has selected the gods, in clockwise order, the server asks all the users for their god. For each user, the server asks for a god (among the remaining gods who haven't been chosen yet). At the same time it sends all the other clients a wait message, specifying why they have to wait and who is currently performing the specific action). The client to which a toDo message was sent response with a choseGod request. The server answers with the accepted/rejected message and at last, all the clients are updated if the request is successful.

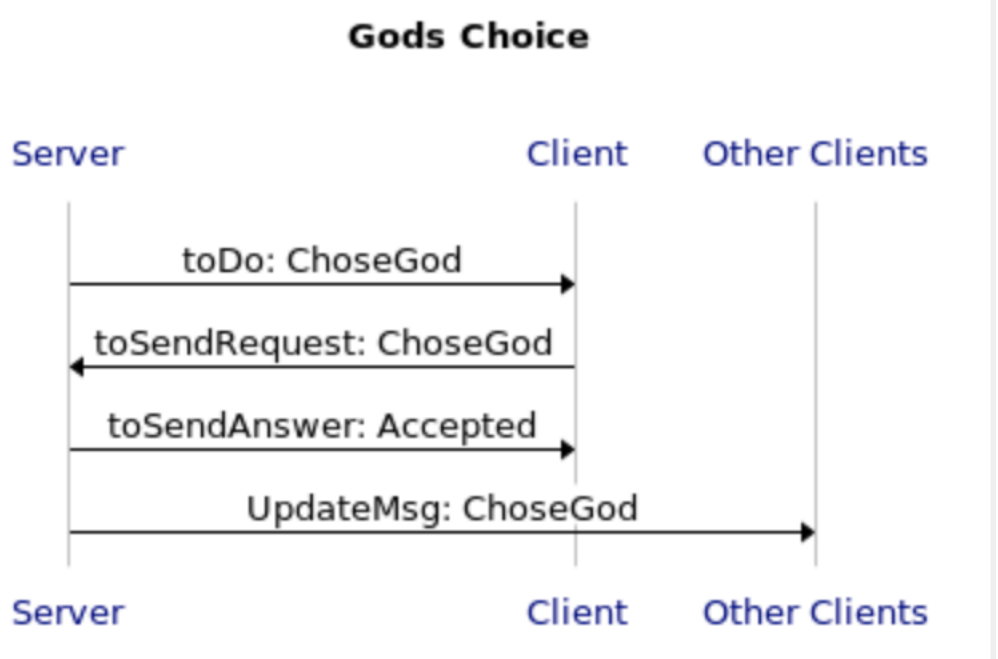

Figure 3.4: Gods Choice Sequence Diagram

3.5 Starter Player Choice

In this phase, the server asks the Challenger to choose the Starter Player. At the same time it sends all the other clients a wait message, specifying why they have to wait and who is currently performing the specific action) The challenger sends a choseStarter request containing the username of the starter player and the server answers with the accepted/rejected message. At last, all the other users are updated if the request has been accepted.

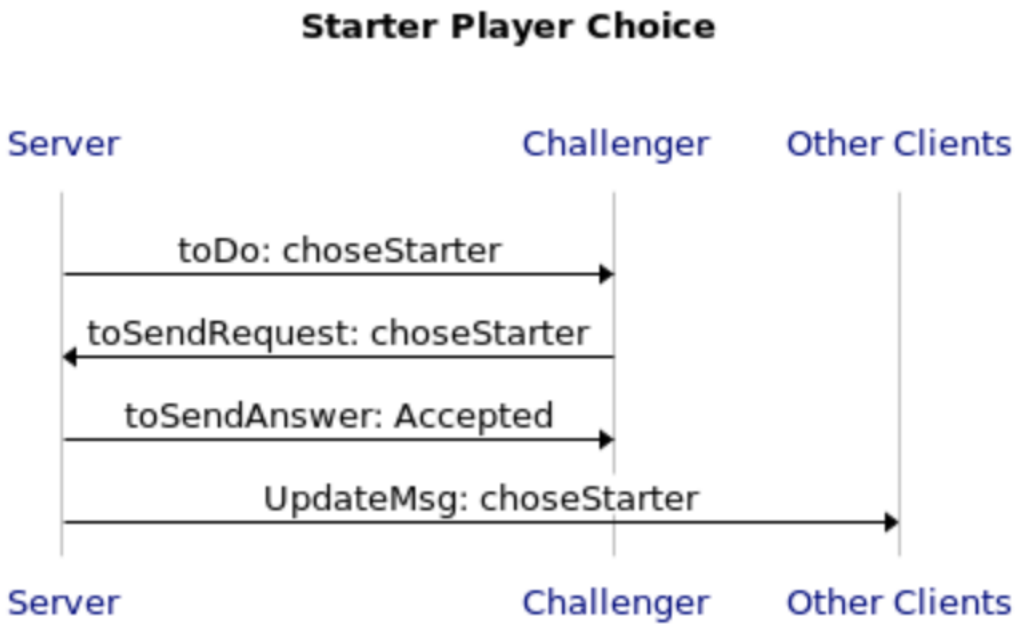

Figure 3.5: Starter Player Choice Sequence Diagram

3.6 Workers Setup

Once the Challenger has selected the Starter Player, the server asks all the users to set their workers on the board (one at a time), the starter player goes first. At the same time it sends all the other clients a wait message, specifying why they have to wait and who is currently performing the specific action). The Player responds with a SetWorkerOnBoard request (containing the position of the Worker and its gender) and the server answers with the accepted/rejected message. At last, all the other clients are updated if the request has been accepted.

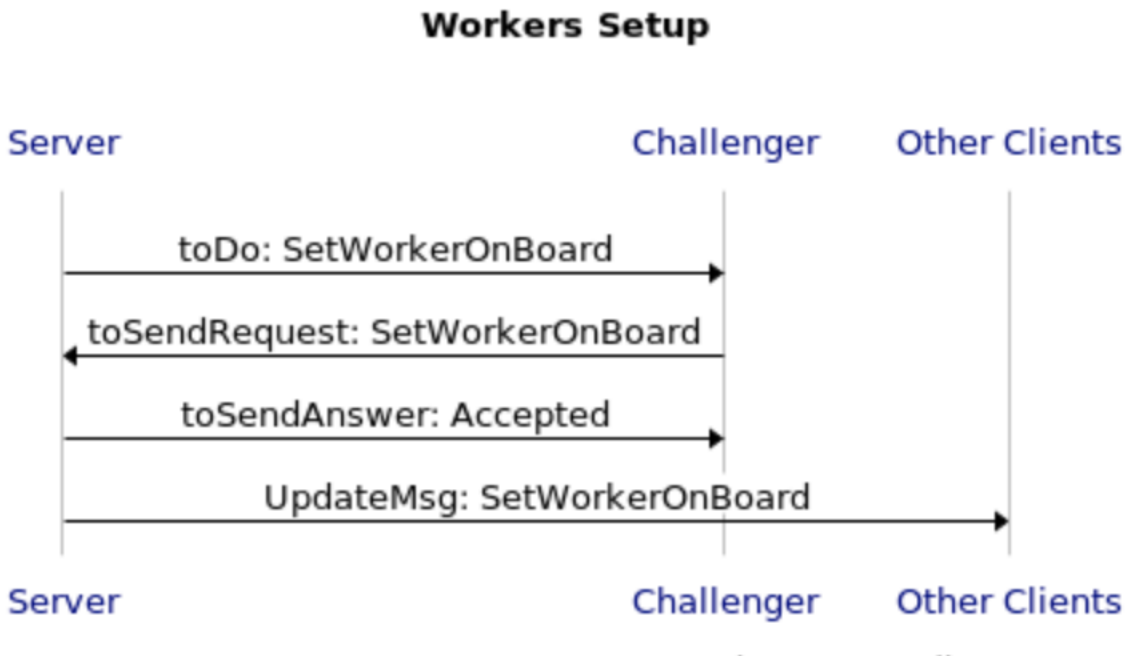

Figure 3.6: Workers Setup Sequence Diagram

4. In-Game

4.1 Start Turn

If it is his turn the server notifies the user with a yourTurn message, that contains the first operation he has to do (or if there is a choice, the first operations he can do), and, at the same time, it sends all the other clients a wait message, specifying why they have to wait and who is currently performing the specific action).

Figure 4.1: Your Turn Sequence Diagram

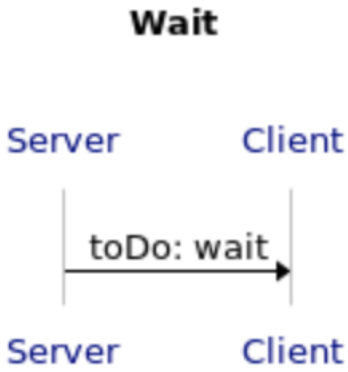

Figure 4.2: Wait Sequence Diagram

4.2 Move

When one of the clients moves, it sends a MoveRequest to the server containing the position and the worker moved. The server answers with the accepted/rejected message and then updates all the other clients if the request was successful. The response for the player in turn also contains the next operation to be performed (move/build or build).

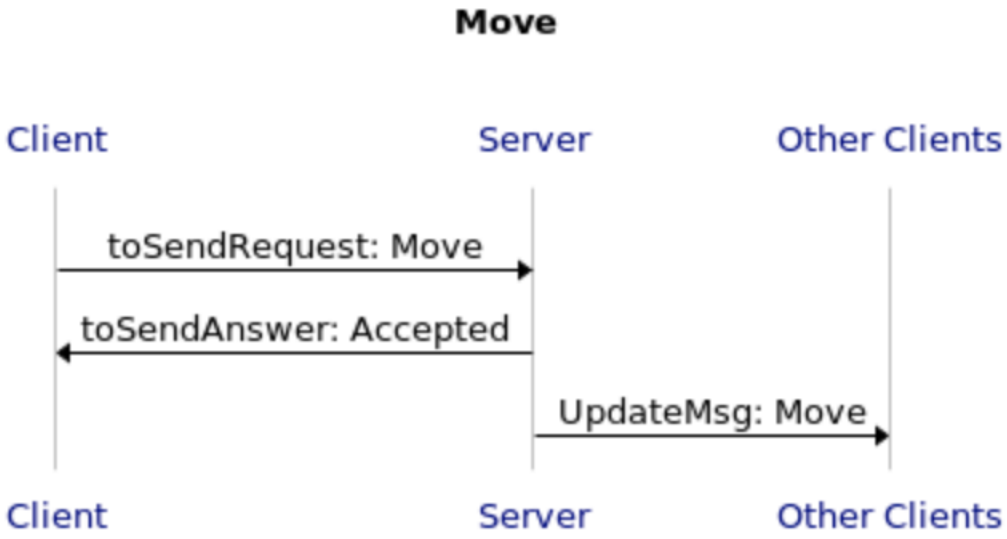

Figure 4.3: Move Sequence Diagram

4.3 Build

When one of the clients builds, it sends a BuildRequest to the server containing the position and the worker used. The server answers with the accepted/rejected message and then updates all the other clients if the request was successful. The response for the player in turn also contains the next operation to be performed (build or build/endOfTurn).

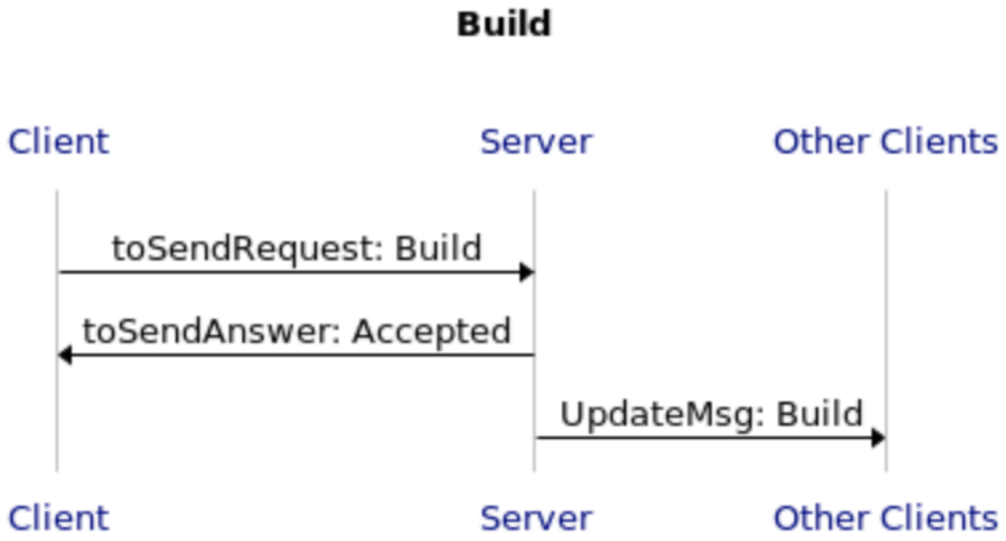

Figure 4.4: Build Sequence Diagram

4.4 End Of Turn

When one of the users ends his turn, it sends an endOfTurnRequest to the server. The server answers with the accepted/rejected message and then updates all the other clients if the request was successful. After this operation, the next player to have to play his turn is warned with a toDo.

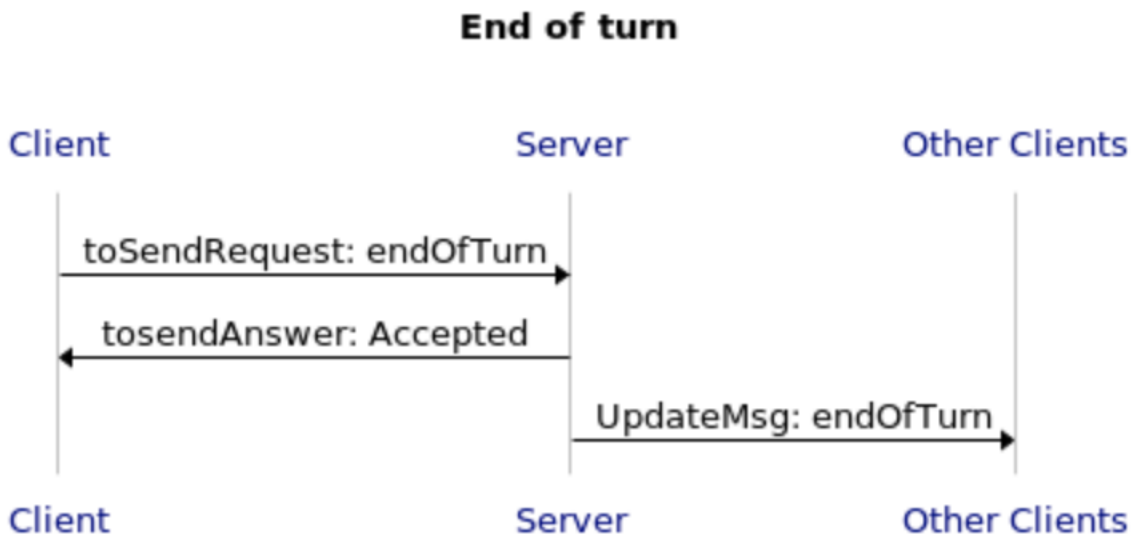

Figure 4.5: End of Turn Sequence Diagram

5. End Match

A match can end up disconnecting a client, winning a player directly, or losing a player in a 2-a-side game.

5.1 Client Disconnection

When a client disconnects from the server during the match, the other clients are warned that a client goes down, then the match ends and the lobby is deleted.

5.2 Win/Lose in a 2-a-side game

A player can win in two different ways, if he wins directly or if he is playing currently a 2-a-side game and the other player loses. If one of these two conditions happens, the server sends a message to the winner to notify him of the victory and a message to loser to notify him of the defeat. After the notifications, the server provides to delete the match and the lobby.