Council of Four

Software Engineering Project
Politecnico di Milano
Academic Year 2015/2016

ps13 group

Michele Ferri
Alessio Mongelluzzo
Tommaso Massari

High level description

- This project is the Java implementation of the board game *Council of Four.*
- Game configuration is loaded from an XML file, which makes it fully customizable.
- The game allows to play online matches using both **RMI** and **Socket** communication.
- Players can choose to run the application in Command Line Interface (**CLI**) or Graphical User Interface (**GUI**).

Architecture

- The project structure is based on the **MVC (Model-View-Controller)** architectural pattern.
- **Model** contains the objects representing the state of the game, and allows to use them according to the rules.
- **Controller** acts as an intermediary between the View and the Model, handles client requests, performs a legality check on every action request, creates the action objects and applies them.
- **View** allows players to interact with the application by entering input, which is converted into appropriate request messages and sent to the server.

Model

- Model objects are created from information contained in a **configuration file**: regions, cities, cards, councillor balconies, bonuses…
- The Game class encapsulates the entire status of a match. It is *Serializable*, so that it can be easily sent over a network.
- In addition to the actual board game rules, model also contains logic for handling a ***market*** *phase*, which allows players to trade special *marketable* items (assistants, politics cards and permit tiles) for coins.

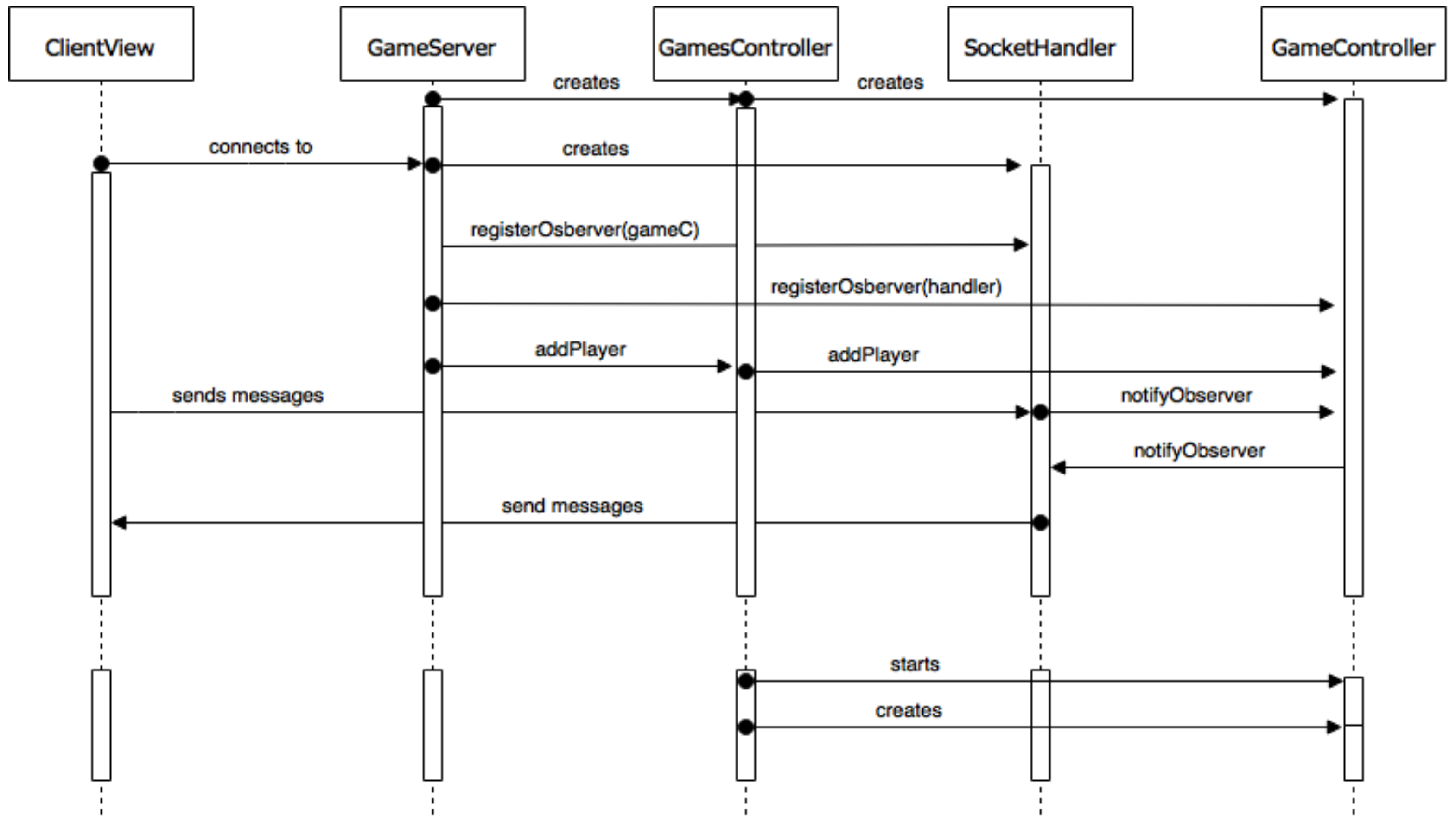
Controller

- For the interaction between view and controller on the server side, we decided to adopt the *Observer* design pattern: the controller is notified by the view with request messages.
- Action request messages are converted into concrete action objects by an action factory using a *Visitor* design pattern.
- **Multiple matches** can be played at the same time: every match has its own game controller once it starts.

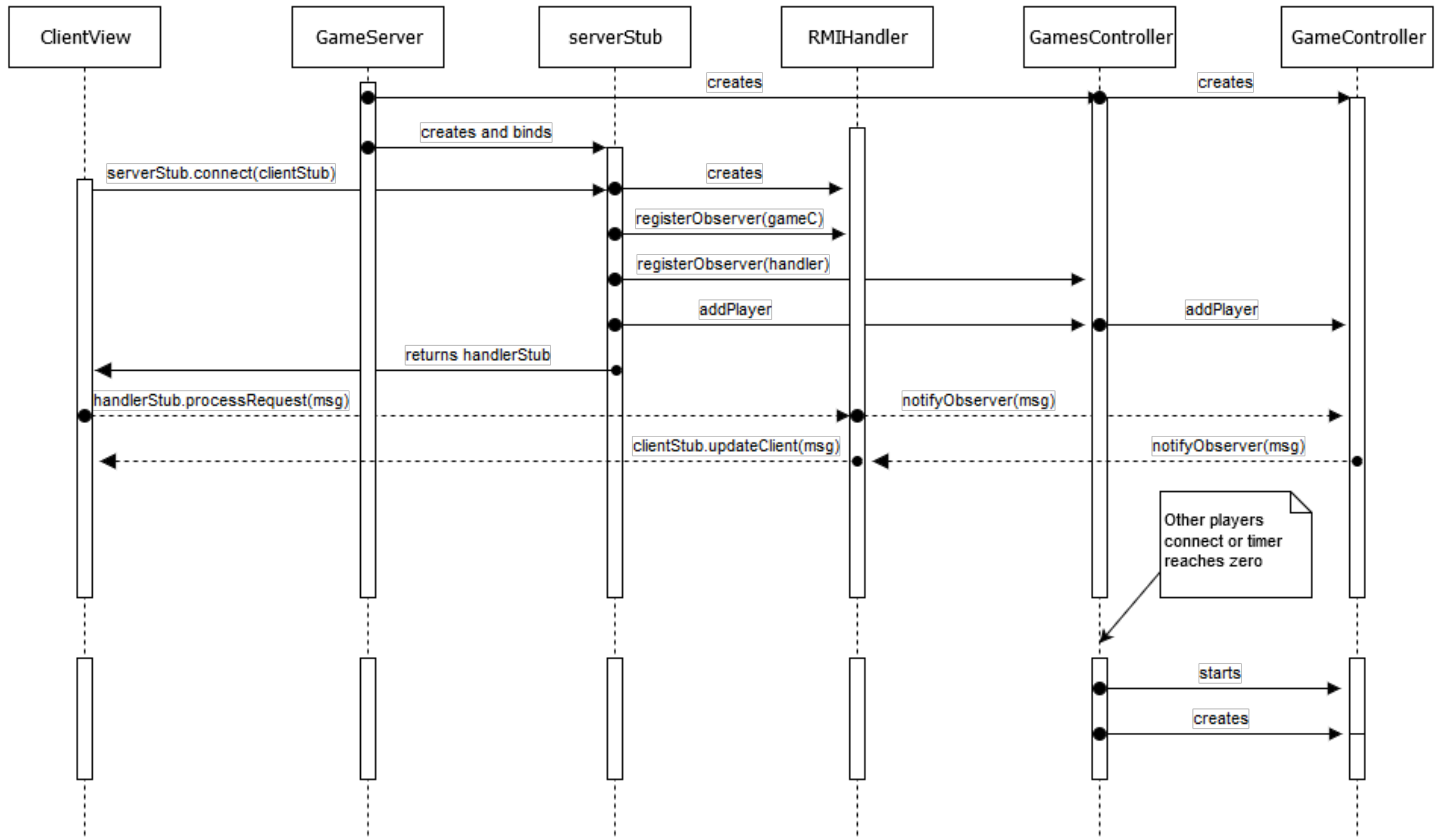
View

- For each client connected to a game a view (handler) on the server side is created. The handler is notified by the controller with response messages, which are then forwarded to the client side view.
- The client side view constantly waits for server updates and, in the meantime, can send every type of request (chat, action…)
- Both the **handler** and the **client side view** have an RMI and a Socket implementation, which can be used in the same way through the Handler abstract class and **ClientConnection interface** respectively.

Socket communication



RMI communication



Features

- Once two players connect to the current waiting game, a **timer** is started. When the timer reaches zero, the game starts and a new waiting game is created for new players to join.
- While waiting for the game to start, waiting players can change their displayed name with the ***rename*** command.
- **Chat** is always available. In CLI, players have to type “chat …”. In GUI, an input form sends entered input as a chat message when Enter key is pressed.

Features

- Every **turn** has a fixed amount of time to be completed and passed. As the **timer** reaches zero, the turn is automatically passed to the next connected player.
- When a player **disconnects** in the middle of the game, its turn will always be skipped by default but it will still be counted as a potential winner of the game. Every player is notified when a player disconnects. If everyone disconnects but one, the game is instantly terminated and the winner is announced.