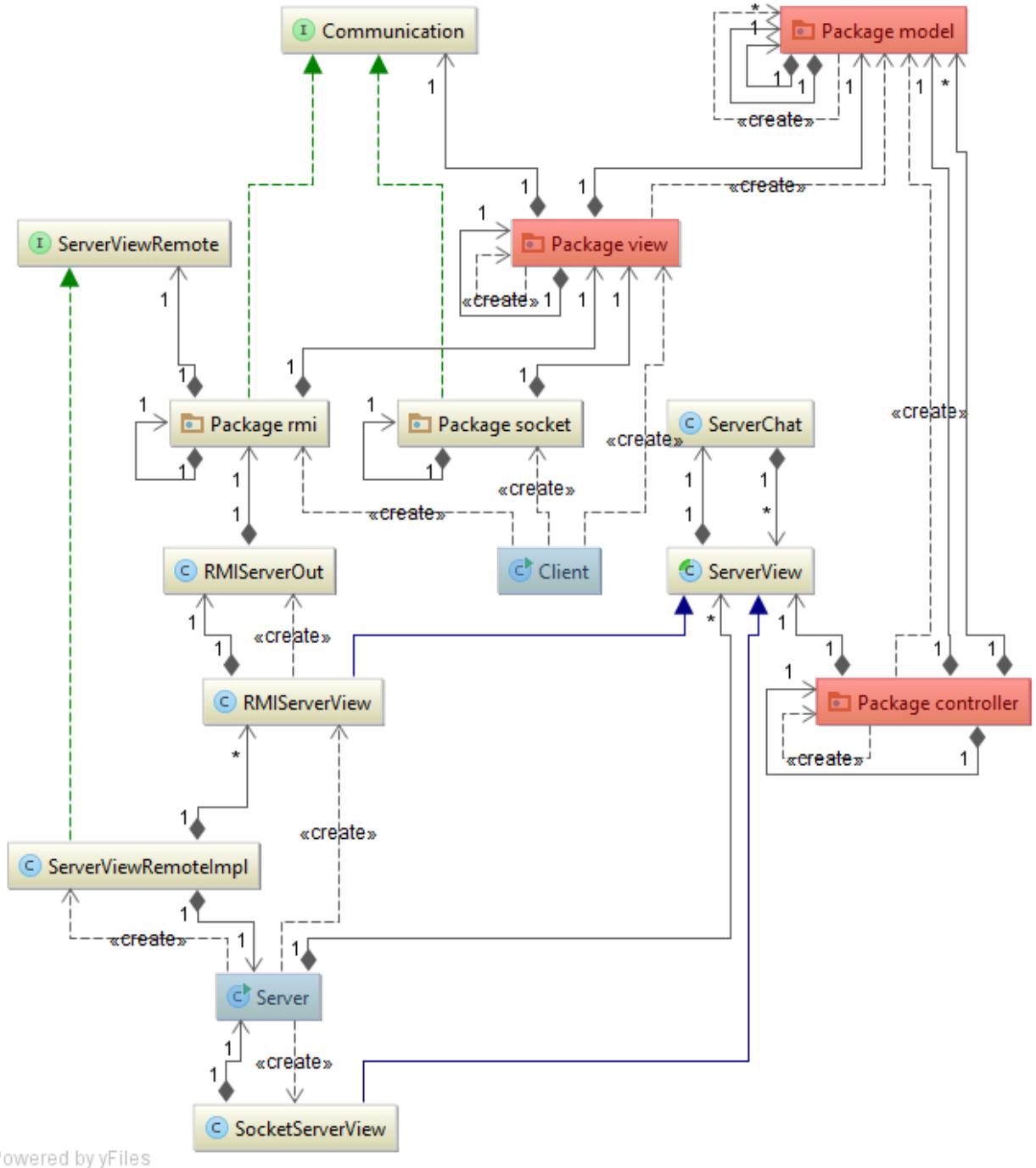
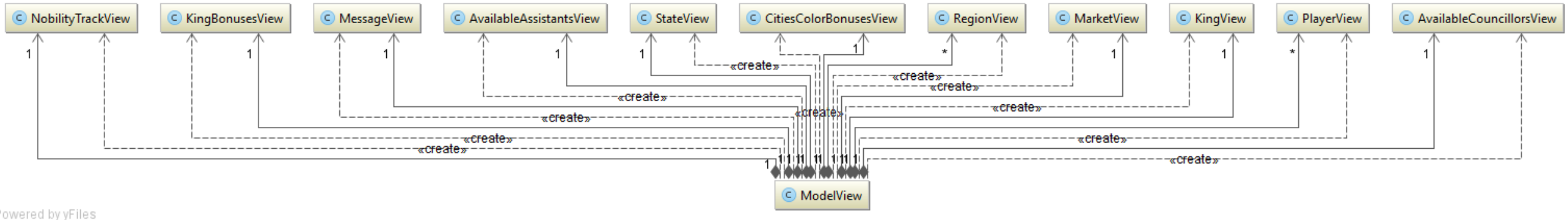
MVC Pattern

The whole architecture of the application follows the **MVC** pattern, the model contains all the information about the game and the controller manages it. The view is composed by two parts, one on the server side and the other on the client side, that communicate with each other.

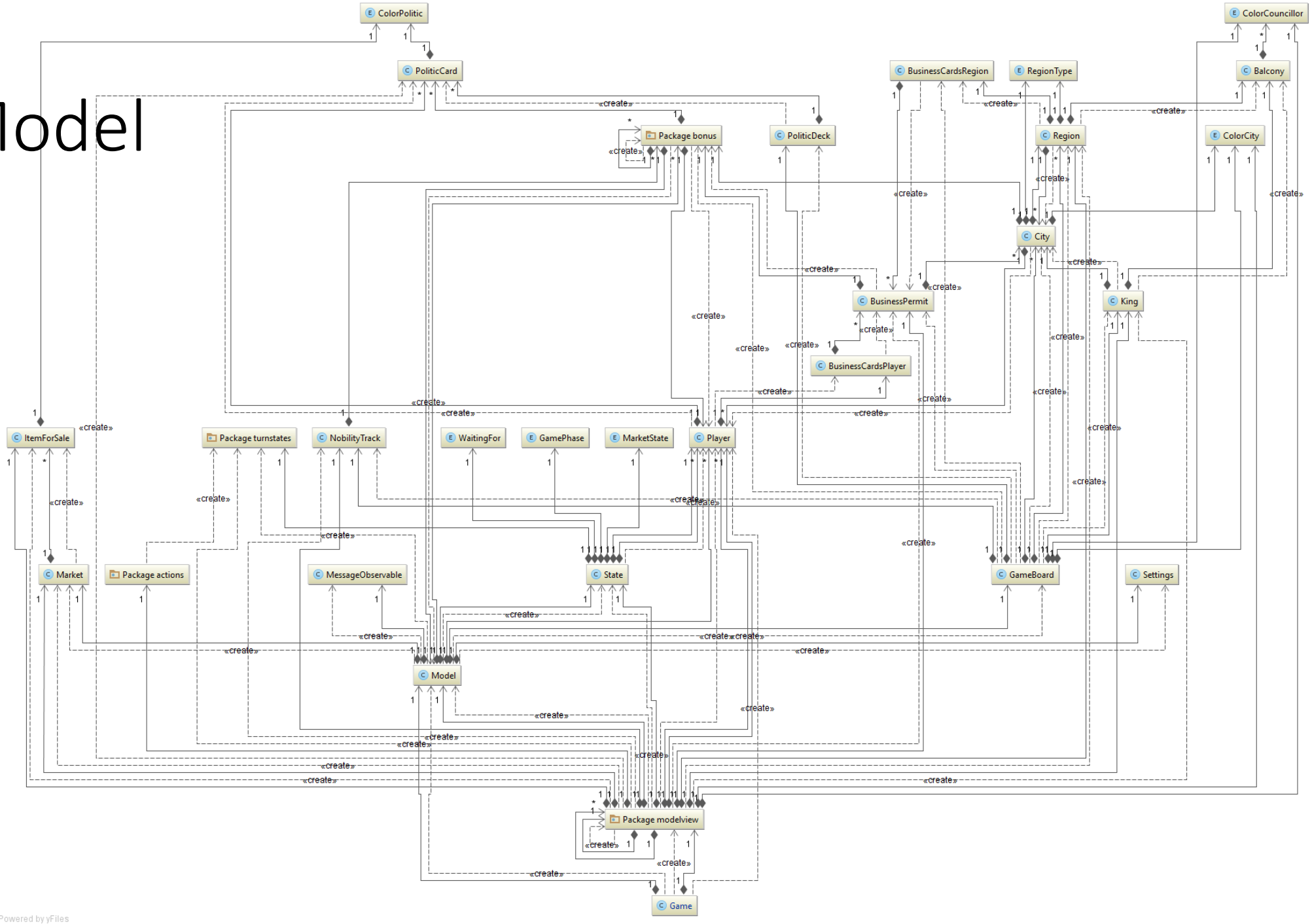


ModelView

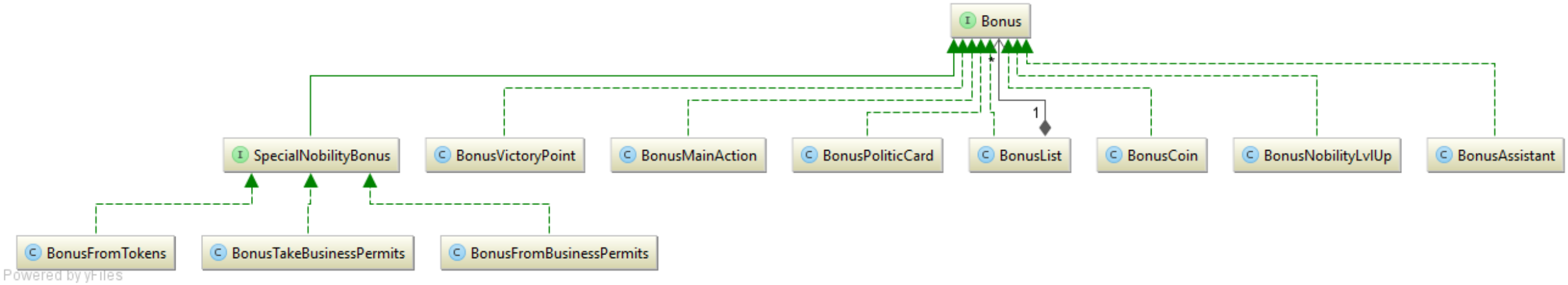


We choose to implement a ModelView class which is an always updated deep copy of the game.
In this way the View hasn't got a reference to the action model but it's still able to refresh itself without communicating with the controller.
Every main component of the game is observed by a *ComponentView* class. Each of them updates itself when needed and notifies the ModelView class that sends the updates to the View.

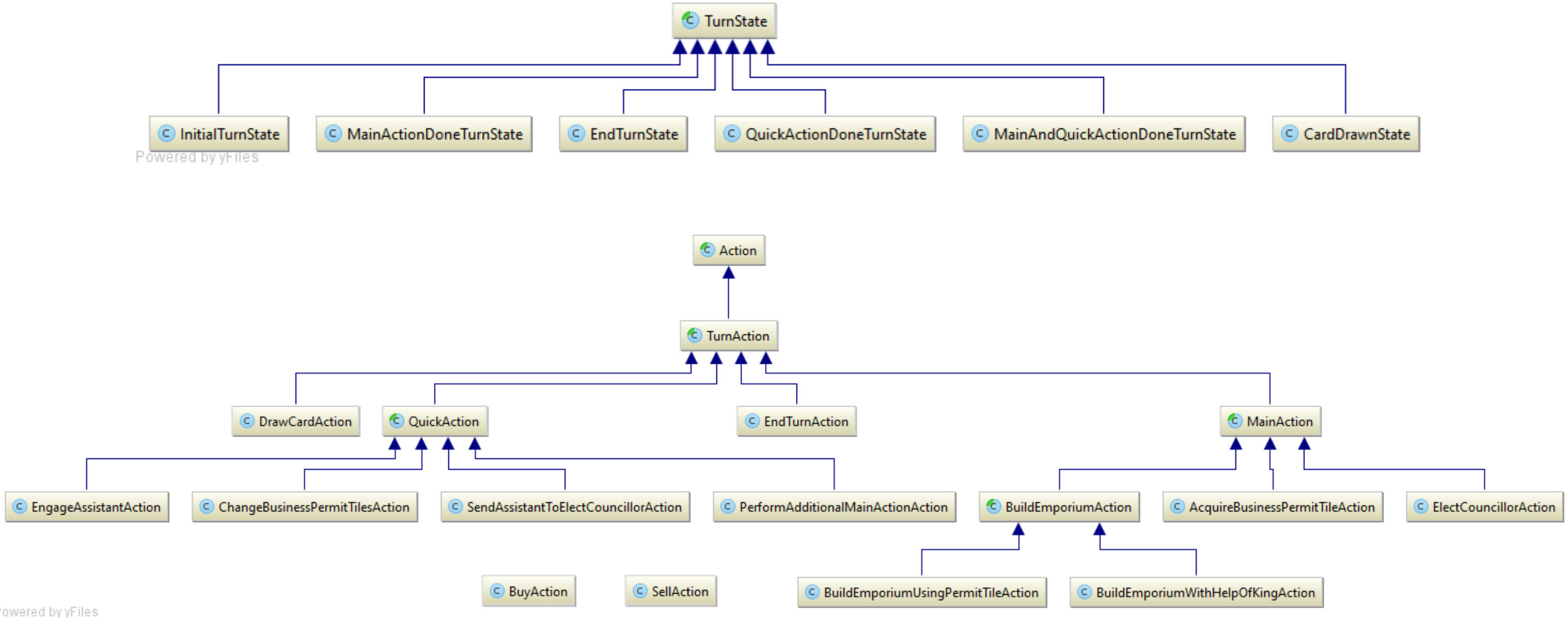
Model



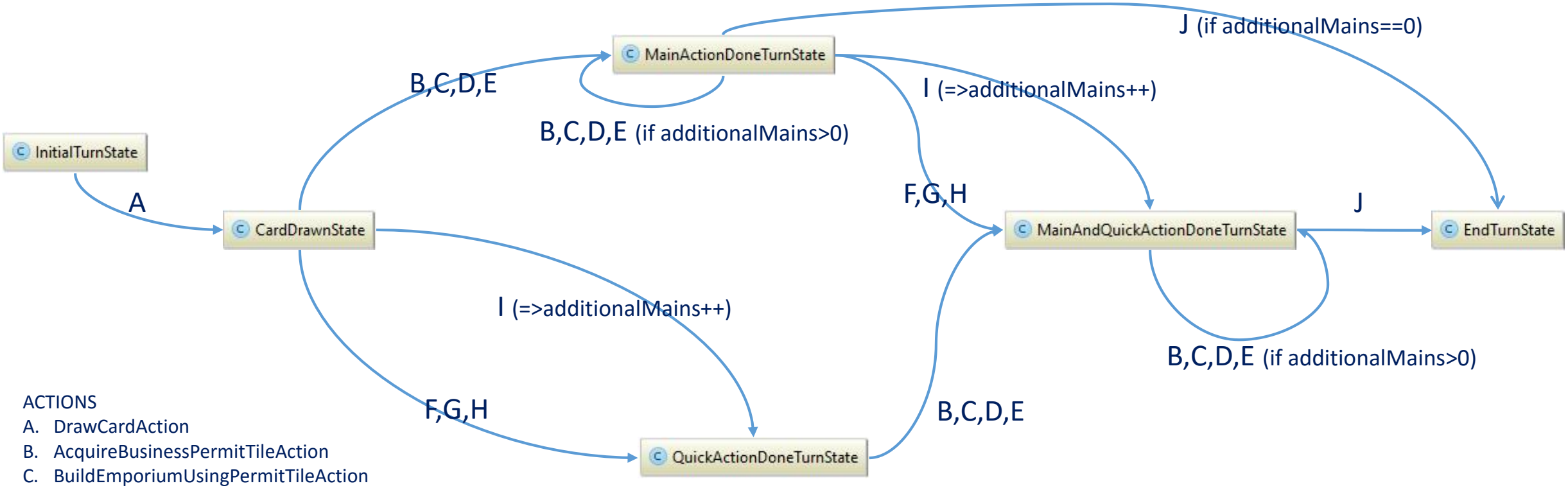
Bonus


Bonus
SpecialNobilityBonus
BonusVictoryPoint
BonusMainAction
BonusPoliticCard
BonusList
BonusCoin
BonusNobilityLvlUp
BonusAssistant
BonusFromTokens
BonusTakeBusinessPermits
BonusFromBusinessPermits
1
*
Powered by yFiles

Turnstate & Action



Turn State Machine



ACTIONS

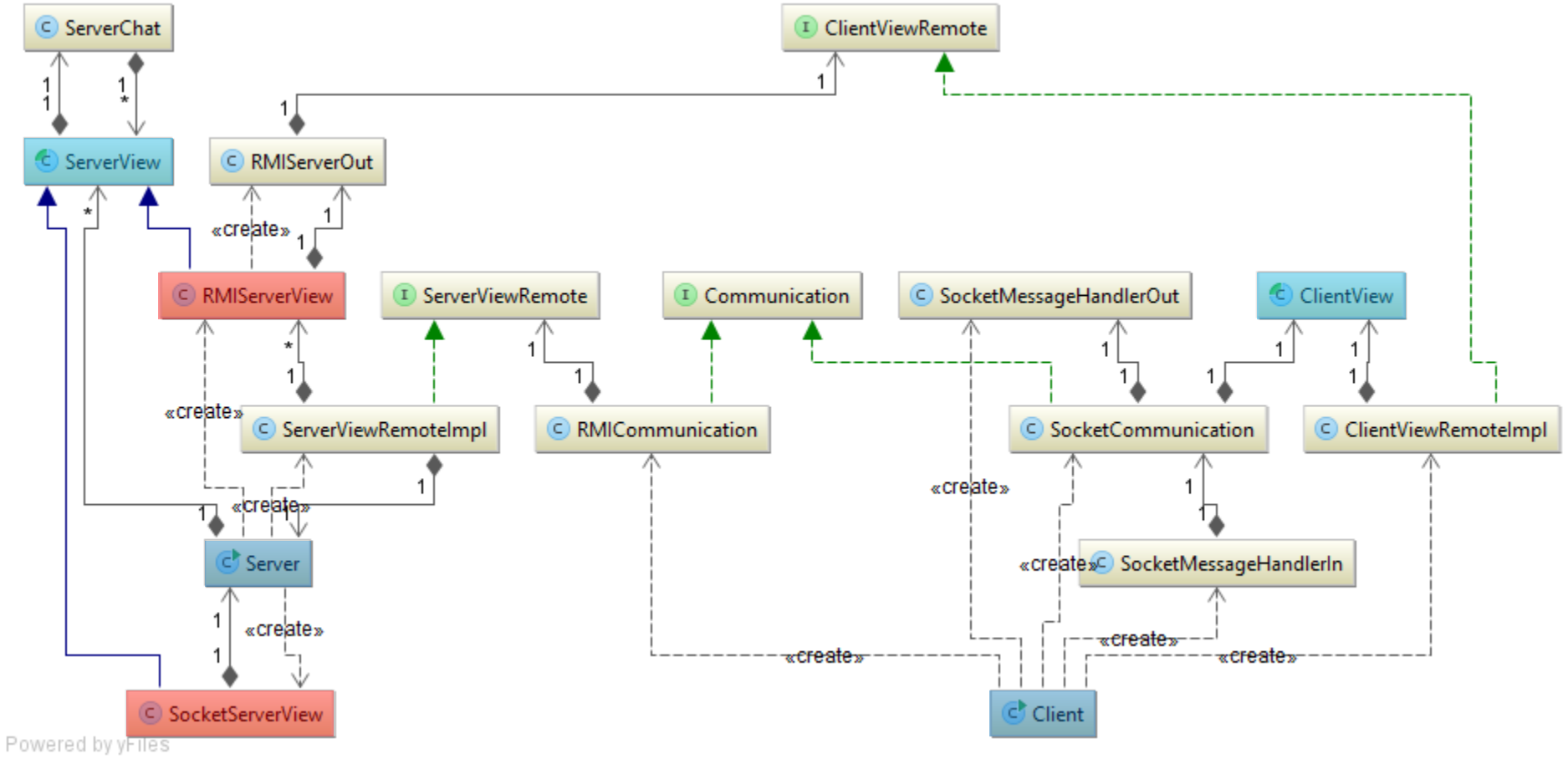
A. DrawCardAction
B. AcquireBusinessPermitTileAction
C. BuildEmporiumUsingPermitTileAction
D. BuildEmporiumWithHelpOfKingAction
E. ElectCouncillorAction
F. ChangeBusinessPermitTilesAction
G. EngageAssistantAction
H. SendAssistantToElectCouncillorAction
I. AdditionalMainAction
J. EndTurnAction

additionalMains is an attribute of MainActionDoneState, QuickActionDoneState and MainAndQuickActionDoneState used as counter for the remaining MainAction.

Implemented features: rmi/socket connection

Server and clients can communicate through RMI or Socket connections.

The information flows between the ServerView and the ClientView classes which work as if they were just one View.

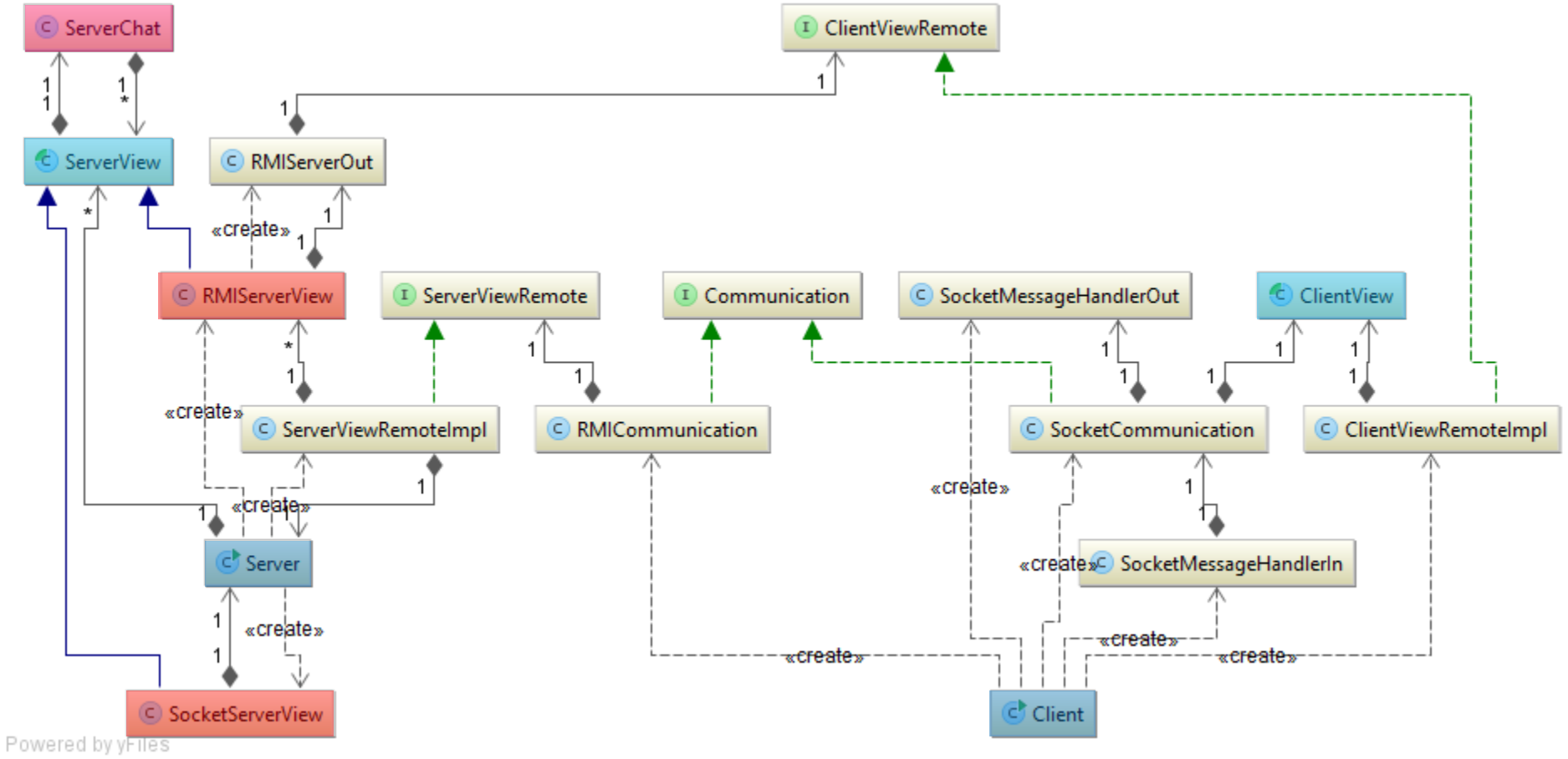


Implemented features: chat

Server and clients can communicate through RMI or Socket connections.

The exchange of information is developed between the ServerView and the ClientView classes which work as if they were just one View.

The ServerChat class is responsible for collecting message from a player and for forwading it to all players.

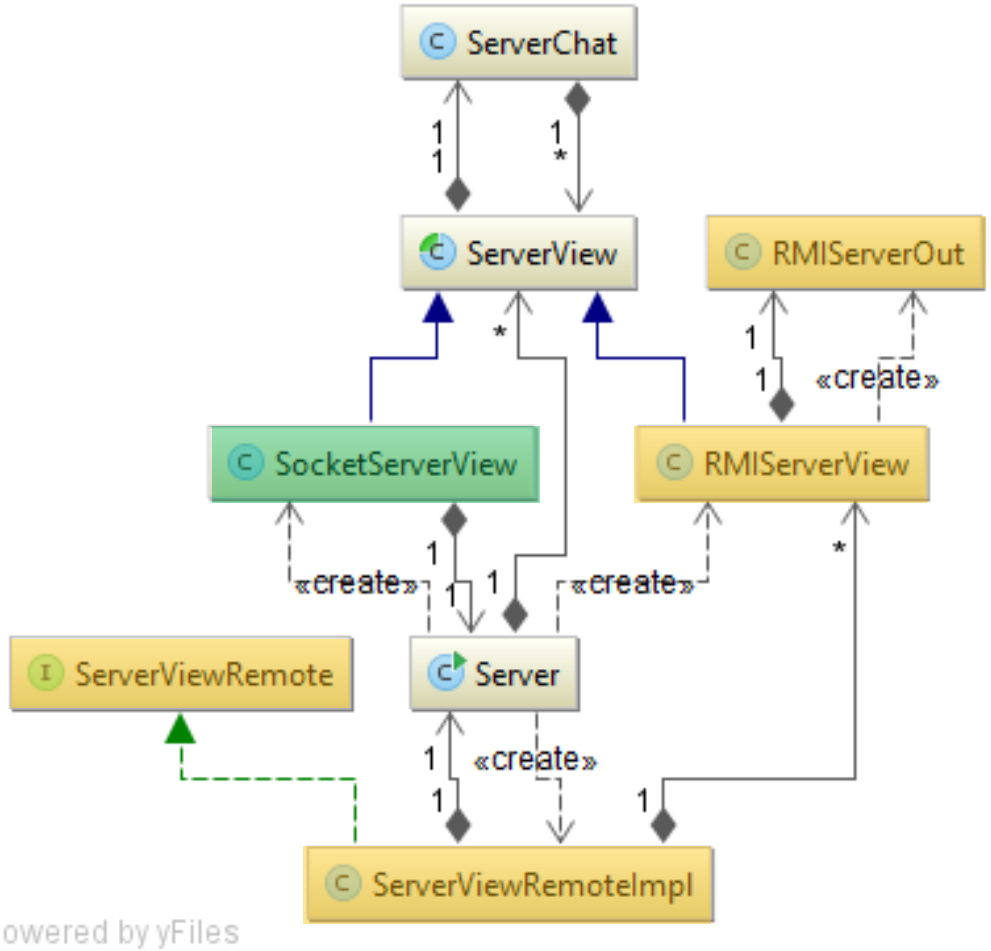


Server connection side

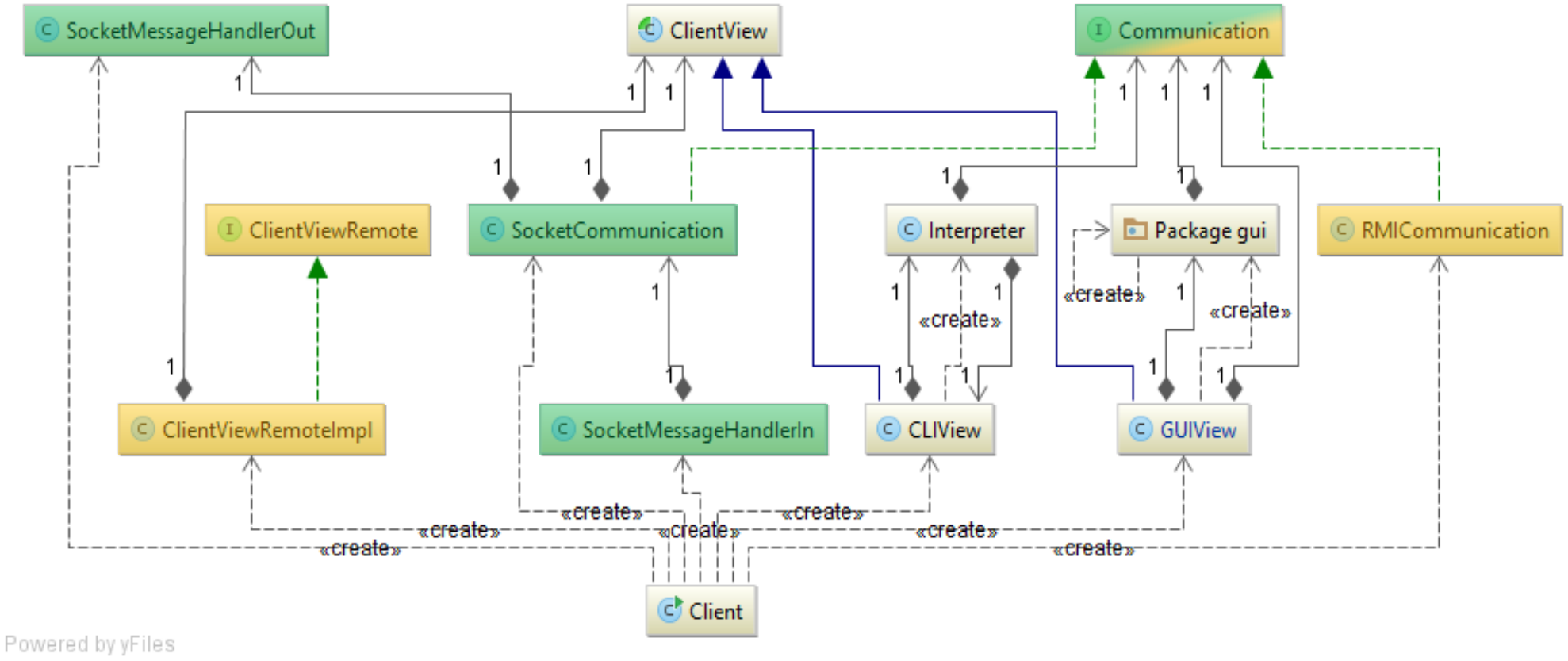
Any ModelView update is received by the ServerView which is responsible for forwarding it to the ClientView. This will result in CLI or GUI updates.

When 2 Clients are connected to the Server a 20 seconds timer is set. After this time the game starts. The number of players in a game can be easily changed.

If there is no available game when a player connects to the server a new one is created.

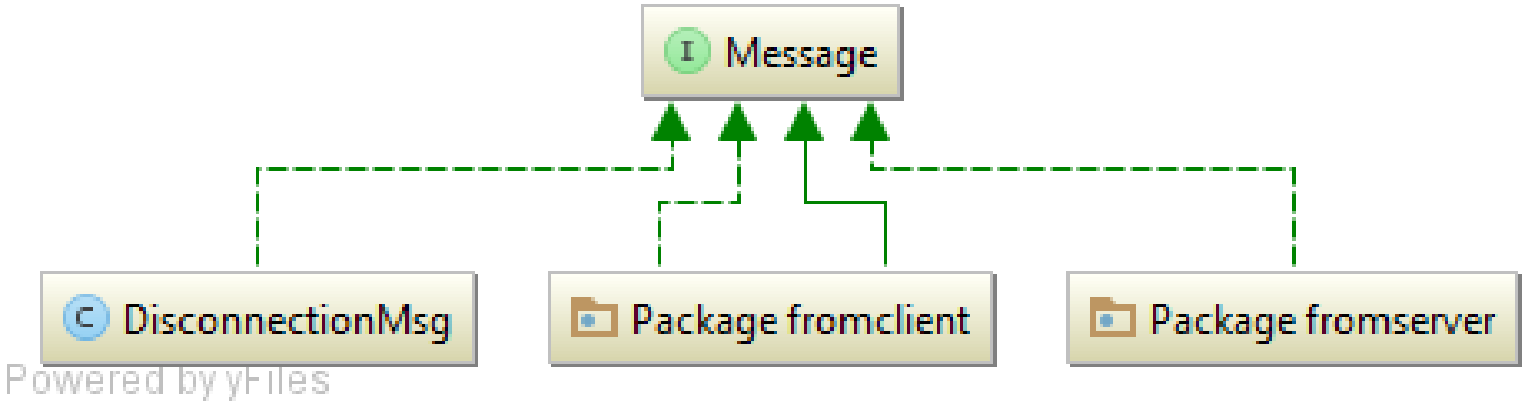


Client connection side



Any message from the Server is handled by the specific socket or RMI class, while the Communication class is responsible for forwarding any message from the ClientView (either CLI or GUI) to the Server using the proper RMI or Socket class.

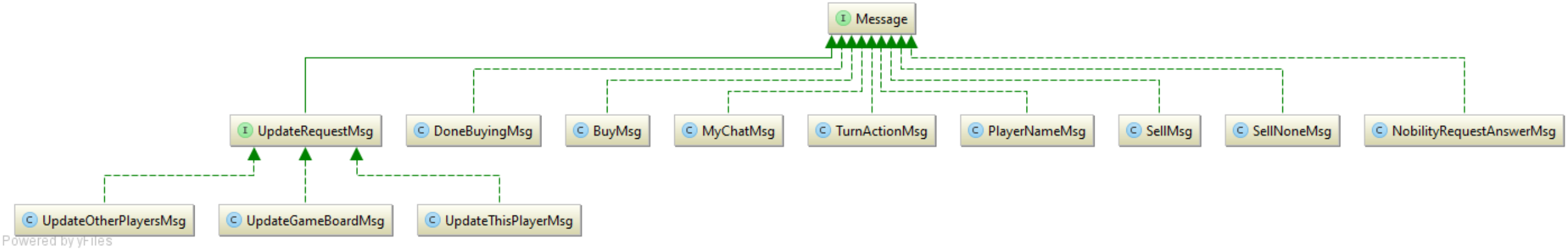
Message



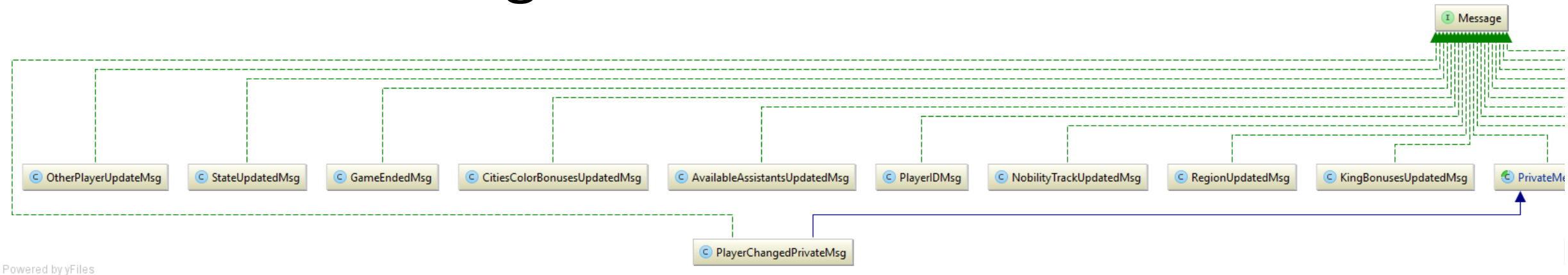
In general Messages allow communication between Server and Clients.

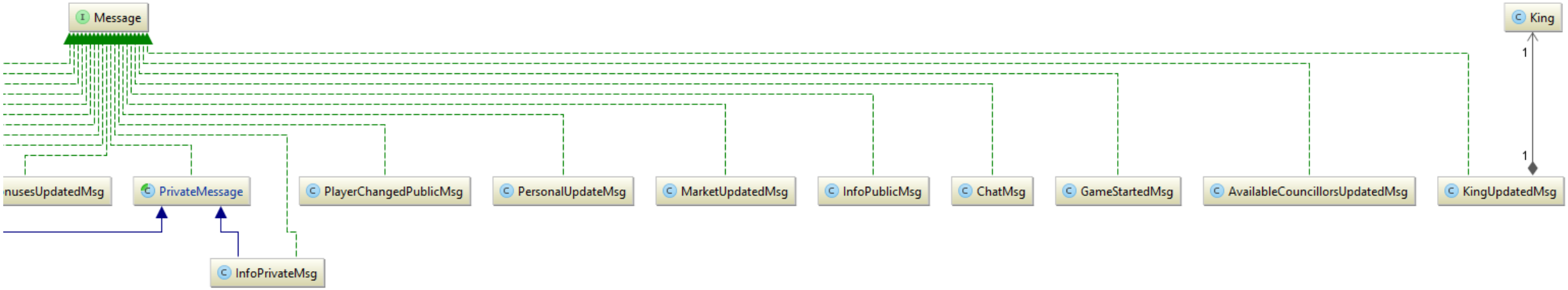
DisconnectionsMsg is specifically used by the Server to inform the Controller class that a player is no longer connected.

Client messages

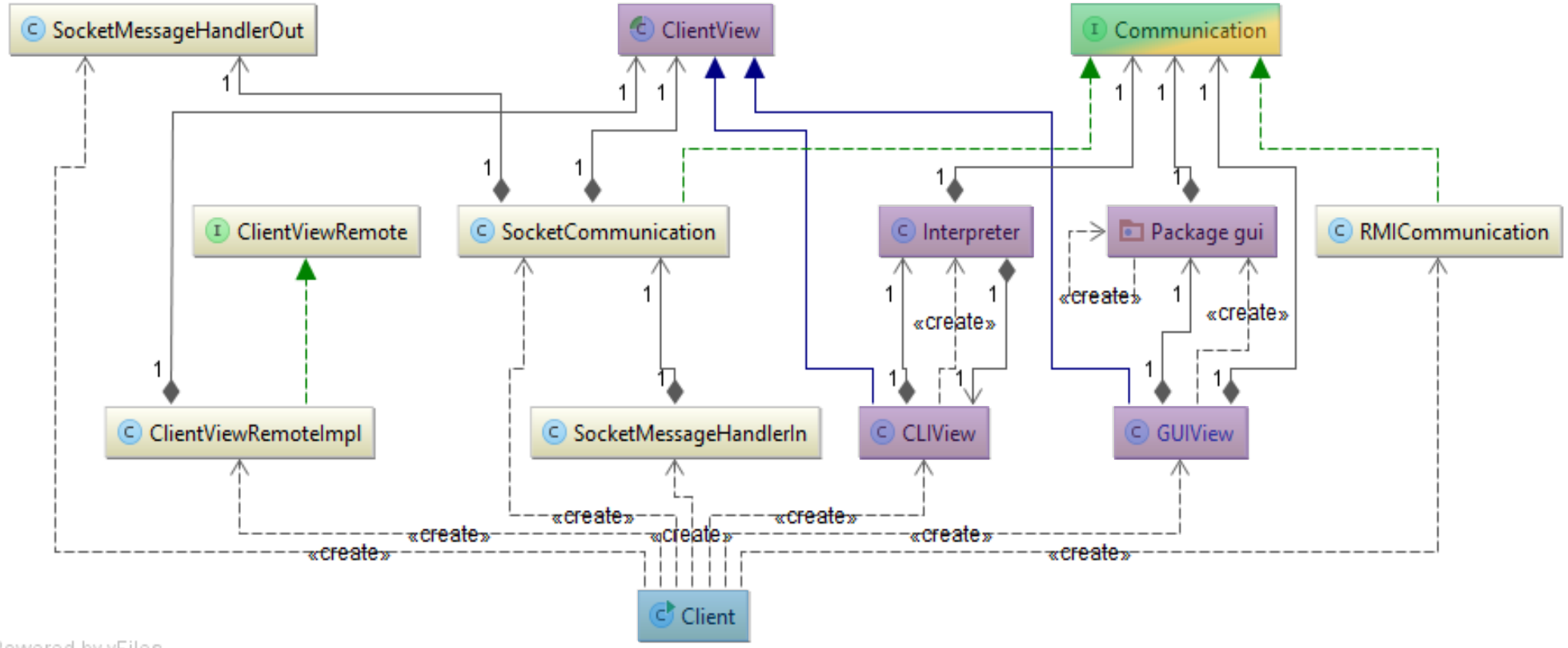


Server messages





Implemented features: GUI/CLI



CLIView and GUIView extends both the ClientView class allowing runtime instantiation of CLI or GUI (strategy pattern). Both register the player's actions and deliver them to the ServerView through the Communication class.

The CLI used an Interpreter class that convert read String into callable Communication's methods.

GUI

The GuiView class contains a reference to the GUI class which is the application main window.

When the game needs information from the player this window creates the proper jDialog.