D2 | Requirements Modelling

Battleship Social Network

Date	30/9/22
Course	SWEN3145
	Software Modelling

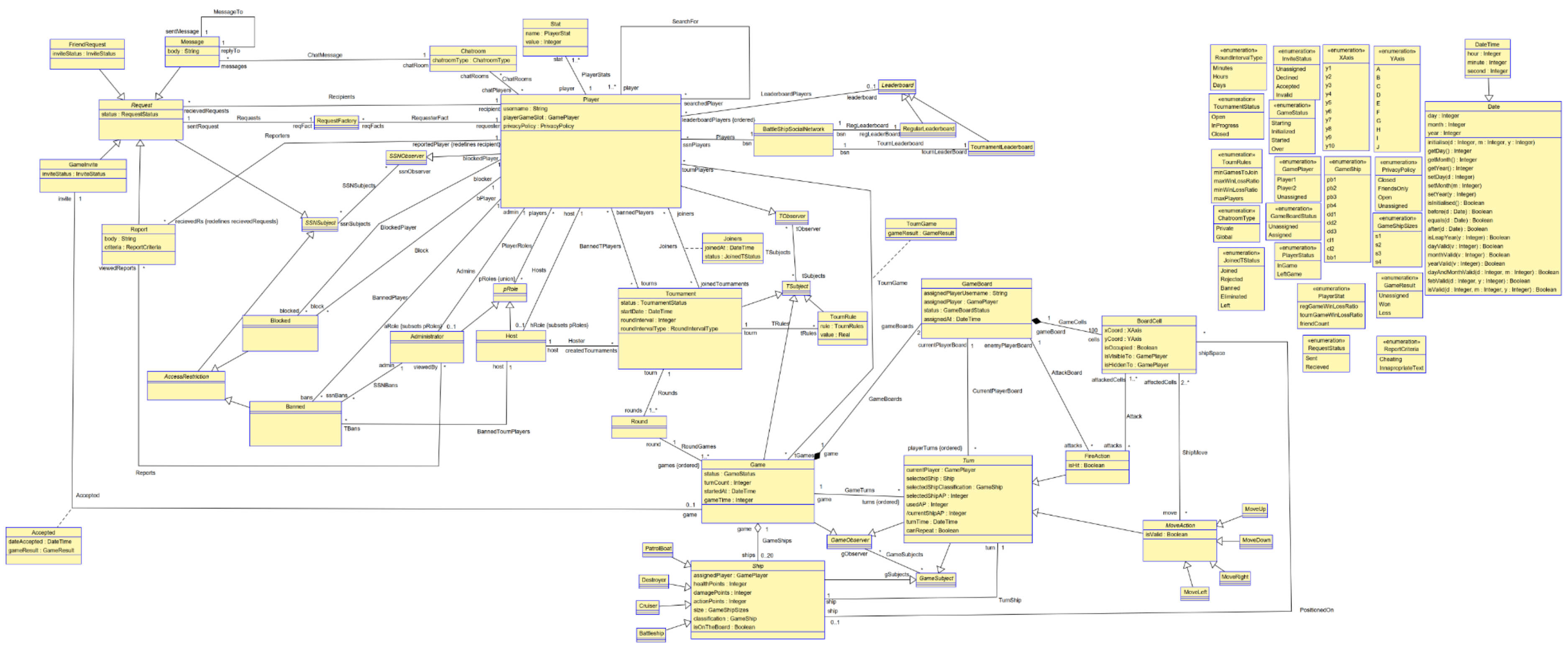
Prepared By:	Kieran Jaggernauth	620138494	kieran.jaggernauth@uwimona.edu.jm
	Patrick Witter	620139423	pwitterop123@gmail.com

Table of Contents

Updates

Change	**Rationale**
D1a | Gameplay	In the gameplay description added that a hit or miss graphic must disappear after a turn is over
D1a | Gameplay	In the gameplay description added that a player should be informed of how many ships the enemy has remaining and which ship they sunk.
D1a | Social Network	Added tournament stats and tournament descriptions to the document
D1a | Access Control	Added ABAC-enforced access control for tournaments
D1b | Functional Requirements	Added in Tournament and Leaderboard requirements. Removed Special Ability Requirements
D1b | Use Case View	Added in tournament and leaderboard use cases. Added in a host user. Added extends arrows

Class Diagram



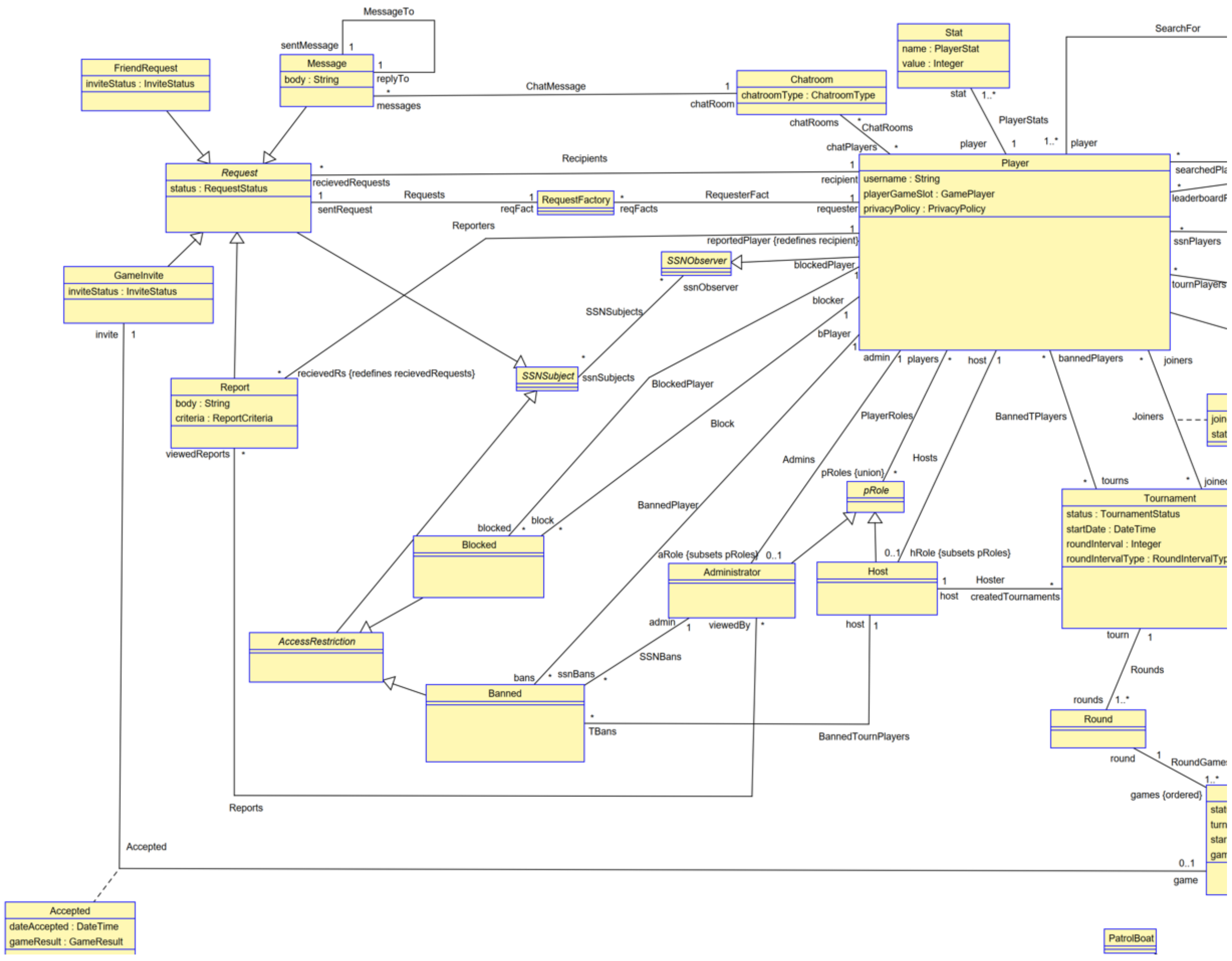
Discussion

This class diagram is a good encapsulation of our requirements document because it accounts for all the static functionality of the Social Network, Leaderboards, Tournaments and Games.

Throughout the class diagram the observer pattern is used to allow different observers such as the player and game to keep track of various subjects such as requests and game objects.

The factory pattern is used as a bridge between a player and a request to determine the type of request the player is trying to send and provide an appropriate request type. A breakdown of the different functionality can be found below.

Social Network



- The search for player functionality is included through the use of the **SearchedFor** association
- The different requests are modelled with a Request superclass which adheres to liskov's substitutability principle(LSP) allowing them to take the place of each other. The only request type that would violate LSP would be the report request type as the semantics of the recipient would not be the same as the player that was reported would not receive anything. This is solved using a redefine to redefine

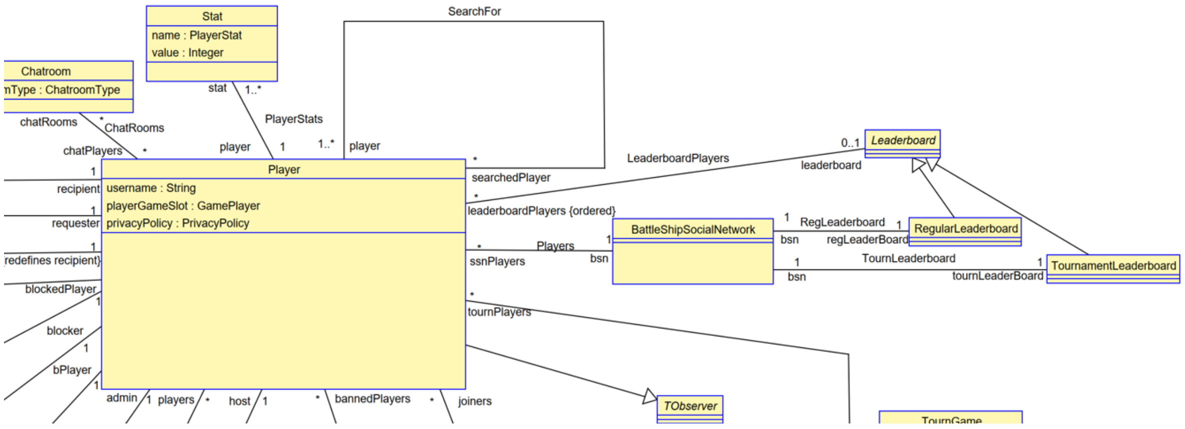
the **Recipients** association. This approach allows us to model the following functionality onto our class diagram.

  - Players can send messages to each other by use of the chatroom. The constraints of the chatroom allow for access control lists to be enforced between 2 players in a private chat.
  - The players can also send messages to a global chat which is seen by all players.
  - The ability to send friend requests, game invites and reports is also modelled here as a request because players can send and receive these.

- The reason the request model is a good implementation here is because it allows us to model that a player can send and receive multiple requests but a request can only exist between 2 players acting as an access control policy enforcement point for communications between players.
- RBAC access control is modelled using the pRole, Administrator and Host class which enforces RBAC in the sense that a player can have multiple roles but cannot have multiple of the same role. This access control scheme allows us to show the following:
  - Only hosts can create tournaments and ban players from tournaments
  - Only admins can view reports and ban players from the social network
  - An issue with this scheme is that a social network ban cannot be differentiated from a tournament ban which would require constraints to be written.

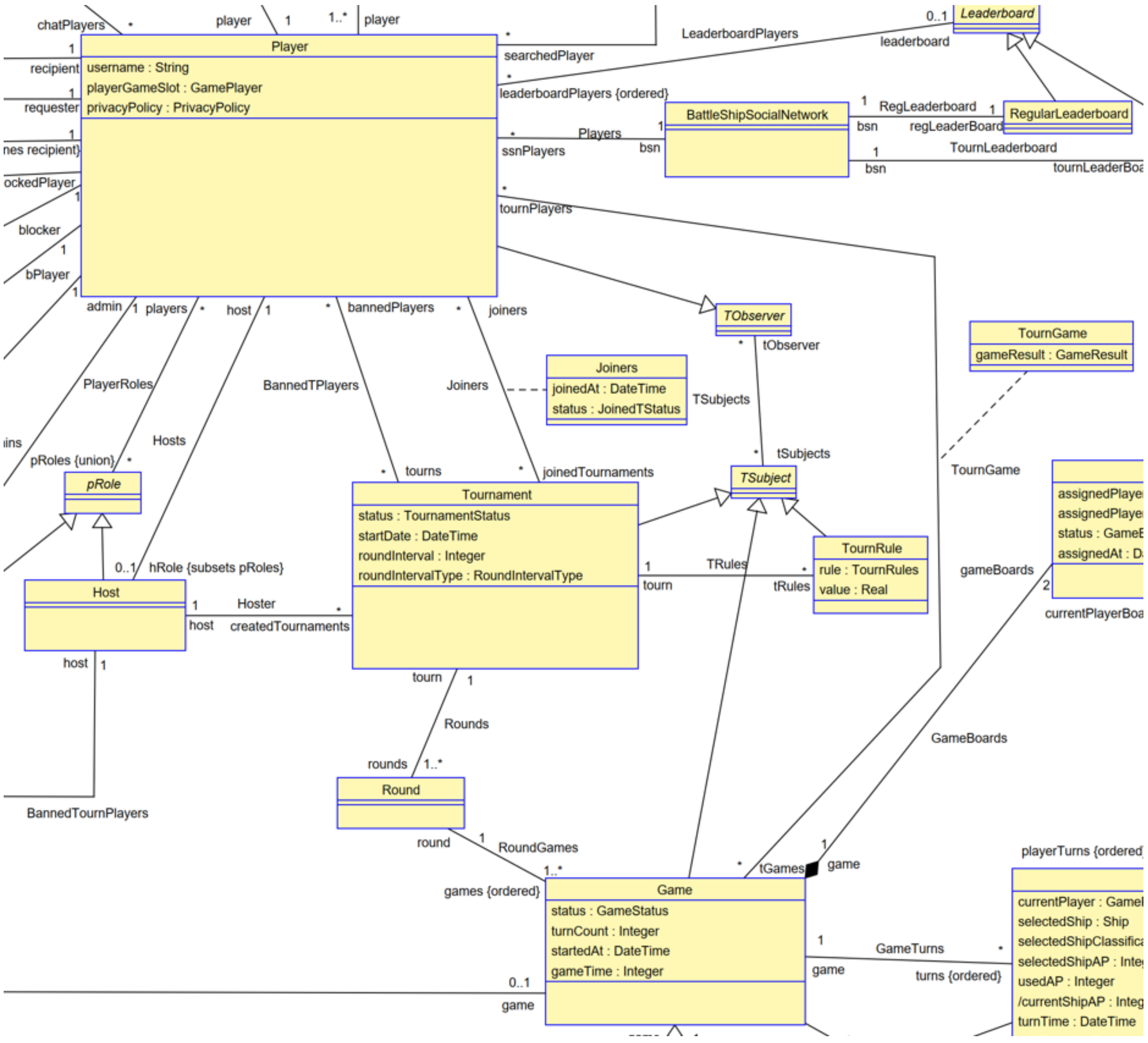
- A ban and a block are both different types of access restrictions and can be seen modelled here as a a subtype of access restriction allowing for operation reuse specific to restricting access.

Leaderboards



- The modelled system only has one regular and tournament leaderboard can exist within the social network. A leaderboard is an ordering of players and we can see this enforced within the class diagram.

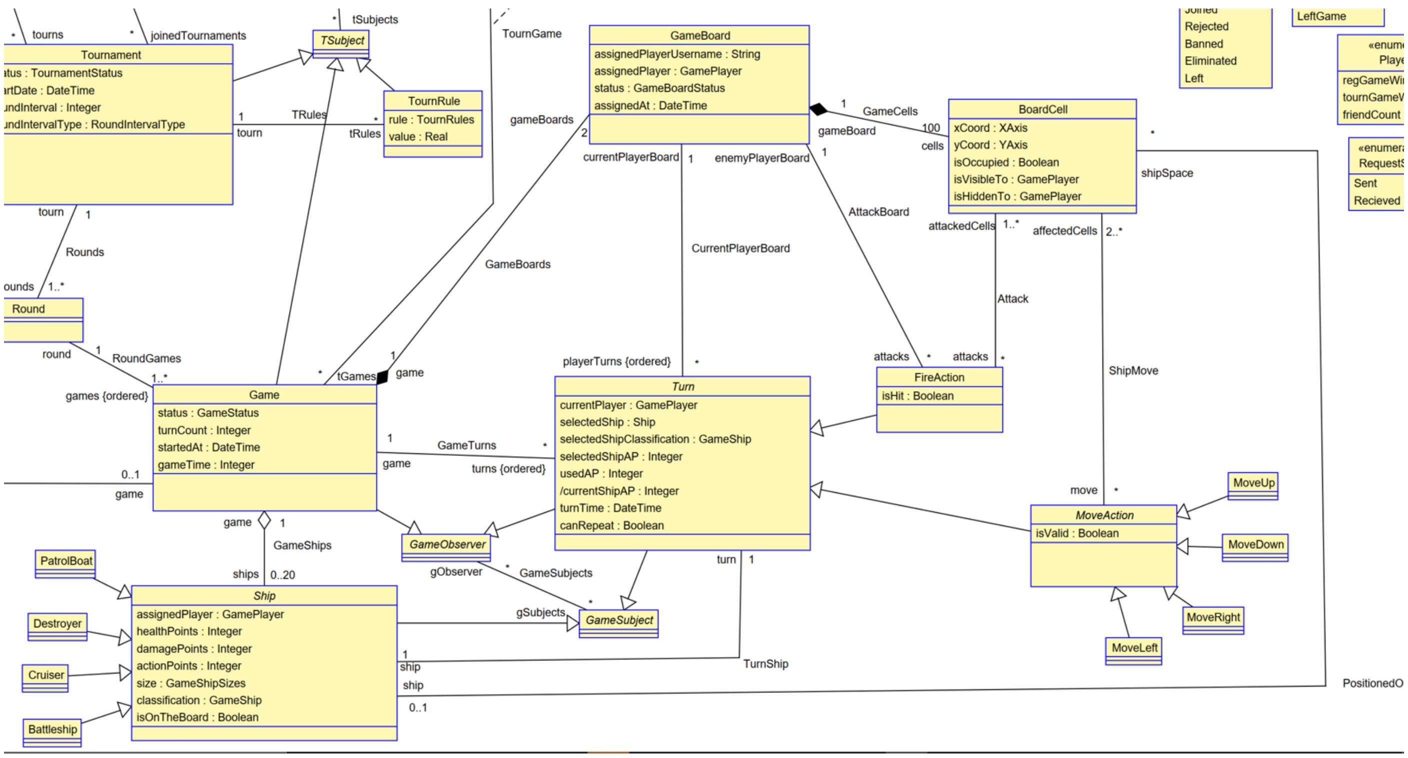
Tournaments



- Tournament access is regulated by ABAC by keeping track of game results for a tournament game for each player. This is enforced using the **TournGame** association class. For more concrete enforcement of this OCL constraints would need to be written.
- The rules around a tournament are enforced using name value pairs to allow for greater modifiability around what rules a tournament host may set.

Games

- Games can start through either tournaments or a game invite in either case the multiplicities enforce that only 2 players can be involved. However in the case of the TournGame association OCL would need to be written to enforce this.



- This representation of the game matches our requirements by allowing a player to make different types of turns, which can be an attack(**FireAction**) or move(**MoveAction**). The classification attribute which is repeated across the ship and turn objects helps to allow checking for which ship a player selects within OCL constraints. It also allows for the placement enforcement of specific ships on the board when intializing a game to ensure that the correct number of ships is placed on each side.

Static Invariants\Constraints

Constraints List

1. Players cannot block themselves
2. Players cannot report themselves
3. Players cannot send a request to themselves
4. Any request must be between only 2 players
5. A private chatroom must only exist between the recipient and requester of a message
6. A player can only send friend requests to a friend
7. A player cannot search for themselves
8. A player cannot be in 2 games at once
9. A private game can only be between 2 players
10. A player who joins a tournament can only join tournament games within that tournament.
11. A banned player cannot join a tournament
12. A host cannot participate in a tournament
13. An eliminated player cannot advance in a tournament
14. A game must have 2 gameboards
15. A game must have 200 cells in total with 100 cells per gameboard
16. A game must be initialized with the correct amount of ships and only 2 players
17. A game cannot start before a player's account was created
18. An even numbered turn is played by player 1 and an odd numbered turn is played by player 2

19. A turn can only repeat if the player has not missed or there is AP remaining for the selected ship.
20. A ship can only move once per turn.
21. Ships can only be placed vertically
22. A ship cannot move into another ship
23. A ship can move 1 cell up, 1 cell down or transform to the left or right
24. A ship can only move on the current player's board
25. A ship may only fire if it has the required amount of action points
26. A ship cannot fire on it's own board
27. A ship can only fire on the enemy board
28. The gameboards must be assigned to the players
29. The game board must have 10 ships on both boards when it is initialized. 20 Cells are occupied in total on the board when all the ships are placed
30. A cell can only be visible to a player if they own that gameboard and hidden if the player does not own that gameboard.

Enforced By

Constraint	Class Diagram Enforcement
Any request must be between only 2 players	This is enforced by the following associations - Recipients - Requests - RequesterFact The strict allowance of a request to only have one requester and one recipient allows there to only be 2 players participating in any request
A private game can only be between 2 players	The GameInvite class is the link between a

	requester and a recipient to a game. This allows only 2 players to be given access to a game instance
A game must have 2 gameboards	This is enforced by the GameBoards composition
A game must have 200 cells in total with 100 cells per gameboard	This is enforced by the GameCells composition and further enforced by the GameBoards composition.

Constraint	OCL
A player cannot block themselves	context Player --- Player cannot block themselves inv: block->forAll(p|p.blockedPlayer<>self)
A player cannot send a friend request to themselves	context Player --- Player cannot send a request to themselves inv reqFacts.sentRequest->asSet().recipient->forAll(r|r<>self)
A player cannot report themselves	context Player --- Player cannot report themselves inv recievedRs.recipient->ForAll(r|r<>self)
A ship may only fire if it has the required amount of action points	`inv validFire: let ship = game.ships->select(s| s.assignedPlayer=self.currentPlayer and s.classification=self.selectedShip)->asOrderedSet()->last() in ship.actionPoints>0`
The game board must have 10 ships on both boards when it is initialized. 20 Cells are occupied in total on the	`inv validInitShipCount: (status=GameBoardStatus::Assigned and game.status=GameStatus::Initialized)`

board when all the ships are placed	
	```
	            implies

	cells->select(c|c.ship->size()=1)->size()=20
	```

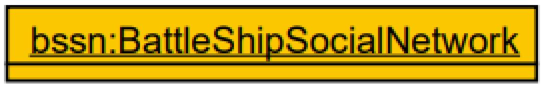
Ships should be placed vertically only	
	```
	    inv validShipPlacement:
	    let
	        y = OrderedSet{YAxis::A, YAxis::B, YAxis::C,
	YAxis::D, YAxis::E, YAxis::F, YAxis::G, YAxis::H,
	YAxis::I, YAxis::J},
	        x = OrderedSet{XAxis::y1, XAxis::y2, XAxis::y3,
	XAxis::y4, XAxis::y5, XAxis::y6, XAxis::y7, XAxis::y8,
	XAxis::y9, XAxis::y10}
	    in
	    ((status=GameBoardStatus::Assigned
	    and
	    game.status=GameStatus::Initialized or
	game.status=GameStatus::Started))
	        implies
	            game.ships->forAll(s|
	                            s.shipSpace->forAll(ss1,
	ss2|

	x->indexOf(ss1.xCoord)=x->indexOf(ss2.xCoord)
	                                and

	(y->indexOf(ss1.yCoord)+1 = y->indexOf(ss2.yCoord)
	                                    or

	y->indexOf(ss1.yCoord)+2 = y->indexOf(ss2.yCoord)
	                                    or

	y->indexOf(ss1.yCoord)+3 = y->indexOf(ss2.yCoord)
	                                    or

	y->indexOf(ss1.yCoord)+4 = y->indexOf(ss2.yCoord)
	                                    or

	y->indexOf(ss1.yCoord)-1 = y->indexOf(ss2.yCoord)
	```

```
                                    or

y->indexOf(ss1.yCoord)-2 = y->indexOf(ss2.yCoord)
                                    or

y->indexOf(ss1.yCoord)-3 = y->indexOf(ss2.yCoord)
                                    or

y->indexOf(ss1.yCoord)-4 = y->indexOf(ss2.yCoord)
                                    )
                                ))
```

Object Models

Sign Up

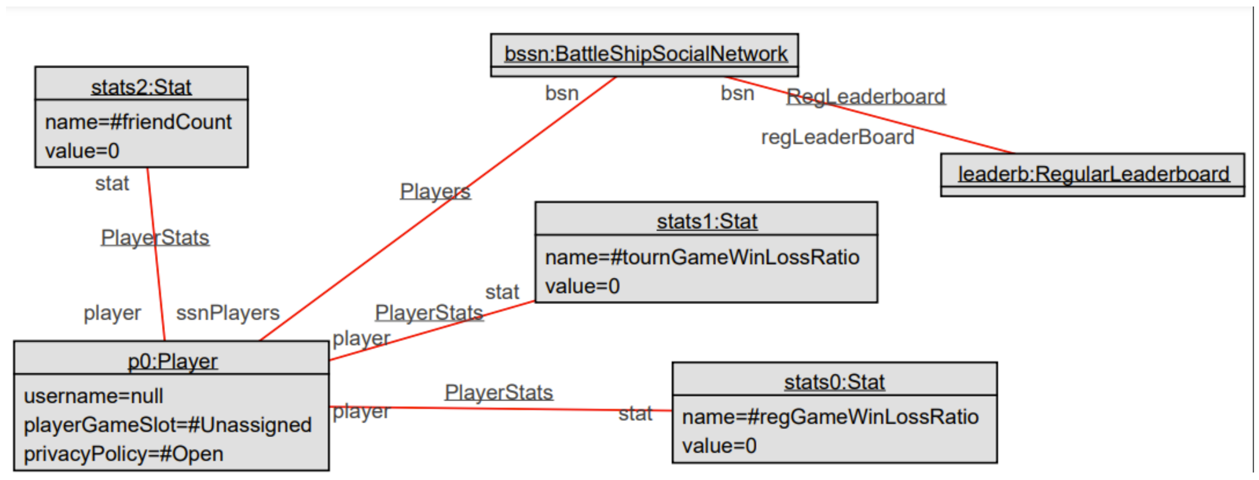
Before



Description

Before Signing in the player object doesn’t exist only the social network.

After



Description

After the player signs in the player object is created and is admitted to the social network and the player stats and leaderboard are intialized.

Login

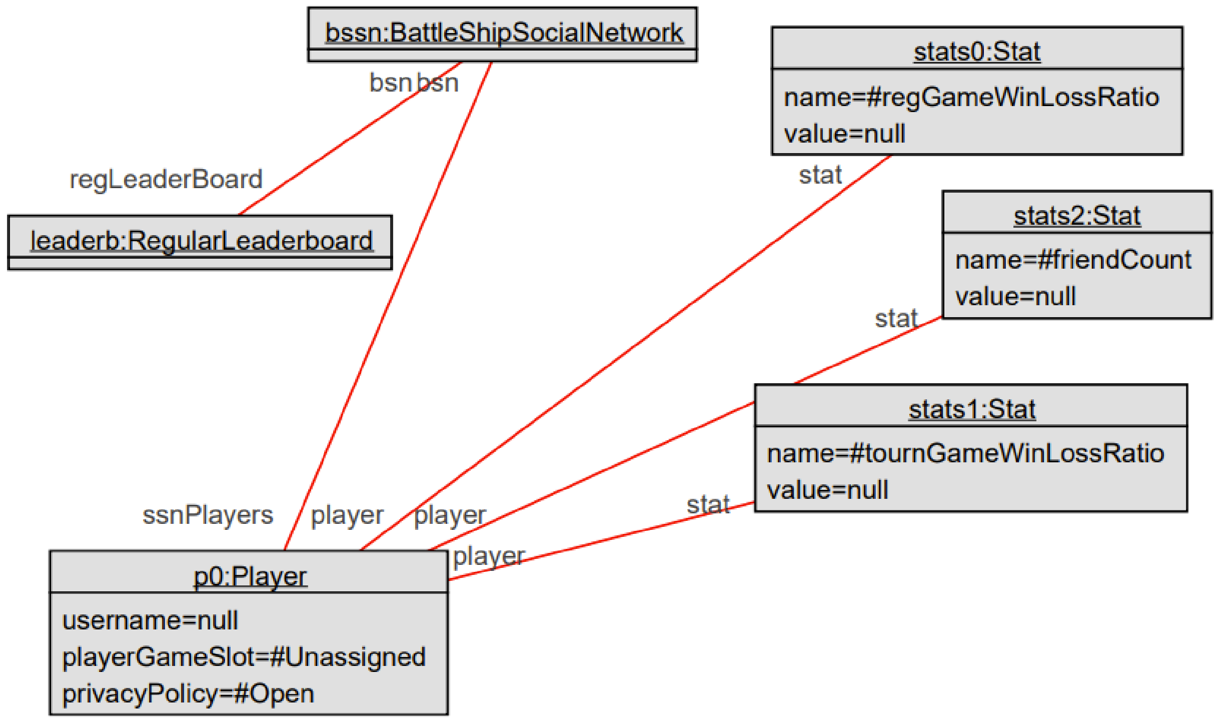
Before

bssn:BattleShipSocialNetwork

Description

Before Loging in the player object doesn’t exist only the social network.

After

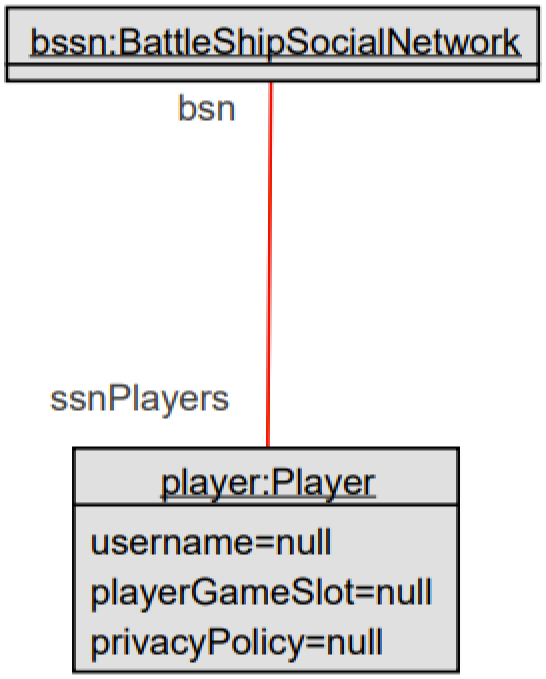


Description

After the player logs in the player object is created and is admitted to the social network and the player stats and leaderboard are intialized.

Search for another player

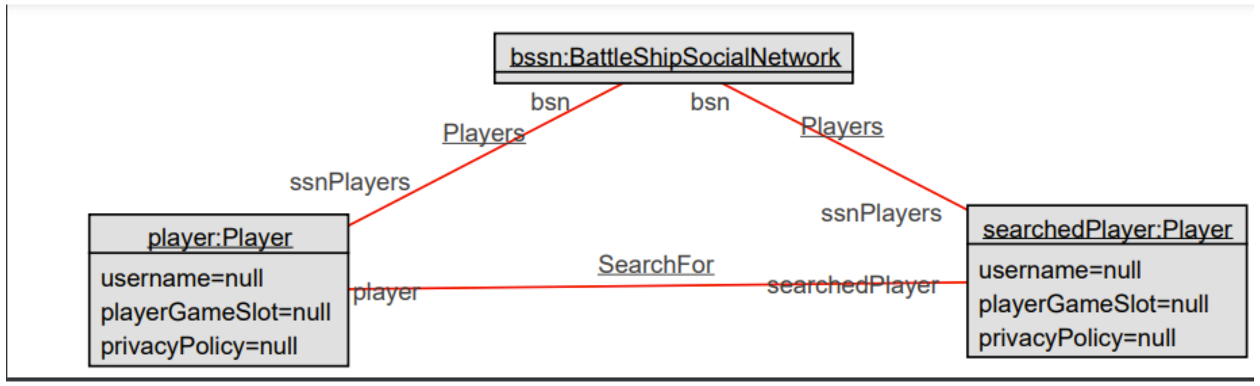
Before



Description

Before searching for the player only the player and the social network exists. The player needs the social network to access other players.

After

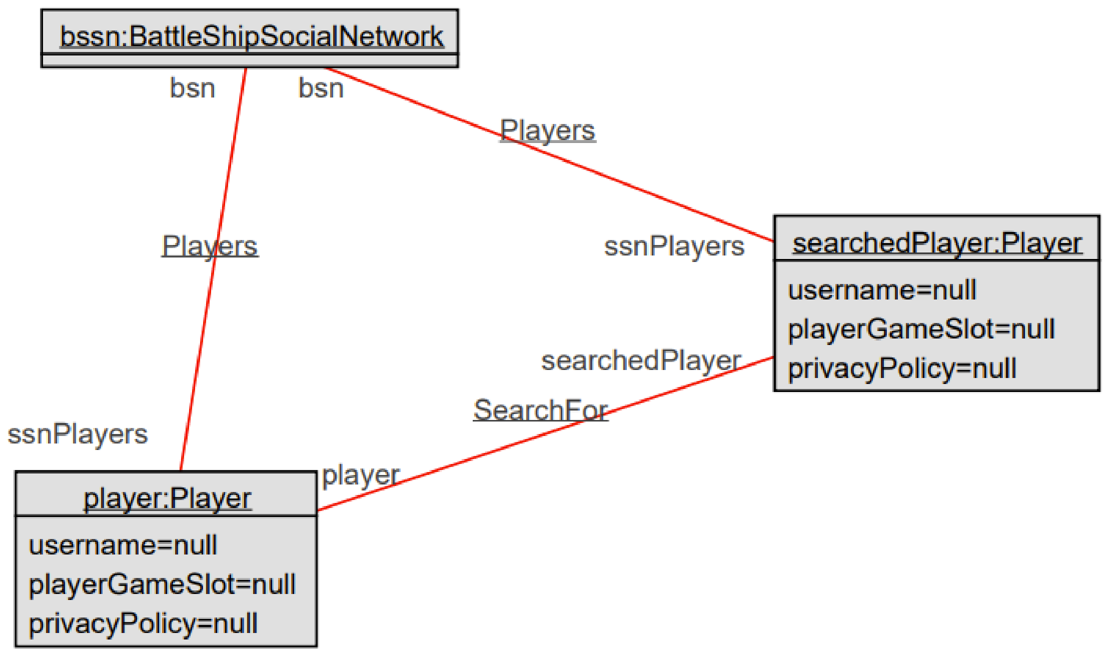


Description

After searching successfully, the searched player is instantiated into the system. They must have been both been apart of the social network for the search to be successful

Send a Friend Request

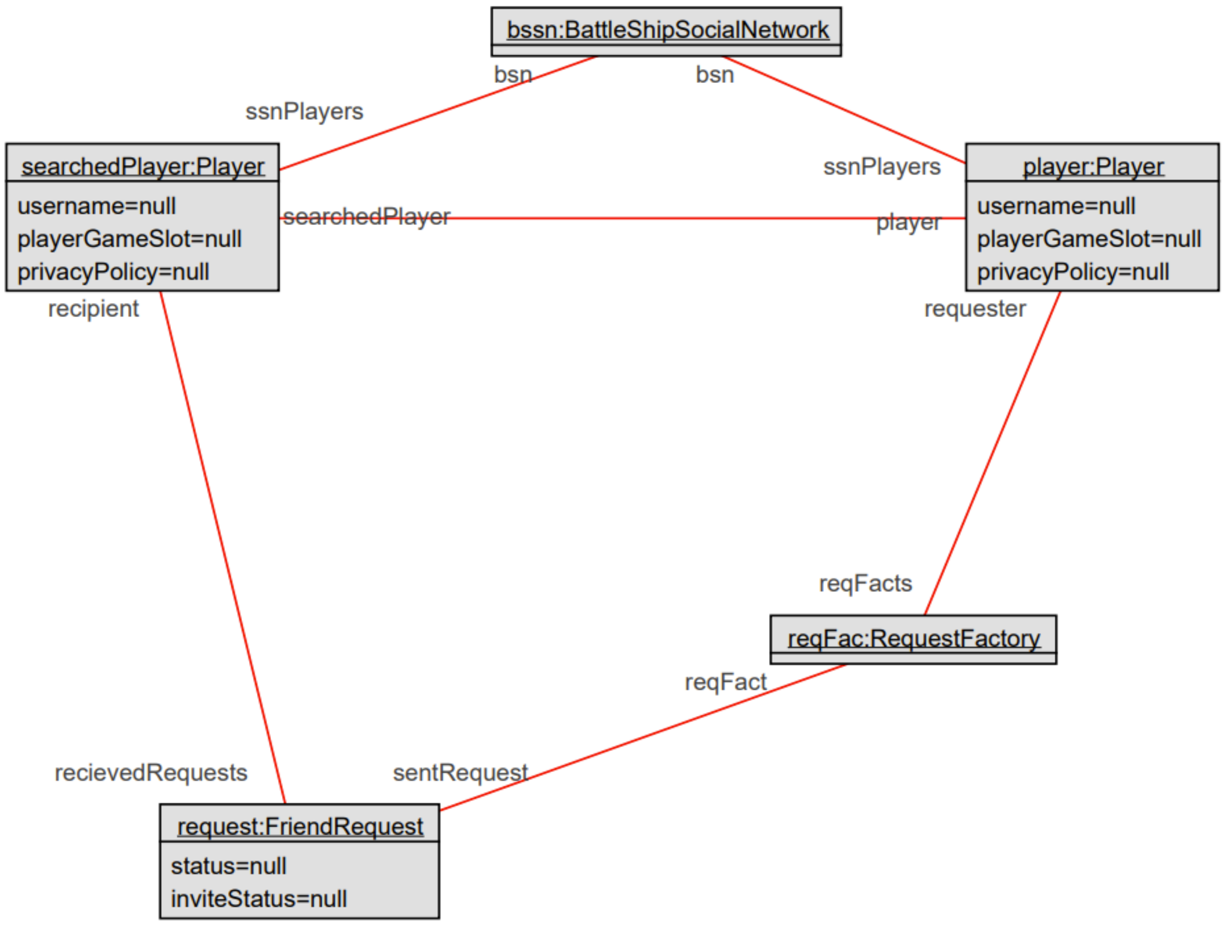
Before



Description

The player searches for the player he wants to send the friend-request to within the social network.

<u>After</u>

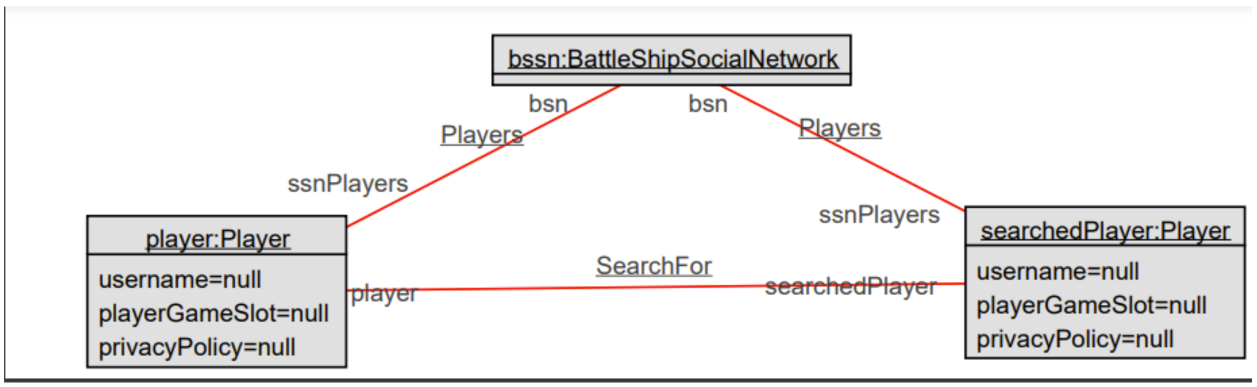


Description

After sending the friend request the request is instantiated with the player as the requester and the searched Player as the recipient

Block a player

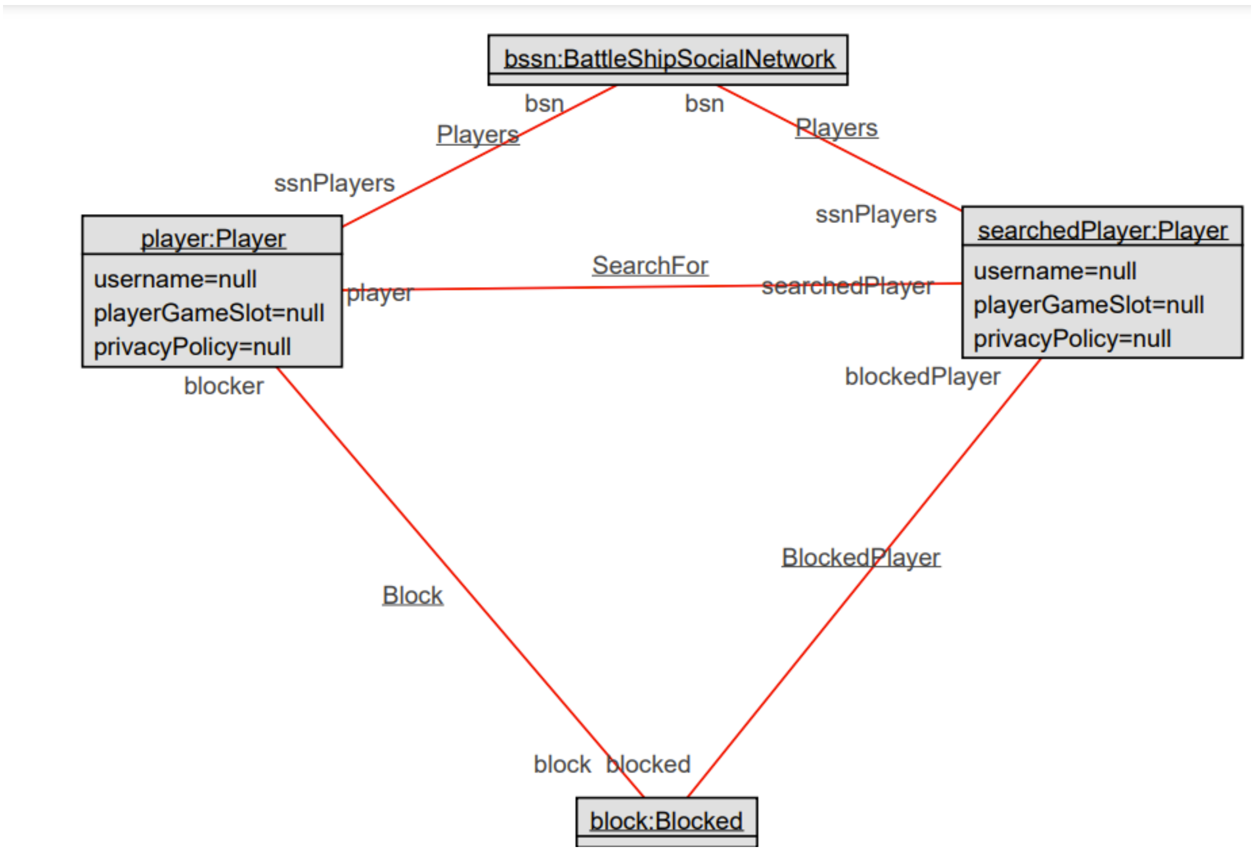
Before



Description

The player searches for the player he wants to block.

<u>After</u>

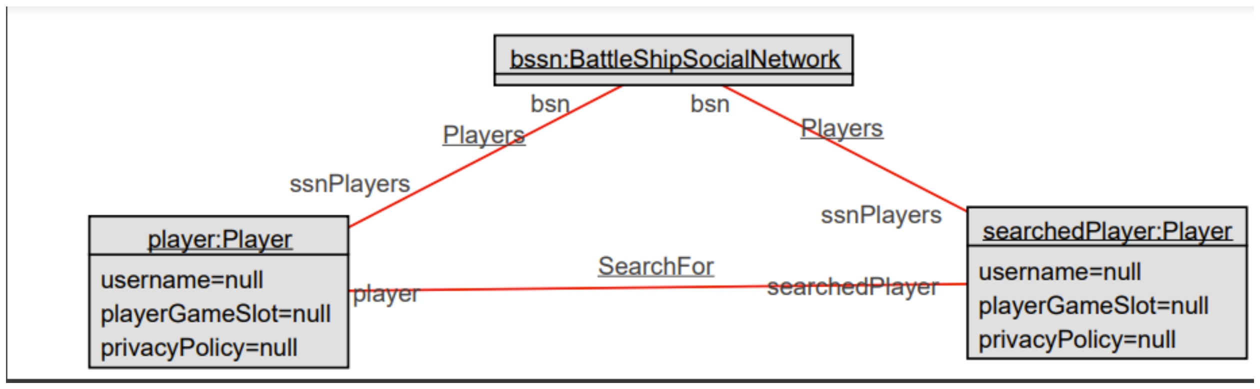


Description

After blocking the searched player, the relationship between the player and searched player is "blocked"

View a player’s details

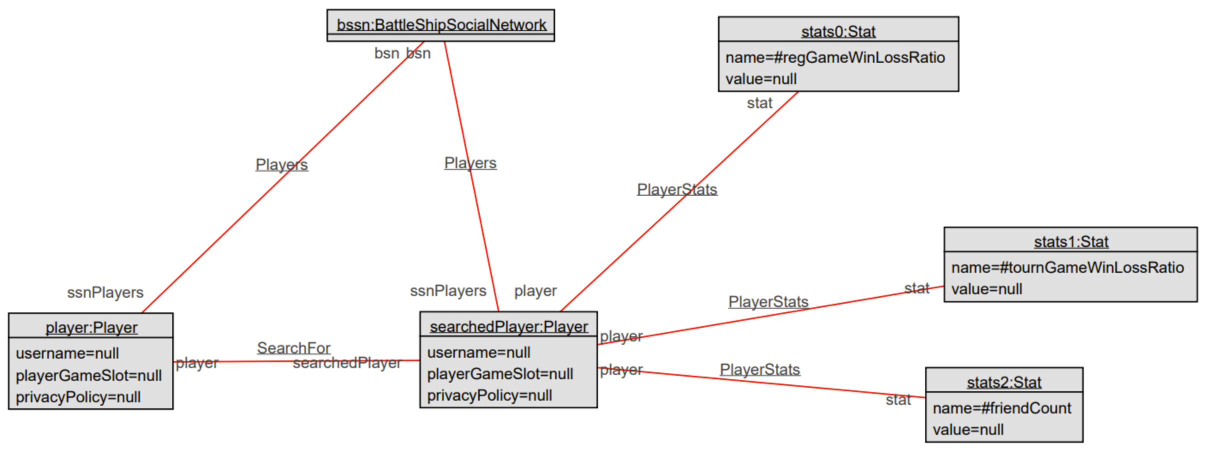
Before



Description

The player searches for the player he wants to finds details from.

After

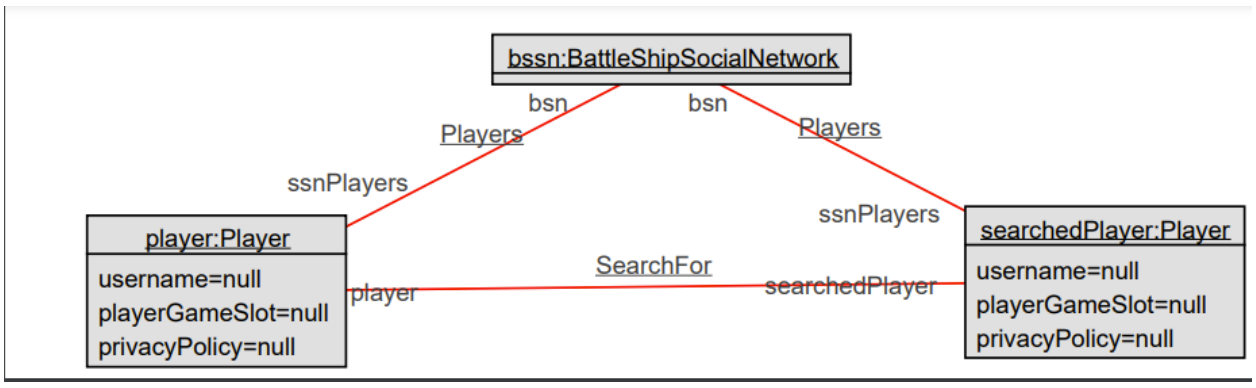


Description

The searched player’s details are instatntiated so the player can access them.

Report a player

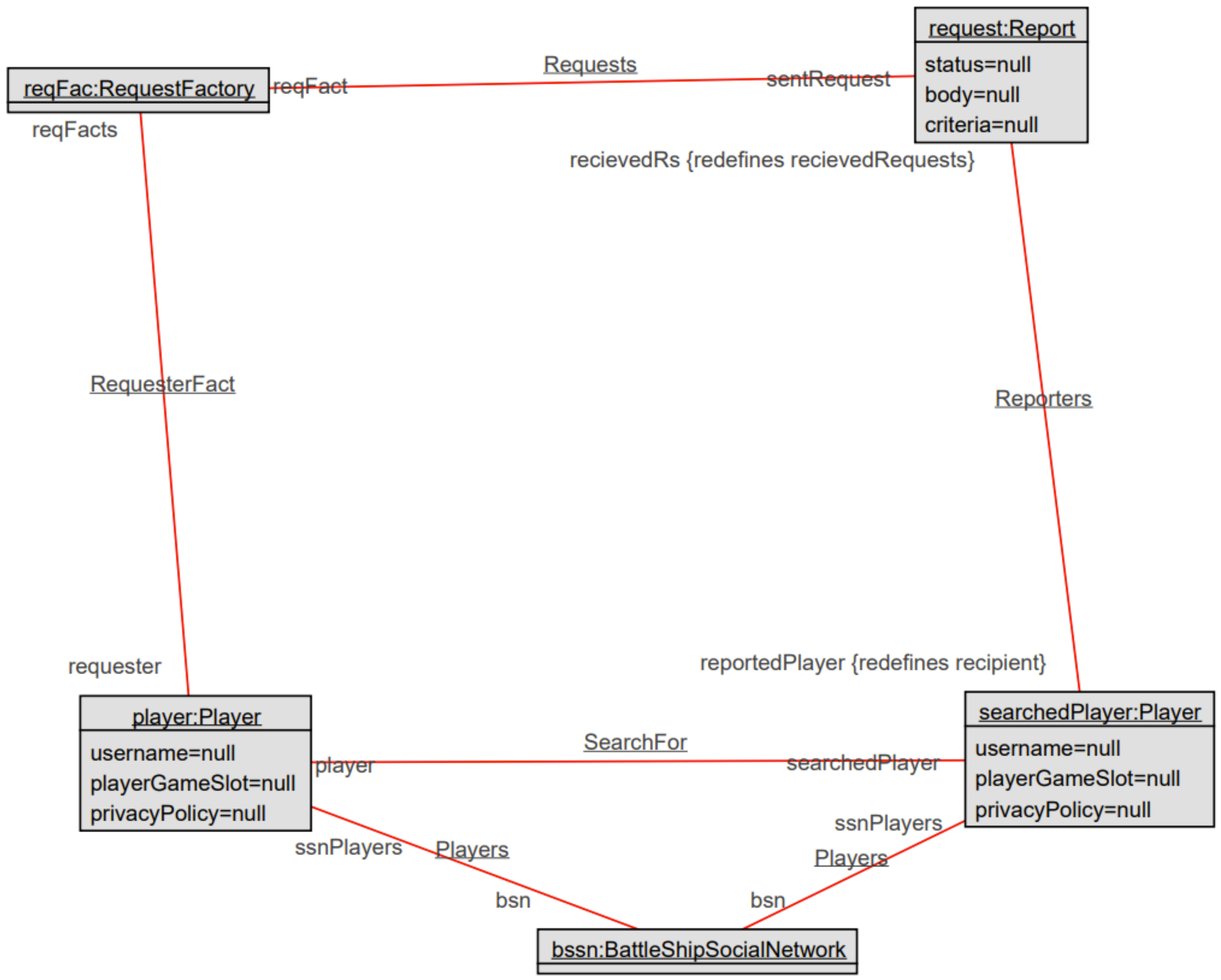
Before



Description

The player searches for the player he wants to report

After

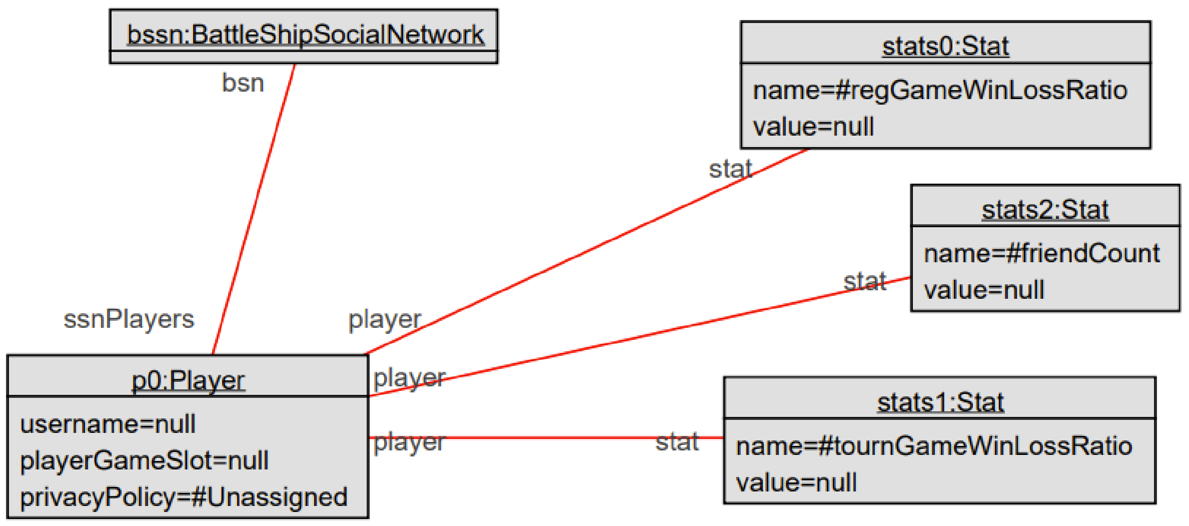


Description

The player's report request is instantiated and the searched player is now reported.

Set Privacy Settings

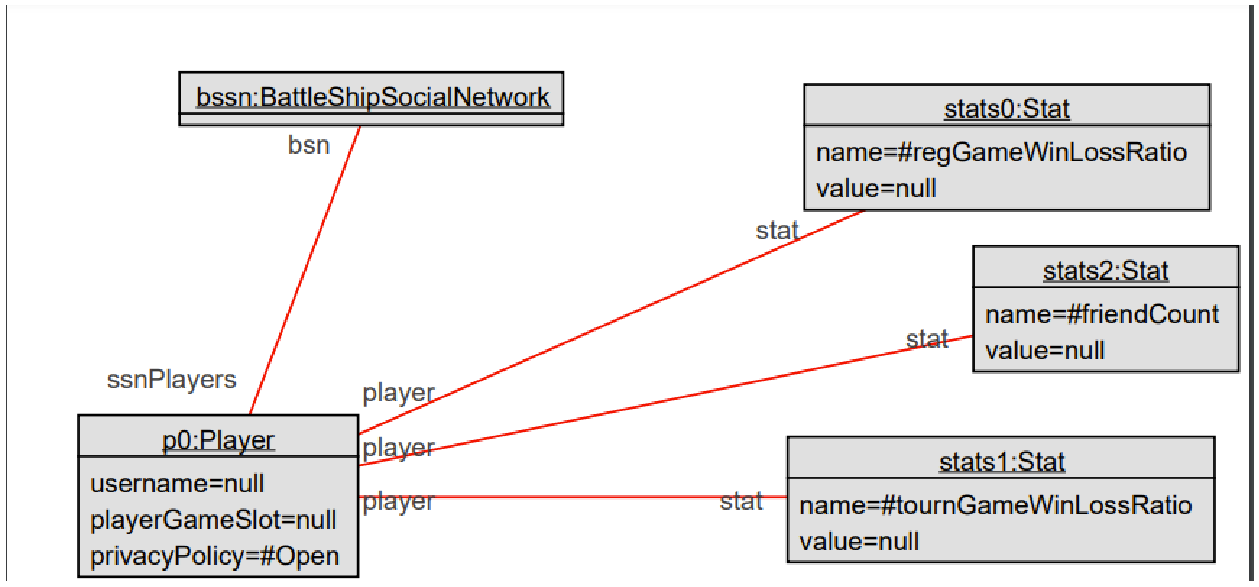
Before



Description

The privacy policy of the player is unassigned.

<u>After</u>

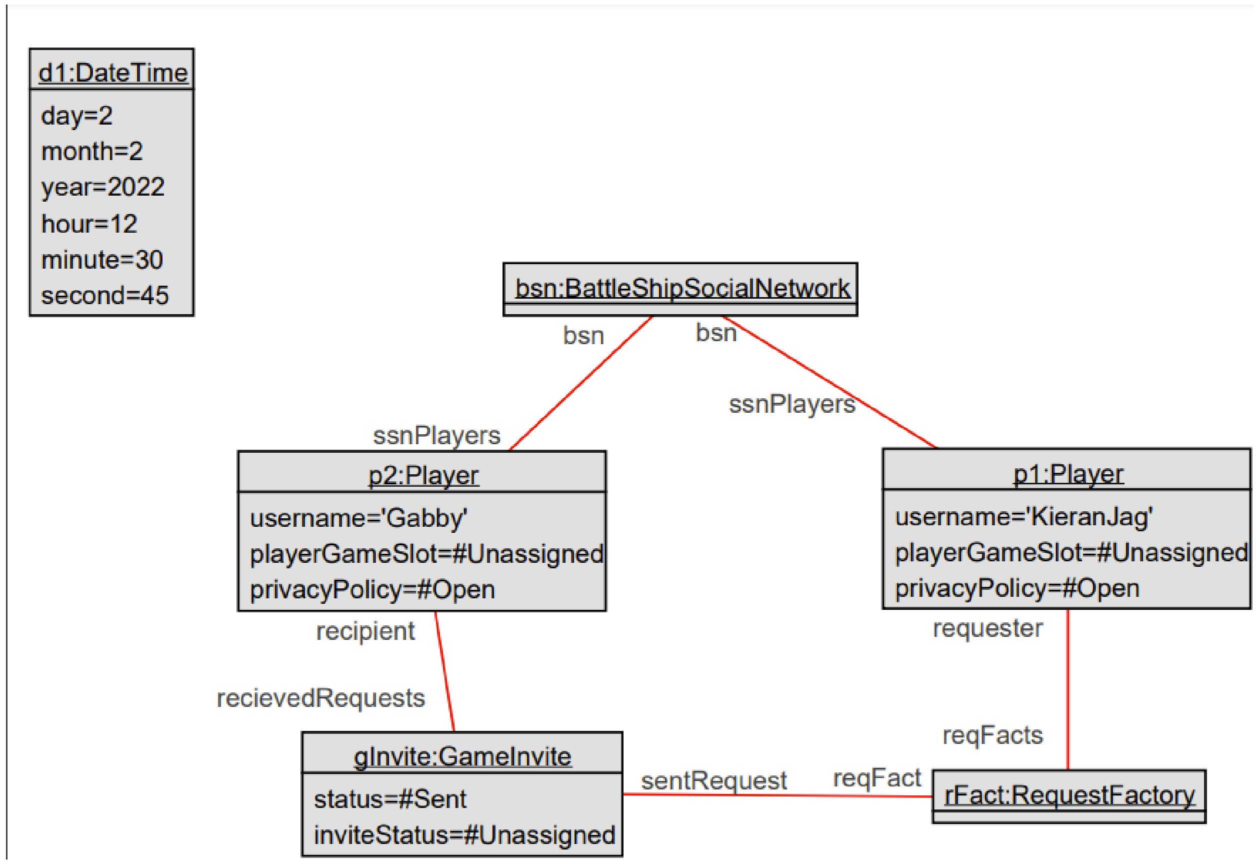


Description

The privacy policy of the player is now assigned and has the value “Open”.

Recieve Game Invite

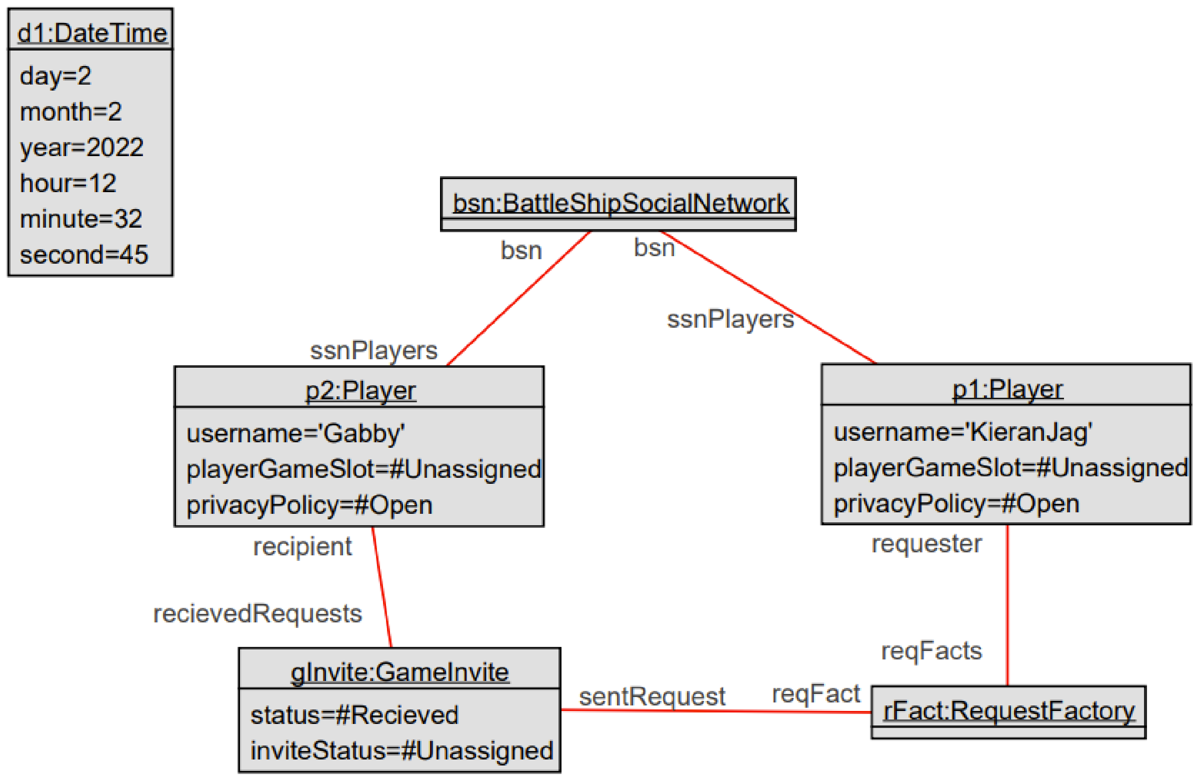
Before



Description

Player 1 sends the game invite however Player 2 has not received it as seen by the status of the game invite “gInvite”.

After

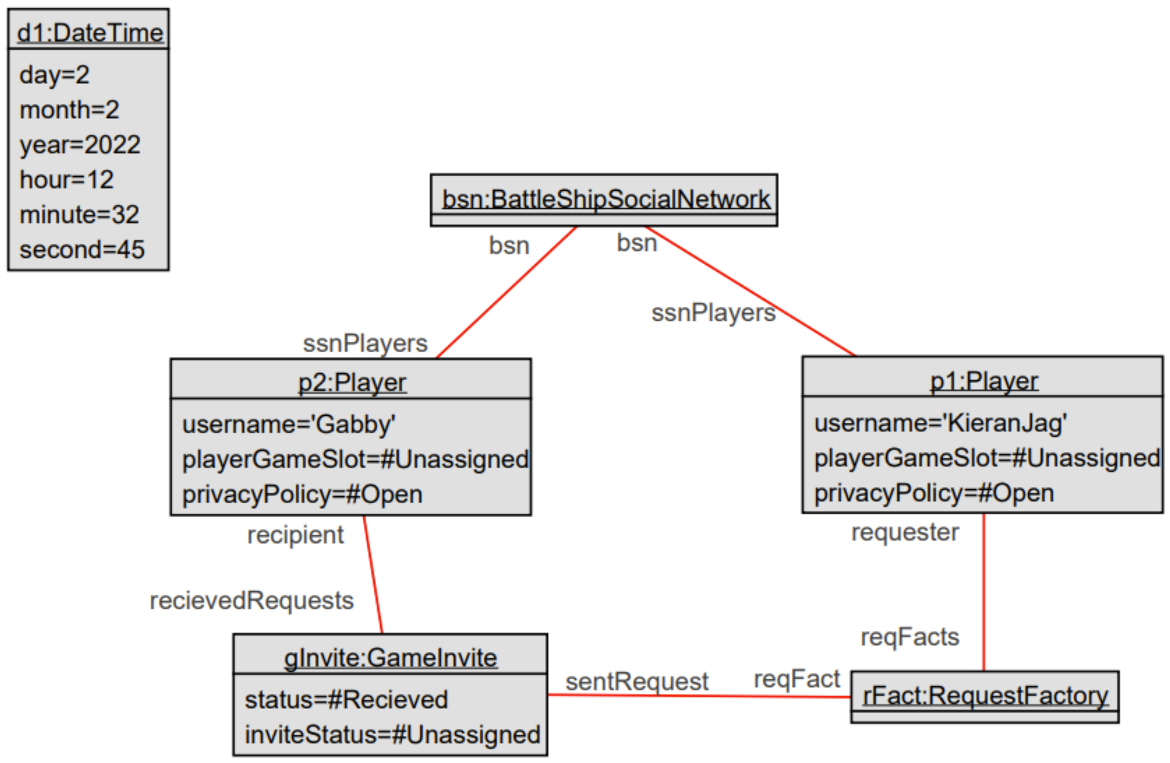


Description

Player 2 now receives the invite as dictated by the status “Recieved” on the game invite

Accept Game Invite

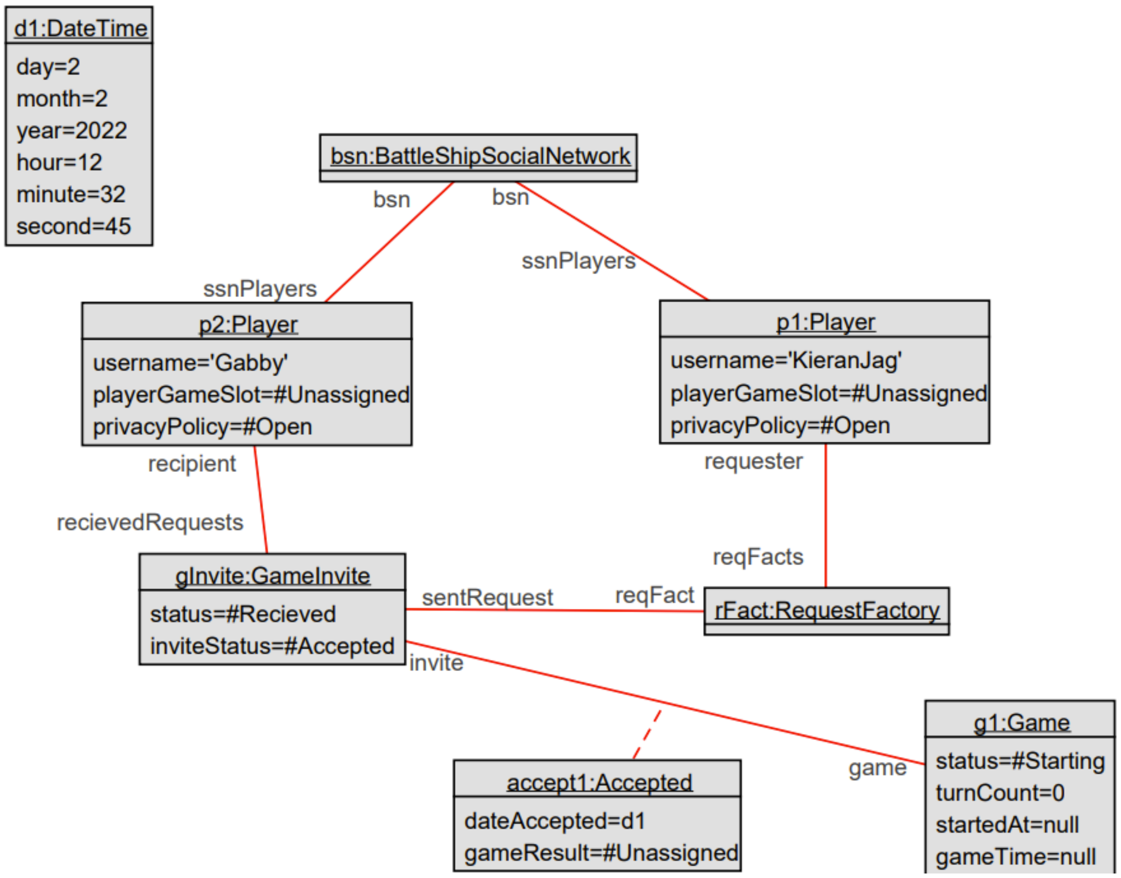
Before



Description

The Player 1 has sent the game invite and Player 2 has received it (as seen by the status ‘Recieved’) but has not taken any action with the invite (as seen by the status ‘Unassigned’)

After

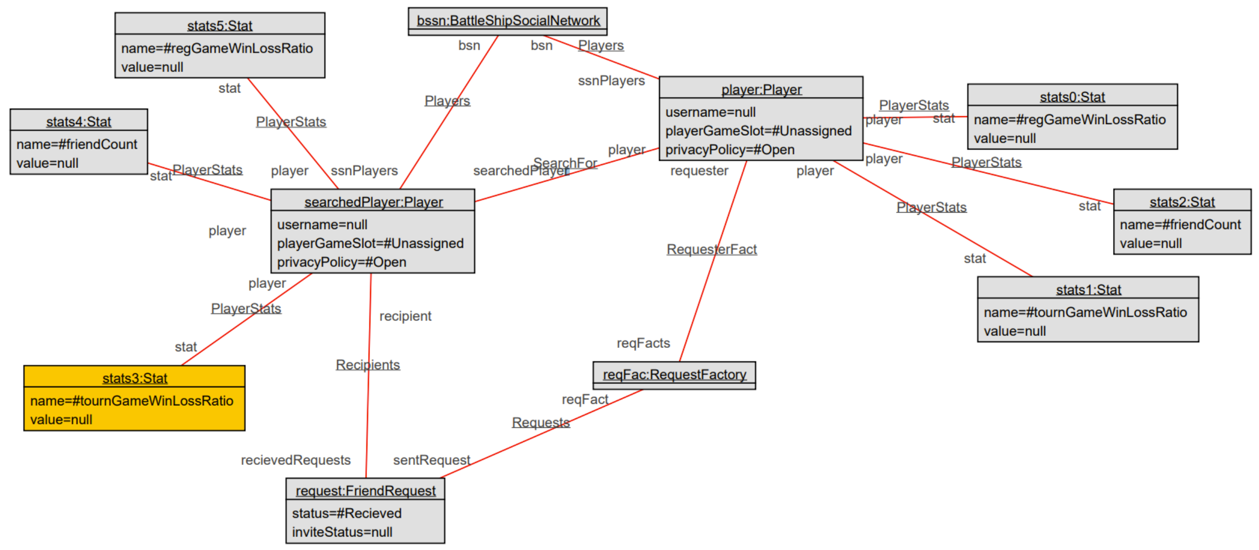


Description

Player 2 has accepted the game invite as seen by the invite status of ‘Accepted’

Accept Friend Request

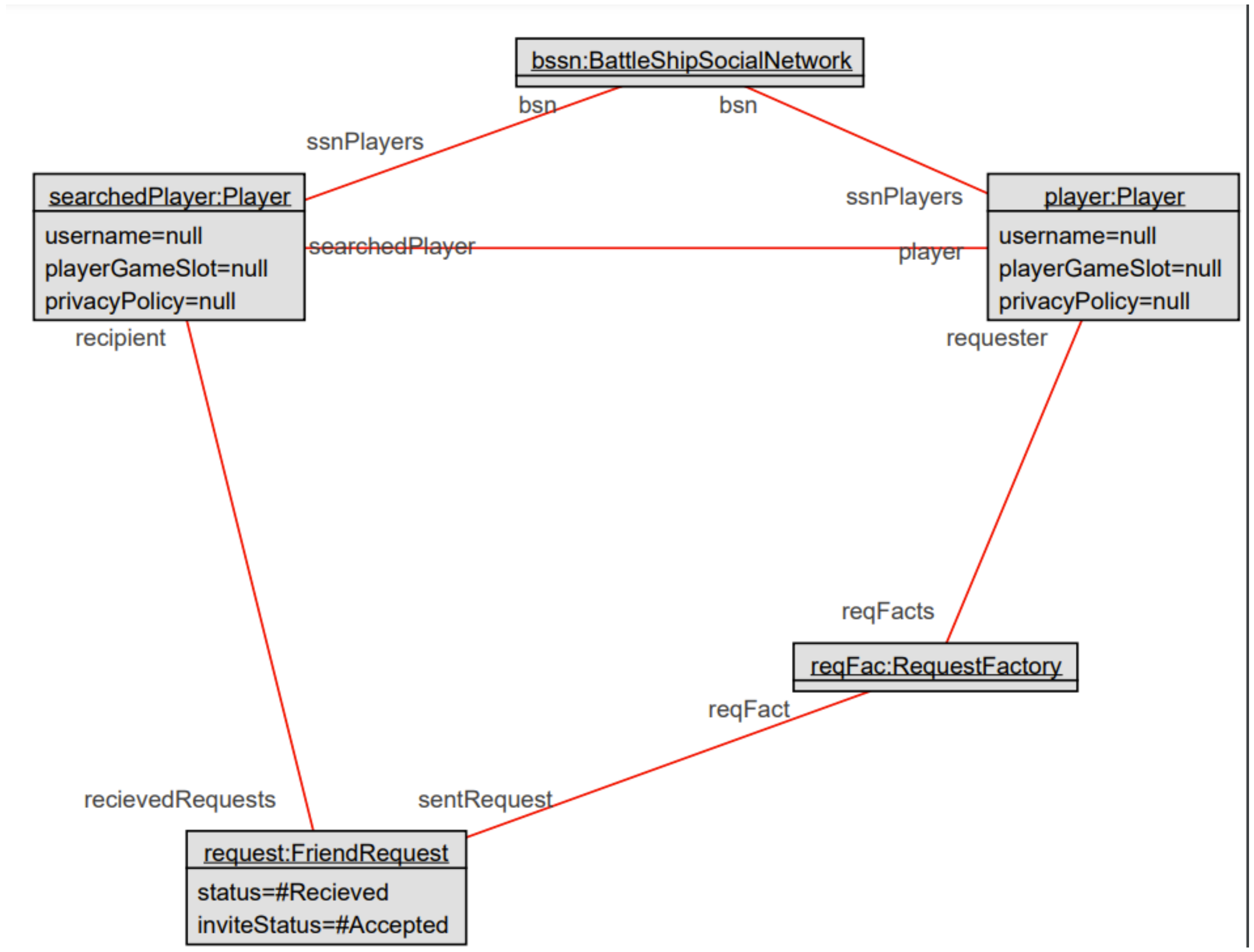
Before



Description

Player has sent the friend request to the searched player. It has been received by the searched player as noted by the status being ‘Recieved’.

After

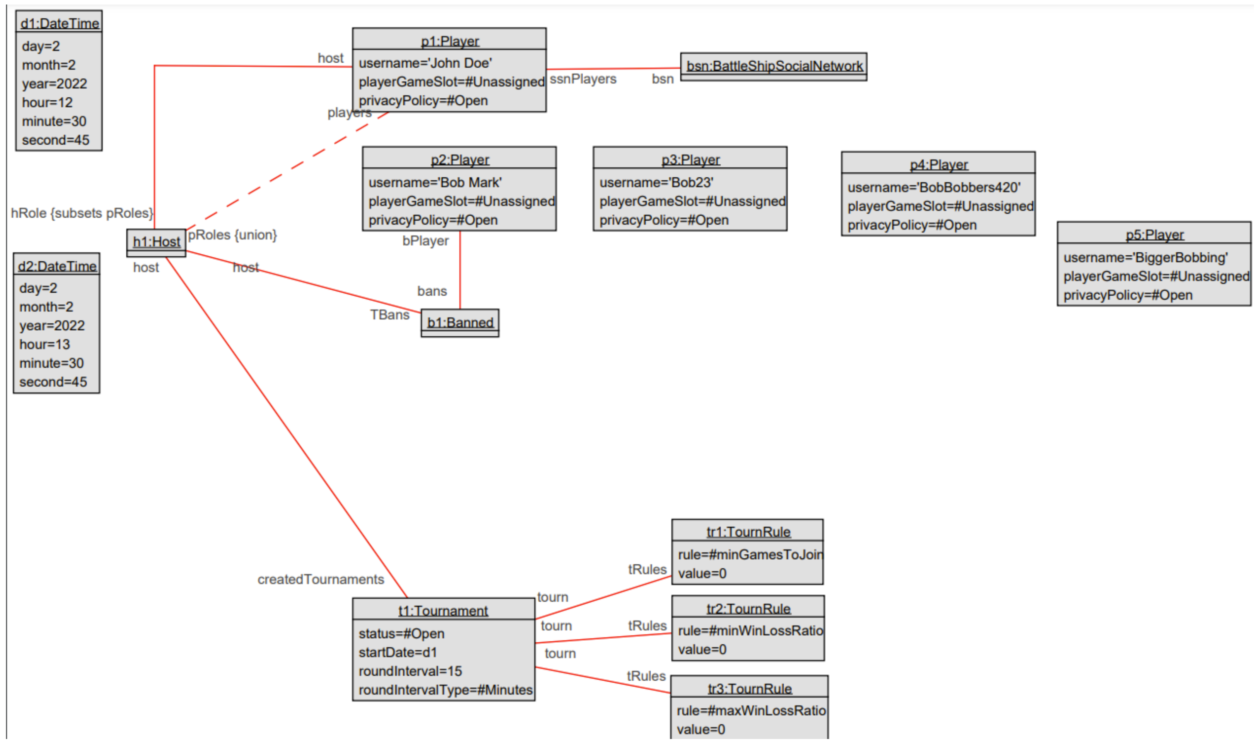


Description

The Searched Player has accepted the invite as seen by the invite status being ‘Accepted’

Register for Tournament

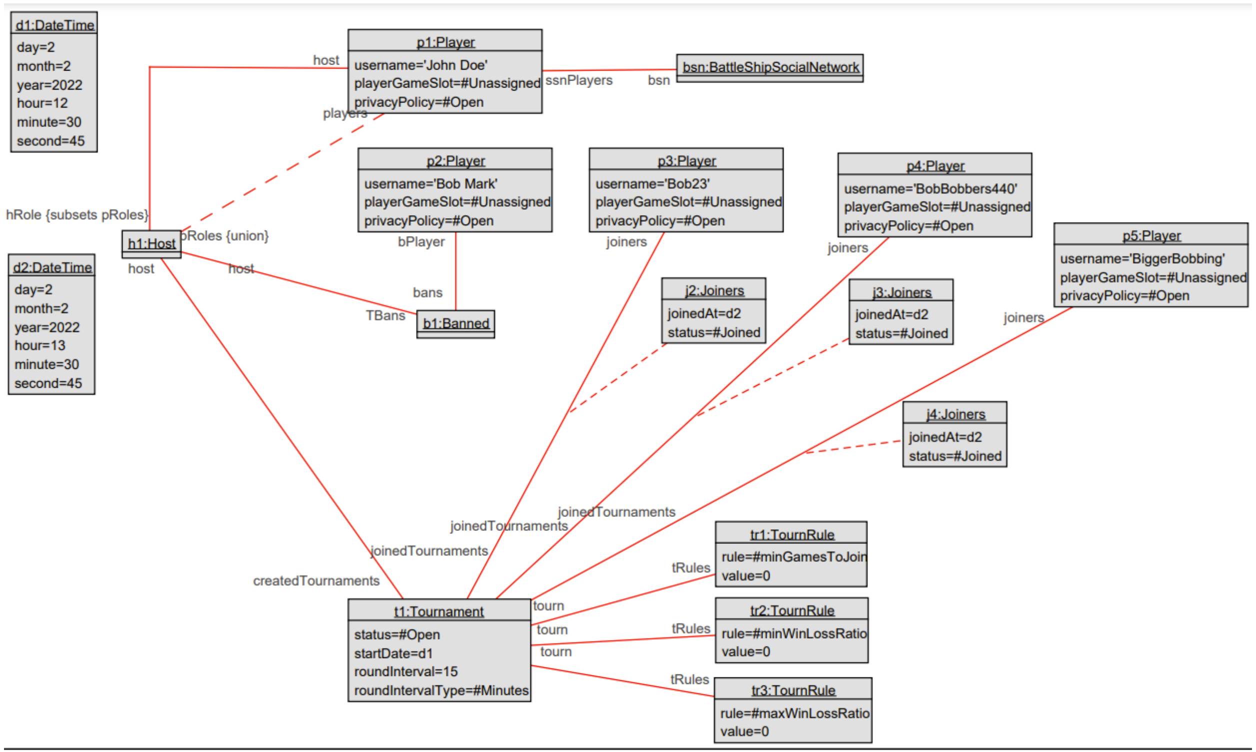
Before



Description

The players are online in the social network and a host has created a tournament with their specified rules

After

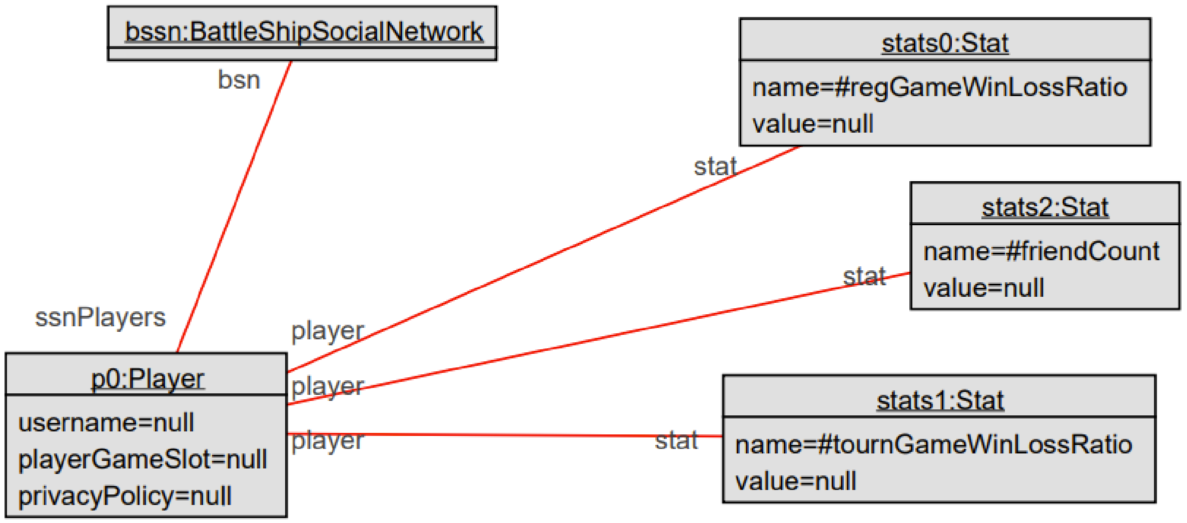


Description

The players have joined the tournament and banned a player from accessing the tournament

View Leaderboard

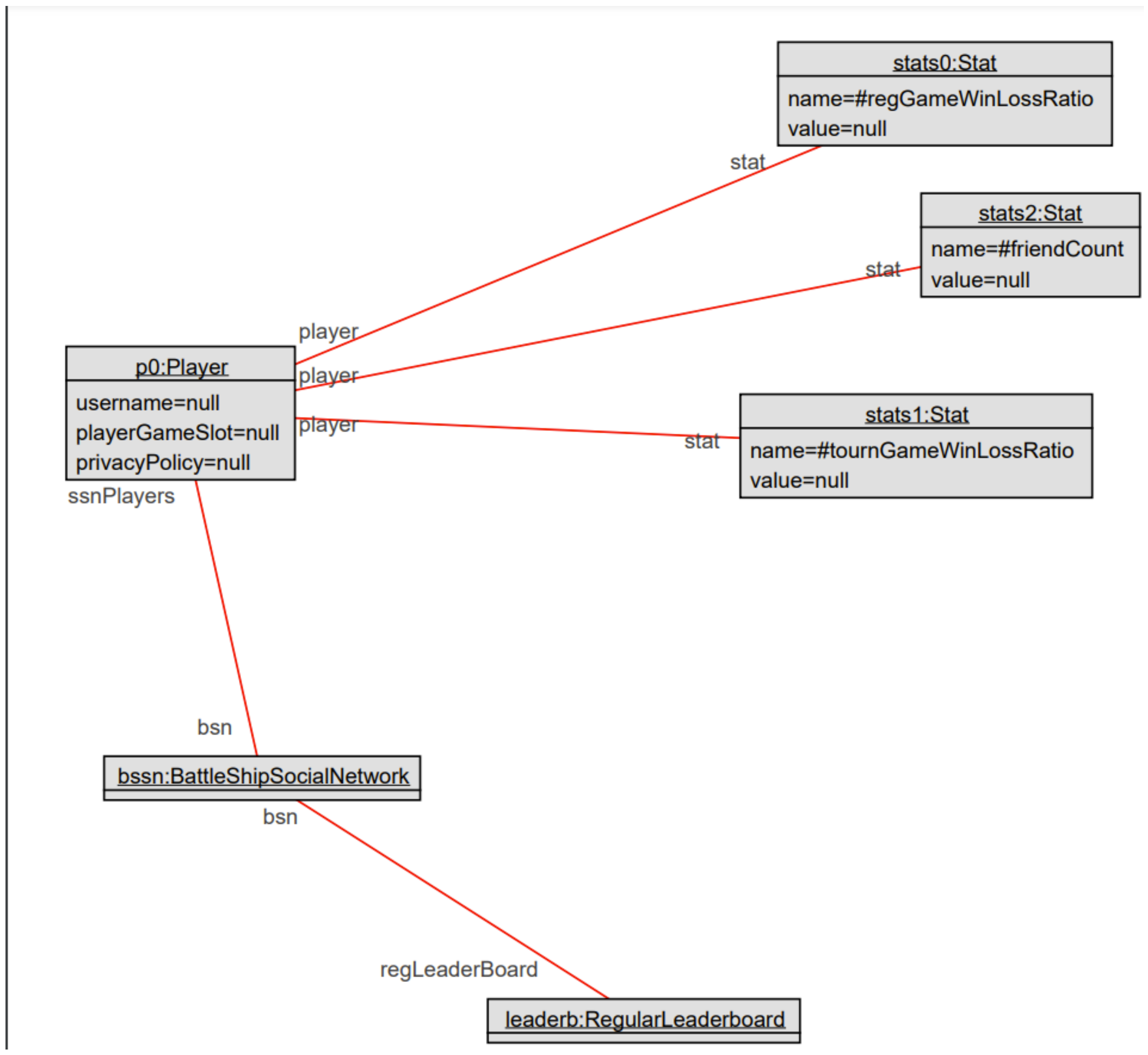
Before



Description

The leader board is not available for the player to see .

<u>After</u>

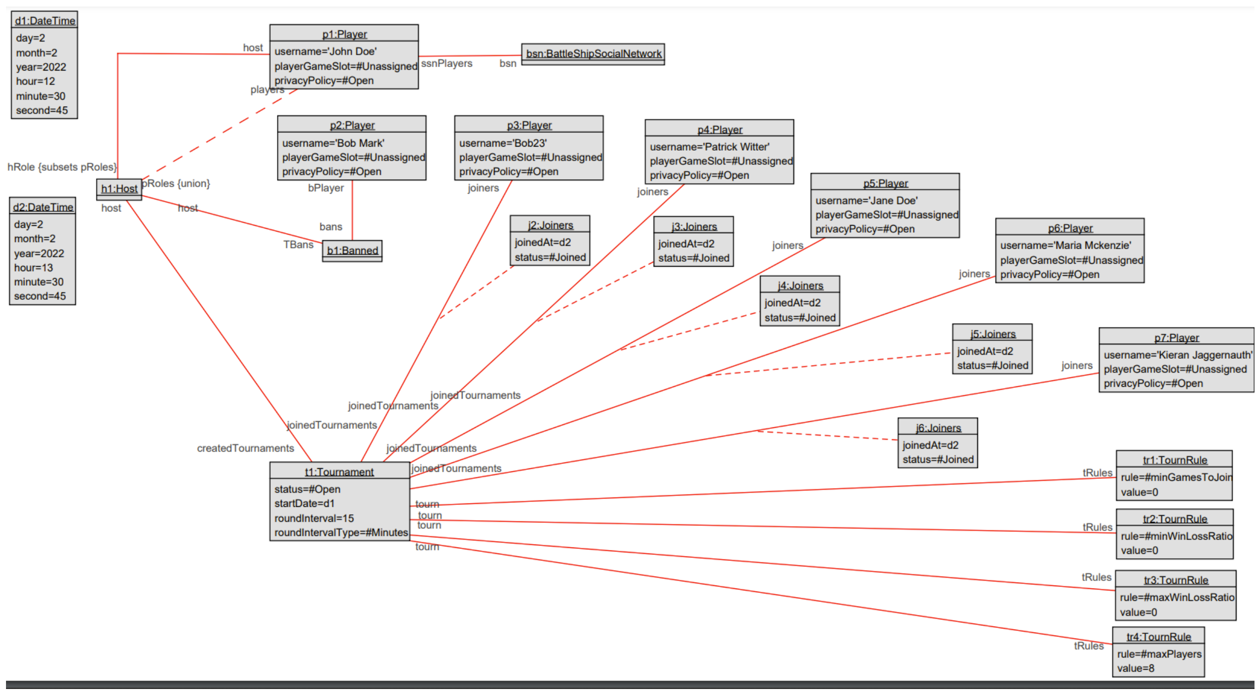


Description

The player can now access the regular leaderboard through the social network

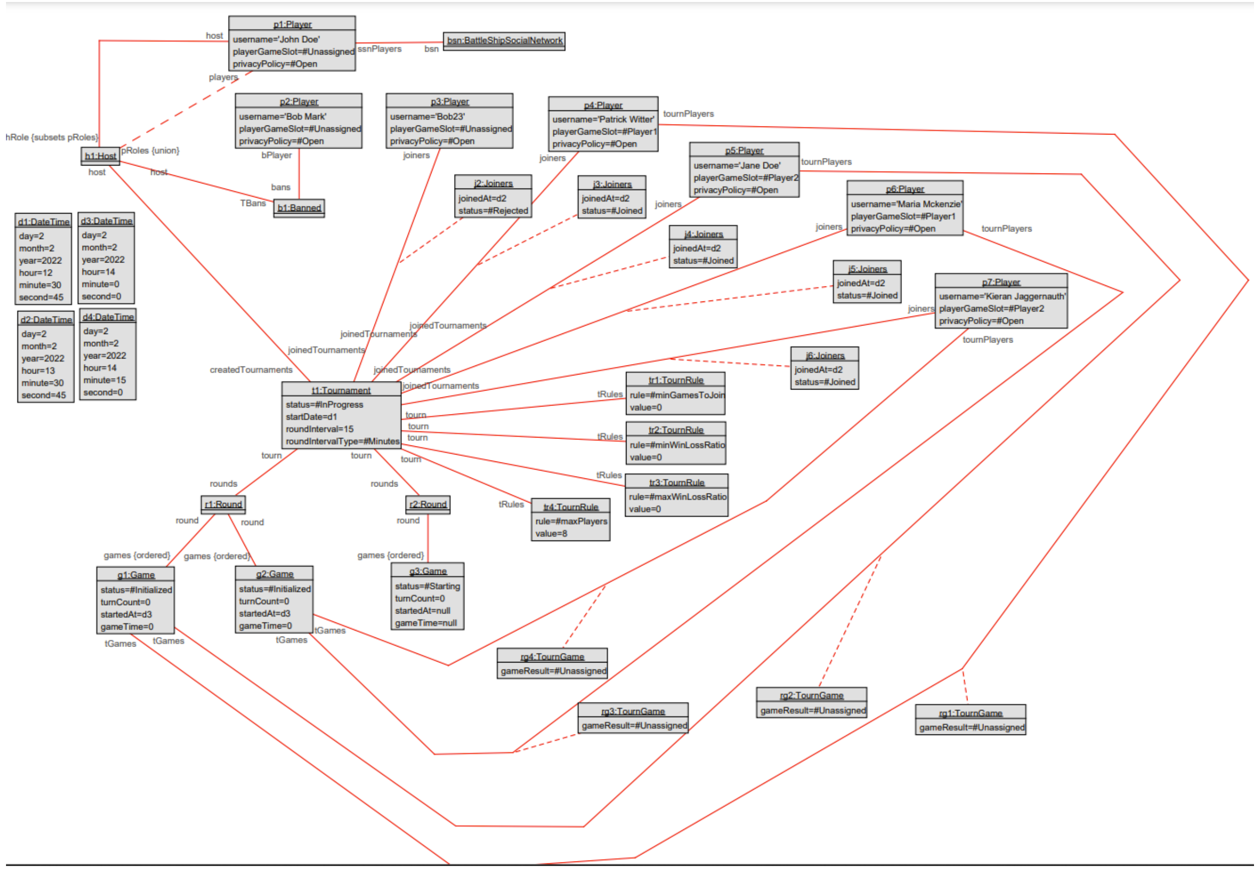
Start Tournament

Before



Description

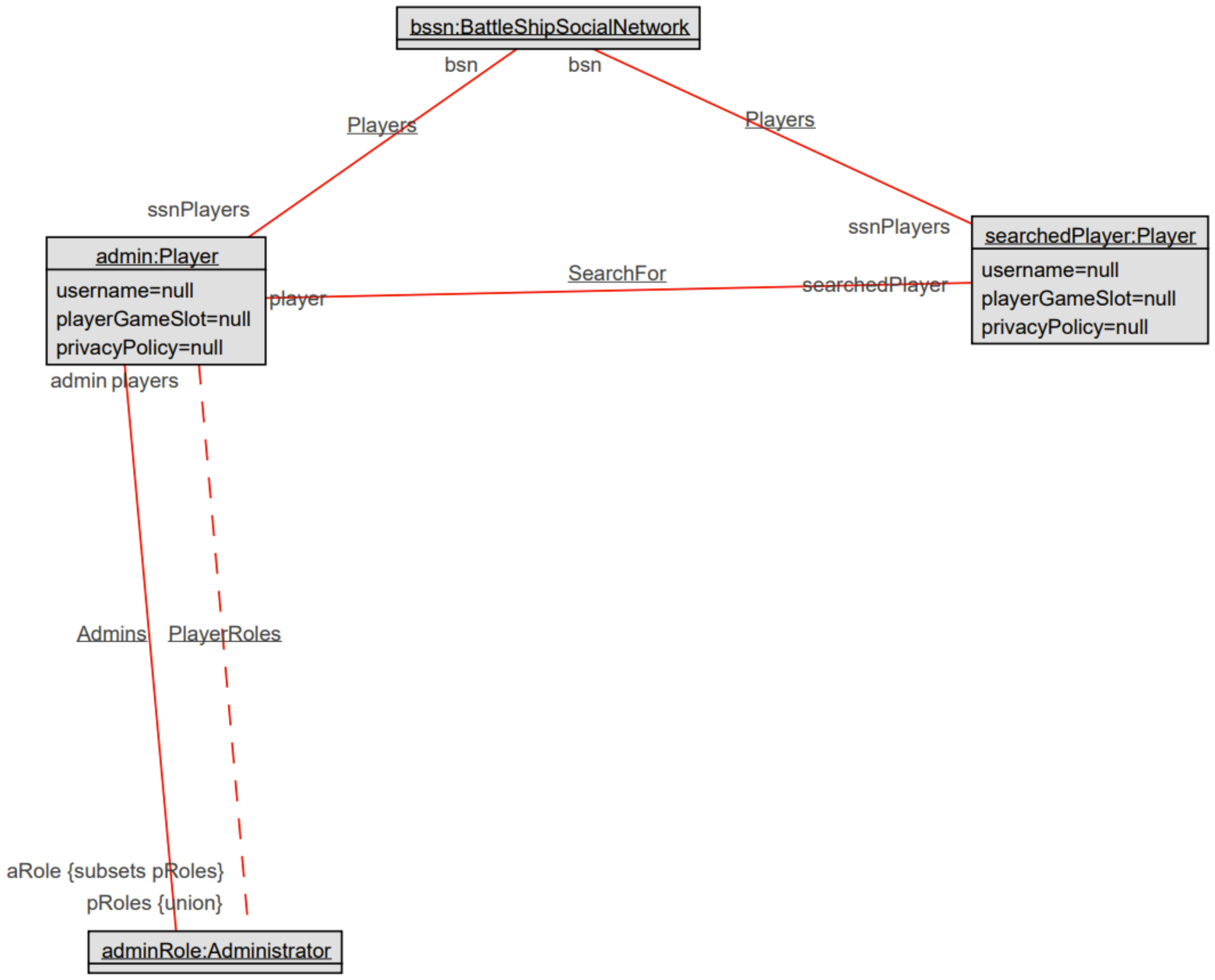
<u>After</u>



Description

Ban a Player

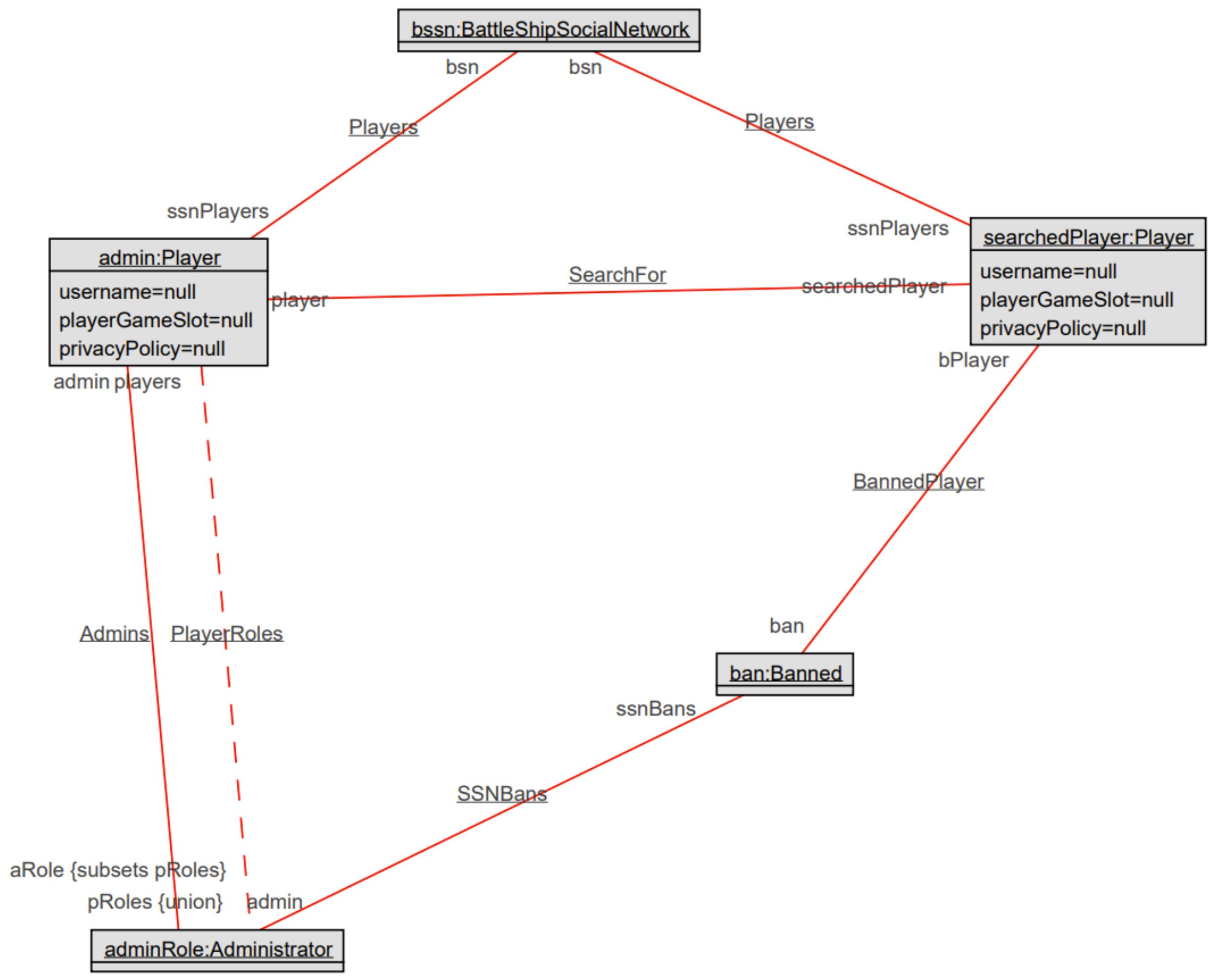
Before



Description

The player with the admin role has searched for the another player.

After

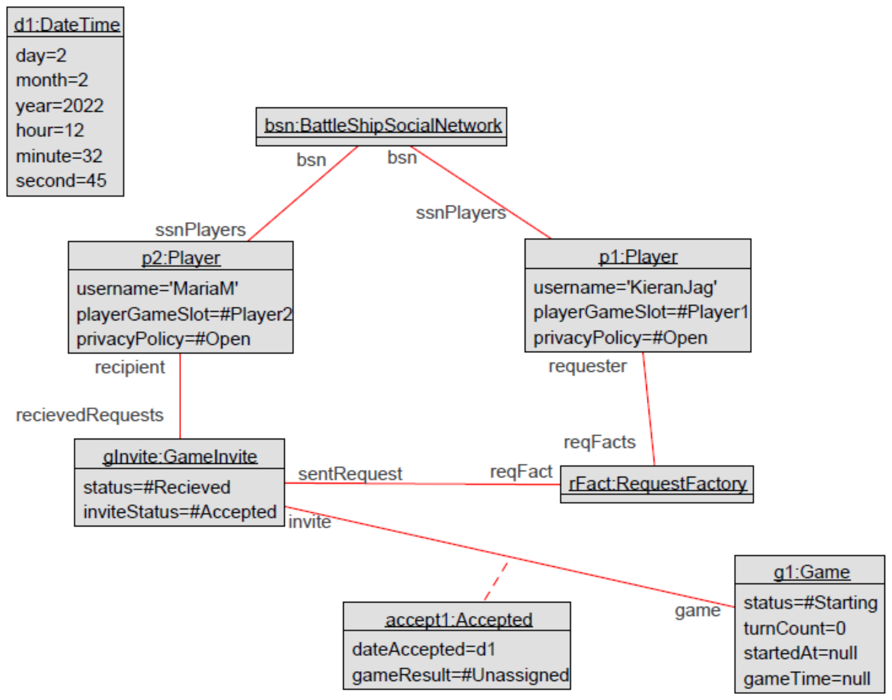


Description

The administrator player has now banned the player

Initialize Game

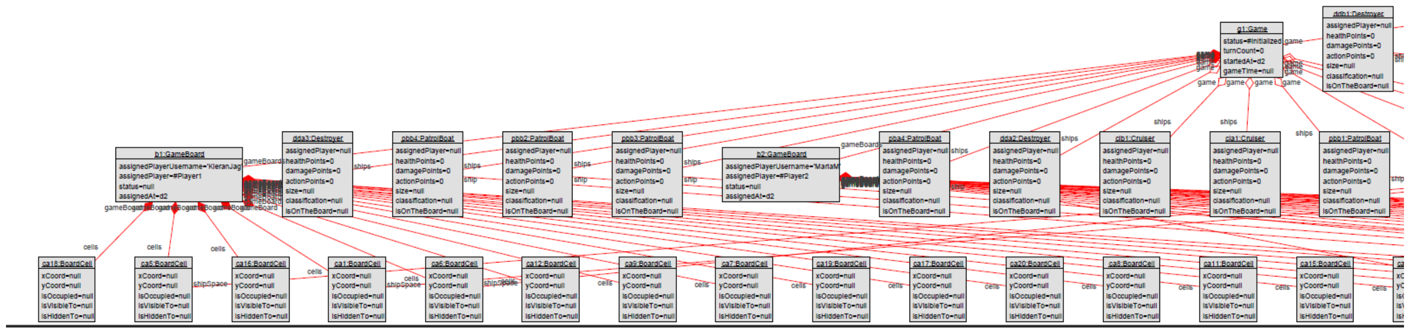
Before



Description

The system is still in the state where a player has accepted an invite from another player

After



Description

After a game is initialized the gameboards are assigned to the 2 players and the cells are generated for each board. All the ships are also placed on valid cells.

Reflection

A major thing we learned about representing values that are constantly changing or optional is that we could represent these values as name-value pairs to allow for adding future values without changing the classes in the class diagram. This allows for a higher level of modifiability within the model.

Another thing we learned is the importance of the observer pattern. This pattern was implemented throughout different sections of our class diagram to allow for optimized recognition of status changes for various subjects such as the different requests and tournament instances for observers which are players.

The last thing we learned was where to optimally use the factory pattern. We modelled our different requests as different instances of a request. Here use found that the factory pattern was very useful to show the creation of different requests.