horizontal line

**The Yilmazing Six**

Video Game Development and Management Platform

**21st April 2020**

# Group Members

Team Leader: Spencer Downey

swd0008@auburn.edu

Requirements Engineer: Kegan Van Ginkel

kzv0015@auburn.edu

Architect: Cannon Sloan

cys0003@auburn.edu

Designer: Davis Platt

adp0040@auburn.edu

Programmer: Cameron Mathis

clm0081@auburn.edu

**Table of Contents**

**Domain Analysis 2**

Concept Statement 2

Conceptual Domain Model (i.e., (Class Domain Model) 4

Class Linguistic Analysis 4

Associations Linguistics Analysis 5

Attributes & Updated UML Class Diagram 8

Updated UML Class Diagram with Specialization/Generalization 9

Constraints 10

Domain State Model 10

**Application Analysis 12**

Application Interaction Model 12

Scenarios 12

High-Level System Sequence Diagrams 15

Analyzation of HLSSDs 24

Scenario Diagram 25

Essential Use Cases 26

Concrete Use Cases 29

Detailed System Sequence Diagrams 35

Application Class Model 41

Application State Model 42

**Consolidated Class Model 43**

**Architectural Design 44**

Collaboration Diagrams 44

Design Class Diagrams 50

Design Patterns 56

Factory Design Pattern 56

Builder Design Pattern 56

Object Design 57

Interface Contract 57

UML Statechart 60

Procedural Behavioral Specification of Methods 63

**Model Review 66**

**Domain Analysis**

**Concept Statement**

We are creating a video game development and management platform. The platform will be a place for people to come and play or view different games in tournament-style gameplay. Each user will be required to login with either their username and password or email and password. Their login information will be verified, and if correct then they will be allowed to access the platform. If it is a user's first time visiting the platform, then they will be required to sign up by entering a valid email address, a username that is not already in use, and a password. Once a user has logged in, they will be able to apply for different user permissions, which allows them to do things such as apply to different leagues, manage leagues, or create advertisements. Leagues will be a way for players to communicate with each other openly or privately about different gameplay topics and will also host tournaments.

The platform will default open up to a Home page that consists of scores of current popular matches (with popularity being determined by overall spectator count), information about large leagues, and banner advertisements. Users will have the ability to search for league pages, which will consist of things such as tournament information and discussion board, and they can also search for players. Each league page will be broken down into further pages. Users will also be able to navigate to the Following page that will contain information about players and leagues that the user follows.

The platform will support five different types of users, each of which will have their own unique username. The first type of user is an operator, who can define new games, define tournament styles (e.g., knock-out tournaments, championships, best of series), define new expert rating formulas, and manage users and assign them a new user type.. The second type of user is a league owner. A league owner can define new leagues, organize and announce new tournaments within a league, conduct a tournament, declare a winner, adjust the rating points of players who played in their tournament, accept players into their league, remove players from their league, and moderate the league’s discussion post. A league owner can also see the rating of players who apply to their league. The third type of user is a spectator. All users start as spectators until they request and are granted permission to be another type of user. A spectator can monitor any match in progress and check scores and statistics of past matches and players. They can also “follow” players and leagues to get notified of when matches within those leagues or containing those players are happening. Spectators do not need to register in an arena to view these things. The fourth type of user is a player, who can register, apply for a league, sign up for a tournament, play the matches that are assigned to them, or drop out of a tournament. Each player has statistics associated with their account that show things like the number of matches played for each game type, the number of matches won and lost for each game type, scores of all previously played matches, and a rating number for each type of game they have played. Both league owners and players also have all of the abilities of a spectator. The fifth and final type of user is an advertiser. An advertiser is capable of uploading new advertisements, selecting new advertising schemes ( e.g., tournament sponsor, league sponsor, banner ad), check balance due, and cancel advertisements. In order to have any permission level besides spectator, a user must request permission from the platform. Operators will handle approving or rejecting user’s permission level requests.

Each league will be associated with only one game type. Each league can host only one official tournament at a time; however, a league can host as many unofficial tournaments as it would like. Each league page will have three main subpages. The first of these subpages will be the default page, called the News page. The news page will have information about upcoming tournaments, results of previous tournaments, any post a league owner makes, information about the league sponsor, and banner ads. The next page users will be able to navigate to is the league Discussion page. This page will be a way for players and league owners to make posts and comment on other players' and league owners’ posts from the league. League owners will also be able to temporarily or permanently ban players from making posts. The final page a user will be able to go to is the league’s Arena page. The arena page will be a place for spectators and players to view any current matches from the league’s tournaments.

Each league will have two different types of tournaments, official and unofficial. Only official tournaments will affect a player's statistics. Official tournaments will have particular styles like single elimination, double elimination, best 3 out of 5, etc. Official tournaments will have the option of automatic player rating formulas, which automatically rates each participating player based on their performance. League owners can also choose to rate each player individually if they do not wish to use the automatic player rating system. Unofficial tournaments can be played in official tournament styles, or can be in different styles created by the operator with different rulesets depending on the game. Both official and unofficial tournaments will be signed up for the same way. League owners will create a tournament by determining the tournament-style and number of players the tournament will have. They will also have the option to set up an automatic player rating system. Once a league owner has set up the tournament, they will be able to post the tournament to the league’s News page. Any player in the league will be able to request a position in the tournament. League owners will then review the players who have applied and approve of any players. Once the tournament has been filled with its maximum number of players, the league owner will close the tournament from applications and start scheduling matches. Once a player’s match is scheduled they will be notified of the time and their opponent.

Advertisers will have their own page called My Advertisements. From here, they will have the ability to choose to create an advertisement, select an advertisement style, save or view advertisement drafts, submit ads for approval, post those approved ads, and takedown existing ads. They can post ads to the home page and the news page, but each place it is displayed results in an increase in balance. They can also choose to sponsor a tournament or league. Their balance can be viewed on My Advertisements and will be affected by how long they choose to leave the ads up, where they display the ads, and how many leagues or tournaments they are sponsoring.

Player’s rating numbers for the games they play are determined through their expert rating points. Expert rating points are awarded or deducted depending on the player’s wins or losses in tournaments, along with how well they do in each match. All of their statistics from each match are taken into consideration when it is determined how many rating points they are given. A player’s rating points will be used to show their overall skill level in certain games, the higher the number the higher the skill level. League owners will take this into consideration when choosing who to accept into their league and whether or not players should be removed from their league.

**Conceptual Domain Model (i.e., Class Domain Model)**

**Class Linguistic Analysis**

**Our linguistic analysis provided identified these tentative classes:**

Videogame, Development, Management, Platform, Place, People, Game, Tournament, Style, Gameplay, User, Username, Password, Email, Login Information, Email Address, Permission, Thing, League, Advertisement, Player, Topic, Home Page, Score, Match, Popularity, Spectator, Spectator Count, Information, Discussion Board, League Page, Page, Type, Operator, Championship, Series, Expert Rating Formula, League Owner, Winner, Rating Points, Discussion Post, Statistics, Arena, Account, Number, Ability, Advertiser, Sponsor, Banner Ad, Balance, Permission Level, Request, Subpage, Default Page, News Page, Result, Post, Arena Page, Performance, Rulesets, System, Position, Application, Time, Opponent, Drafts, Following Page

**We chose to keep these classes:**

Platform, Game, Tournament, Account, League, Advertisement, Player, Home Page, Match, Spectator, Discussion Board, League Page, Page, Operator, League Owner, Discussion Post, Arena, Advertiser, News Page, Post, Arena Page, Rating Formula, Ruleset, Following Page

**We eliminated these classes (listed by reason):**

*Too vague:* Place, Gameplay, Thing, Topic, Information, Number, Ability, System

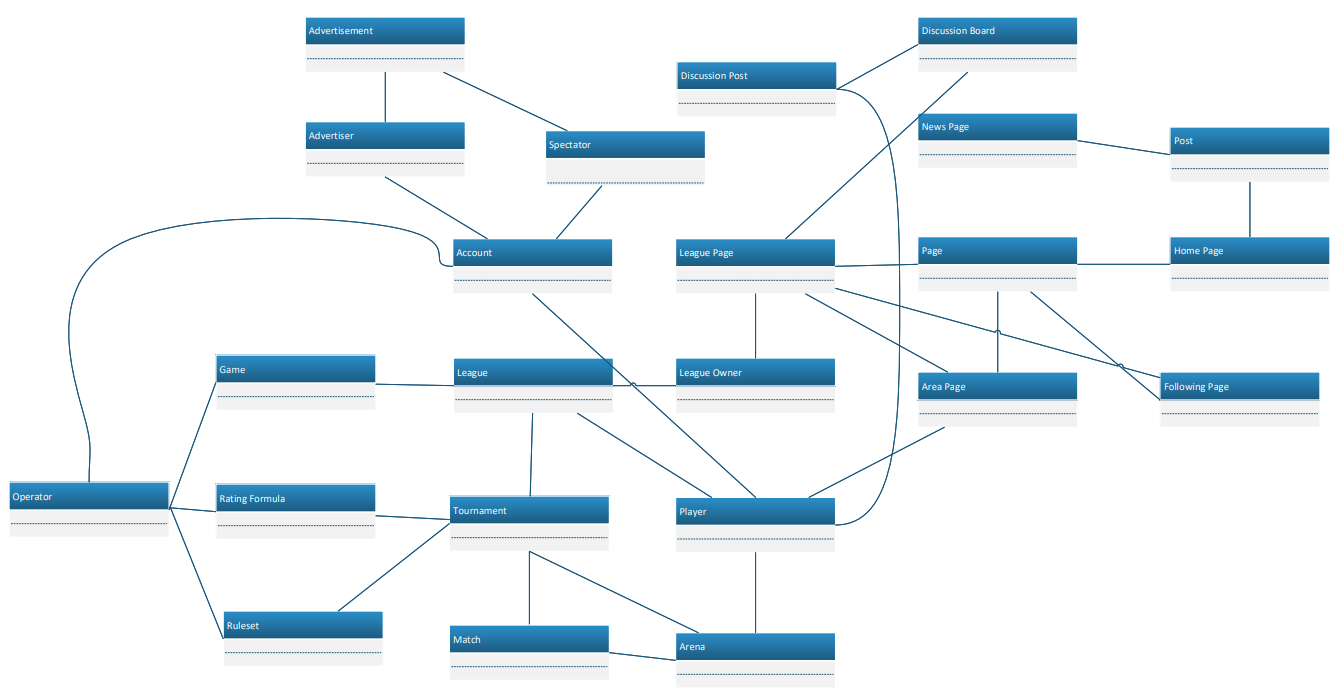
*Attributes:* Username, Password, Email Address, Score, Spectator Count, Rating Points, Balance Due, Time, Statistics, Style

*Irrelevant:* Login Information, Popularity, Type, Expert, Winner, Result, Performance, Draft, Championship, Series

*Redundant:* Videogame, People, Email, User, Sponsor, Subpage, Position, Opponent

*Implementation:* Development, Management, Permission, Banner Ad, Permission Level, Request, Default Page, Application

**UML Class Diagram:**

****

**Associations Linguistics Analysis**

**Our linguistic analysis provided identified these tentative associations:**

An account will log in to the platform. **x**

The platform will support five different types of users. **x**

A spectator will apply for a new account type. **x**

A spectator can follow a player. **x**

A spectator can view a match. **x**

League owners will manage leagues. ✔

Advertisers will create advertisements. ✔

Advertisers can sponsor a League/Tournament. ✔

Players will communicate with each other. **x**

Leagues will host tournaments. ✔

The Home page will consist of scores. **x**

The Home page will consist of information about leagues. **x**

The Home page will consist of banner advertisements. ✔

Users will search for League pages. **x**

Users will navigate to the Following page. **x**

The Following page will contain information. **x**

The platform will support five different types of users. **x**

An operator can define new games. ✔

An operator can define rulesets. ✔

An operator can define new expert rating formulas. ✔

An operator can manage users. **x**

An operator can grant a user a new account type. **x**

A league owner can define new leagues. **x**

A league owner can organize new tournaments. **x**

A league owner can announce new tournaments. **x**

A league owner can conduct a tournament. **x**

A league owner can declare a tournament winner. **x**

A league owner can adjust the rating points of players. **x**

A league owner can accept players into their league. **x**

A league owner can remove players from their league. **x**

A league owner can see the rating points of players who apply to their league. **x**

Users will request a different account type. **x**

Operators approve or reject user’s permission level requests. **x**

Each league can host only one official tournament at a time. **x**

Each league can host as many unofficial tournaments as it would like. **x**

Each league page will have three main subpages. ✔

The news page will consist of information. **x**

The news page consists of posts. ✔

The news page consists of banner ads. ✔

Users can navigate to the discussion page. **x**

Users can make posts on other users' posts. **x**

League owners can ban players from making posts. **x**

Users can go to the league Arena Page. **x**

Each league will consist of two different types of tournaments, official and unofficial. ✔

Official tournaments will affect a player’s statistics. **x**

Official tournaments will have particular styles. **x**

Official tournaments will have the option of automatic player rating formulas. **x**

League owners can rate each player individually based on their performance. **x**

Unofficial tournaments can be played in many different tournament styles. **x**

League owners will create tournaments. **x**

League owners can post about tournaments. **x**

Players can request positions in their leagues’ tournaments. **x**

League owners will approve or deny players. **x**

League owners can close off tournaments from applicants. **x**

League owners will schedule matches. **x**

Advertisers will select an advertisement style.**x**

Advertisers can save and review advertisement drafts. **x**

Advertisers can submit ads for approval. **x**

Advertisers can post ads. **x**

Advertisers can take down existing ads. **x**

Advertisers can sponsor leagues. **x**

**We chose to keep these associations:**

League owners will manage leagues.

Advertisers will create advertisements.

Advertisers can sponsor a League/Tournament.

Leagues will host tournaments.

The Home page will consist of banner advertisements.

An operator can define new games.

An operator can define new rulesets.

An operator can define new expert rating formulas.

Each league page will have three main subpages.

The news page consists of posts.

The news page consists of banner ads.

Each league will consist of two different types of tournaments, official and unofficial.

**We eliminated these associations (listed by reason):**

*Associations between eliminated classes:*

An account will log in to the platform.

An operator can manage users.

The Following page will contain information.

A league owner can see the rating points of players who apply to their league.

The news page will consist of information.

Official tournaments will affect a player’s statistics.

Official tournament will have particular styles.

Official tournaments will have the option of automatic player rating formulas.

Unofficial tournaments can be played in many different tournament styles.

An operator can define new tournament styles.

*Irrelevant or implementation associations:*

The platform will support five different types of users.

The Home page will consist of scores.

The Home page will consist of information about leagues.

*Actions:*

A spectator will apply for a new account type.

A spectator can follow a player.

A spectator can view a match.

Players will communicate with each other.

Users will request a different account type.

Operators approve or reject user’s permission level requests.

Users can navigate to the discussion page.

Users can make posts on other users' posts.

Users can go to the league Arena Page.

League owners can rate each player individually based on their performance.

League owners will create tournaments.

League owners can post about tournaments.

Players can request positions in their leagues tournaments.

League owners will approve or deny players.

League owners can close off tournaments from applicants.

League owners will schedule matches.

Advertisers will select an advertisement style.

Advertisers can save and review advertisement drafts.

Advertisers can submit ads for approval

Advertisers can post ads.

Advertisers can take down existing ads

*Derived Associations:*

An operator can grant a user a new account type.

A league owner can define new leagues.

A league owner can organize new tournaments.

A league owner can announce new tournaments.

A league owner can conduct a tournament.

A league owner can manage its league.

A league owner can declare a tournament winner.

A league owner can adjust the rating points of players.

A league owner can accept players into their league.

A league owner can remove players from their league.

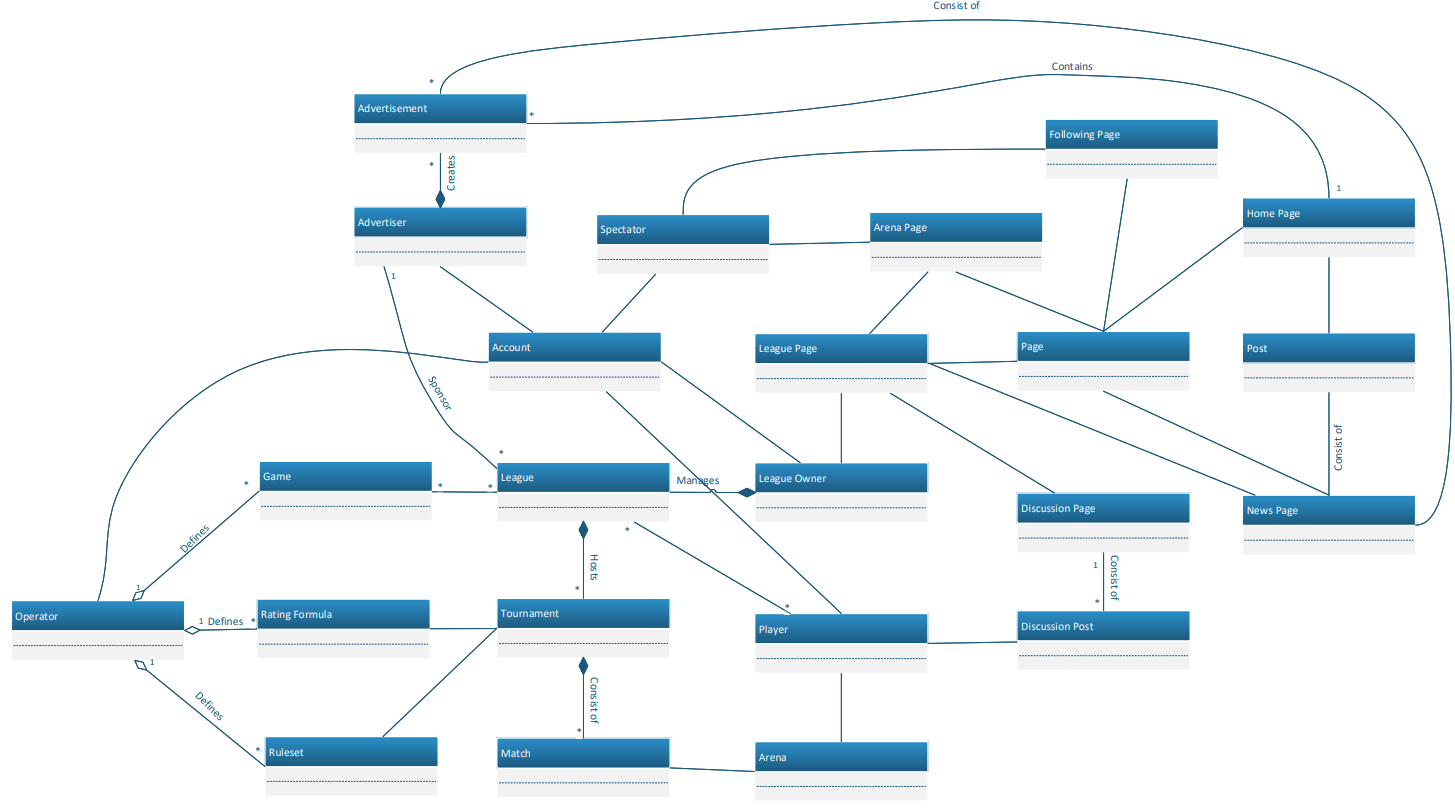
League owners can ban players from their league.

Each league can host only one official tournament at a time.

Each league can host as many unofficial tournaments as it would like.

Advertisers can sponsor leagues.

**UML Class Diagram:**



**Attributes and Updated UML Class Diagram**

**We identified these attributes:**

Account: Username, Password, Email Address

Operator: Username, Password, Email Address

Player: Username, Password, Email Address, Rating Points, Statistics

League Owner: Username, Password, Email Address

Advertiser: Username, Password, Email Address, Balance Due

Spectator: Username, Password, Email Address

Match: Score, Spectator Count, Time

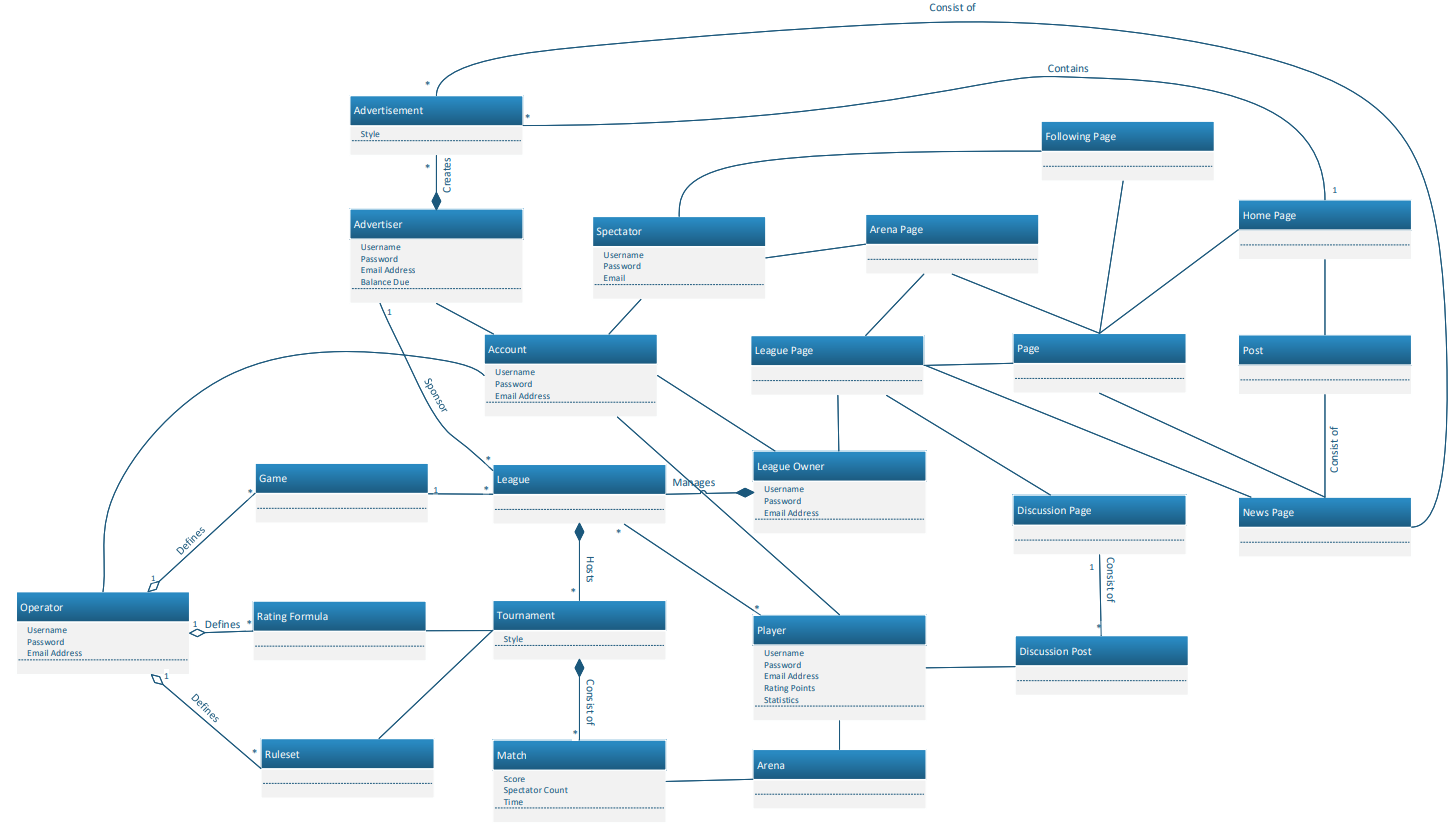
Tournament: Style

Advertisement: Style

League: Members

Discussion Post: Creator, Text

**UML Class Diagram:**



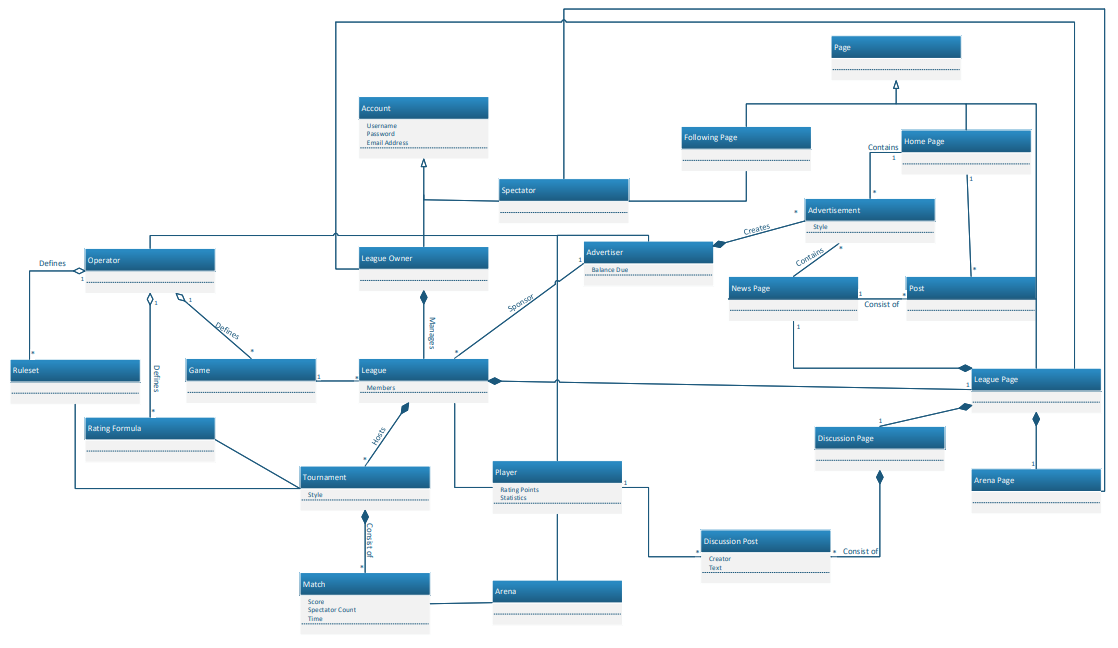
**Updated UML Class Diagram with Specialization/Generalization**

**We identified these opportunities for specialization/generalization:**

User: Generalizes Operator, Player, League Owner, Advertiser, and Spectator with the common attributes of Username, Password, and Email Address. Player and Advertiser then have specific attributes of Player: Rating Points and Statistics, and Advertiser: Balance due.

Page: Generalizes League Page, Following Page, Home Page

**UML Class Diagram:**



**Constraints**

**We identified these constraints:**

Constraints: (1) All usernames should be unique, (2) A player must be a member of the league in order to write a discussion post, (3) All players in a tournament must be unique (i.e. a player can join the tournament only once)

**OCL Expressions:**

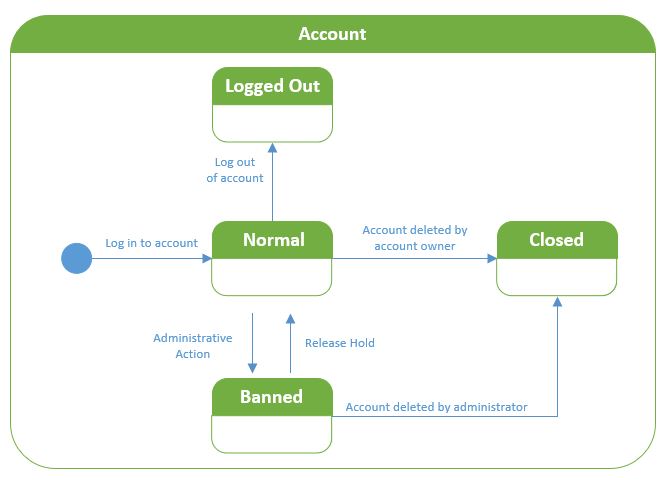
OCL Expression: For all ((x, y) | x.username =/= y.username)

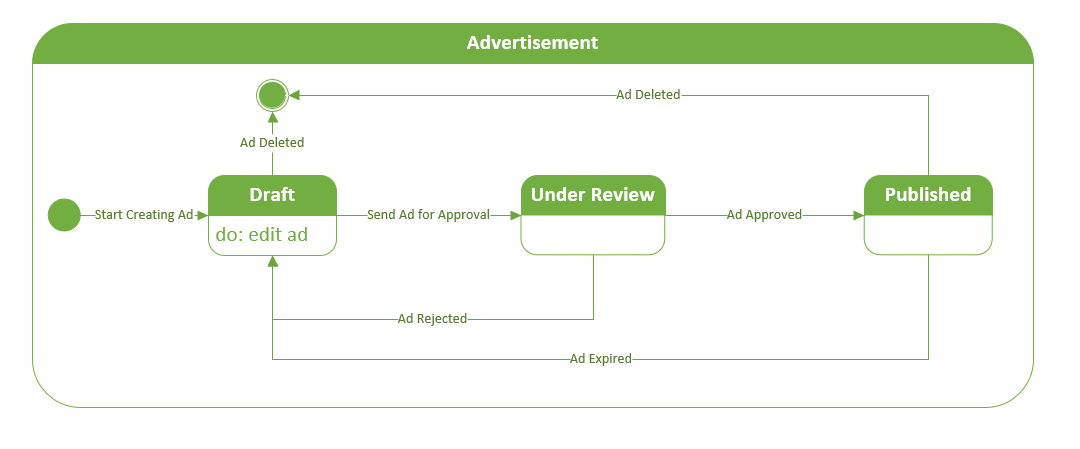
OCL Expression: aLeague.LeaguePage.DiscussionPage.DiscussionPost -> for all ((x) | x.Creator -> aLeague.Members)

OCL Expression: aTournment.members -> for all ((x,y) | x =/= y)

**Domain State Model-**







****

**Application Analysis**

**Application Interaction Model**

**Scenarios**

1. Develop ten to twelve top-level scenarios based on major functionalities/features of value to actors. (See the Organize Tournament-Scenario example posted under Modules/Supplementary Materials on Canvas)
2. **organizeTournament** [Actors: LeagueOwner, Advertiser]

* LeagueOwner signs into the system.
* System verifies login.
* LeagueOwner requests a new tournament from the system
* System verifies LeagueOwners request
* LeagueOwner begins the process of starting a new tournament for their league.
* System requests tournament name, ruleset, and other relevant attributes from the LeagueOwner.
* LeagueOwner provides information to the system.
* System verifies the information and reports to the LeagueOwner that the tournament is now created.
* LeagueOwner communicates with the system that it needs a tournament sponsor.
* System contacts advertiser with this request.
* Advertiser provides a sponsorship to the system.
* System informs the LeagueOwner that they have a sponsor.
* LeagueOwner communicates with the system that the tournament is now open to players.
* Once the tournament is full, the tournament will be successfully created, sponsored, and have a playerbase.

1. **joinMatch**[Actors: Player, System]

* Player logs in to System.
* System verifies login.
* Player browses arena page for registered matches.
* Player requests the System to join a match.
* System verifies request.
* Player joins and participates in the match.
* Match ends, System boots player.

1. **spectateTournamentMatch** [Actors: Spectator, Advertiser]

* Spectator signs into the system.
* System verifies login.
* Spectator navigates the homepage to view all of the running tournament matches.
* Spectator requests the specific match that they want to view.
* System verifies this request and provides the match for the spectator to view.
* Spectator views the match for it’s lifespan.

1. **participateInTournament** [Actors: Player]

* Player signs into the system.
* System verifies login.
* Player signs up to play in a new tournament through the system.
* System communicates with the tournament to verify there is still space left.
* Tournament verifies to the system there is still space remaining.
* System adds Player to the tournament.
* System communicates with the player that they are successfully signed up.
* The tournament begins, and the player participates in the tournament.
* The tournament ends and communicates with the system that the player is no longer a part of the tournament.
* System removes Player from the tournament.

1. **purchaseAdvertisement** [Actors: Advertiser]

* Advertiser signs into the system.
* System verifies login.
* Advertiser begins the process of purchasing a new advertisement.
* System requests advertisement scheme and duration.
* Advertiser provides the scheme and duration.
* System verifies scheme.
* Advertiser provides upload of advertisement.
* System verifies upload.
* System calculates and reports balance due.
* Advertiser begins payment.
* System checks with the interface to see that there is ad space, if there is, the ad will then be displayed on the interface until the duration is finished.

1. **defineGame** [Actors: Operator]

* Operator signs into the system.
* System verifies login.
* Operator requests a new game to be added to the system.
* System verifies request
* Operator begins the process of defining a new game.
* System requests game information and attributes from the operator.
* Operator provides game information and attributes to the system.
* System communicates the addition of a new game to the interface.

1. **defineTournamentRuleset** [Actors: Operator]

* Operator signs into the system.
* System verifies login.
* Operator requests a new ruleset to be added to the system.
* System verifies request
* Operator begins the process of defining a new tournament ruleset.
* System requests tournament ruleset information from the operator.
* Operator provides ruleset information to the system.
* System requests rating values for the new ruleset.
* Operator provides rating values to the system.
* System communicates the addition of a new tournament ruleset to the interface.

1. **createAccount** [Actors: Operator, Spectator]

* New user begins the registration process within the system.
* System requests username, password, and email address for the account.
* New user provides username, password, and email address to the system.
* System sends the username and email address to the interface to make sure they aren’t taken.
* Interface reports to the system that the information isn’t taken.
* System sends the new user’s information to the Operator for review.
* Operator grants the new user the “Spectator'' account type.
* System verifies account information and creates the new account on the interface.

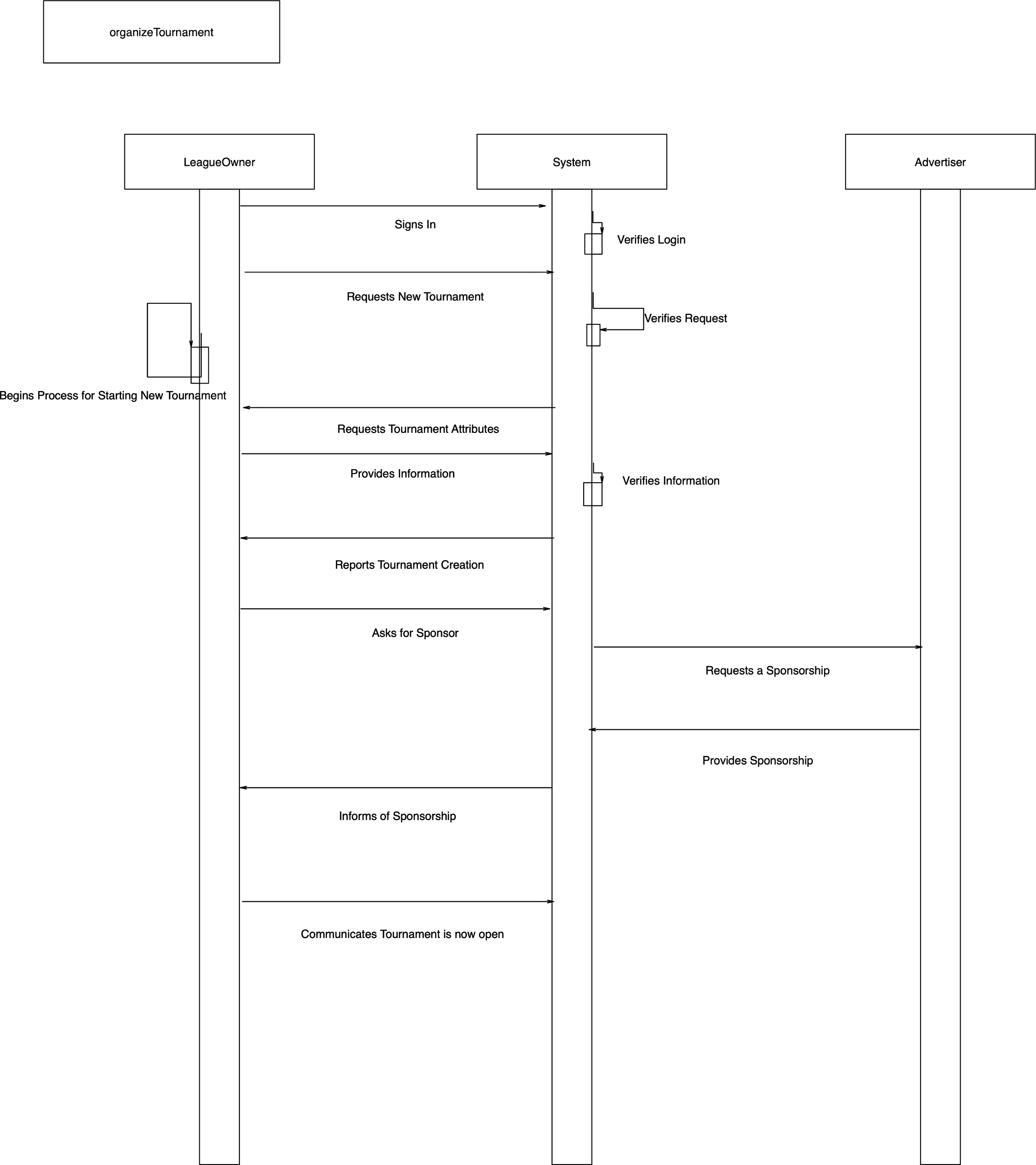
1. **organizeNewTournamentLeague** [Actors: LeagueOwner, Advertiser]

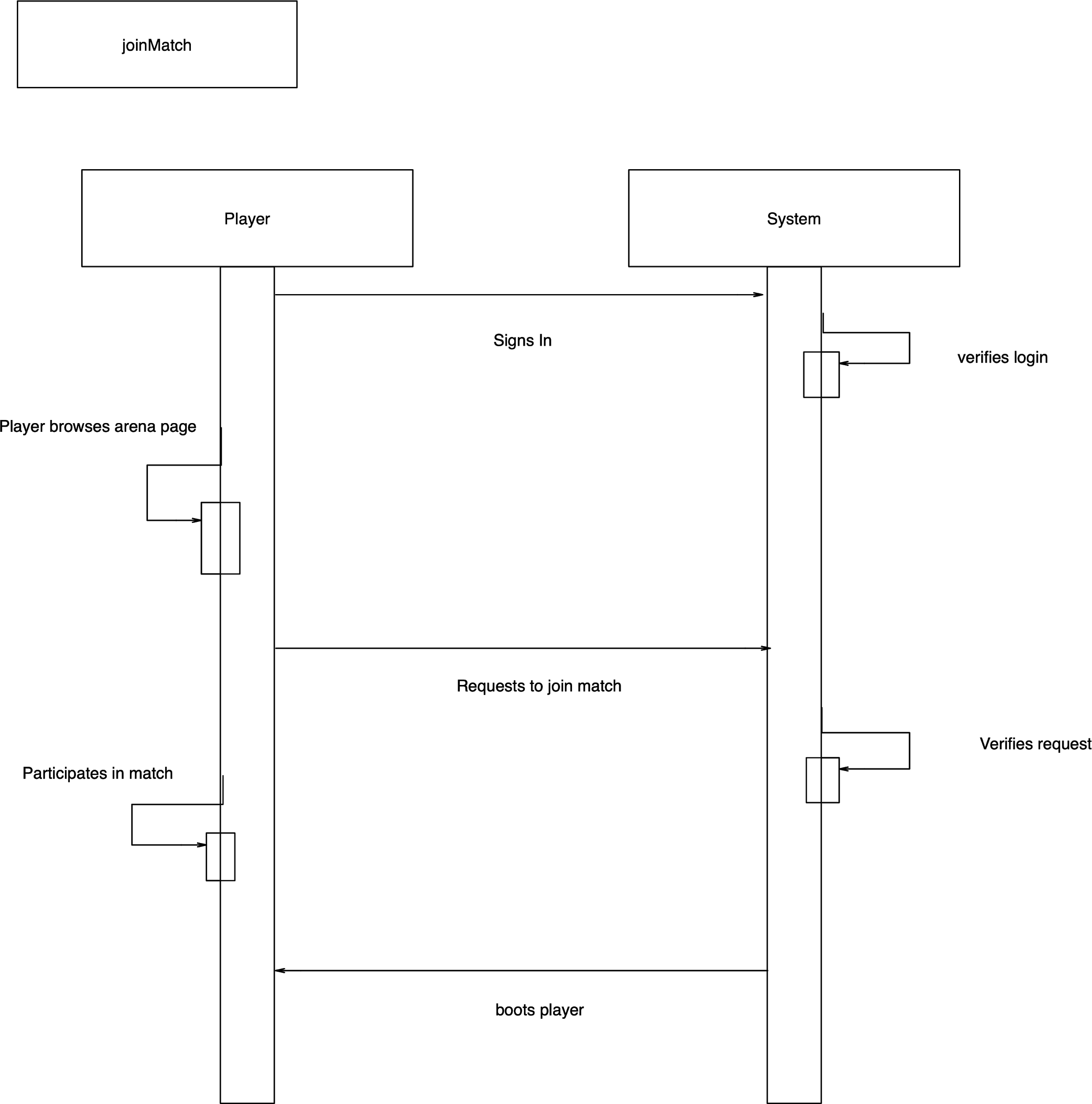
* LeagueOwner signs into the system.
* System verifies login.
* LeagueOwner requests new tournament league from system
* System verifies LeagueOwner’s request.
* LeagueOwner begins the process of starting a new league
* System requests league name and relevant attributes from the LeagueOwner.
* LeagueOwner provides league name and relevant attributes to the system.
* System verifies the information and reports to the LeagueOwner that the league is now created.
* LeagueOwner communicates with the system that it needs a league sponsor.
* System contacts advertiser with this request.
* Advertiser provides a sponsorship to the system.
* System informs the LeagueOwner that they have a sponsor.
* League is successfully made and sponsored.

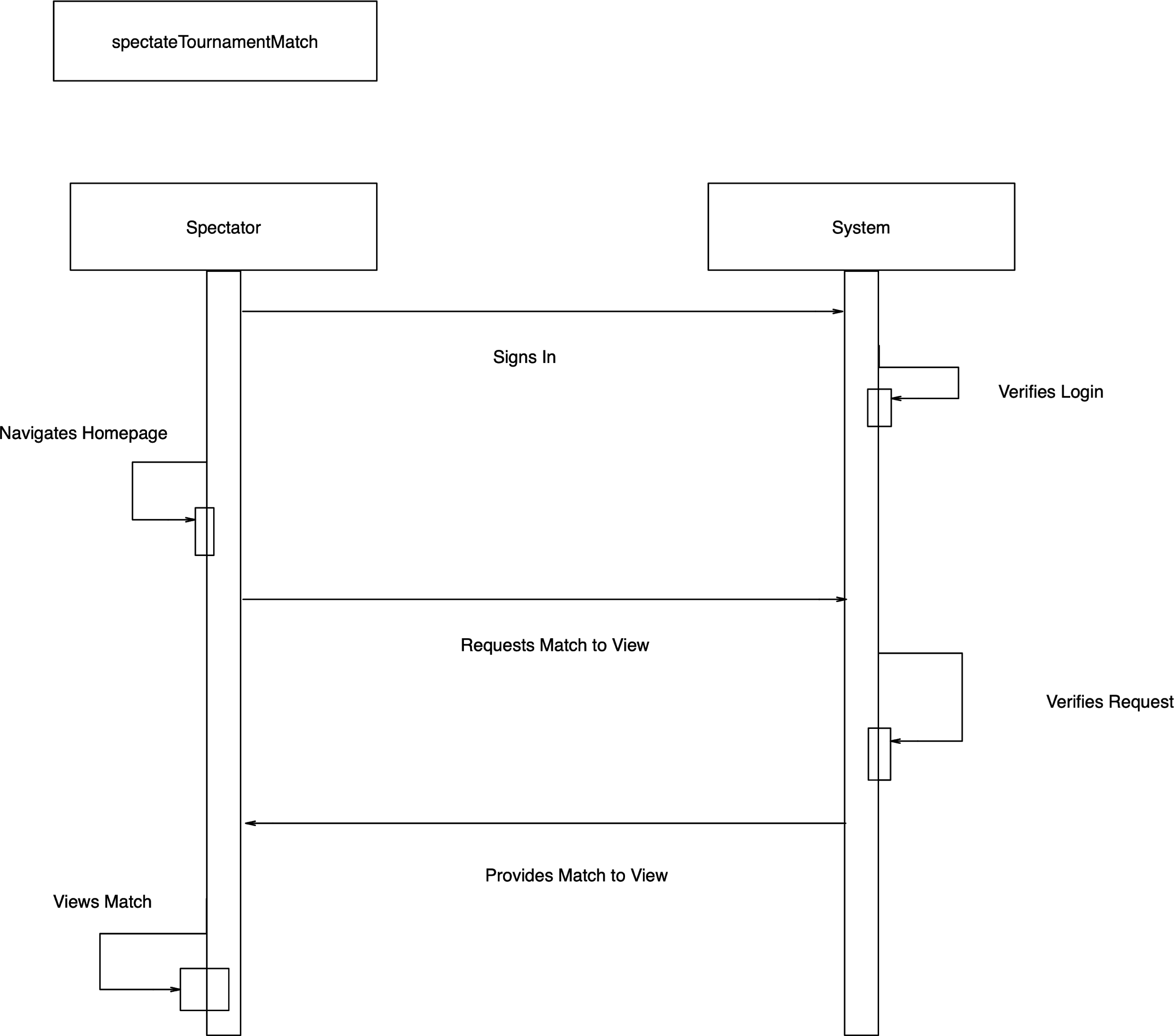
1. **changeUserType** [Actors: Operator, Spectator]

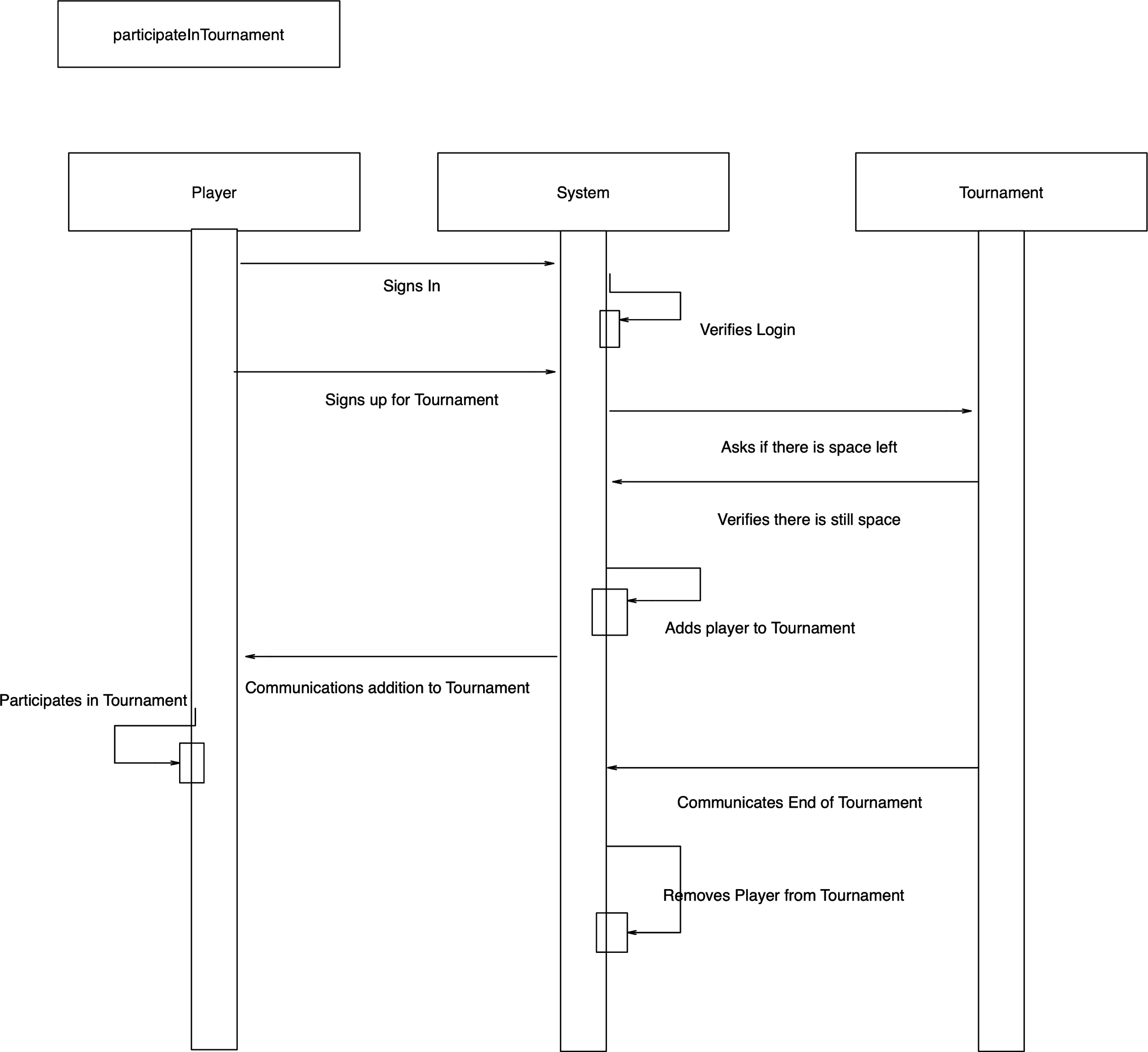
* Spectator signs into the system.
* System verifies login.
* Spectator requests from the system to be changed to Operator.
* System sends Operator request
* Operator verifies the change and informs the change
* System verifies the change and informs the Spectator

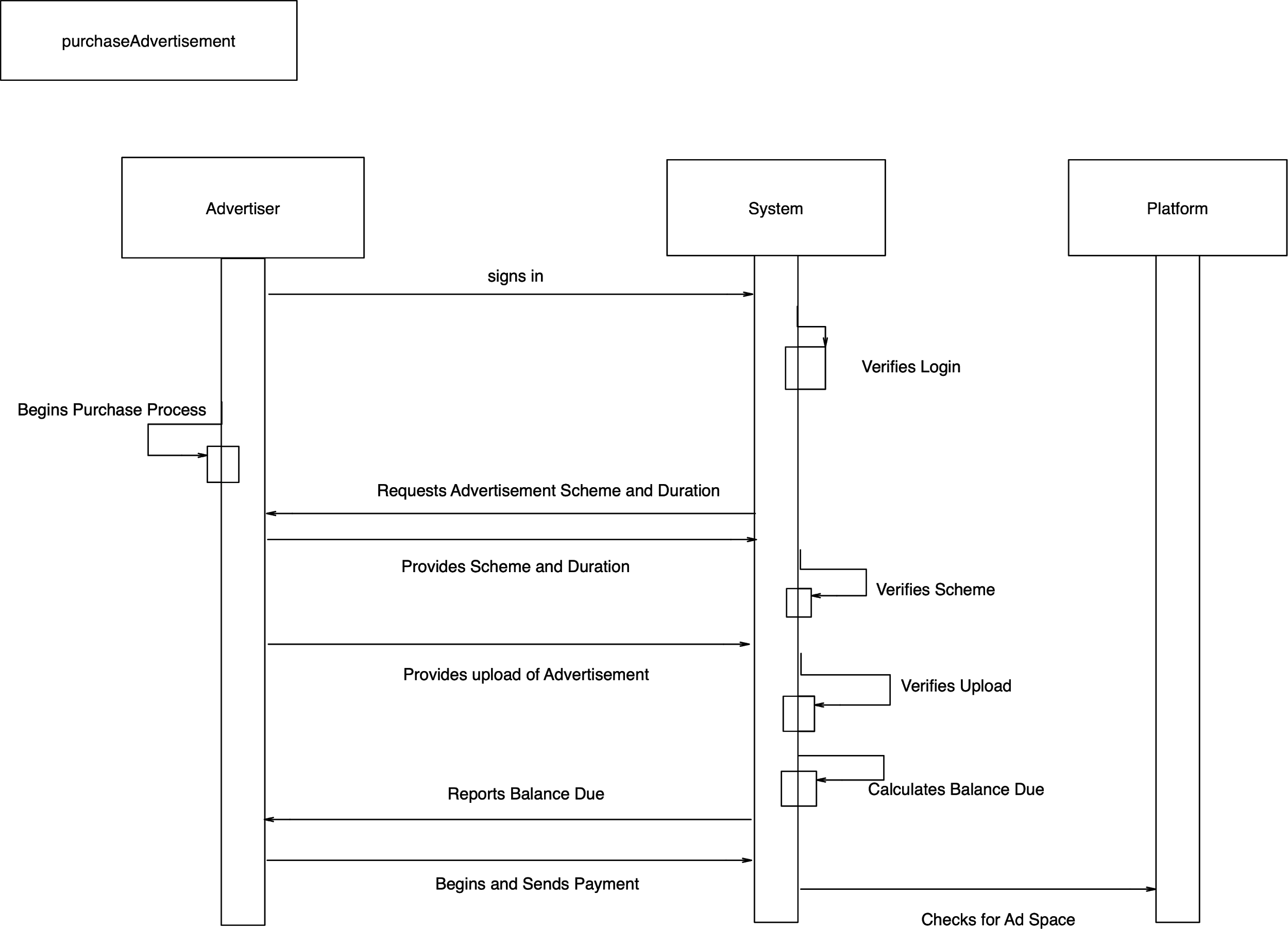
**UML High-Level System Sequence Diagrams**

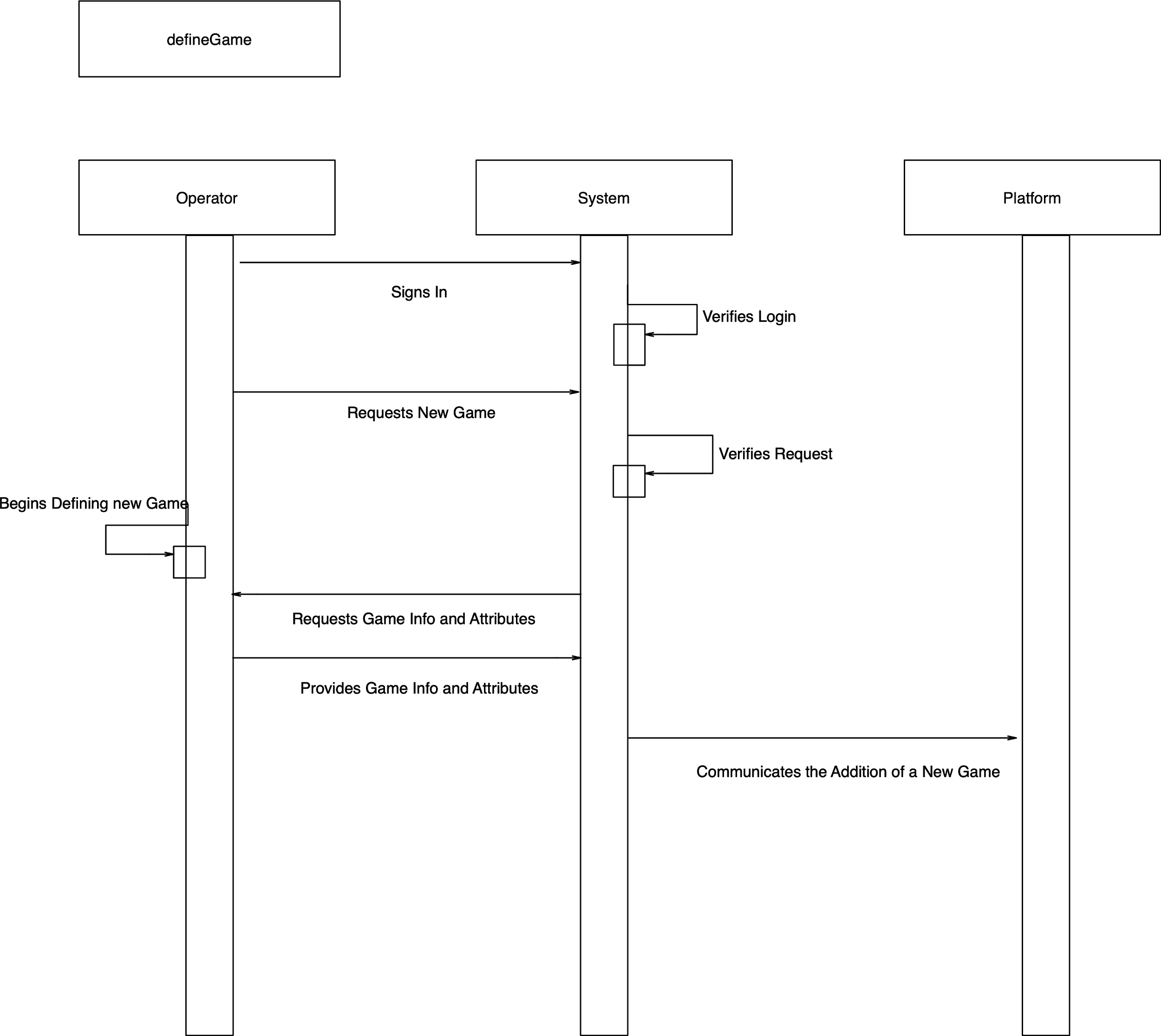


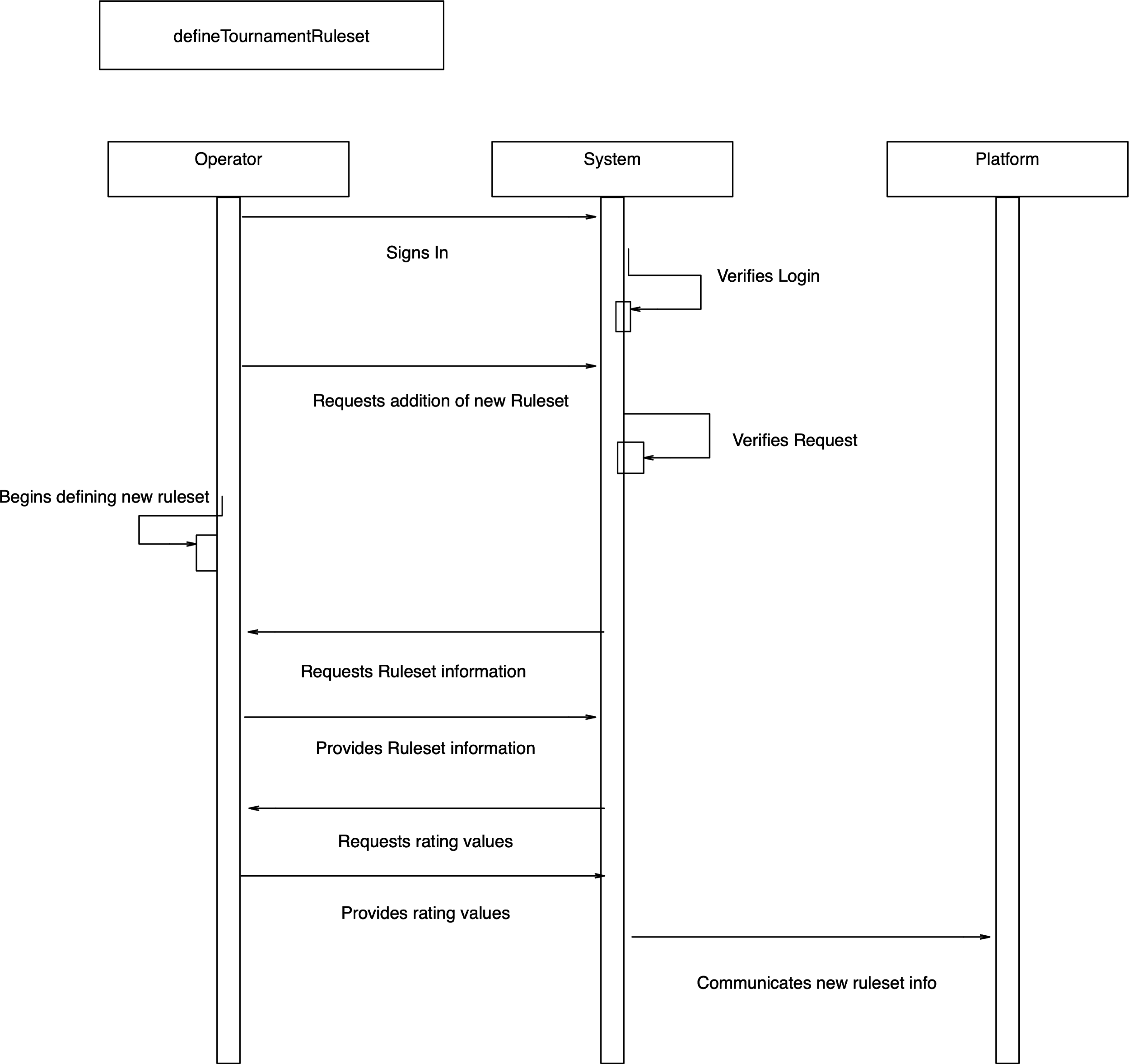


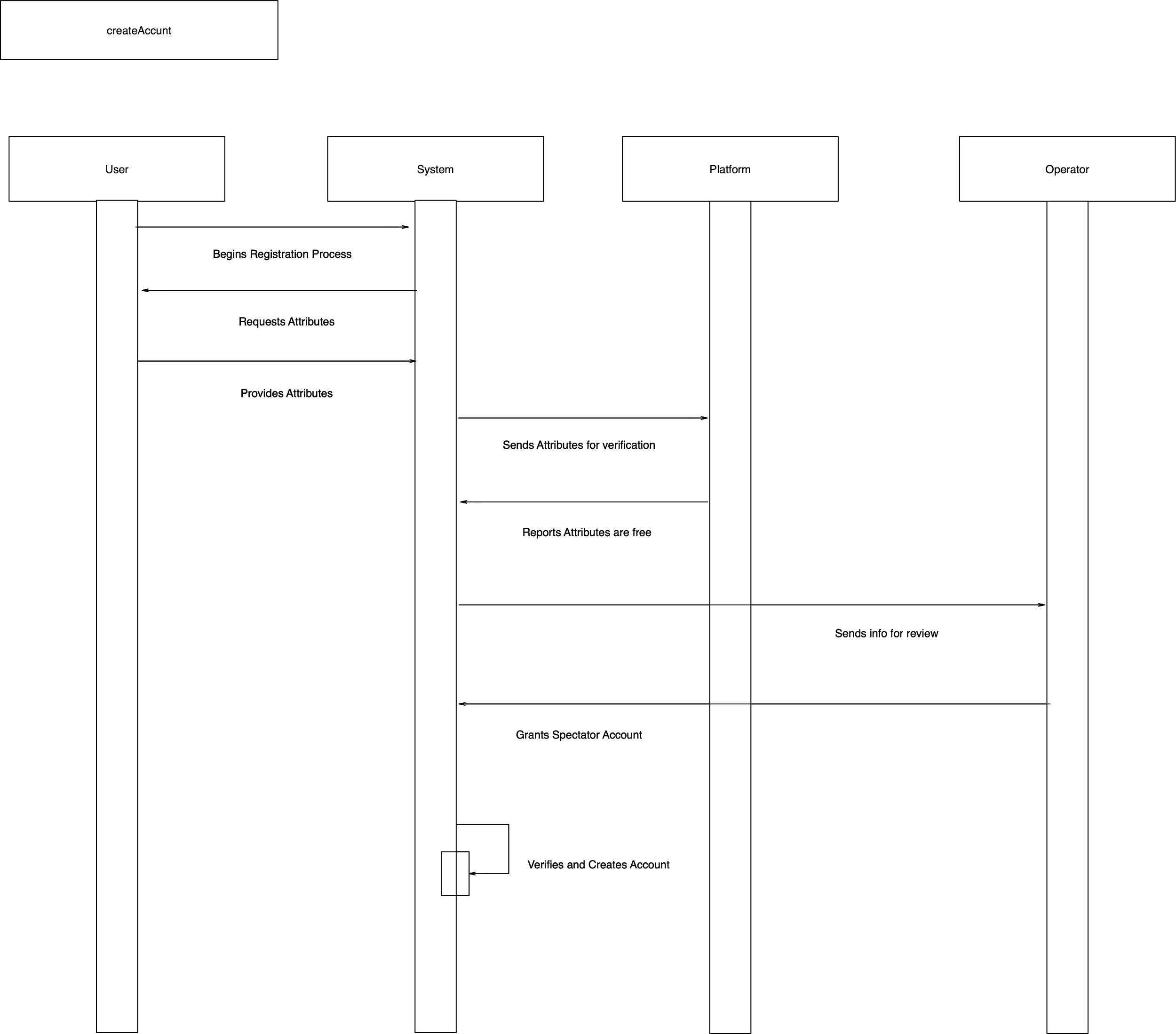


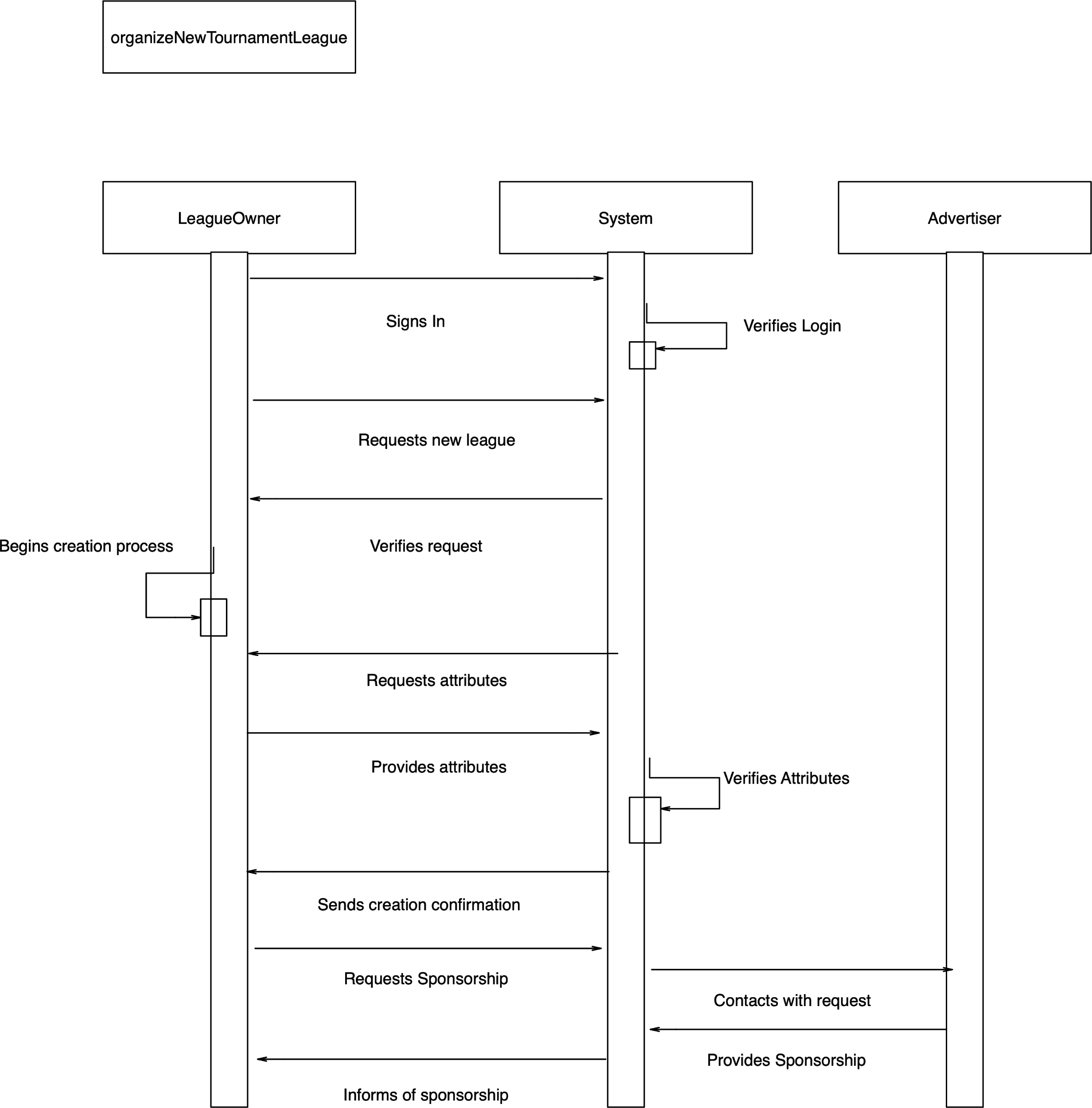


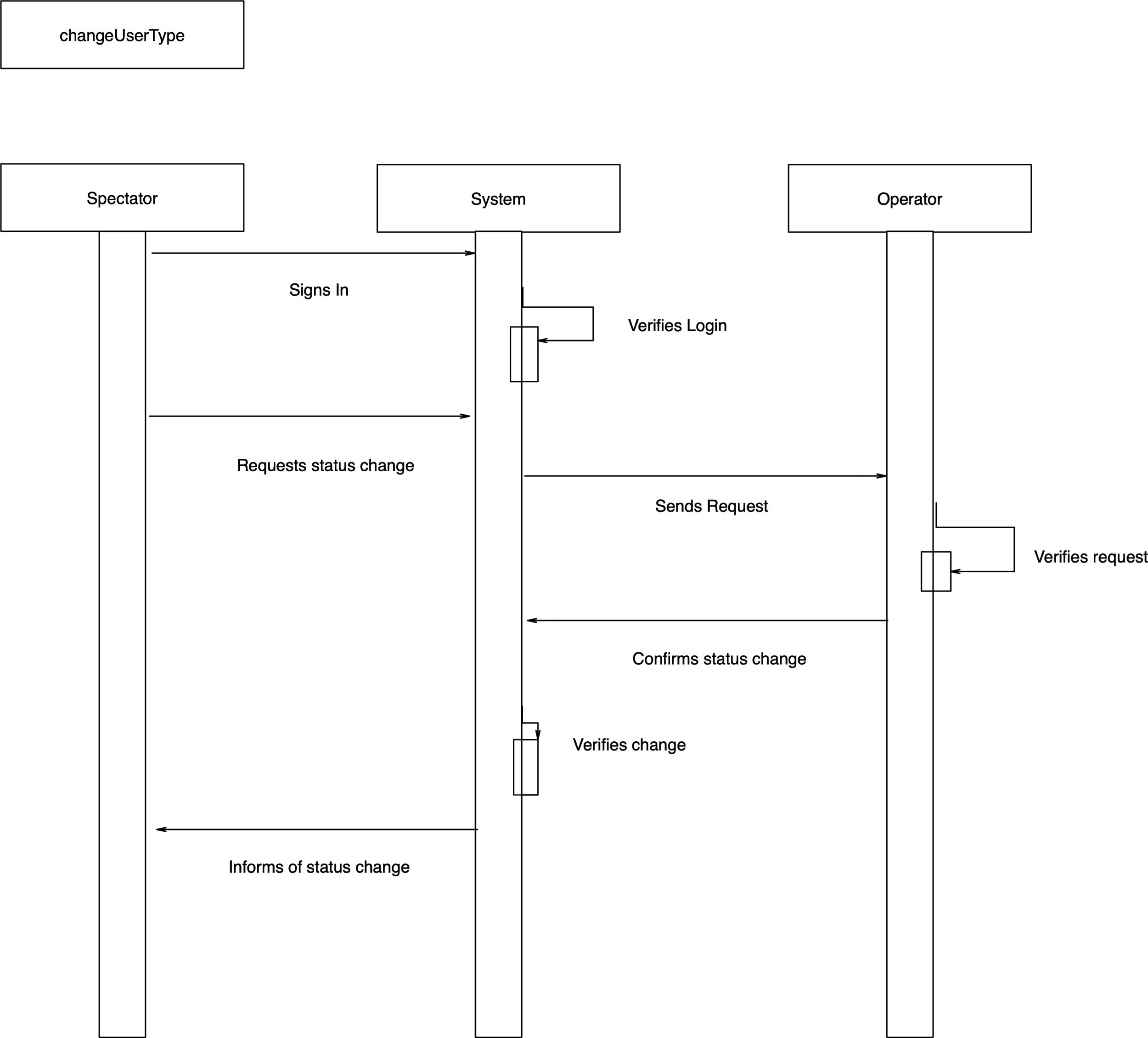












**Analyzation of High-Level System Sequence Diagrams**

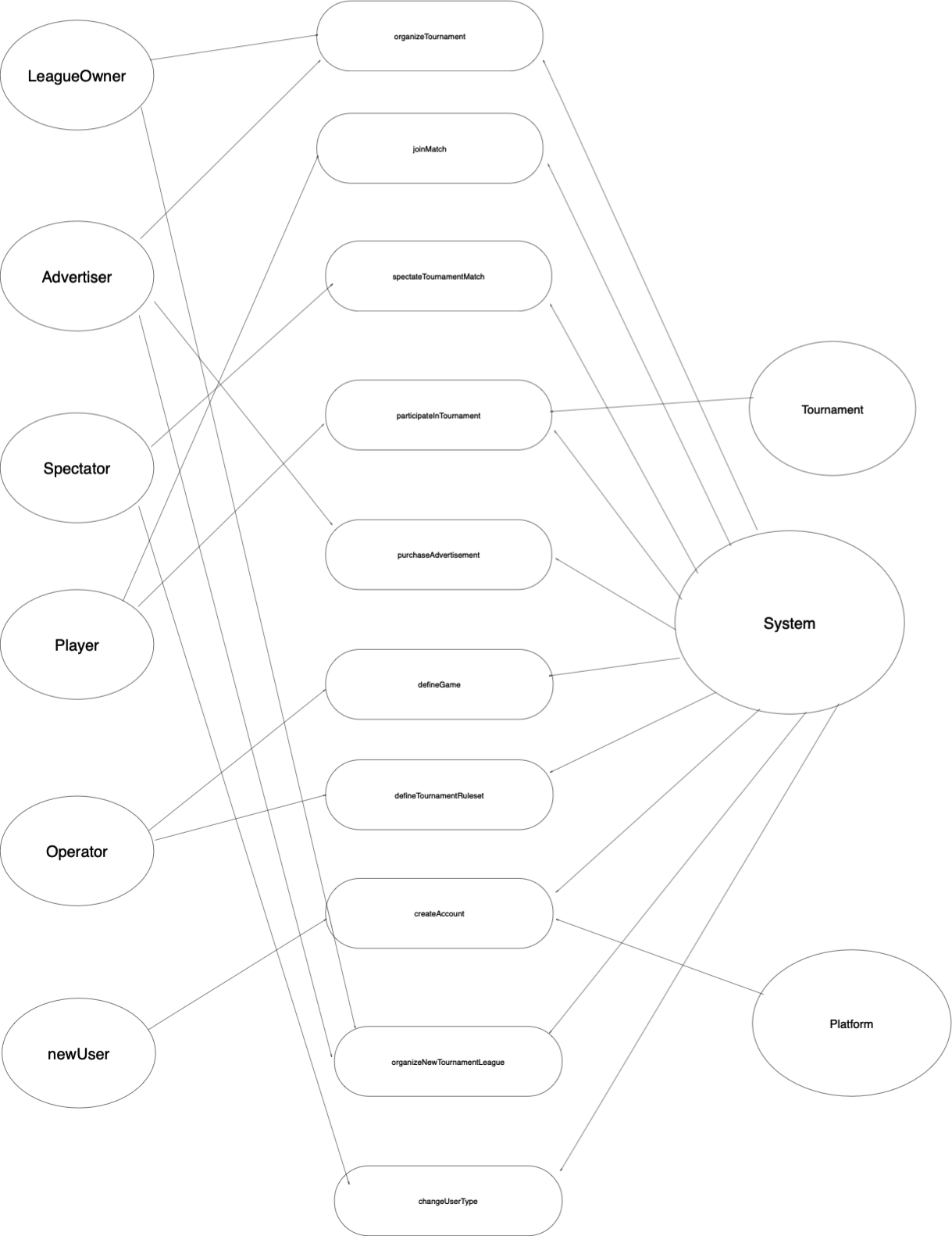
**Similarities:**

Log In: User signs in to the system and system verifies the login

Request for a change: User requests a change from the system and system verifies the change

Request for attributes: System asks user for attributes and user provides attributes

**Scenario Diagram**



**Essential Use Cases**

**Use case name**: OrganizeTournament

**Participating actors**: LeagueOwner, Advertiser, and System

**Flow of Events**

1. The LeagueOwner Logs in.
2. The LeagueOwner starts to make a new tournament.
3. LeagueOwner defines tournament name, ruleset, and other attributes.
4. System takes the information in and creates the tournament.
5. System reports that a tournament has been created.
6. League owner requests a sponsor from the system.
7. System contacts Advertiser for a sponsorship
8. Advertiser sponsors the tournament (include purchaseAdvertisement use case).
9. LeagueOwner opens up the tournament to players.
10. Players can now join the tournament (include participateInTournament use case).
11. Once the tournament is full it will be ready and closed to players.

**Entry Condition:** LeagueOwner is logged into the system.

**Exit Condition:** Tournament is ready and entry is closed.

**Use case name**: joinMatch

**Participating actors**: Player, System

**Flow of Events**

1. Player Logs in.
2. Player goes to the arena page for the tournament they are participating in.
3. System grants the request.
4. Player is now in the match and participates.
5. Match ends and the player is returned to the arena page.

**Entry Condition:** Player must be logged into the system and participating in a tournament.

**Exit Condition:** Tournament match ends.

**Use case name:** spectateTournamentMatch

**Participating actors**: Spectator, Advertiser

**Flow of Events**

1. Spectator Logs in.
2. Spectator views the running tournament matches on the homepage.
3. Spectator views requests to view the match.
4. System grants request.
5. Spectator sees an advertisement for a product.
6. Spectator views the match in its entirety.

**Entry Condition:** Spectator Logs in.

**Exit Condition:** Match ends.

**Use case name:** participateInTournament

**Participating actors:** Player

**Flow of Events**

1. Player Logs in.
2. Player goes to the tournaments and signs up for one.
3. Space is available and the Player is added to the tournament.
4. The tournament starts.
5. The player plays in the tournament.
6. Tournament ends.

**Entry Condition:** Player Logs in.

**Exit Condition:** Tournament is over and all players are removed.

**Use case name:** purchaseAdvertisement

**Participating actors:** Advertiser

**Flow of Events**

1. Advertiser Logs in.
2. Advertiser picks a scheme and duration.
3. Advertiser uploads the advertisement.
4. System shows how much the advertisement will cost.
5. Advertiser pays for the advertisement.

**Entry Condition:** Advertiser Logs in.

**Exit Condition:** Advertisement is paid for.

**Use case name:** defineGame

**Participating actors:** operator

**Flow of Events**

1. Operator Logs in.
2. Operator creates new game.
3. Operator enters in information and attributes for the game.
4. Game is added.

**Entry Condition:** Operator Logs in.

**Exit Condition:** Game is created.

**Use case name:** defineTournamentRuleset

**Participating actors:** operator

**Flow of Events**

1. Operator Logs in.
2. Operator starts creation of new ruleset.
3. Operator enters information for ruleset.
4. Operators enters rating value for the ruleset.
5. New ruleset is added.

**Entry Condition:** Operator Logs in.

**Exit Condition:** New ruleset is present.

**Use case name:** createAccount

**Participating actors:** operator, newUser

**Flow of Events**

1. newUser starts the registration process.
2. newUser enters valid username.
3. newUser enters valid password.
4. newUser enters valid email.
5. System sends the newUser’s information to the to confirm validity.
6. Operator reviews newUser’s username, password, and email.
7. Operator grants newUser the “Spectator” account type (include changUserType use case).
8. System updates newUsers’ account type.

**Entry Condition:** New User reaches the login portal.

**Exit Condition:** User is added to Spectator account type.

**Use case name:** organizeNewTournamentLeague

**Participating actors:** LeagueOwner, Advertiser

**Flow of Events**

1. LeagueOwner Logs in.
2. LeagueOwner requests new league.
3. LeagueOwner enters name and attributes of the league.
4. System verifies and creates league.
5. LeagueOwner requests a sponsor for the league.
6. Advertiser receives request and purchases and advertisement (include purchaseAdvertisement use case).
7. League is created and sponsored.

**Entry Condition:** LeagueOwner Logs in.

**Exit Condition:** New League is in system.

**Use case name:** changeUserType

**Participating actors:** operator, spectator

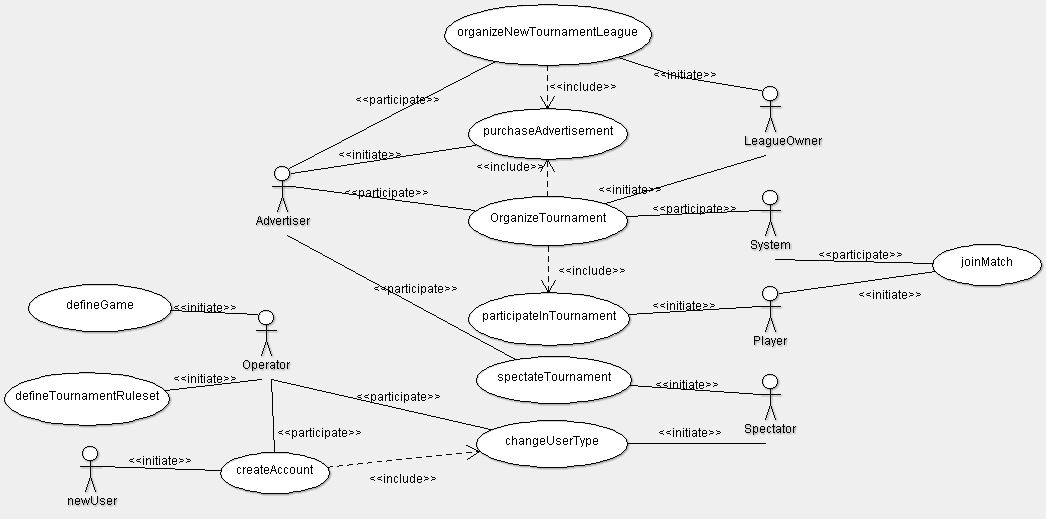
**Flow of Events**

1. Spectator Logs in.
2. Spectator requests to be changed to player.
3. Operator grants request and changes spectator to player.

**Entry Condition:** Spectator Logs in.

**Exit Condition:** Spectator’s userType is changed to operator.

**Use case Diagram**



**Concrete Use Cases**

**Use case name:** organizeTournament

**Participating actors**: LeagueOwner, Advertiser, and System

**Entry Condition:** LeagueOwner logs into system and System verifies the login

**Flow of Events**

1. LeagueOwner sends a request to the System for a new tournament containing tournament attributes such as tournament name, official/unofficial, style, automatic or individual player rating, and number of players through the organizeTournament form.
2. The System verifies the tournament, creates the tournament, and sends a verification message back to the LeagueOwner.
3. LeagueOwner then submits a sponsorship request form back to the system containing the tournament name.
4. System posts a sponsorship request form to the news page where advertisers may view the request and choose to sponsor the tournament through purchaseAdvertisement(). Once sponsored, the sponsorship is verified by the System.
5. Verification of Sponsorship message is now sent to LeagueOwner from the System containing the amount sponsored and name of Advertiser. The tournament name and balance is also now shown on the Advertisers My Advertisement page.
6. LeagueOwner now makes tournament status open on the news page, where players can navigate open tournaments and choose to joinTournament(). When joined by players using joinTournament(), System iterates numberOfPlayers until the closing condition of maxPlayers is met. Once the limit is reached tournament status becomes closed.
7. System now changes tournament status to ready and the tournament begins.

**Exit Condition:** System sends a notification to Players and LeagueOwner that the tournament entry is closed and has begun.

**Use case name:** joinMatch

**Participating actors:** Player, System

**Entry Condition:** Player logs in to the System through the log in portal and System verifies the login.

**Flow of Events:**

1. layer navigates to the arena page to view the matches he is registered in.
2. Player sends a request to the System containing the name of the match they would like to join by clicking the joinMatch button.
3. System verifies the request and allows the Player to enter into the match, where the Player can view the match and statistics.
4. Player interacts with the system to participate in the match.
5. Match ends when one player emerges victorious.

**Exit Condition:** Match’s status is changed to closed, and System redirects the Player back to the arena page.

**Use case name:** spectateTournamentMatch

**Participating actors:** System, Spectator, and Advertiser

**Entry Condition:** Spectator logs in to the System through the log in portal and System verifies the login.

**Flow of Events:**

1. Spectator navigates through the homepage to view the tournaments with status running.
2. Spectator sends a request to the System containing the name of the match they would like to view by clicking the viewMatch button.
3. System verifies the request and allows the spectator to enter into a viewing status, where the Spectator can view the match and statistics.
4. System then pauses the Spectators viewing of the match, checks the sponsor, and redirects them to the sponsors Ad for a duration of time. After time has completed, System redirects the Spectator back to the match viewing.
5. Spectator continues viewing the match.

**Exit Condition:** Match’s status is changed to closed, and System redirects the Spectator back to the homepage, where they may repeat the process of spectateTournament.

**Use case name:** participateInTournament

**Participating actors:** Player and System

**Entry Condition:** Player logs in to the system through the login portal and the System verifies the login.

**Flow of Events:**

1. Player navigates through the homepage viewing tournaments with status open. Once the tournament is selected through joinTournament(), a request is sent to the System with the name of the tournament they would like to join.
2. System receives the request form and checks the tournaments conditions of maxPlayers and status. If the status of the tournament is open and the maxPlayer condition is not met then the System enters the Player into the tournament, iterates the numberOfPlayers, and sends a verification message back to the Player.
3. Once the tournament status is changed to closed and ready the System notifies the Player that the tournament has begun.
4. Player enters into the tournament and participates in matches. During this time, System checks tournament attributes of automatic/ individual player statistics and if automatic then System updates player statistics in each match.
5. Once tournament status is changed to closed by the System, System notifies Player of tournament status ending and redirects them back to the homepage where they may repeat the process of participateInTournament.

**Exit Condition:** Tournament status is changed to closed and all players are redirected to the homepage by the System.

**Use case name:** purchaseAdvertisement

**Participating actors:** System and Advertiser

**Entry Condition:** Advertiser logs into System through the login portal and System verifies the login

**Flow of Events:**

1. After login, System redirects the Advertiser to their My Advertisement page where they may view their existing ads, balance, or choose to purchaseAdvertisement by clicking the purchaseAdvertisement button.
2. When Advertiser clicks purchaseAdvertisement, a request is sent to the System with the Advertisers name. System verifies the request and sends them back a request containing missing attributes of scheme and duration for the Advertiser to fill in.
3. Advertiser then receives the request from the system and fills in the missing attributes and resends the request back to the System.
4. System then alotts space in the My Advertisement page for an ad with the scheme and duration requested by the Advertiser. System then redirects the Advertiser to an upload portal, where the Advertiser can upload the ad of their choosing using the upload button.
5. After the upload button is pressed, System calculates the balance due based on scheme and duration, and changes the balance under the Advertiser’s My Advertisement page.
6. System then sends a request to the interface for the advertisement to be posted. The interface checks the conditions of spaceAvailable and if true then the Ad is posted to the Interface.

**Exit Condition:** Ad duration end condition is met after the time boundary has been reached and Ad is removed from the Interface.

**Use case name:** defineGame

**Participating actors:** System Operator and Interface

**Entry Condition:** Operator logs in to the System through the log in portal and the System verifies the login

**Flow of Events:**

1. Once entered into the System, System redirects the Operator to their homepage. Operator presses the defineGame button, which sends the System a request to allocate space for a new game.
2. System verifies the request, redirecting the Operator to a defineGame process request form with missing attributes such as gameStyle, expert rating formulas, and other attributes.
3. Operator provides the attributes for the request form and submits the form to the System through the submit button
4. System receives the form and creates a game, in the already allocated space, with the attributes the operator desires.

**Exit Condition:** System posts the game to the Interface, where everyone is notified of the new games creation.

**Use case name:** defineTournamentRuleset

**Participating actors:** Operator and System

**Entry Condition:** Operator logs in to the System through the login portal and System verifies the login.

**Flow of Events:**

1. Once the operator is logged in, the System redirects the Operator to the homepage. The Operator then presses the defineTournamentRuleset button sending a request to the System for allocation of space for a new ruleset.
2. System receives the request, allocates space for the new ruleset, and redirects the Operator to defineTournamentRuleset process request form with missing attributes for the Operator to fill in.
3. Operator provides these attributes for the request form and submits the form to the System through the submit button.
4. System receives this form and redirects the Operator to a new request form for rating values.
5. Operator provides the information and submits it to the System through the submit button.
6. System receives the information and creates a new ruleset in the already allocated space.

**Exit Condition:** System posts the ruleset to the Interface, where everyone is notified of the addition of a new ruleset.

**Use case name:** createAccount

**Participating actors:** Operator, Spectator, and System

**Entry Condition:** New User reaches the login portal and clicks the createAccount button.

**Flow of Events:**

1. When the createAccount button is clicked, a request is sent to the System for allocation of space for a new account. System verifies the request and redirects the New User to createAccount request form with the missing attributes of type, username, email, and password.
2. New User provides the information and submits it to the System using the submit button.
3. System receives the form and attributes, and checks the conditions within the Interface of attributeNotTaken. If True the Interface sends the System a confirmation.
4. When the System receives the confirmation, they then resend the the submitted request form to the Operator for review.
5. When Operator receives the verification form they verify the request, and the Operator is redirected to a request form to approve creation.
6. Operator submits the request form to the system using the submit button.
7. System receives the confirmation and creates the new account in the allocated space.

**Exit Condition:** System notifies the New User of account creation and redirects them to the home page

**Use case name:** organizeNewTournamentLeague

**Participating actors:** LeagueOwner, and System

**Entry Condition:** LeagueOwner signs into the System through the login portal and the System verifies the login.

**Flow of Events:**

1. After login, the System redirects the LeagueOwner to their home page. When the LeagueOwner clicks the organizeNewTournamentLeague button, a request is sent to the System to verify and allocate space for a new tournament league.
2. System receives the request and verifies the request. After verification, the System allocates space for the new tournament league and redirects the LeagueOwner to request form with missing attributes such as league name, etc;
3. LeagueOwner provides the information and submits the attributes to the System through the submit button.
4. System receives the submitted request form and verifies. Once verified, it creates the new tournament league in the allocated space and posts it to the Interface. System then sends the LeagueOwner a confirmation message.

**Exit Condition:** League Owner is redirected to the home page.

**Use case name:** changeUserType

**Participating actors:** Operator, Spectator, and System

**Entry Condition:** Spectator signs in to the System through the login portal and the System verifies the login

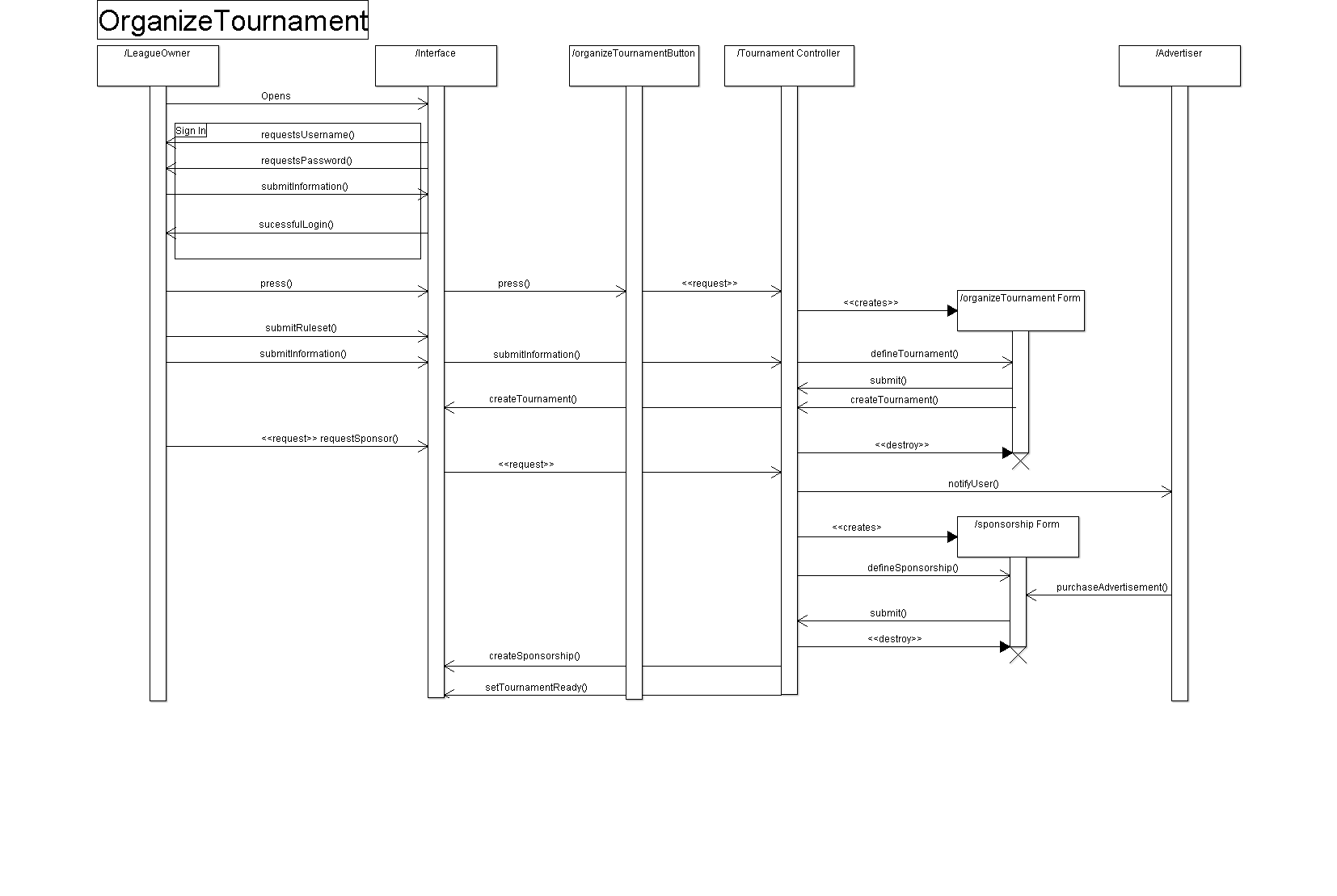
**Flow of Events:**

1. Once logged in, the System redirects the Spectator to the home page.
2. Spectator then clicks the changeUserType button which sends a request to the System for a change in userStatus from Spectator to Player.
3. System receives the request form and forwards it to the Operator for verification.
4. When the Operator receives the request, they are redirected to a userType form where they can change the userType of the Spectator. Operator changes userType from Spectator to Player and clicks submit, sending the verification to the System.
5. System receives the verification.

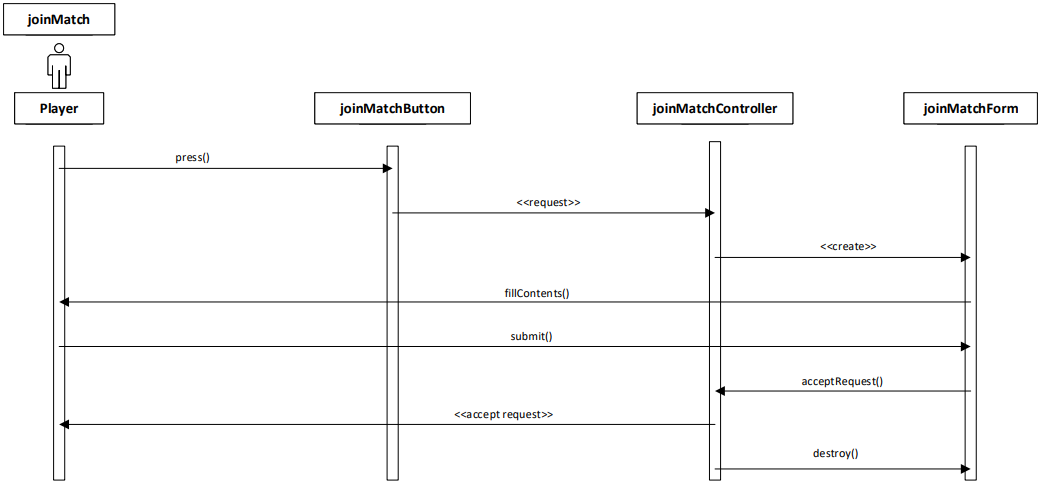
**Exit Condition:** System notifies Spectator that their userType has been changed from Spectator to Player.

**Detailed System Sequence Diagrams**

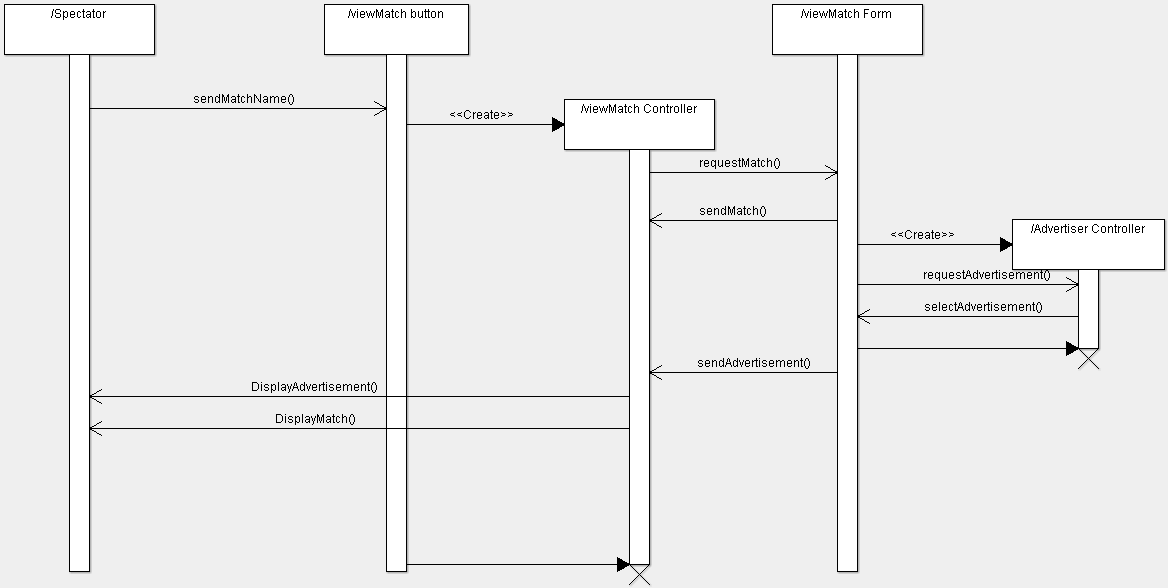
**OrganizeTournament DSSD**

****

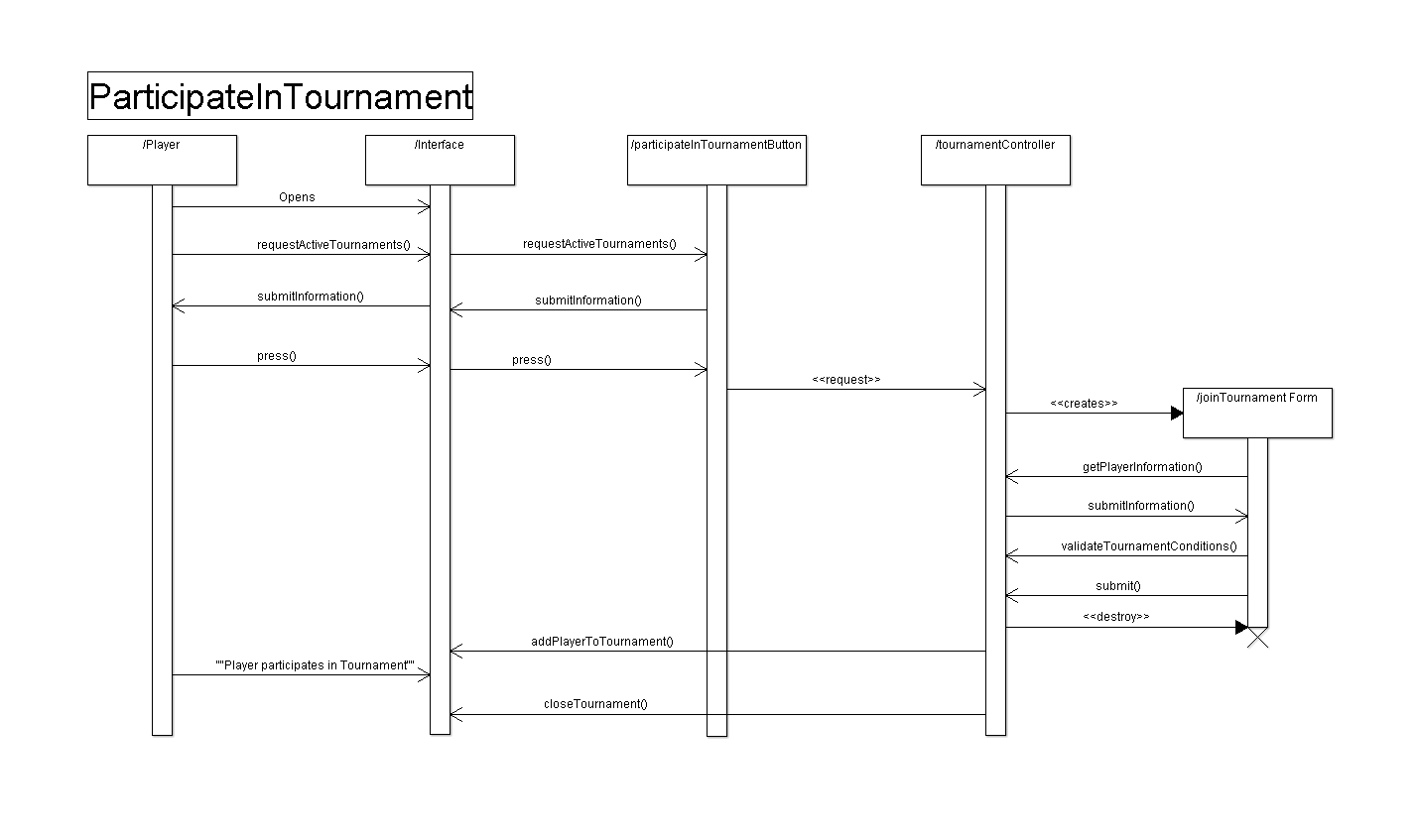
**JoinMatch DSSD**



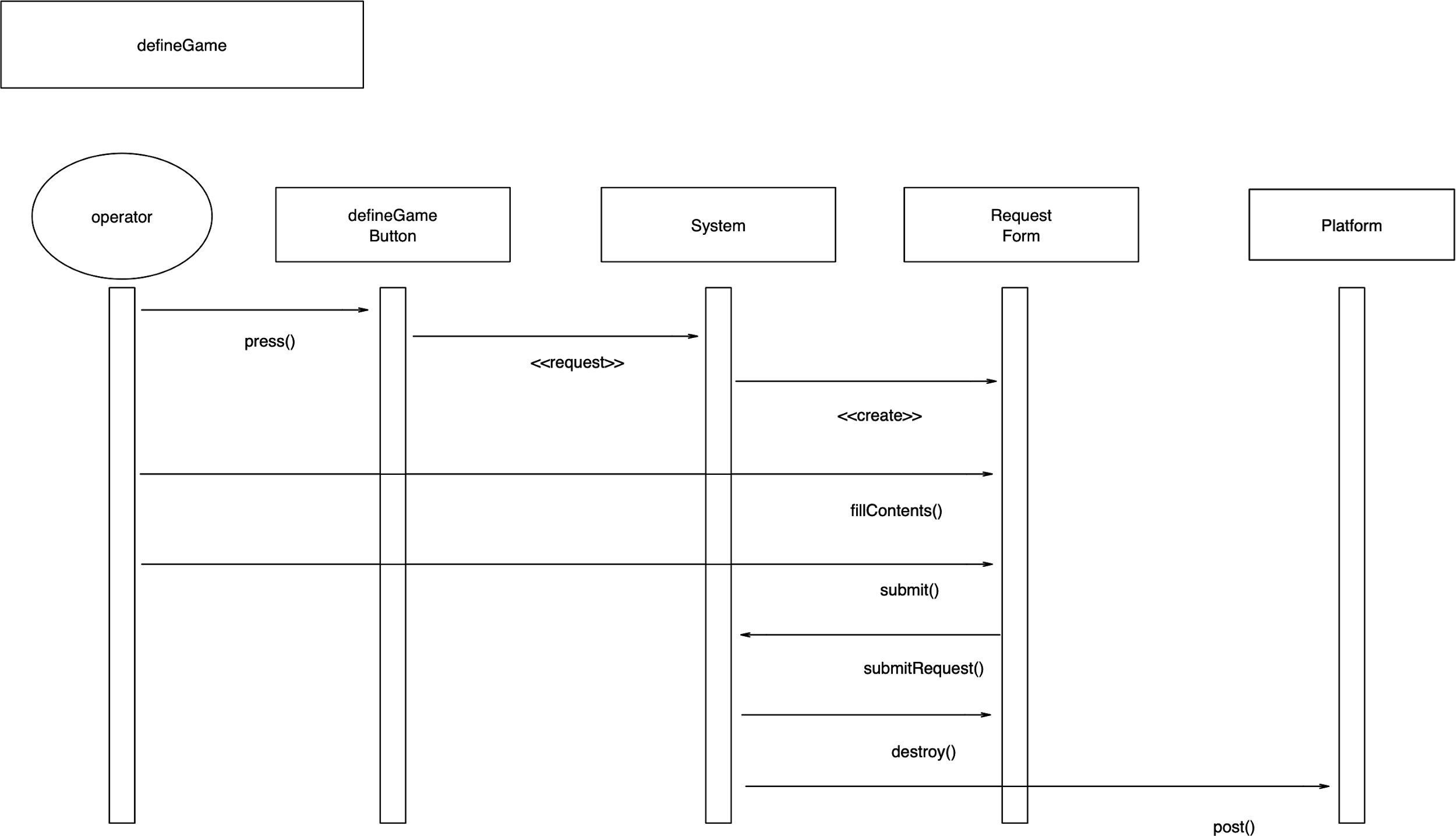
**SpectateTournamentMatch DSSD**



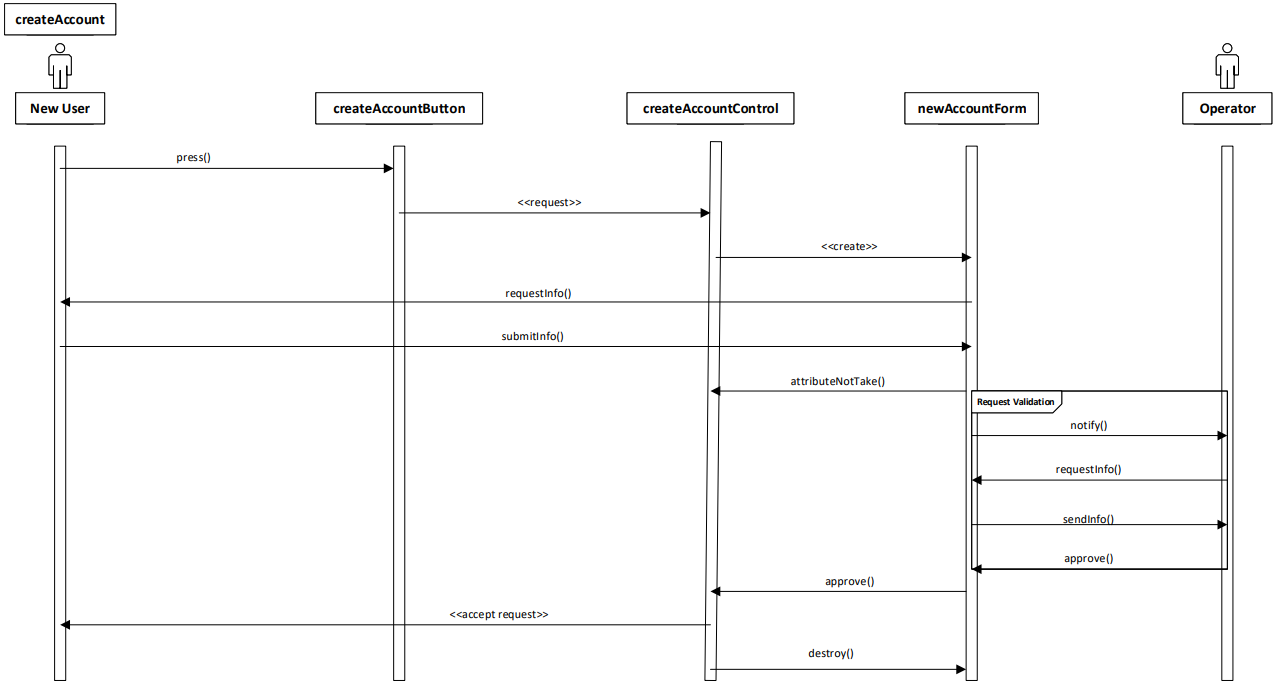
**ParticipateInTournament DSSD**



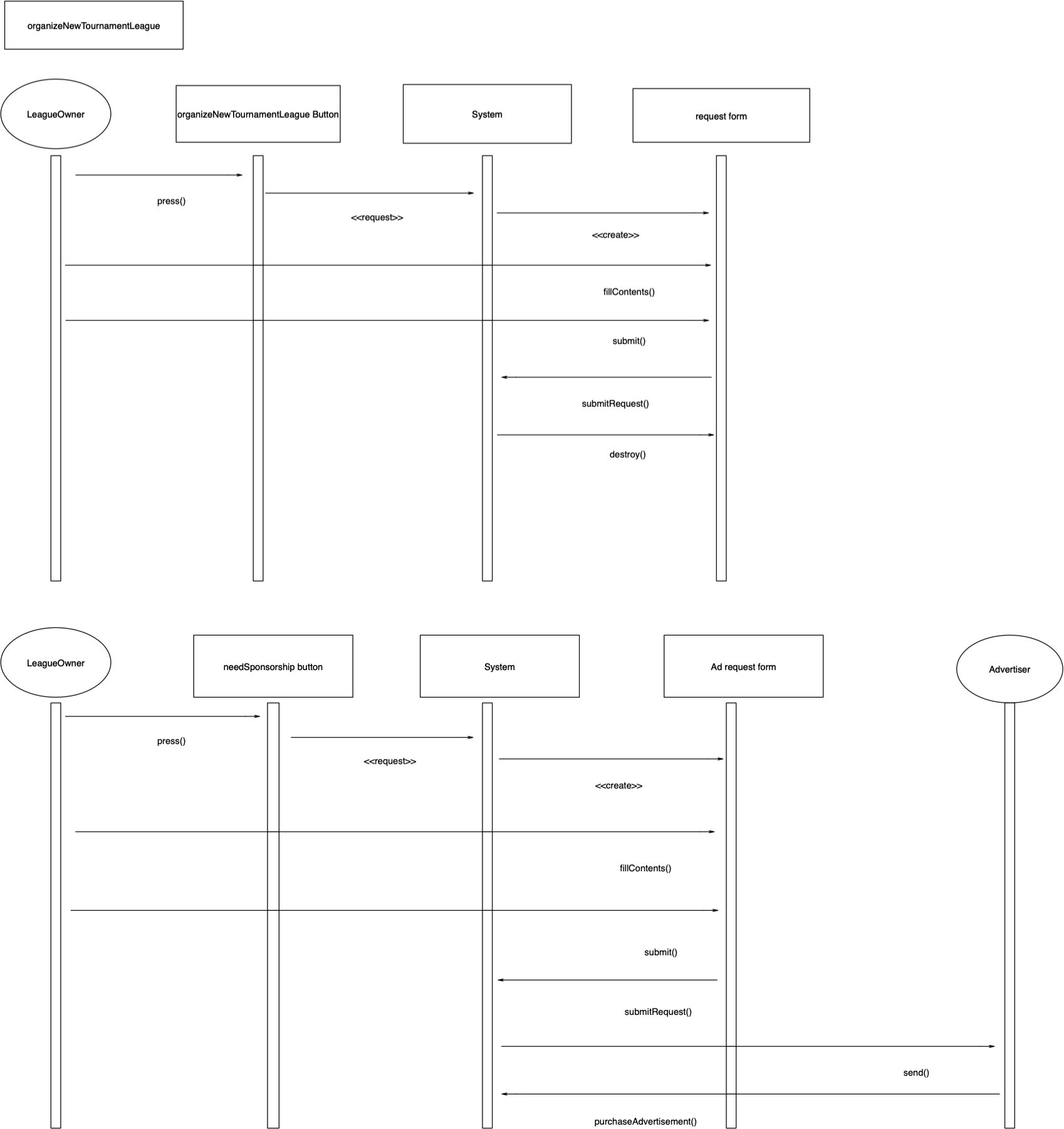
**DefineGame DSSD**



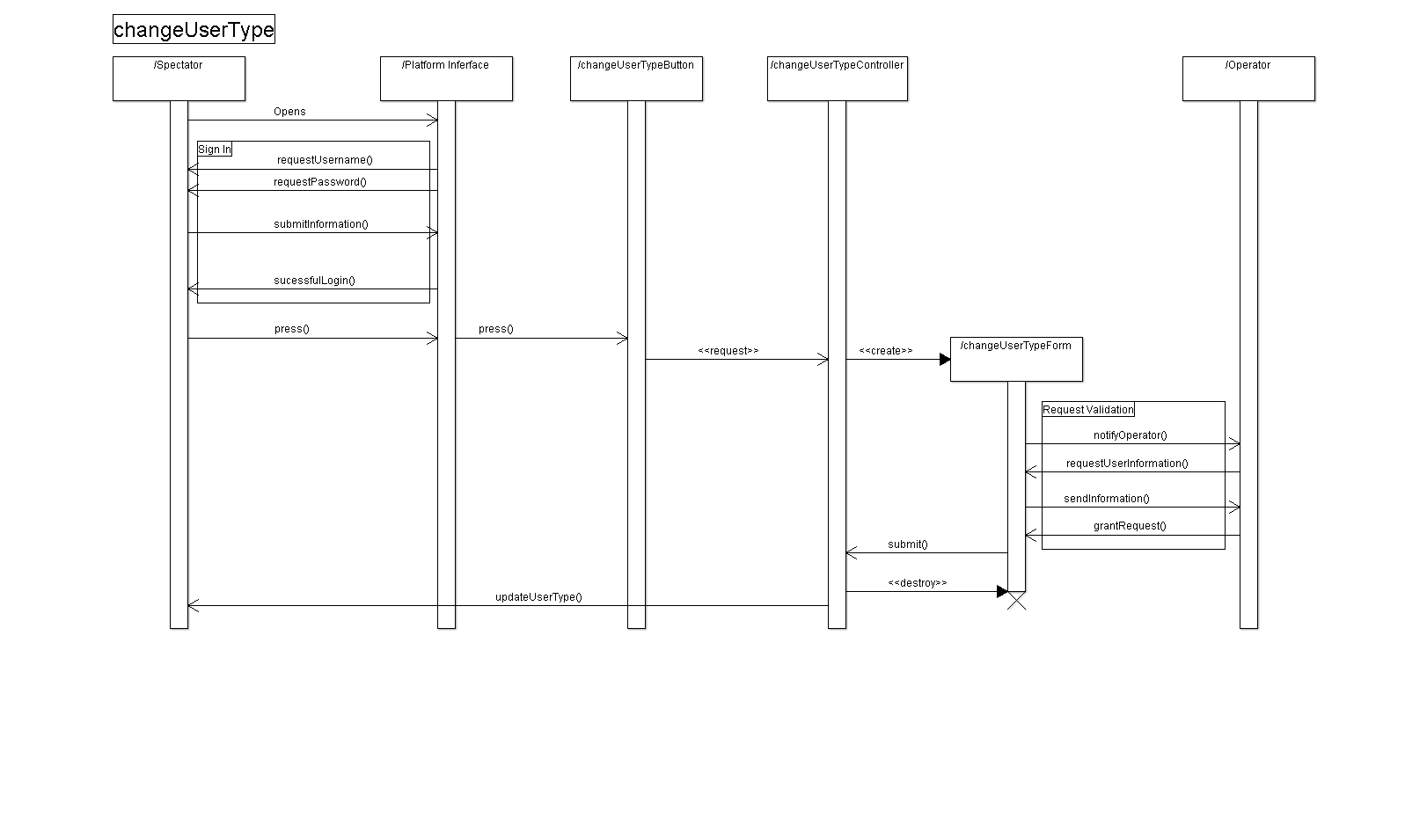
**CreateAccount DSSD**



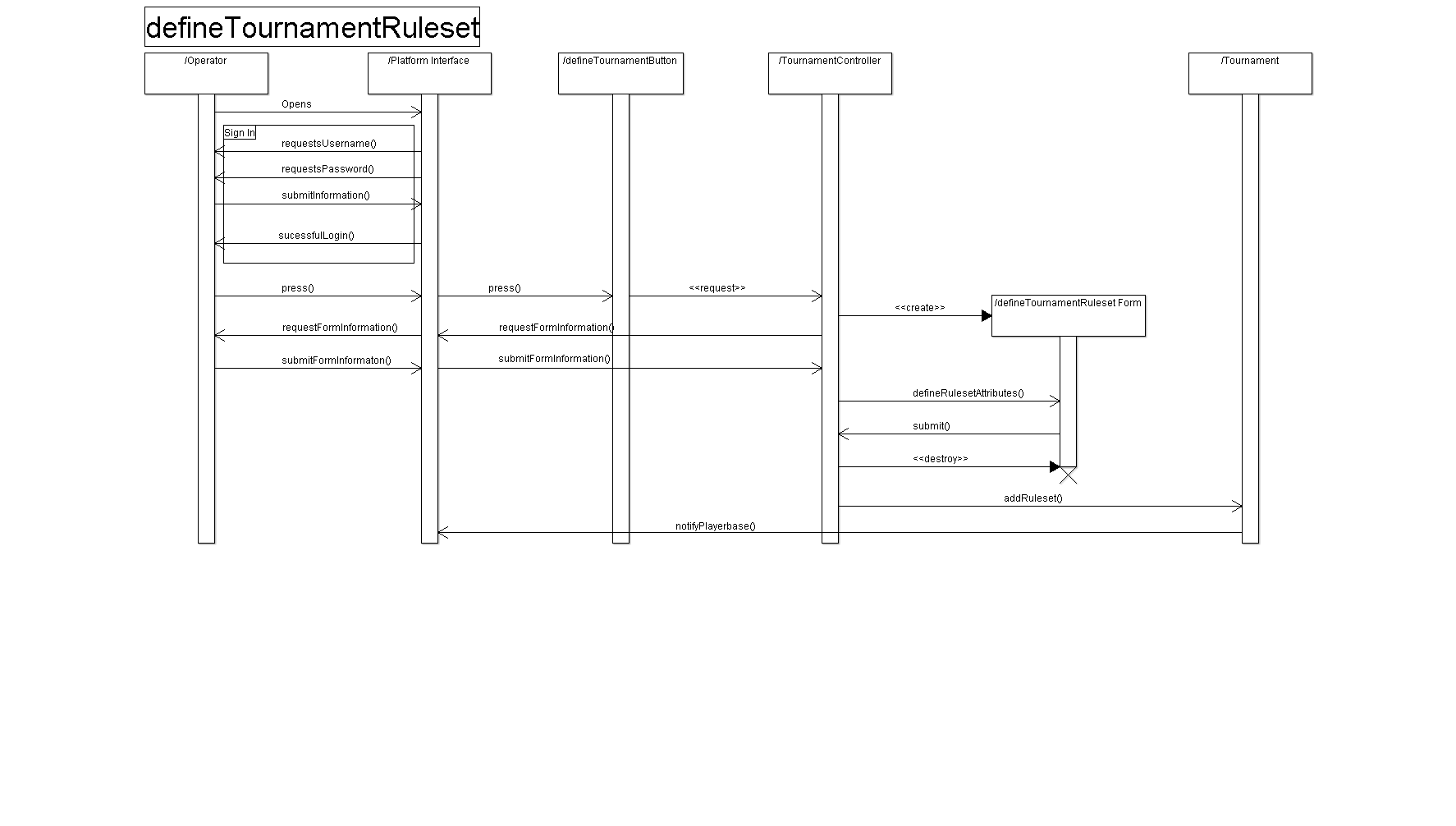
**OrganizeNewTournamentLeague DSSD**



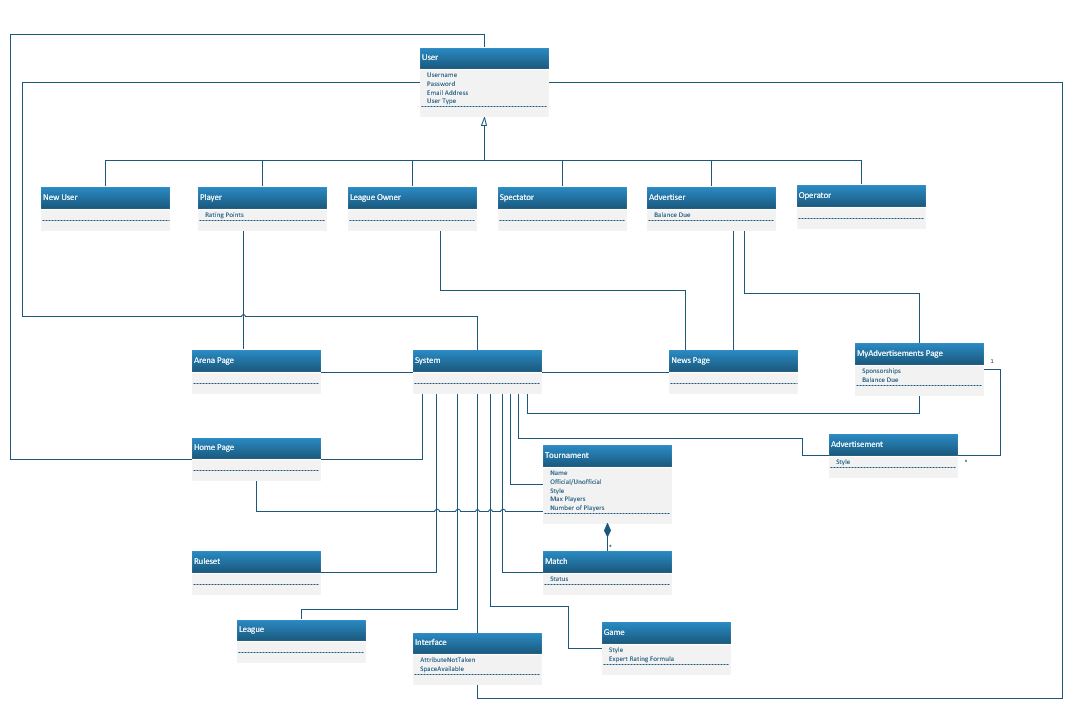
**ChangeUserType DSSD**



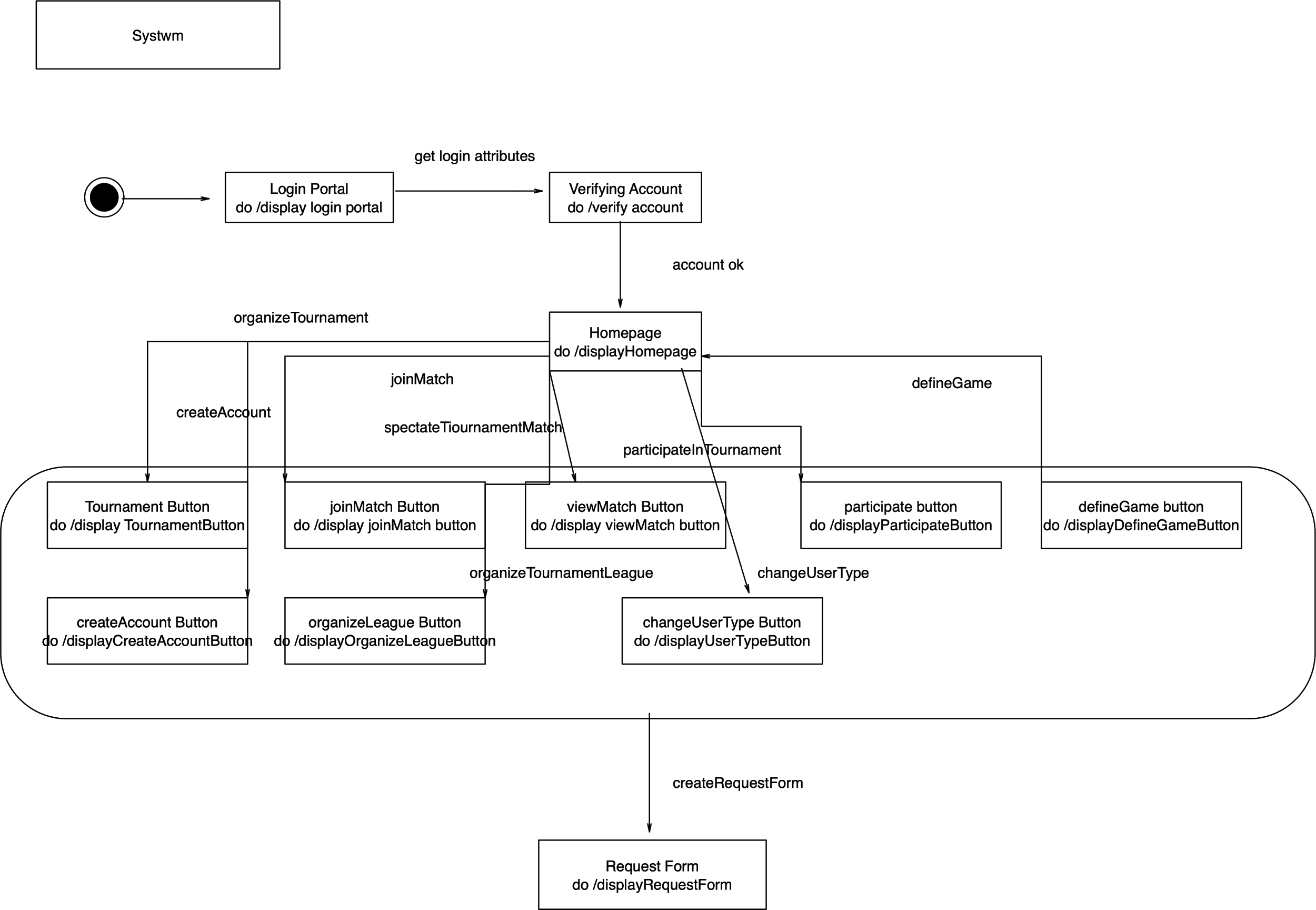
**DefineTournamentRuleset DSSD**

****

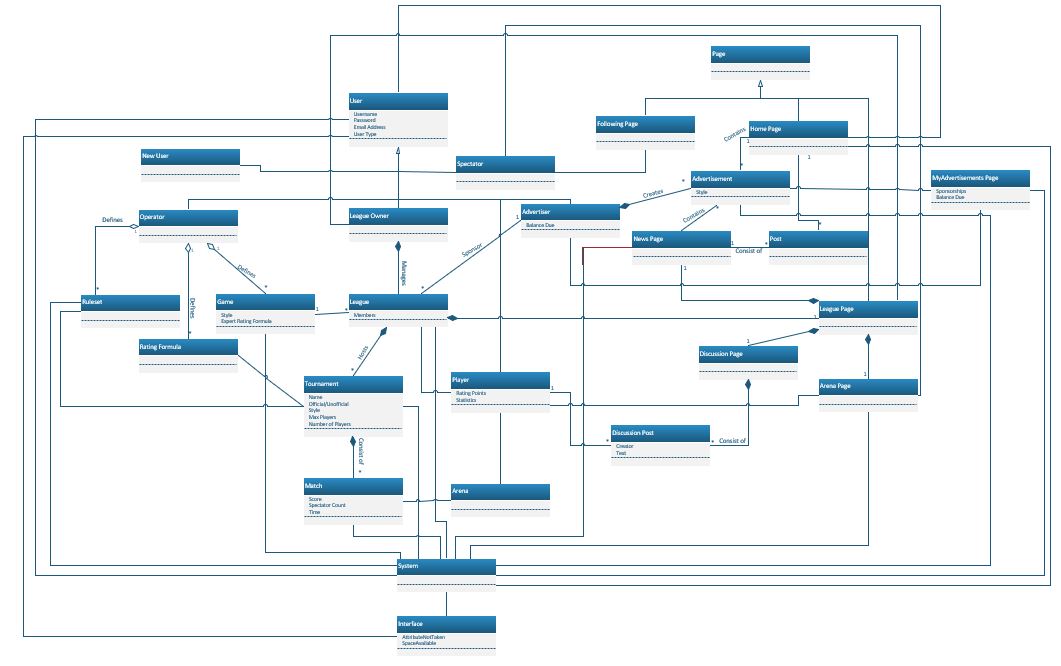
**Application Class Model**



**Application State Model**



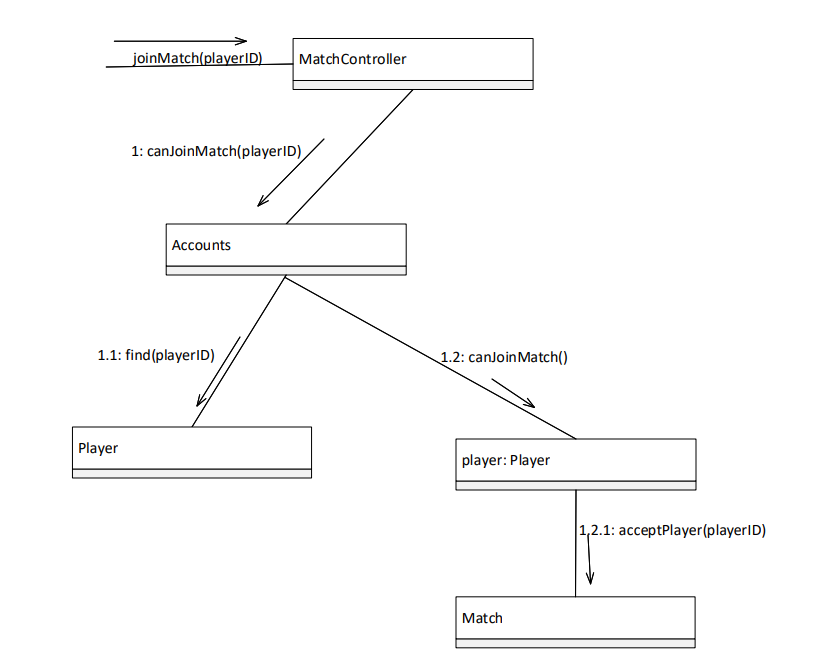
**Consolidated Class Model**



**Architectural Design**

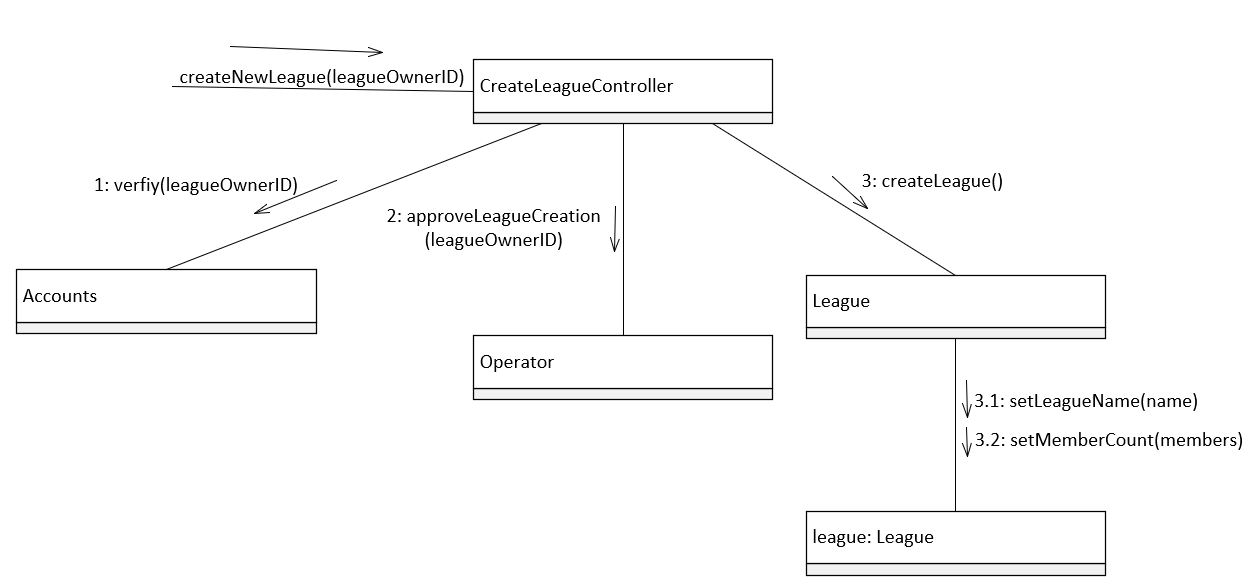
**Collaboration Diagrams**

**Join Match Collaboration Diagram**

****

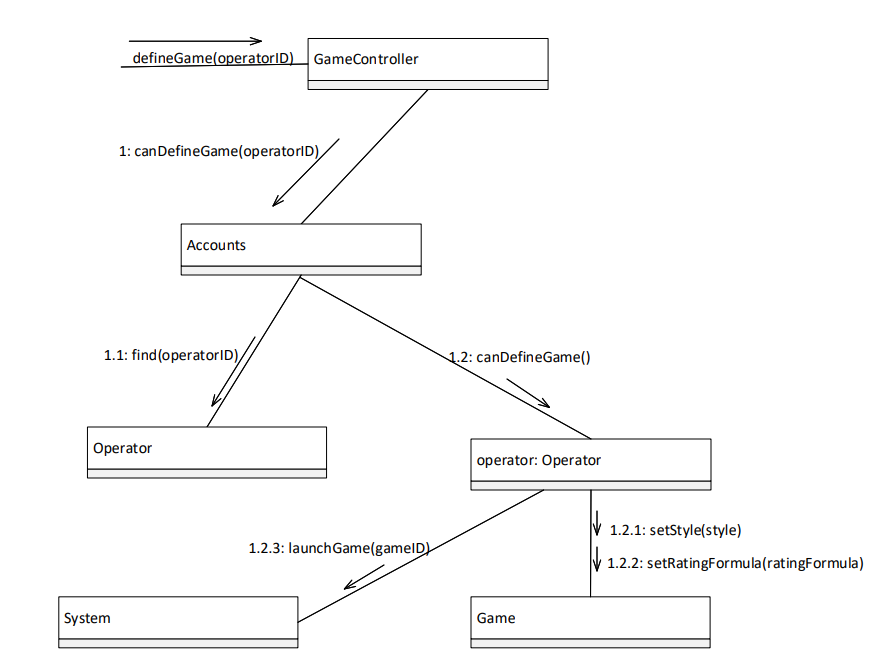
We used the Controller GRASP guideline for the joinMatch(playerID) responsibility. We used the Expert GRASP guideline for the canJoinMatch(playerID), find(playerID), and canJoinMatch() responsibilities. We used the Low Coupling GRASP guideline for the acceptPlayer(playerID) responsibility.

**Organize Tournament League Collaboration Diagram**

****

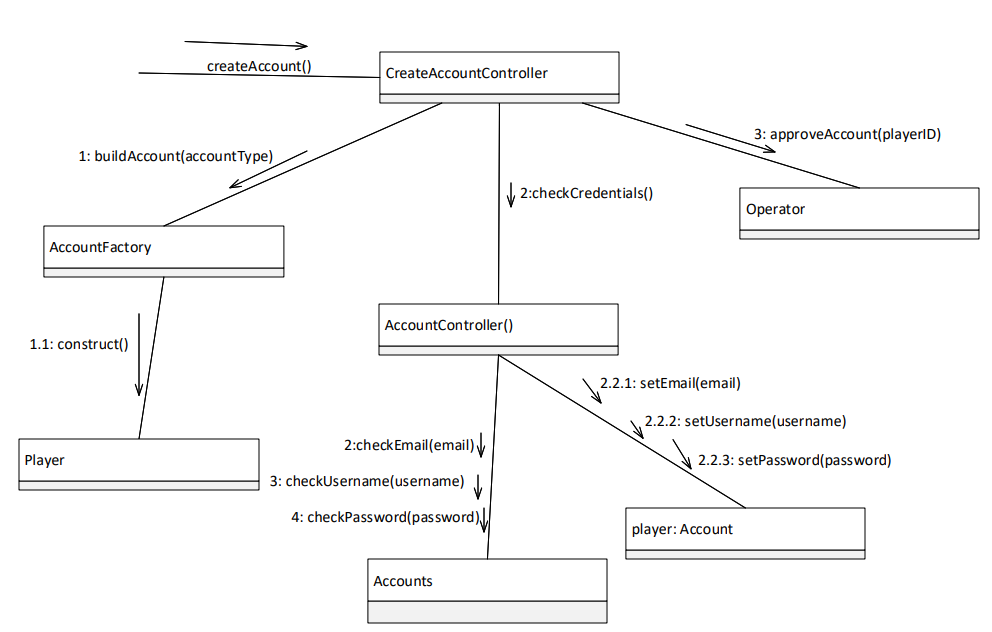
We used the Controller GRASP guideline for the createNewLeague(leagueOwnerID) responsibility. We used the Expert GRASP guideline for verify(LeagueOwnerID), approveLeagueCreation(leagueOwnerID), and createLeague() responsibilities. We used the Low Coupling GRASP guideline for the setLeagueName(name) and setMemberCount(members) responsibilities.

**Define Game Collaboration Diagram**

****

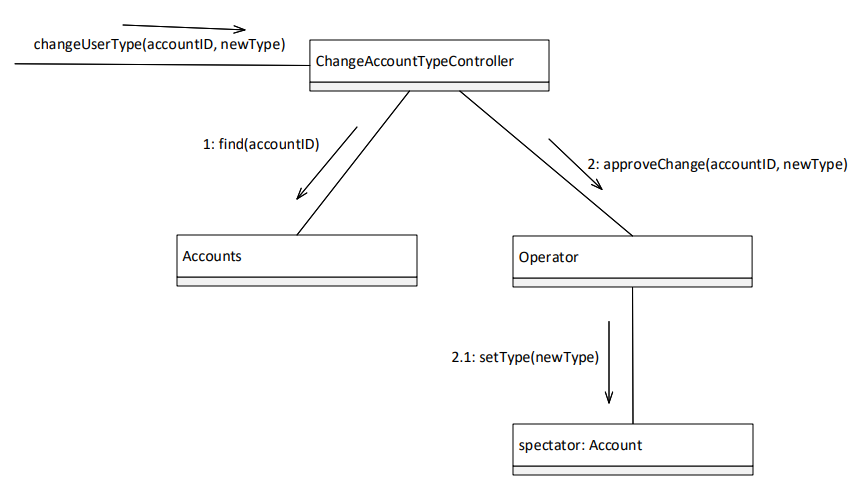
We used the Controller GRASP guideline for the defineGame(operatorID) and launchGame(gameID) responsibilities. We used the Expert GRASP guideline for the canDefineGame(operatorID), find(operatorID), and canDefineGame() responsibilities. We used the Low Coupling GRASP guideline for the setStyle(style) and setRatingFormula(ratingFormula) responsibilities.

**Create Account Collaboration Diagram**

****

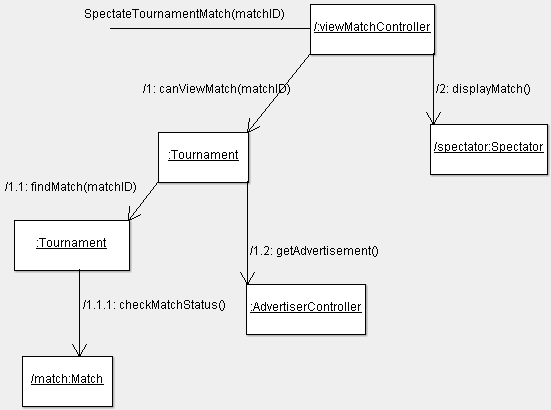
We used the Controller GRASP guideline for the createAccount and checkCredentials() responsibilities. We used the Expert GRASP guideline for the buildAccount(accountType), construct(), checkEmail(email), checkUsername(username), checkPassword(password), setEmail(email), setUsername(username), setPassword(password), and approveAccount(playerID) responsibilities.

**Change User Type Collaboration Diagram**

****

We used the Controller GRASP guideline for the changeUserType(accountID), newUserType() responsibility. We used the Expert GRASP guideline for find(accountID), approveChange(accountID, newType), and setNewType(newType) responsibilities.

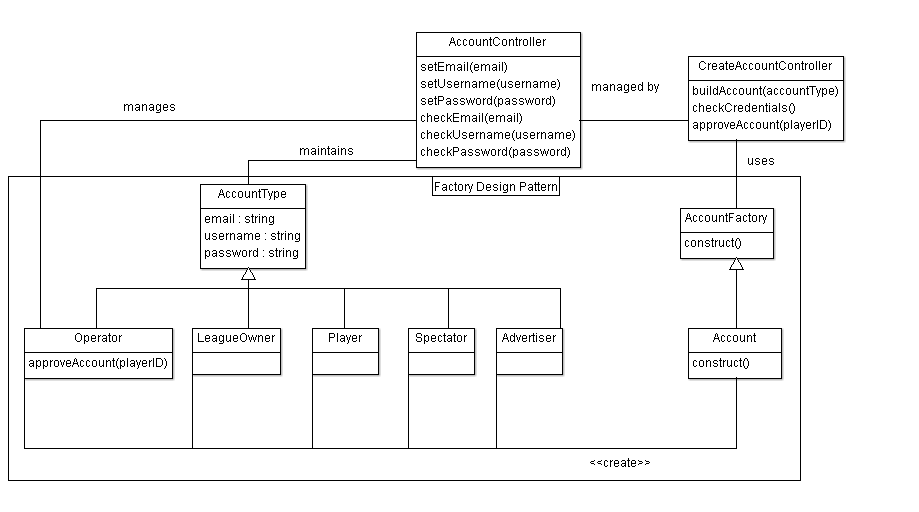
**View Match Collaboration Diagram**

****

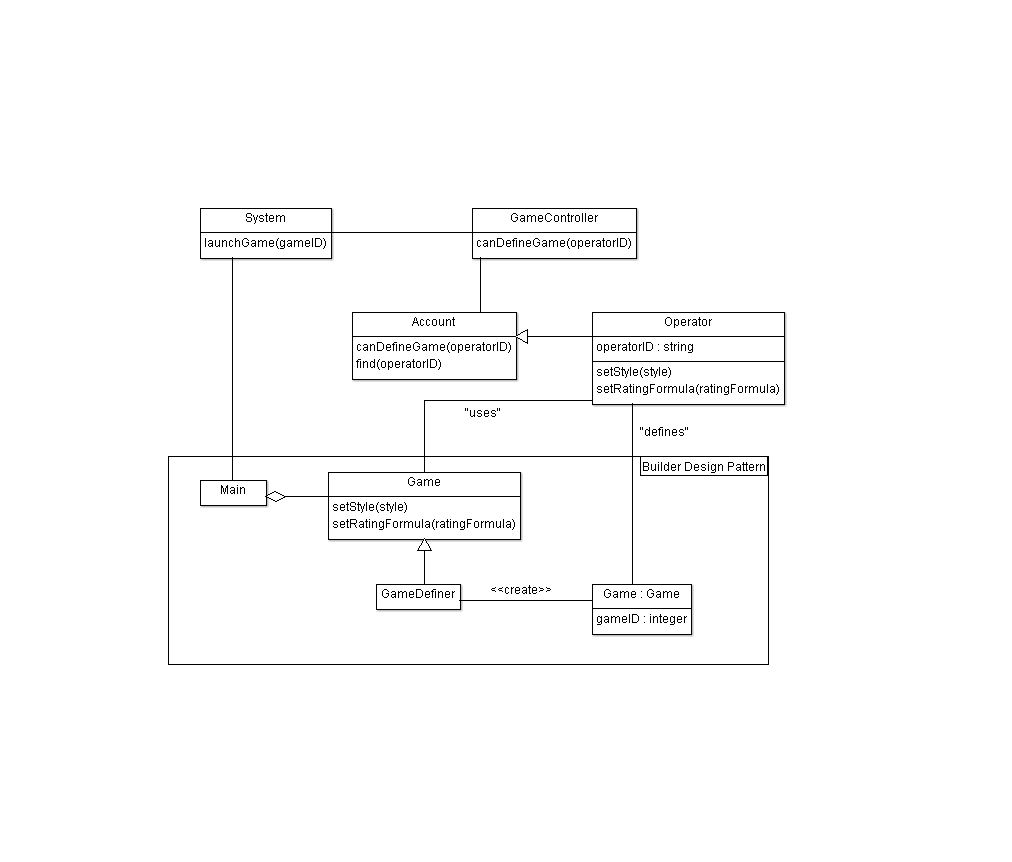
We used the Controller GRASP guideline for the spectateTournamentMatch(matchID) responsibility. We used the Creator GRASP guideline for the findMatch(matchID) responsibility. We used the Expert GRASP guideline for the canViewMatch(matchID), displayMatch() and getAdvertisement() responsibility.

**Design Class Diagrams**

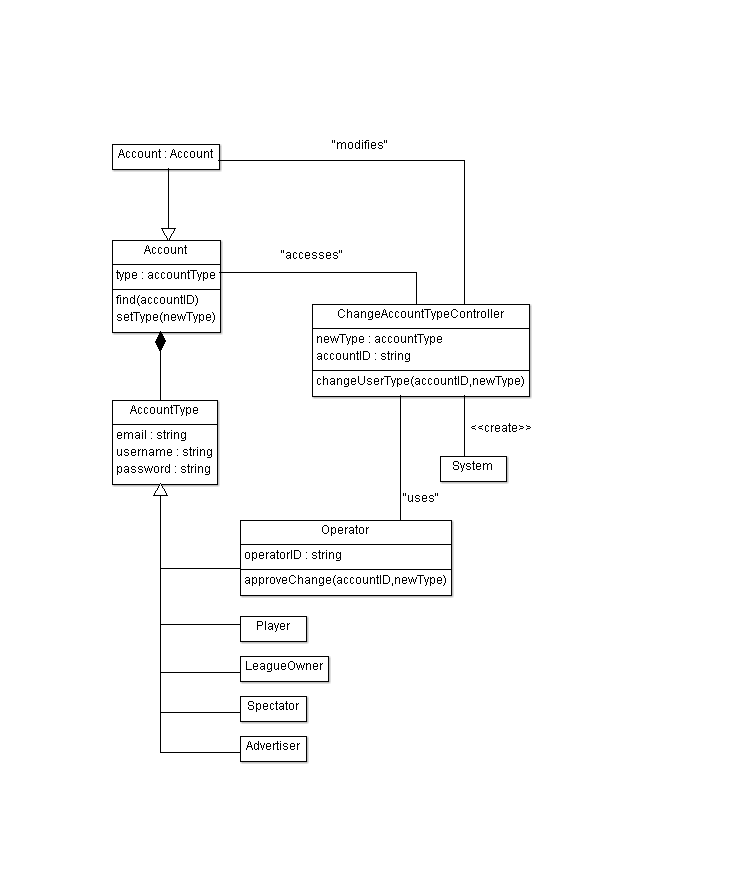
**Create Account Design Class Diagram**



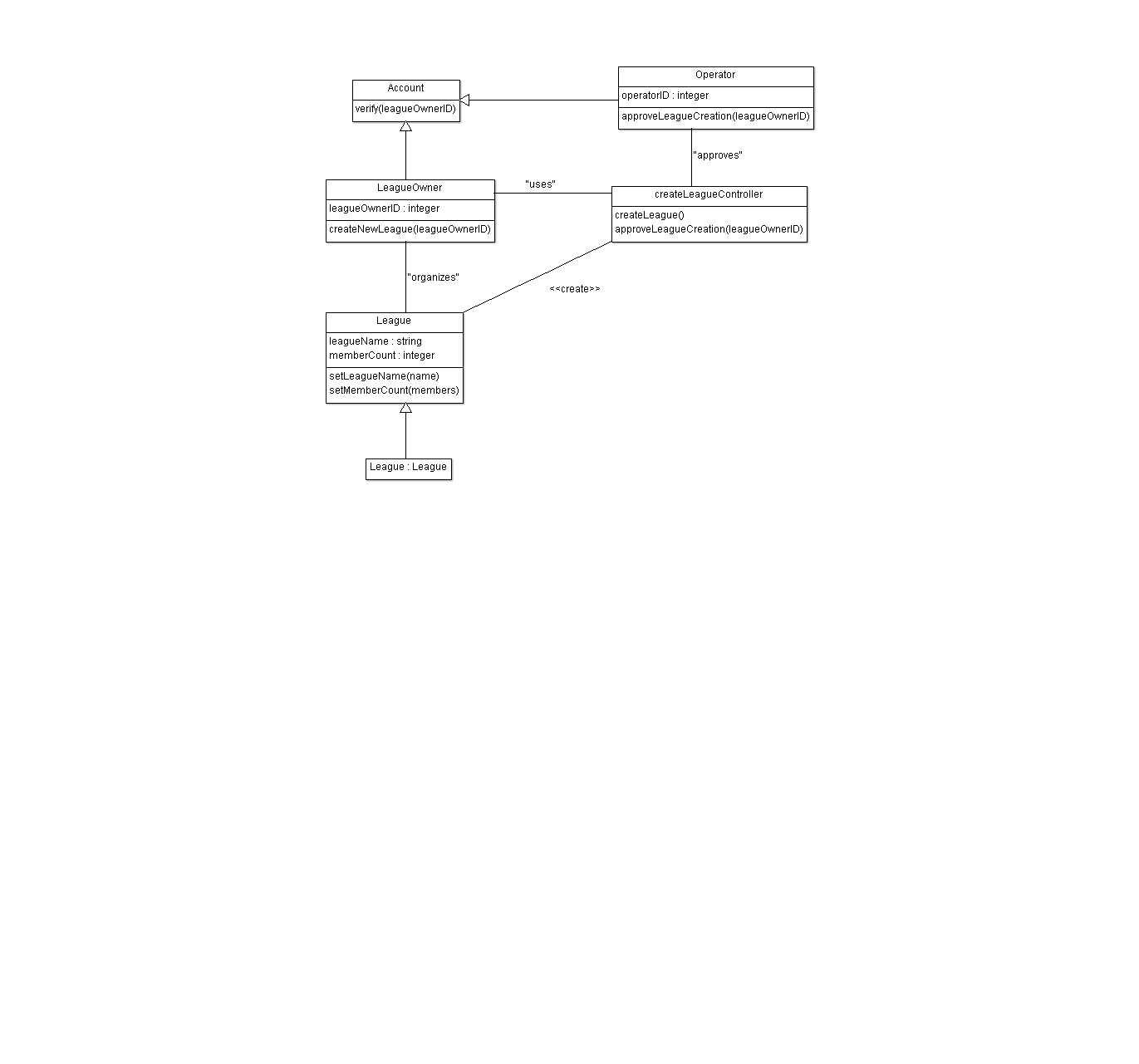
**DefineGame Design Class Diagram**



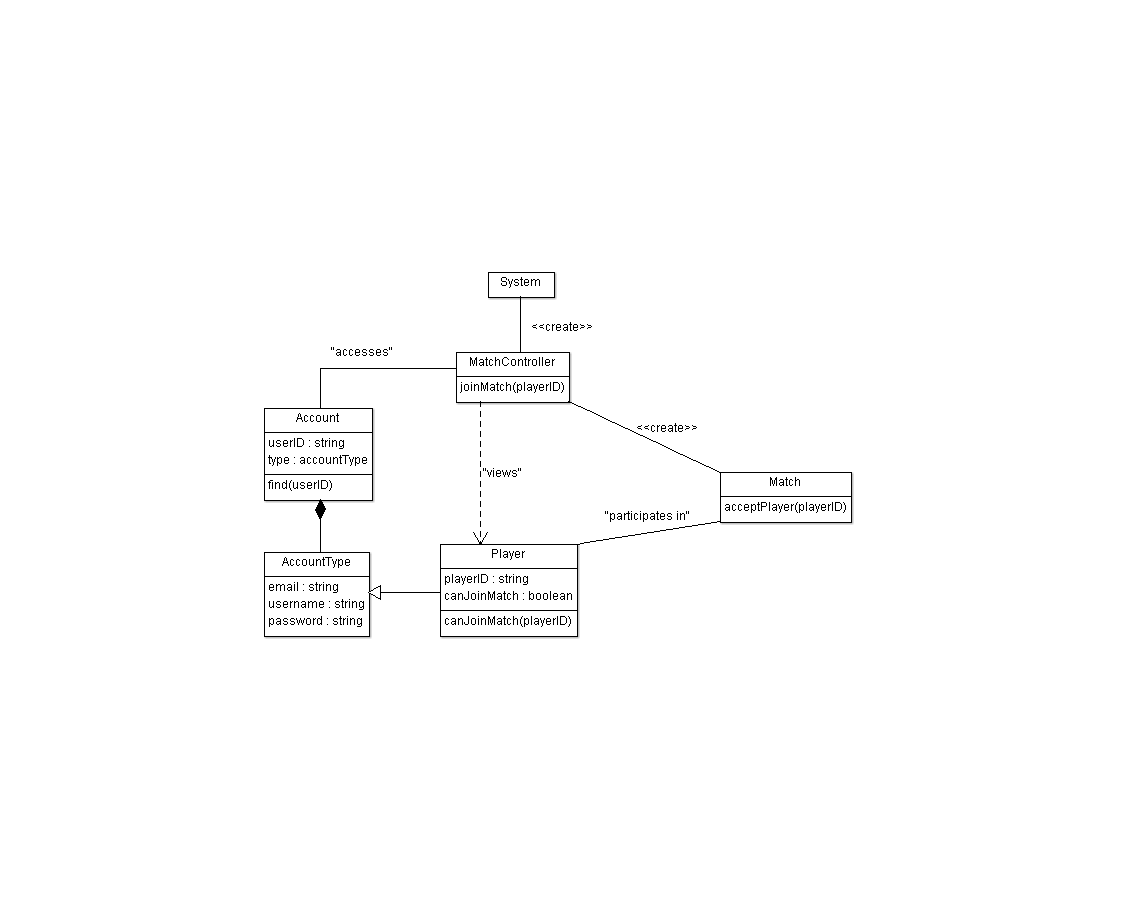
**Change User Type Design Class Diagram**



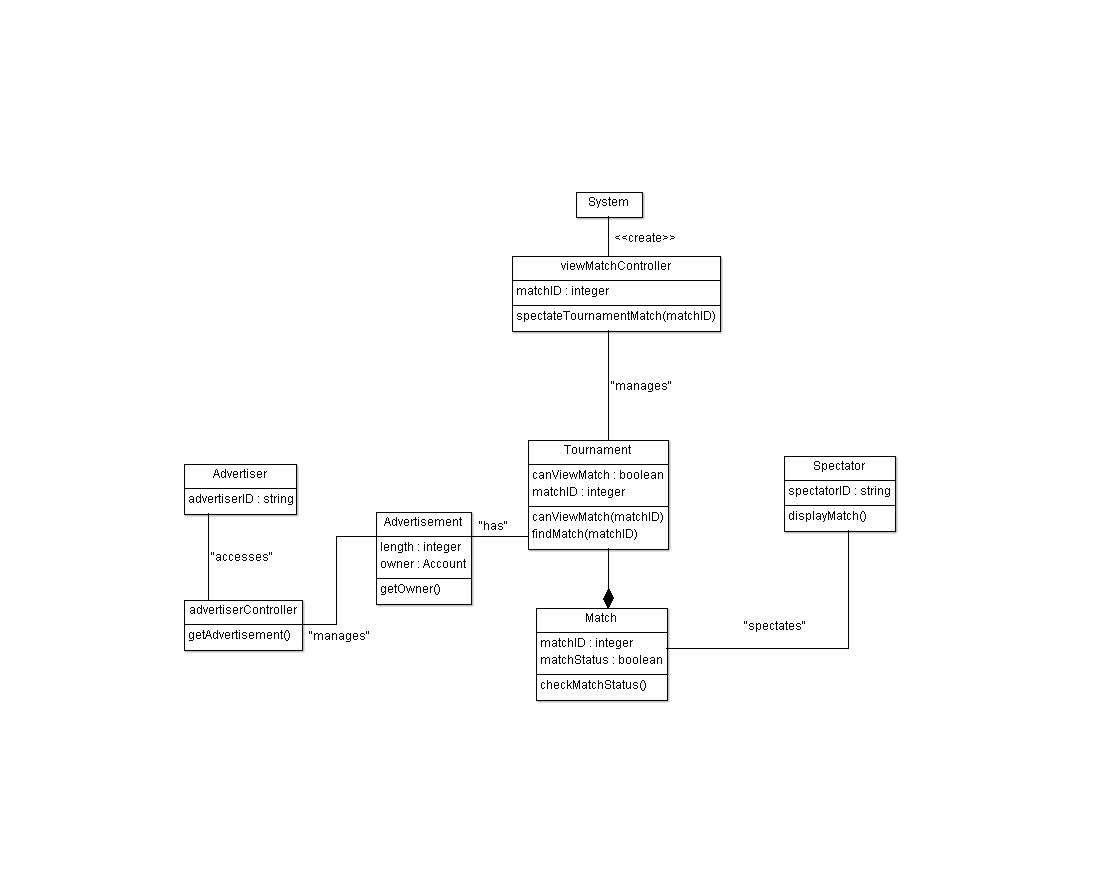
**Organize Tournament League Design Class Diagram**

****

**JoinMatch Design Class Diagram**

****

**View Match Design Class Diagram**



**Design Patterns**

**Factory Design Pattern**

We chose to use the Factory Design Pattern for the creation of accounts. We implemented the class AccountFactory as our creator. We implemented the class Account as the concrete creator. We implemented the AccountType class as our product. Finally, we implemented the classes Operator, LeagueOwner, Player, Spectator, and Advertiser as the concrete products. The reason we chose to use the Factory Design Pattern is because our AccountFactory class cannot anticipate the type of account it needs to create. Another reason is because its subclasses specify the objects it creates. A third reason is because accounts need to be mutable. A pro of using this design pattern is that the creation of a new account type does not require a large amount of new code.

**Builder Design Pattern**

We chose to use the Builder Design Pattern for defining game types. We implemented the class Game as our builder. We implemented the class GameDefiner as our concrete builder. We used the Main class as our director. And the actual games that are created are the products (example: TicTacToe). The reason we chose to use the Builder Design Pattern is because our construction process has to allow different representations for each game object. Another reason we chose the Builder Design Pattern is because the games need to be immutable. A pro of using this design pattern is that it allows you to have required and optional characteristics for each game. Another advantage is that it is impossible to make incomplete games since you do not use setter methods to give the games attributes.

**Object Design**

**Interface Contract**

**Account**

find(operatorID)

Pre-Conditions: operatorID exists

Post-Conditions: an operator object is returned

Invariant: operatorID

find(accountID)

Pre-Conditions: accountID exists

Post-Conditions: an account object is returned

Invariant: accountID

find(userID)

Pre-Conditions: userID exists

Post-Conditions: an user object is returned

Invariant: userID

canDefineGame(operatorID)

Pre-Conditions: operator exists

Post-Conditions: operator is able to define games

Invariant: operatorID

setType(newType)

Pre-Conditions: Account exists

Post-Conditions:accountType == newType

Invariant: email, username, and password

verify(leagueOwnerID)

Pre-Conditions: leagueOwner exists

Post-Conditions: leagueOwner verified and returned

Invariant: leagueOwnerID

**Advertisement**

getOwner()

Pre-Conditions: advertisement exists

Post-Conditions: owner identified and returned

Invariant: Owner

**Tournament**

canViewMatch(matchId)

Pre-Conditions: match exists

Post-Conditions: match verified and canViewMatch set to true or false

Invariant: matchId

findMatch(matchId)

Pre-Conditions: match exists

Post-Conditions- match found and returned

Invariant: matchId

**Match**

acceptPlayer(PlayerId)

Pre-Conditions: player exists

Post-Conditions: player verified and accept == true or accept == false

Invariant: PlayerId

checkMatchStatus()

Pre-Conditions: match exists

Post-Conditions: match found and status == true or status == false;

Invariant: matchId

**Game**

setStyle(Style)

Pre-Conditions:game exists

Post-Conditions;game verified and style set

Invariant: gameId

setRatingFormula(ratingFormula)

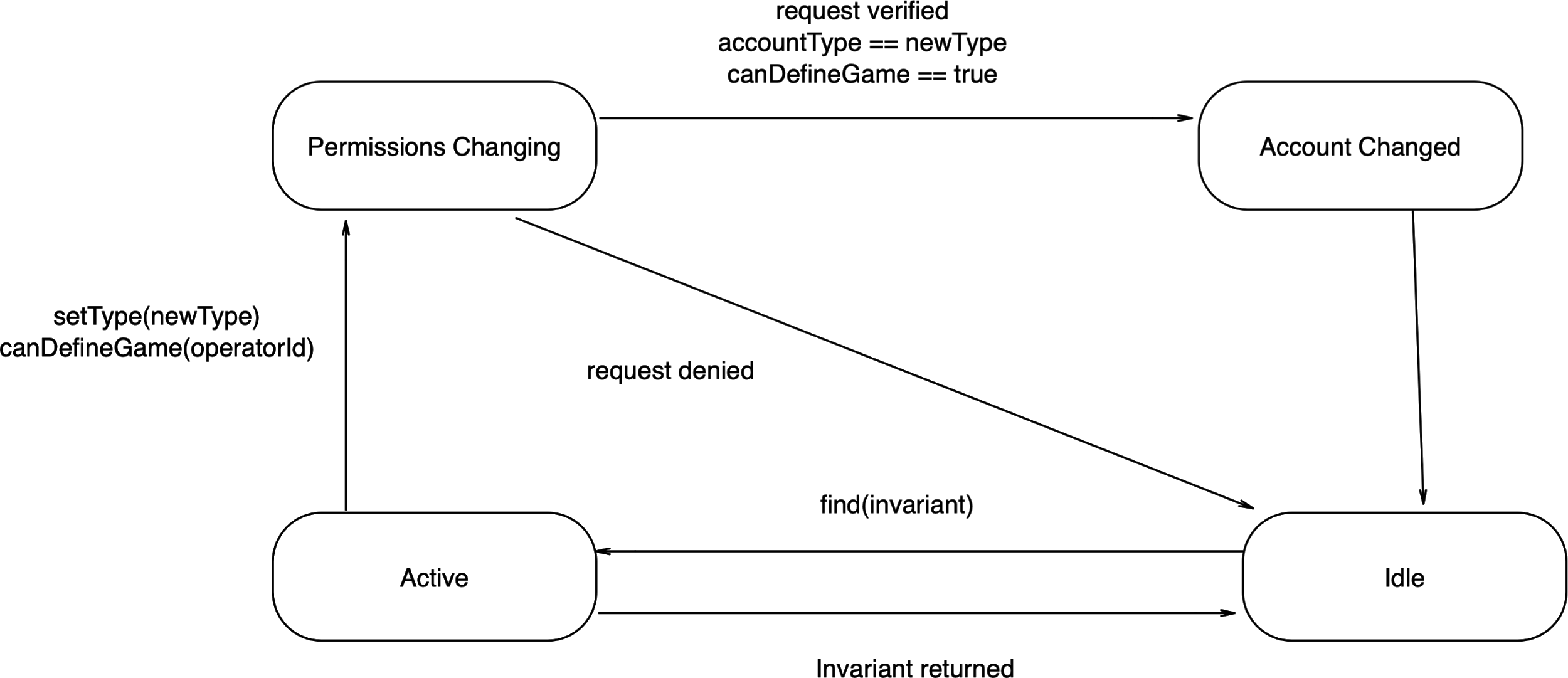
Pre-Conditions: game exists

Post-Conditions: game verified and ratingFormula set

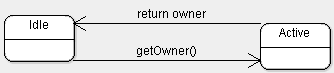
Invariant: gameId

**UML Statechart**

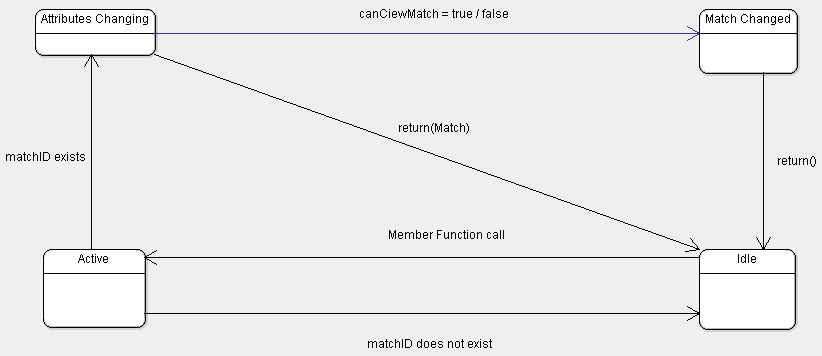
**Account**



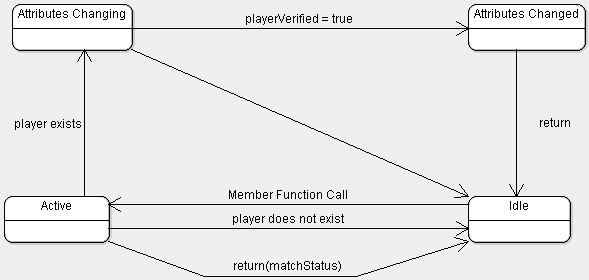
**Advertisement**



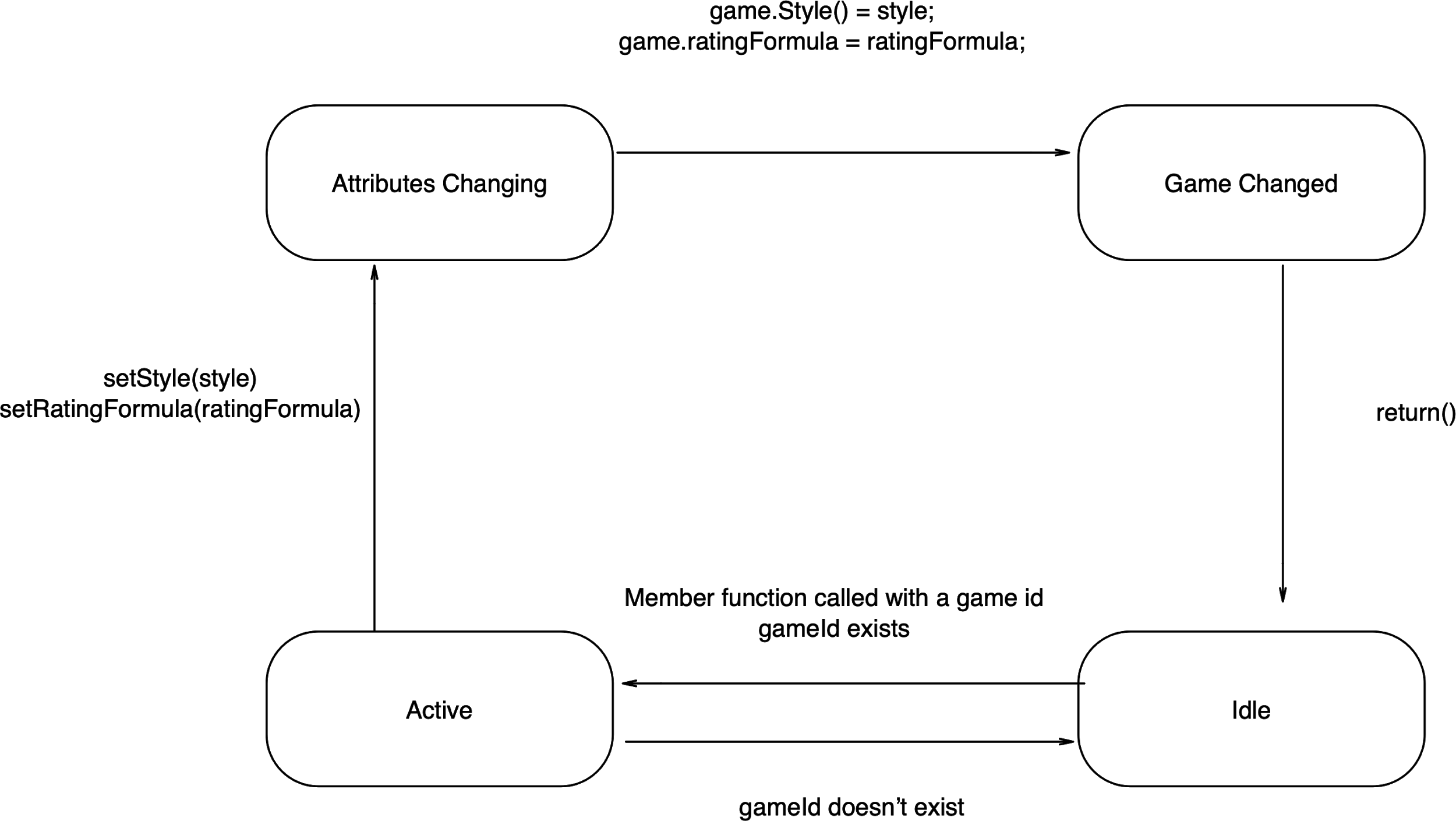
**Tournament**



**Match**



**Game**

****

**Procedural Behavior Specification of Methods**

**Account**

setType(newType) {

if(accountId does not exist) {

Return;

Else {

Switch(newType) {

Case operator: accountType == Operator

Case leagueOwner: accountType == leagueOwner

Case spectator: accountType == Spectator

Case player: accountType == Player

Case advertiser: accountType == Advertiser

}

Print “Account type changed”

Return;

}

**Advertisement**

getOwner() {

If (advertisement does not exist) {

Print “can not find advertisement/advertisement doesn’t exist”

Return;

}

Else {

If (advertisement.owner() does not exist)

Return “cant find owner of advertisement”

else{

Return advertisement.owner()

}

}

}

**Tournament**

canViewMatch(matchId) {

If (matchId does not exist) {

Return “The match does not exist”

}

Else {

If (match is allowed to be view) {

canViewMatch == true;

}

Else {

canViewMatch == false;

}

}

}

**Match**

acceptPlayer(playerId) {

If (player does not exist) {

Return “Player does not exist”

}

Else {

If (player does not have permission to join) {

acceptPlayer == false

}

Else {

acceptPlayer == true;

}

Return

}

}

**Game**

setRatingFormula(ratingFormula) {

If (game does not exist) {

Return “game does not exist”

}

Else {

game.ratingFormula() == ratingFormula

Return “rating formula set to “ + ratingFormula

}

}

**Model Review**

Our application analysis follows all the guidelines for being correct. The names of all of our concepts, classes, methods, and controllers are understandable by the user. Our abstract classes correspond to user-level concepts. For example, the class Account and all its attributes correspond to each account type that it is the parent class of. All of our entity and boundary objects have meaningful noun phrases as their names. Some of our entity and controller object names are: Interface, newAccountForm, and joinMatchForm. All of our use cases and controller objects have meaningful verb phrases as their names. Some of our use case and controller object names are: Create Account, createAccountButton, Join Match, and joinMatchButton.

Our application analysis also follows all the guidelines for being complete. All of our objects are used in at least one use case. Ruleset is used and created in the use case of OrganizeTournament. Operator is used in the use case of DefineGame. Rating Formula is used and created in the use case of DefineGame. Game is used and created in the use case of DefineGame. Match is used in the use case of JoinMatch. Tournament is used and created in the use case of OrganizeTournament. League is used in the use case of OrganizeNewTournamentLeague. LeagueOwner is also used in the use case of OrganizeNewTournamentLeague. Account is used and created in the use case of CreateAccount. Arena is used in the use case of JoinMatch. Player is used in the use case of ParticipateInTournament. Spectator is used in the use case of SpectateTournament. Advertiser is used in the use case of PurchaseAdvertisement. Advertisement is used in the use case of PurchaseAdvertisement. Home Page is used in the use case of SpectateTournamentMatch. Arena Page is used in the use case of JoinMatch. News Page is used in the use case of OrganizeTournament. Post is used in the use case of DefineGame. Every association is used and each multiplicity makes sense. Each controller object is capable of accessing the other classes needed. Another reason that our application analysis is complete is because in each basic state there is a transition defined for all possible events. Each state is also targeted by at least one transition. Initial states have an incoming transition from the initial pseudo-state. In each basic state, considering also inherited transitions, guards of transitions triggered by the same event form a tautology.

Our application analysis follows all the guidelines for being consistent as well. None of our classes have the same names as any of our other classes. Every use case, class, and attribute that has a similar name also denotes a similar concept. For example, DefineGame and DefineTournamentRuleset both have similar concepts. Another example is that all of the pages have extremely similar concepts also. If any of our classes have similar attributes and associations then they are generalized, which include the account and page generalizations. Our Detailed System Sequence Diagrams are consistent with our UML Statecharts. Our Detailed System Sequence Diagrams are also consistent without our UML Class Models. If an object needs to be able to contact another object or class in one of our Detailed System Sequence Diagrams, then they are properly associated in our UML Class Models. Another reason our application analysis is consistent is that if there are two or more transitions that are originating from the same state and triggered by the same even, then their guards could not be true at the same time.

Our designs for our Video Game Development and Management Platform is correct. We carefully looked through all of our designs to make sure there are no mistakes. Every design we made follows the guidelines for UML designs. All of our pseudo code is correct and performs the task it is supposed to perform. When implemented into an object oriented language the code will have no problems. All of our design patterns follow the rules for implementation and have the appropriate classes. All of our collaboration diagrams follow the GRASP guidelines for each responsibility.

Our designs for our Video Game Development and Management Platform is also complete. Our design ensures that our program always does everything that it is supposed to do. As we created different diagrams and charts to help us model and design our use cases, we were very careful to ensure that, when implemented into code form, our designs will accomplish all the requirements that they are supposed to.

Our designs for our Video Game Development and Management Platform is also consistent. Our design ensures that our program always behaves as expected. Each design diagram, design pattern, method, and state chart is consistent. As we moved from one step to another we made sure to check that we were accurately transforming each diagram from one form to another without changing any key features about what the diagrams represent. This ensures that we have transformational accuracy throughout our entire project, and especially from one model to another. All of our design patterns are present in our class diagrams and are implemented accordingly.

Finally, all of our diagrams and models are consistent throughout our entire application analysis and design.