Software Requirements Models
"Spaceships: Battle for the Frontier"

Prepared for Professor Jörg Kienzle
Computer Science Department
McGill University

By

The Dangling Pointers

Sundaram Sankarapapa
Tina Latif
Avery Morin
Andrew Fogarty
Emir Aydin

1. Introduction

1.1 Purpose

The purpose of this document is to provide a general understanding between the requirements and the final product for the video game **Spaceships: Battle for the Frontier.** We shall present the use cases in a manner that shall hopefully allow all parties to be comfortable and content with the product to be developed and submitted before the end of the winter 2014 semester at McGill University.

We attempt to create a general consensus among all major parties involved in the development of this product, including, any Teaching Assistants, Professors, and Students involved in the development and / or marking of this software project.

1.2 Scope of the Document and System

This software to be developed is a Computer Video Game to be developed for either the Windows Operating System or Macintosh Operating System.

This document will contain and discuss all parts of the requirements elicited in the **Battleships v1.2** document available on the course webpage. The software will involve the interaction of multiple players *across two different machines* over an online server that will connect the players. The players will be able to store usage information, including but not limited to, wins and losses.

This document includes a: **Use Case Model, Environment Model, Concept Model, Operation Model, and Protocol Model.** These five (5) models will interact to present a general and thorough description of the project to be developed.

1.3 Definitions, Acronyms, and Abbreviations

Player:
When we discuss the players, we are describing the actual players who are interacting with the system by playing the game. They are distinguished from other important actors in that they have simple goals of *joining the game, playing the game, and exiting the system*.

Current Player:
The player who has the active turn.

Other Player:
The player who is waiting for the Current Player to finish their turn.

System:

This refers to the actual game to be developed. It encapsulates all the game logic (i.e. anything regarding playing the game), and all structural and software logic (i.e. how the system creates and interface between the players and how it develops and supports the network allowing this interaction).

Server:

This refers to a specific portion of the Game or System attributed to allowing two players simultaneous action to a game interface. We differentiate because, though the System encapsulates the System, it may arise that a distinction must be made between the two.

Game:

Another portion that is encapsulated by the System but needs to be distinguished, the Game refers to the portion of the system that deals with Game Logic. When a Player plays the Game, we refer to the player interacting with the Game Logic portion of the system with the implementation and Server logic generally abstracted out.

Environment:

Anything not directly included within the overreaches of the System (i.e. those not included within the system boundary). This includes any external entities like Players who have some goal that they wish the system to satisfy. Typically the environment is presented an interface (dependent on the individual actor) that outlines the possible actions they are able to take on the system. The system will encapsulate and abstract the rest.

SBF:

Spaceships: Battle for the Frontier

1.4 Overview

Section 2: Contains the Use Case Models and Diagrams. This is a general elicitation of the major ways the System interacts with the environment and *vice-versa*. Section also includes all possible actions a user can take while playing *SBF***.**

Section 3: Contains the Structural Requirements. This is a thorough understanding of the System Architecture and the main designs thereof.

Section 4: Contains the Behavioral Requirements. This section delineates the conceptual behavior of the software system.

2. Use Case Model and Elicitation

2.1 Introduction to Section

The following section outlines the major interactions between *the system* and *the environment* while playing *SBF*.

This section will include all major actions demanded by the Requirements Milestone, however it will not represent each action the *Player* can take within gameplay, in an effort to preserve brevity. As such, only major goals are presented including by not limited to: Players logging in, Players reviewing their information, Players choosing an opponent, Players playing the game, etc.

2.2 List and Descriptions of Major Actors

Figure 2.1 – Actor Description Table

Actor	Description	Multiplicity
Player (Primary)	The primary players of the game. The actors who have goals of logging in, playing, and logging out of the game.	1 Player interacts with the system. They play as if they are playing against the system, where in reality the "system" is another player doing the same thing.
Screen (Secondary)	The screen where the program is displayed. The screen is the main hardware interface between the player and the game.	1 there is one active per player.
Mouse / Keyboard (Secondary)	These are the operational tools of the player. They are what the player does to navigate the screen (i.e. the game interface).	1 Set of mouse and keyboard per player.

2.3 List of all Interactions to be specified by specific Use Cases

Figure 2.2 – Use Case Table of Contents

Name Id	Description	Details	Figure Number
Play SBF	The overall summary interaction of one player as he navigates the system to login, play and exit the game.	A broad understanding of the major decisions a player takes when interacting with **SBF.**	**[S1]**
Login	The specific user goal of logging into the system.	Requires User to have Login Credentials. If not, depends on Sign-Up	**[U2]**
Sign Up	The player wishes to create a new account.	Accessed through the Login Screen, given player does not have valid credentials.	**[U3]**
Review Statistics	The User wishes to review game statistics for Player.	Player can review stats for themselves, or other players.	**[Sub2]**
Get Matched	The User selects another player from a list to play against in the game.	Two players must agree to play against each other. This requires notification of both players.	**[U5]**
Game Conditions	The two users review the game decisions. Deciding on an appropriate map.	Two users are provided a map, and both must agree on it. The players also agree on the amount of gold they can spend.	**[U6]**
Play Game	The summary interaction of the two players playing the actual game.	Differentiated by the Login phase and the exit phases. Contains the brunt of the interactions.	**[S2]**
Select Ships	The two players select the ships they are taking into battle.	The players select their ships using the gold amount they agreed on.	**[U7]**
Place Ships	The Users place their ships on the battlefield, according to game rules.	Placement of ships is asynchronous. Players merely alert the system when they are done.	**[U8]**
Play Turn	The player plays their turn.	The player has a host of actions they can take while playing the game. **See figure 2.4**	**[U9]**

Figure 2.3 – In Game Actions Players can take.

Action	Description	Must be Current Player?
Move Ship	The Player selects and moves (or turns) one of his ships. This action is no longer possible if the Ship has already moved its max distance.	
Shoot (Torpedo or Cannon)	The player shoots one of his projectiles. The System informs player of allowed choices. Current player is notified if hit, other player is notified that shot was taken.	Yes, only the Current player can shoot.
Deploy/ Retrieve Mine	The Current player uses one of his mine ships to deploy or retrieve a mine in the surrounding area. System does *not* notify other player that mine was deployed/	Yes, only the current player can deploy and retrieve mines.
Activate / Deactivate Sonar	The current player uses one of her radar ships in order to assess the map.	Yes, only the current player can activate or deactivate the sonar.
Save / Load Game State	Both the Current and the Other Player agree to save or load the game state.	**No,** both players must together decide whether they want to save or load the game at any time.
Repair Ship	The Current Player navigates their ship to the base, and selects the repair option. Allowing their ship to be repaired over time.	Yes, only the Current Player may be able to activate reparations at any time.
Send Message	The players send a text message to one another.	No, either player can send and receive messages at any time.
Review Game Rules	Either player decides to review the game rules or the possible decisions they are able to make.	No, either player can review the game rules at any given time.

2.4 Use Case Model

Play SBF Use Case [S1]

Use Case: Play Spaceships: Battle for the Frontier
Scope: SBF
Level: Summary
Intention in Context: The player wishes to receive some enjoyment by playing *Spaceships: Battle for the Frontier* hoping also to win the game.
Multiplicity: One Player interacts with their *System* at any given time.
Primary Actor: Player
Pre-Condition: Player has valid program downloaded and installed on an appropriate operating system.
Main Success Scenario:

1. *Player* logs into [U2] *System*.
2. *Player* gets matched [U5] to an opponent from list provided by *System*.
3. *Player* plays a game [S2] of **SBF**.
   Steps 2-3 can be repeated indefinitely.
4. *Player* exits from *System*.

Extensions
3a. *Player* wishes to save game state. Proceed to (4). Use Case ends in Success.

Login Use Case [U2]
Use Case: Login to System.
Scope: SBF
Level: User Goal
Intention in Context: The player wishes to retrieve gameplay details, credentials, and identity, from a previously created account.
Multiplicity: Multiple users can be logged in at one time.
Primary Actor: Player.
Pre-Condition: The user has already created an account with the *System* – specifically the server.
Post-Condition: The player will be logged into the System identified by their unique identification name provided in login.
Main Success Scenario:

1. *System* displays user login page to *User*
2. *Player* provides *System* with login details.
3. *System* validates user credentials.
4. *System* notifies *Player* of valid login details.
5. *Player* interacts with *System* as identifiable client.

Extensions
2a. Player does not have login details. Player Signup [U3]
3a. *System* determines invalid User. *System* displays login failure. Use case starts again.

Sign Up Use Case [U3]
Use Case: Create a new identifiable profile with the System.
Scope: SBF
Level: User Goal
Intention in Context: The player wishes to create a new profile with the System.
Multiplicity: The *System* can have multiple New Users. *System* allows sign up for one user at a time.
Primary Actor: Player
Pre-Condition: The Player does not already have a valid account recognized by the *System*. Player wants to be able to play SBF.
Post-Condition: The System will store and recognize details for a new player.
Main Success Scenario:

1. *System* displays New-User signup page to *Player*.
2. *Player* provides *System* with required credentials.
3. *System* validates user credentials.
4. *System* alerts *Player* that credentials are validated.
5. *Player* Logs into [U2] *System*.

Extensions:
2a. User does not provide full and appropriate credentials. Case resumes at 1.
3a. *System* determines some major error. Use case ends in failure.

Review Statistics [Sub1]
Use Case: Review specific player stats and performance.
Scope: SBF
Level: Sub-Function of Get Matched.
Intention in Context: The player wishes to review the game statistics of some other Player actor including perhaps themselves.
Multiplicity: The *System* can provide multiple user profiles. The *System* can display one stat section at a time.
Primary Actor: Player
Pre-Condition: User has successfully logged into the system.
Main Success Scenario:

Steps 1-2 can be repeated in sequence.

1. *Player* notifies *System* about a specific player statistic to review.
2. *System* presents appropriate game statistic to *Player*.
3. Player finishes reviewing statistics, returns to Get Matched [U5]

Get Matched [U5]
Use Case: Match players in order to start a game.
Scope: SBF
Level: User Goal
Intention in Context: The player wishes to find an opponent to play a game against.
Multiplicity: *System* can match multiple players. *System* displays a table such that one player can browse multiple opponents.
Primary Actor: Player
Secondary Actor: Other Player

Pre-Condition: Each individual player has appropriately logged into the System.
Main Success Scenario:

1. *Player* notifies *System* that she wants to select a new opponent.
2. *System* responds by presenting *Player* a list of possible opponents.
3. *Player* notifies *System* of *Other Player* she wishes to challenge.
4. *System* responds confirmation that *Other Player* has accepted challenge.
5. *Player* plays a game [S2].

Extensions
2a. *System* determines there are no other *players* logged in. Use case ends in Failure.
3a. Player Reviews Statistics [Sub1] about other player to determine compatibility. Use case resumes per step 2.
4a. *Other Player* does not accept invitation to play game. *System* notifies *Player*. Use case resumes per step 2.
4ai. *Player* cannot send more then 2 requests to *Other Player* in a span of 1 hour.

Play Game Use Case [S2]
Use Case: Play a game of SBF.
Scope: SBF
Level: Summary Level
Intention in Context: Two *Players* have been matched up. The *Players* want to engage in a game of SBF.
Actor: *Player*
Secondary Actor: *Other Player*
Pre-Condition: System has matched up two Players to engage in a game. Both players have agreed to face each other.
Post-Condition: Game statistics for the *Player* will be updated including rankings.
Main Success Scenario:
At this point, the Players may also decide to Load a game. This scenario means use case begins at step 5.

1. *System* executes Game Conditions [U6]
2. *System* Ship Selection [U7]
3. *System* starts Place Ships [U8]
4. *Player* alerts *System* she is ready to Start Game
   Step 5 is repeated until one player wins the game.
5. *Player* takes a turn [U10]
6. *System* alerts *Player* of end of game.

Extensions
1a. Users want to load a game. Use case resumes at step 5.
1ai. System determines some load error. Users are notified. Use case begins at 1.
5a. *User* saves game state. System exists. Use Case ends in success.
5ai. *System* determines some save error. User is alerted. Use case ends in failure.

Game Conditions Use Case [U6]
Use Case: Review the landscape and total money.
Scope: Play Game
Level: User Goal.

Intention in Context: The *Player* is presented with a list of game terms. These terms include the Map and the amount of money the players have to buy ships for that game. The player reviews these conditions and makes any appropriate alterations.
Multiplicity: 2 Players are interacting with the System simultaneously. One player at a time gets updated game conditions.
Primary Actor: Player
Secondary Actor: Other Player.
Pre-Condition: Both players have been properly matched up, and have agreed to start a game.
Post-Condition: The System has registered the appropriate conditions to produce for the specific game instance.
Main Success Scenario
Steps 1 and 2 are repeated until a unanimous decision is made by both players.

1. *Player* is notified of game conditions by System.
2. *Player* reviews and sends acceptance to the *System*.
3. Move to *Select Ships* [U7]

Extensions
2a. *Player* does not like conditions, makes modifications alerts *System* of changes.
2ai. *System* notifies *Other Player* of changes. *Other Player* makes changes of his own. Use case begins again.
2aii. *System* notifies *Other Player* of changes. *Other Player* accepts changes. Use case ends in success.

Select Ships Use Case [U7]
Use Case: Selection of ships for battle.
Scope: Play Game
Level: User Goal.
Intention in Context: The *Player* selects ships from a roster of available brigs according to a declining balance decided upon in the *Game Conditions*. These are the ships the player will be bringing into battle.
Multiplicity: In the context of a game instance: two players simultaneously interact with the *System* to choose ships. Each *Player* interacts with his personal *System*.
Primary Actor: Player
Main Success Scenario:
Steps (1-3) repeated until player runs out of credit or appropriate ships.

1. *System* notifies *player* of valid and presents appropriate ships.
2. *Player* selects ships based on credit balance and alerts *System* of choice.
3. *System* validates ships selection. *System* notifies player of validity.
4. Move to *Deploy Ships* [U8].

Extensions:
3a. *System* determines invalid choices. *System* notifies *Player*. Start again at 2.

Deploy Ships [U8]
Use Case: Deployment of ships on the battle screen.
Scope: Play Game.
Level: User Goal
Intention in Context: The *Player* wishes to deploy his ships on the battle map.

Multiplicity: Both *Players* are deploying ships simultaneously through their individual *Systems*.
Primary Actor: Player
Main Success Scenario:

1. *System* provides *Player* with the list of selected ships from Select Ships [U7].
   Number (2-3) is repeated until all ships are placed on the map surrounding player base.
2. *Player* selects a remaining ship from the list, and informs the *System* where on the map he would like it to be placed.
3. *System* confirms valid ship placement.
4. *Player* declares to *System* that he is ready to Start Game.
5. *System* start game. Players start *Playing Turns*.

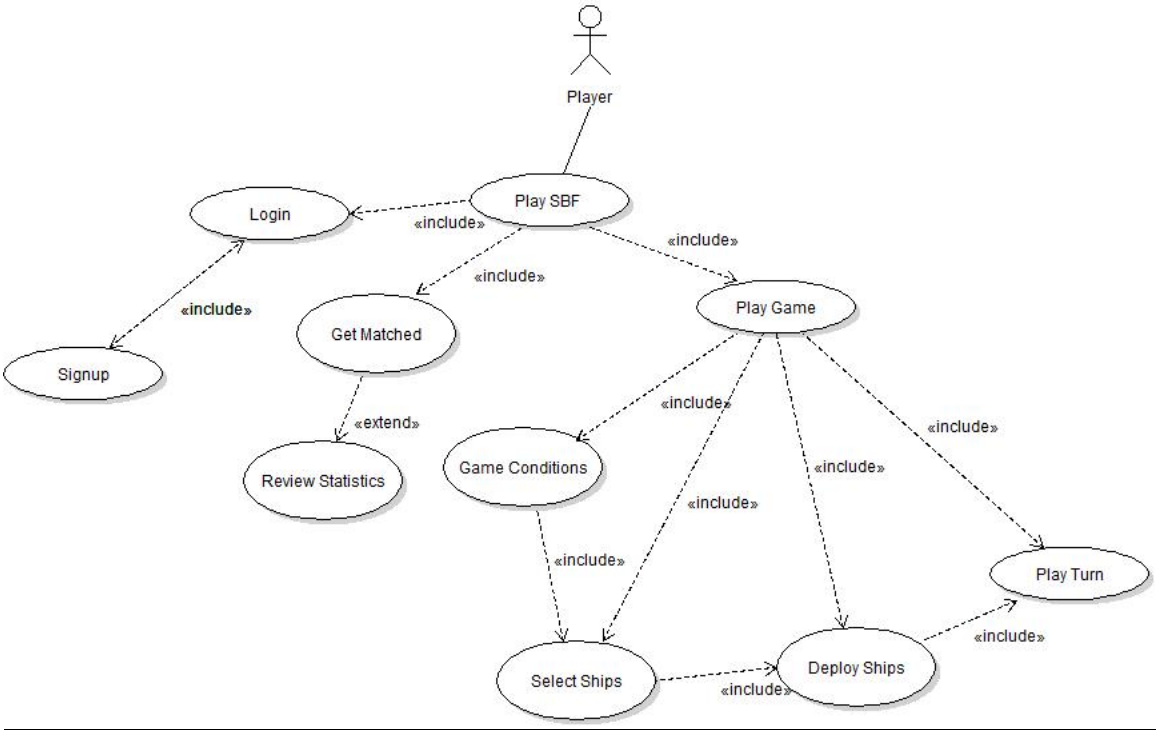
Extensions:
2a. *Player* alerts *System* he want so change the position of the ship. *System* reflects changes. Use case begins again at 2.
3a. *System* Determines invalid ship placement. *System* alerts *Player*. Use Case begins again at (2).

Play Turn [U10]
Use Case: Play a specific turn in the turn based game.
Scope: Play game
Level: User Goal
Intention in Context: The *Player* plays his turn in **SBF**. She wishes to take appropriate action in order to maximize chances of winning the game.
Multiplicity: One Player is interacting with *his System* at any given time in the game instance to take a turn.
Main Success Scenario:
Steps (1-4) are repeated until one player has won game.

1. *System* notifies *Player* that it is his turn.
   Actions for (2) can be selected from Fig. 2.4
   Number (2) is repeated until Player has made all wanted decisions / actions
2. *Player* notifies *System* of the action he wishes to take.
3. *System* returns action confirmation message to *Player*.
4. *Player* notifies *System* that he is ready to end his turn.

Extensions:
2a. *System* determines that *Player* is not able to take any additional actions. Proceed to 4.
3a. *System* has determined that *Player* has won game. Use Cast Ends in Success. Game Ends.
3b. *System* determines that *action* was not successful or invalid. *System* alerts *Player*. Return to step 2.

Figure 2.4 – Use Case Diagram – See UseCase.jpeg for full size



3. Structural Requirements

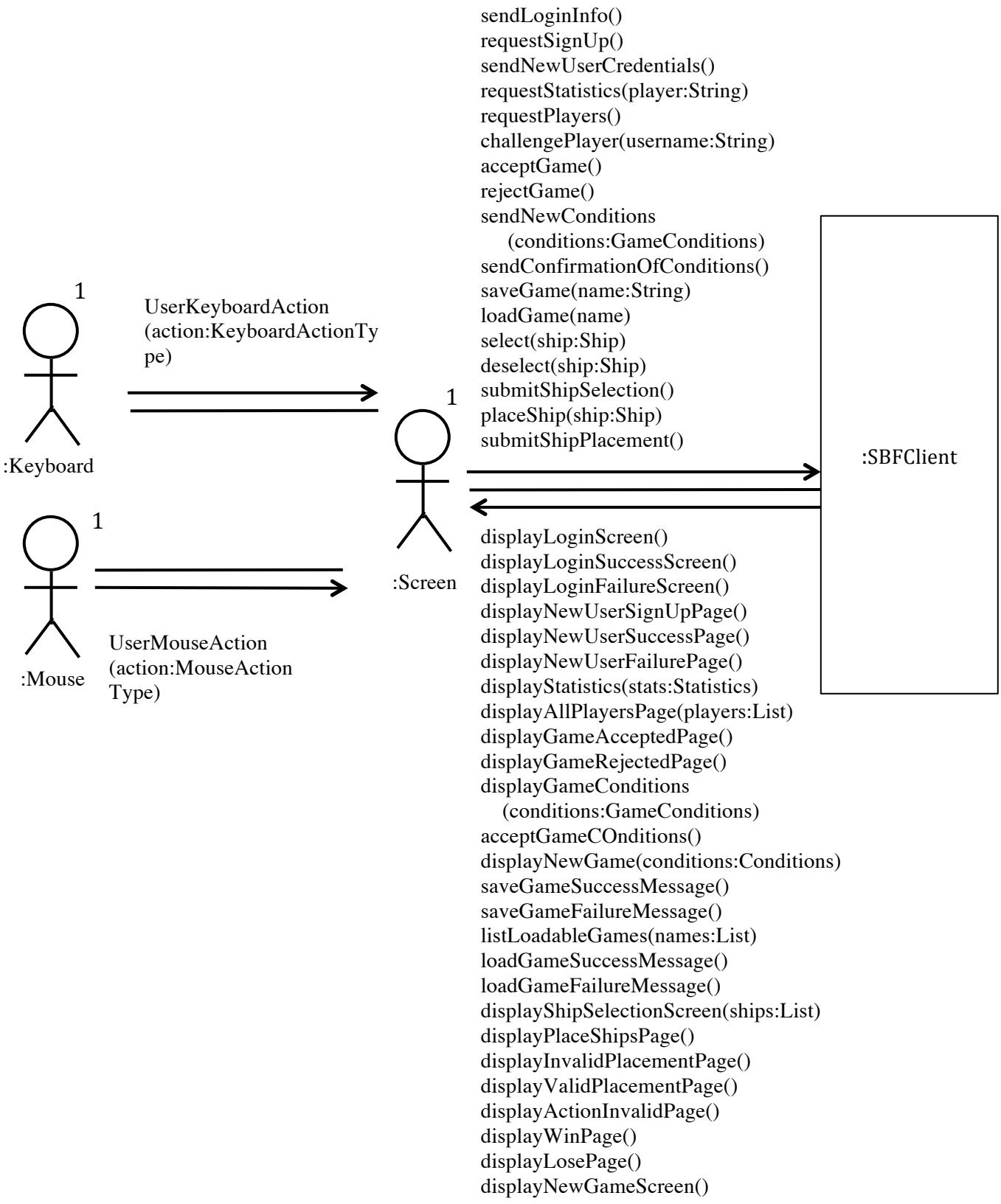
3.1 Introduction

In this section we will describe in detail what the system does. That is, we will define the internal structure of the information held within the system and the relationships between them, as well as the boundaries of the system and what the system is capable of doing, including major interactions between the client and the system.

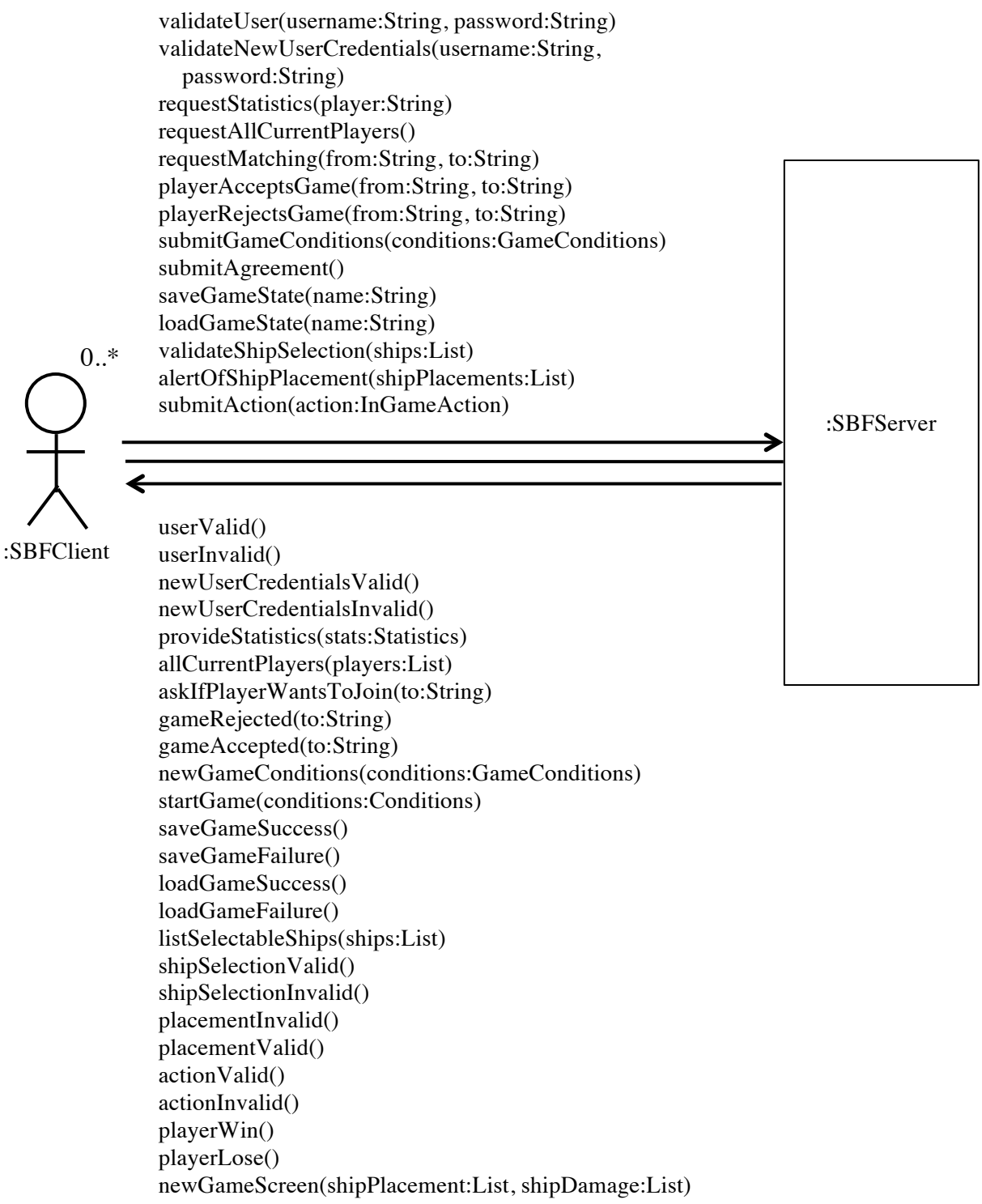
3.2 Environment Model

The environment model defines the system interface. It is a high-level black-box overview of the system that breaks down the system functionality that describes the interactions and communications made between the actors and the system through the exchange of messages. It will be expressed below through 2 models, one for the Client and one for the System.

3.2.1 Client Environment Model



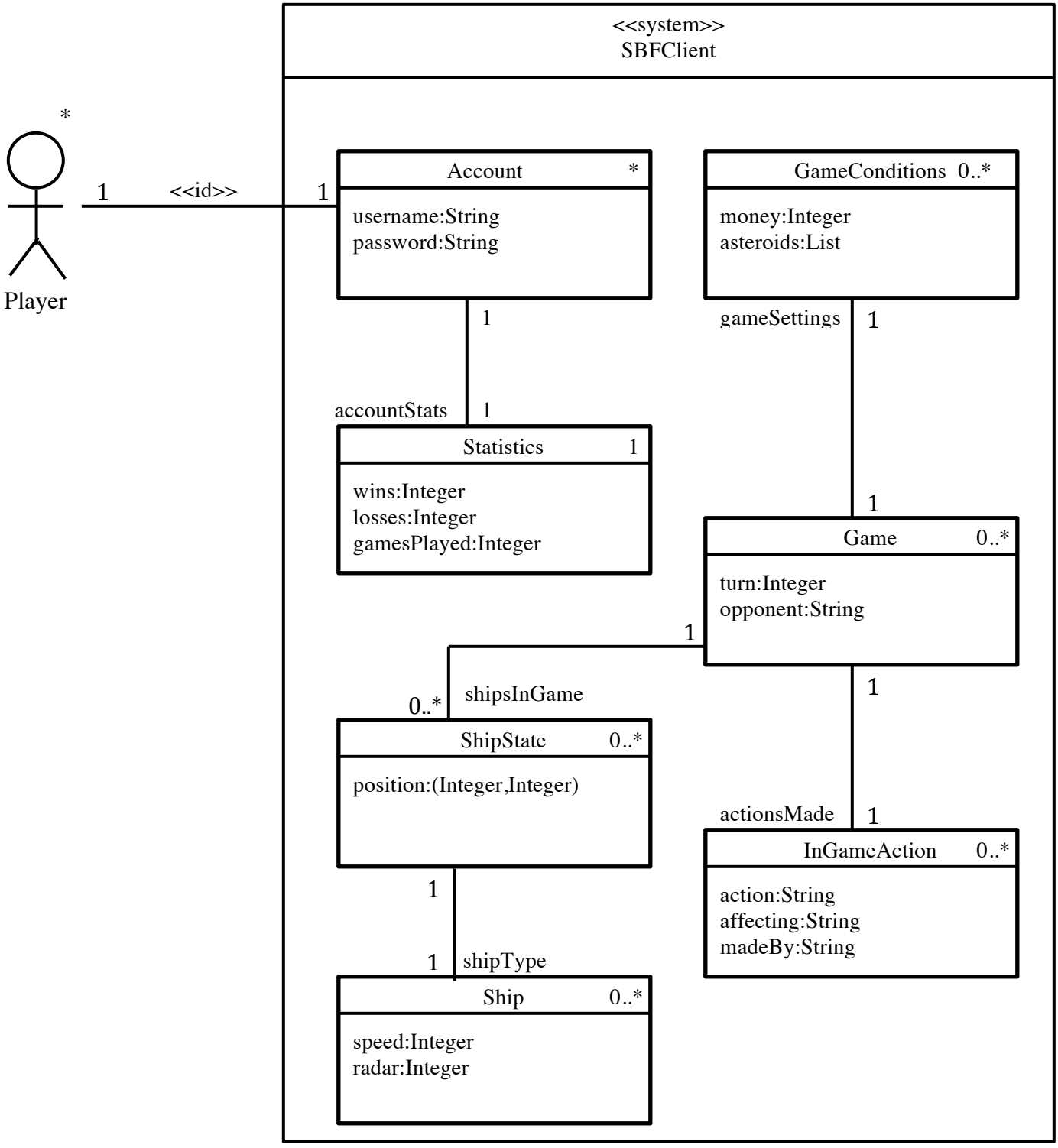
3.2.2 Server Environment Model



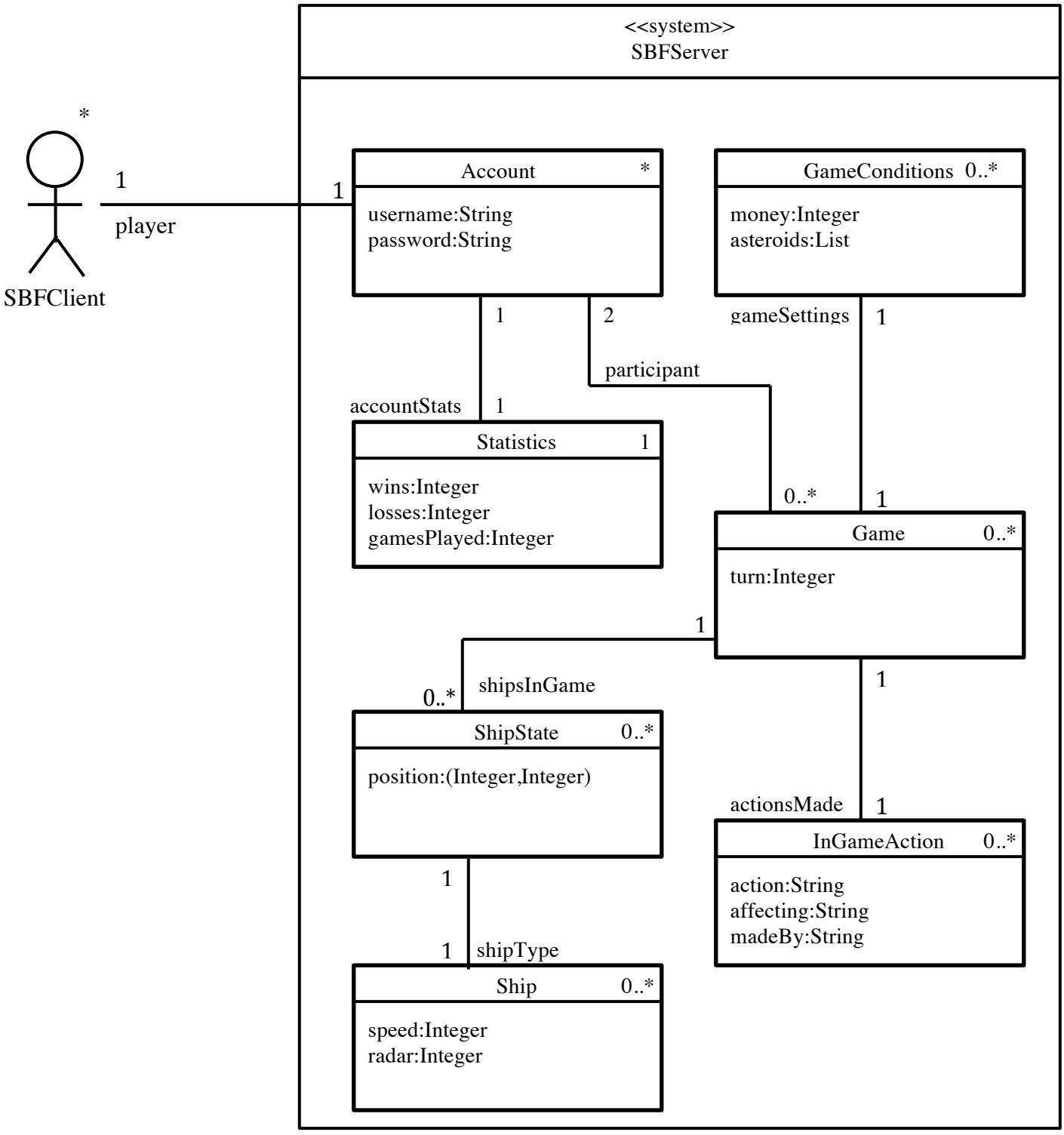
3.3 Concept Model

The concept model is the description of the classes (and their associations) that model the system's conceptual state in order to provide the required functionality. It further separates the objects and classes between what is within the system and what is externally available in the environment. It as well will be expressed through 2 models, for the Client and Server.

3.3.1 Client Concept Model



3.3.2 Server Concept Model



4. Behavioral Requirements

4.1 Introduction

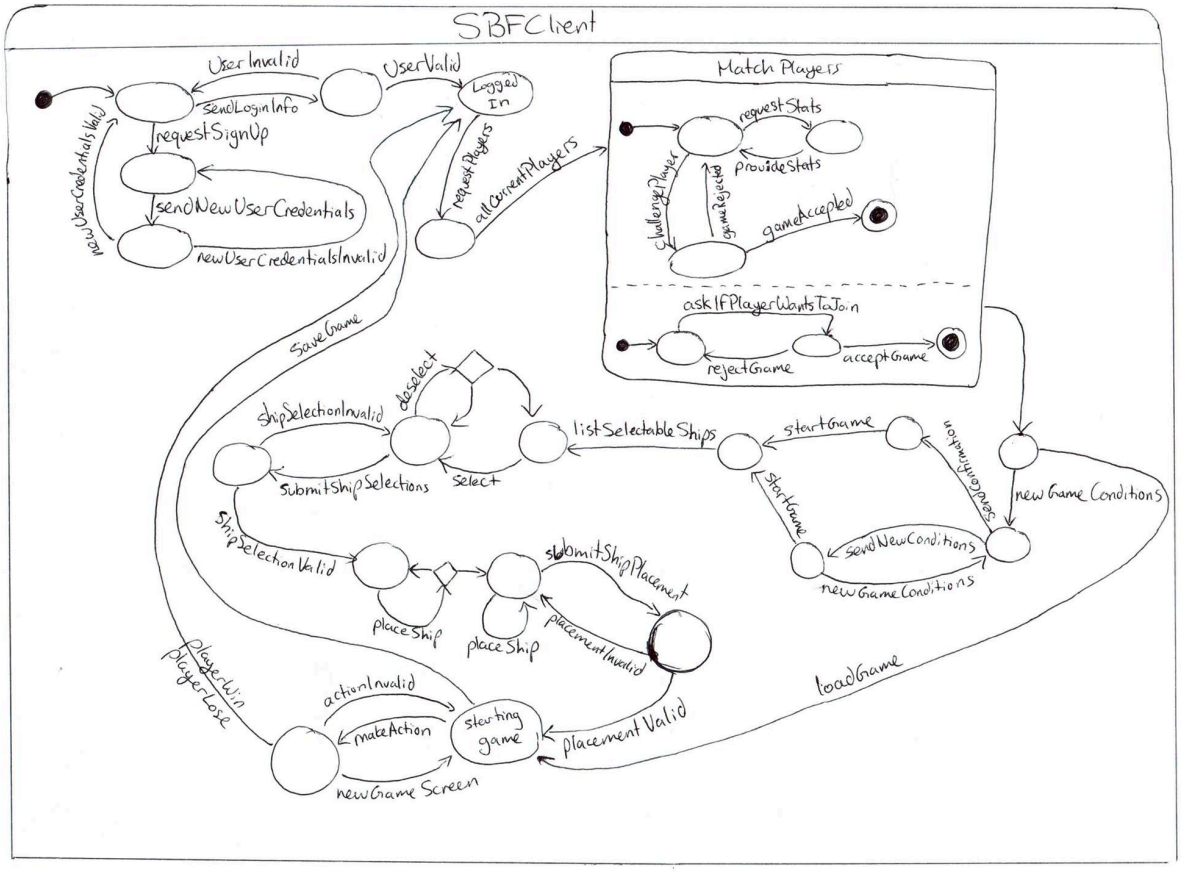
The purpose of the behavioral requirements is to express the appropriate sequences of system operations, as well as their desired effects on the state of the system. This is elaborated and expressed through the Protocol Model and the Operations Model below.

4.2 Protocol Model

We define here the steps that the Game System will attempt to take in order to execute the various messages connecting the environment and the System. There are two figures presented: The Client Figure and the Server Figure. Both play an important role in defining the allowable sequences of interactions that the system may have with its environment.

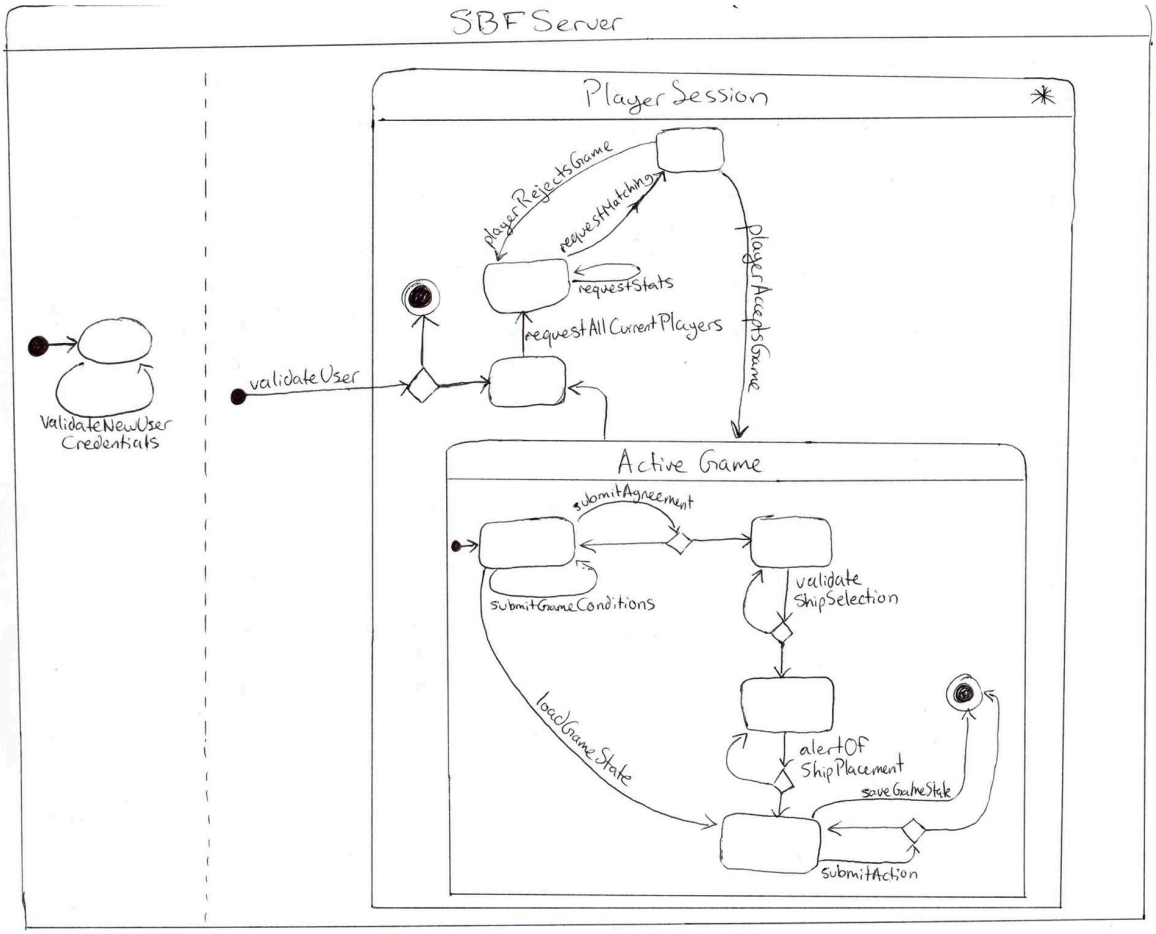
4.2.1 Client Protocol Model

Figure 4.1 – Protocol Client Diagram. See ProtocolClient.jpeg for full size.



4.2.2 Client Server Model

Figure 4.2 – Protocol Server Diagram. See ProtocolServer.jpeg for full size.



4.3 Operational Model

We describe here the desired effects of the execution of each system operation on the system state. Every operation performed in the system has some effects on the system, and in this model, we define this very specifically.

4.3.1 Operation Schemas

Operation: User :: logIn (username: String, password: String)
Scope: Account;
Messages: SBFClient :: { sendLoginInfo(); userValid(); userInvalid_e() }; SBFServer :: { validateUser(username:String, password:String) }; Screen :: { displayLoginScreen(); displayLoginSuccessScreen(); displayLoginFailureScreen() }
Pre: The user is currently not logged in.
Description: The operation shows the log in form on the screen, and if the entered username and password are validated by the server, the user is logged in while a success message is displayed on the screen. Otherwise an error message is displayed on the screen.

Operation: User :: signUp (username: String, password: String)
Scope: Account;
Messages: SBFClient :: { newUserCredentialsValid(); newUserCredentialsInvalid_e() }; SBFServer :: { validateNewUserCredentials(username:string, password:string) }; Screen :: { displayNewUserSignUpPage(); displayNewUserSuccessPage(); displayNewUserFailurePage() }
Pre: The user is currently not logged in.
New: newAccount: Account
Description: The operation shows the sign up form on the screen, and if the new credentials entered by the client are validated by the server, a new account is created and a success message is displayed on the screen. Otherwise, an error message is displayed on the screen.

Operation: User :: reviewStatistics (player: String)
Scope: Account; Statistics;
Messages: SBFClient :: { requestStatistics(player:String); provideStatistics(stats:Statistics) }; SBFServer :: { requestStatistics(player:String) }; Screen :: { displayStatistics(stats:Statistics) }
Pre: The user is currently logged in.
New: gameStats: Statistics
Description: The client requests the statistics from the server and after it is provided, it is displayed on the screen.

Operation: User :: matchup (opponent: String)
Scope: Game; Account;
Messages: SBFClient :: { requestPlayers(); challengePlayer(username:String); acceptGame(from:String); rejectGame(from:String); allCurrentPlayers(players:List); askIfPlayerWantsToJoin(to: String); gameRejected(to:String); gameAccepted(to:String) }; SBFServer :: { requestAllCurrentPlayers(); requestMatching(from:String, to: String); playerAcceptsGame(from:String, to:String); playerRejectsGame(from:String, to:String) }; Screen :: { displayAllPlayersPage(players:List); displayGameAcceptedPage();displayGameRejectedPage() }

Pre: The user is currently logged in and does not have any set up games.
Description: The screen shows all the players after requesting the information. The client can request a matching or can accept/reject a request. Result of the matching is displayed on the screen.

Operation: User :: setupGame ()
Scope: Game; GameConditions;
Messages: SBFClient :: { sendNewConditions(conditions:GameConditions); sendConfirmationOfConditions(); newGameConditions(conditions:GameConditions); startGame(conditions:Conditions) }; SBFServer :: { submitGameConditions(conditions:GameConditions); submitAgreement() }; Screen :: { displayGameConditions(conditions:GameConditions); acceptGameConditions(); displayNewGame(conditions:Conditions) }
Pre: The user is currently logged in and does not have any set up games.
New: newConditions: GameCondition; newGame: Game
Description: New game conditions are generated and displayed, if all clients agree, a new game is started. Otherwise, new conditions are generated and a new game creation awaits the confirmation for the newly created conditions of all clients.

Operation: User :: loadGame (gameName: String)
Scope: Game; GameConditions; Ship; ShipState;
Messages: SBFClient :: { loadGame(name: String); loadGameSuccess(); loadGameFailure_e() }; SBFServer :: { loadGameState(name: String); }; Screen :: { saveGameSuccessMessage();saveGameFailureMessage(); listLoadableGames(names:List); loadGameSuccessMessage(); loadGameFailureMessage() }
Pre: The user is currently logged in and does not have any set up games. Also the opponent saved in the game has to be logged in.
Description: All the loadable/saved games requested from the server and are displayed on the screen. After the client chooses which game to load, it is loaded.

Operation: User :: saveGame (gameName: String)
Scope: Game; GameConditions; Ship; ShipState;
Messages: SBFClient :: { saveGame(name:String); saveGameSuccess(); saveGameFailure_e() }; SBFServer :: { saveGameState(name:String); }; Screen :: { saveGameSuccessMessage();saveGameFailureMessage(); listLoadableGames(names:List); loadGameSuccessMessage(); loadGameFailureMessage() }
Pre: The user is currently logged in and is enrolled in the game.
Description: The client chooses to save the game displayed on the screen, if successful, a success message is displayed on the screen. Otherwise a failure message is displayed on the screen.

Operation: User :: selectShips (ships: List)
Scope: Game; GameConditions; Ship; ShipState;
Messages: SBFClient :: { listSelectableShips(ships:List); shipSelectionValid(); shipSelectionInvalid_e(); select(ship:Ship); deselect(ship:Ship); submitShipSelections() }; SBFServer :: { validateShipSelection(ships:List) }; Screen :: {

displayShipSelectionScreen(ships:List) }
Pre: The user is currently logged in, matched up with another user and the game is already set up.
Description: Selectable ships are displayed on the screen, then the client makes and submits a selection. After that, the server lets the client know if the ship selection is valid or invalid.

Operation: User :: placeShips (ships: List)
Scope: Game; GameConditions; Ship; ShipState;
Messages: SBFClient :: { placeShip(ship:Ship); removeShip(ship:Ship); submitShipPlacement();placementValid(); placementInvalid_e(); } ; SBFServer :: { alertOfShipPlacement(shipPlacements:List) } ; Screen :: { displayPlaceShipsPage(); displayInvalidPlacementPage(); displayValidPlacementPage() }
Pre: The user is currently logged in, matched up with another user and the game is already set up. The user has also selected some ships.
New: newShip: Ship
Description: After the ship placements page is displayed on the screen, the client places/removes ships and submits the final placement to the server. If the placements are confirmed to be valid by the server, a valid placement page is displayed on the screen; otherwise, an invalid placement page is displayed on the screen.

Operation: User :: makeAction (action: InGameAction)
Scope: Game; InGameAction;
Messages: SBFClient :: { makeAction(action: InGameAction); actionValid(); actionInvalid_e(); playerWin(); playerLose() }; SBFServer :: { submitAction(action:InGameAction) }; Screen :: { displayActionInvalidPage() }
Pre: The user is currently logged in, matched up with another user and the game is already set up.
New: userAction: InGameAction
Description: The user chooses to make an in game action (as displayed in Figure 2.3 in the Use Cases Section). If the action is valid, it is submitted; otherwise, an action invalid page is displayed on the screen. Also, if an action defines end of game, the client is let know of win/lose status.

Operation: User :: updateTurn
Scope: Game; GameConditions;
Messages: Screen :: { displayNewGameScreen() }; SBFClient :: { newGameScreen(shipPlacement:List, shipDamage:List) }
Pre: The user is currently logged in, matched up with another user and the game is already set up.
Description: The server sends the client the placement of ships and the damaged ships. This information is then sent to the screen to display a new game screen.