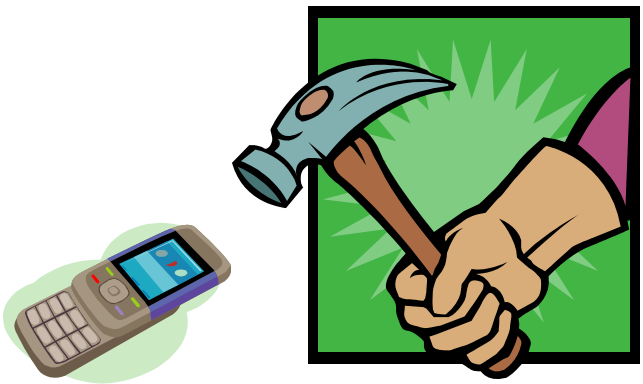
Documentation of the

IKILLU

system

Autumn 2007



by group 4:

Rayner(raynerv), Maja (majasm), Lavdim(lavdima), Vegard(vegaaa), Vincent(vincentg), Tonje(tonjeek)

Table of contents

1 User manual

1.1 Introduction – purpose of the game

I KILL U, the name says it all.

Your purpose is to kill your game opponents using your cell phone and SMS messages.

When the game starts you will receive an initial sum of points that you can use to pay for different services in the quest to kill one or more of your opponents. You also will receive an initial sum of basic shield, which is what keeps you alive.

Your mission is to wipe out anyone in the game, to do this you have to strike them. And if you are the one who finishes them off (bringing their basic shield value below 1) you will take their points and your shield will receive a boost based on your received points.

But beware (!) there are some catches….

1.2 Different roles in the game

User you have registered as interested in playing the game and have to wait for an invitation for a game.

Player you have been invited to a specific game and have accepted the invitation by registering as a player to the game

Admin you have registered as the admin of the current game as you have "created" the game. There can only be one admin at a time.

1.3 Services

1.3.1 Services for user

Register for the player pool: *reg <user name>*
You can only be registered once with the same mobile phone at the time, and you have to choose a unique user name. When you are registered you can be invited to a game or become an administrator by creating a game. You will receive a SMS message with the result.

Join game: *join*
You can only join if you received an invitation to the game on SMS. In the invitation you will find the game name. You will receive confirmation on SMS. If the game has started or ended, it is too late to join the game. You will receive a SMS message with the result.

Unregister for the player pool: *unreg*
You can't unregister from the game system if you still are a registered player. If you try, you will receive an error message. You will have to use *leave* first, and then you can unregister. You will receive a SMS message with the result.

Create new game and be game administrator: *create <game name>*
To create a new game and be a game administrator you have to first be a registered user. There can only one game administrator at any time. Creating the game means taking the role as the game administrator and giving the game a name. After you have created the game, you can start inviting people to the game by announcing it. You will receive a SMS message with the result.

1.3.2 Services for player

Light up area: *lightup <range>*
When you light up an area you will be able to see who of the players are nearby and how far away they are, so that you can choose a player to strike tactically. You give the *range* in meters. In addition to being able to see the players in the area, you will receive information on how far away they are and how many points they have left. You will receive a SMS message with the result. *Also read the **'what happens if…'** section*

Boost your shield for protection: *shield <strength> <duration>*
To give yourself some extra protection you can use some of your points to buy some for a given duration. This extra protection will wear off gradually to zero during this duration. You will receive a SMS message with the result.

Strike a given opponent: *strike <user name> <force>*
Now the time has come for some action. You have decided on an opponent to strike. If you spotted him by lighting up the area you also know how many points he has and can choose the force tactically. If you strike the opponent's shield so that it reaches 0, you have succeeded to kill him. His remaining points will be added to your points and you will receive a boost in your shield. You will receive a SMS message with the result. *Also read the **'what happens if…'** section*

Leave game: *leave*
You can leave a game at any time, but you will then surrender from the game and have no possibility to join in again before you are invited again. You are still a registered user. You will receive a SMS message with the result.

Evaluate your rank in the game: *eval*
Knowing how you rank is an interesting thing to know during the game. Here you get a SMS in return with the number of players alive and how you rank between them. You will receive a SMS message with the result.

1.3.3 Services for admin

Announce game: *announce <game name>*
After creating the game, you need to get the word out to the players that there is a new game ready for registration. When you announce the game, you send a SMS message to all registered users of the game system. You will receive a SMS message with the result.

Start game: *start <game name>*
After the game is announced and if there are two players or more registered for the game you may start the game. After this time, no more players may join the game and the registered users will receive a message that the game is on! You will receive a SMS message with the result.

End game: *end <game name>*
As administrator you may end the game at any time. If the game has been announced, a message is sent to the registered players and they are kicked out of the game. You will receive a SMS message with the result. . If the game is running, it will be evaluated and the all players are informed about their score.

Make a map over the all the player's positions: *kml <game name>*
You can here generate a map that will show you the player's latest known position and their points so far in the game. You will receive a SMS message with the result.

1.4 What happens if…

.. I am being struck
If you are struck the force of the strike will affect you shield. If you have an extra shield, it will affect the force you have on your extra shield first, before it affects the basic shield. The strike has a hit/miss functionality, so that even if you are struck, the strike may miss you and your shield isn't that much affected. Then again, it may hit bullseye and strike you with double force.

..the area I am in is being lit up
If someone lights up an area and you are in it, they have full control of the distance you are from them at light up time and how many points you have left. But so do you, you get their information for free. This is the time to think, strike or shield,or run for it….

…I light up the area
The price you pay for a light up is half the range. This will be deducted from your points.
The players you light up are warned…so beware…they can strike first. They are also informed of your position and your remaining points. You can be the victim of many strikes at once.

…I strike someone
The cost of the strike is based on the distance and the force you decide to lay on them. Beware that your strike might not be a direct hit, there is an added feature where you can hit straight on the target and your hit will do twice the damage to the opponent's shield as you intended, but it can also miss the target and make no or little damage to the opponent's shield.

…I strike a user I know is playing, but don't know where is…
Striking without lighting up first may cost you dearly as the cost is calculated on how far the player you wish to attack are from you.

…I strike but don't kill
You may do damage to your opponent's shield and make him an easier target for yourself or others later on. No change in your points other than the deduction of the cost of the strike.

…I strike and kill
You will receive all the points from the opponent you killed and you will get a shield boost based on those points too.

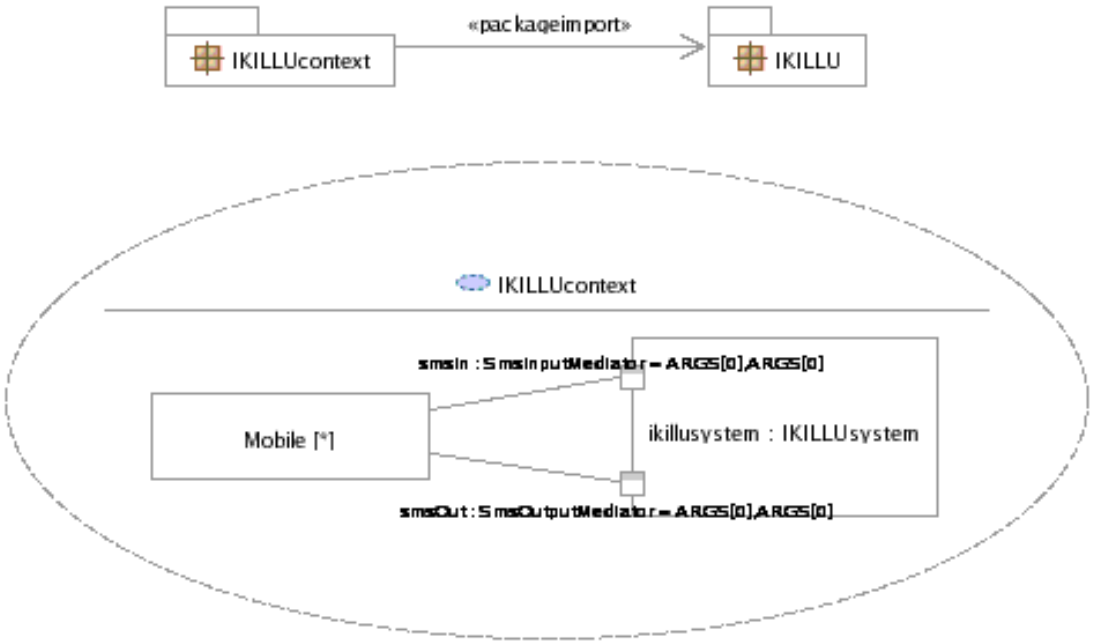
1.5 Specialties

While striking, there is a random effect simulating a hit or miss. As described in the user manual you can hit straight on the target and your hit will do twice the damage to the opponent's shield as you intended, but it can also miss the target and make no or little damage to the opponent's shield.

2 The IKILLU system

2.1 Environment

The environment illustrates the iKILLUcontext. To make a short description about what this system does in general, we have included a box labeled Mobile, and another one labeled ikillusystem. Here we want to illustrate that our system is handling sms's from different users through their mobiles. When our system receives a sms, it will then be sent to and handled within the ikillusystem.



Figur 1 - IKILLU Enviroment

2.2 Functions

In order to respect the fairness and the playability of the game, we had to define some functions. We assumed that the game is meant to be played in the Oslo area (useful for the cost of a light up) and we've been aware of the cost of a strike with respect to the amount of points you get at the beginning of the game.

2.2.1 Cost of a light up

- **Input**: the range of the wanted area (meters)
- **Output:** the cost of this operation (points)

The function is: cost = range/2
It is performed in the user session before sending the *LightUpCheck* internal signal (since the cost is a parameter of that signal).

2.2.2 Cost of an extra shield

- **Input:** duration of the wanted shield (seconds) and strength of the wanted shield (shield points)
- **Output:** cost of this operation

The function is: cost = (duration * strength)/10
This is a method of the player object named *costShield*. It's called inside the game session when receiving a *ShieldUp* signal.

2.2.3 Cost of a strike:

- **Input:** distance from the player to strike (meters) and the strength of the strike (points)
- **Output:** cost of this operation

The function is: cost = distance/2 + strength
There is a guard that prevent distance from being less than one (in this case, we put it equals to 1).
This is a method of the player object named *costStrike*. It's called in the user session just before sending the *Strike* internal signal (the cost is a parameter of the signal).

2.2.4 Effect of a strike:

The effect a strike has on a struck player cannot be deduced directly from the distance/strength of the strike. Indeed we introduced a function *getForce* (in the player object) that gives a random element to the whole process.

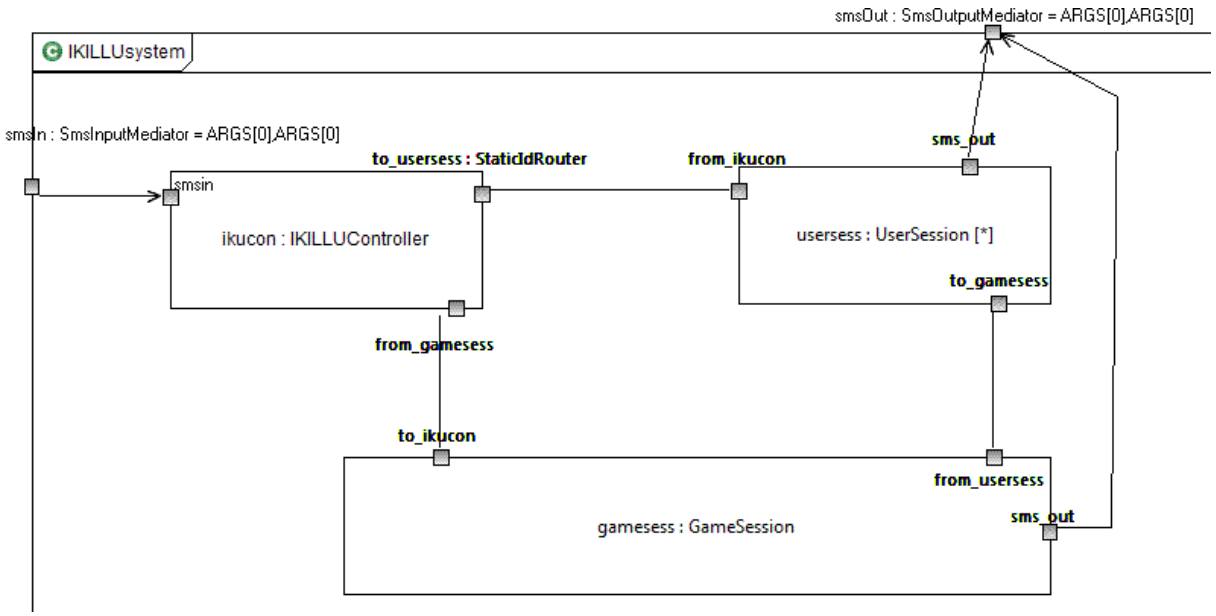
- **Input:** strength of the strike (points)
- **Output:** force of the strike (points)

The function is: force = (random number between 0 and 2) * strength
The force is then returned and the strike is proceed by calling the *beingAttacked* method.

2.3 Composite structure

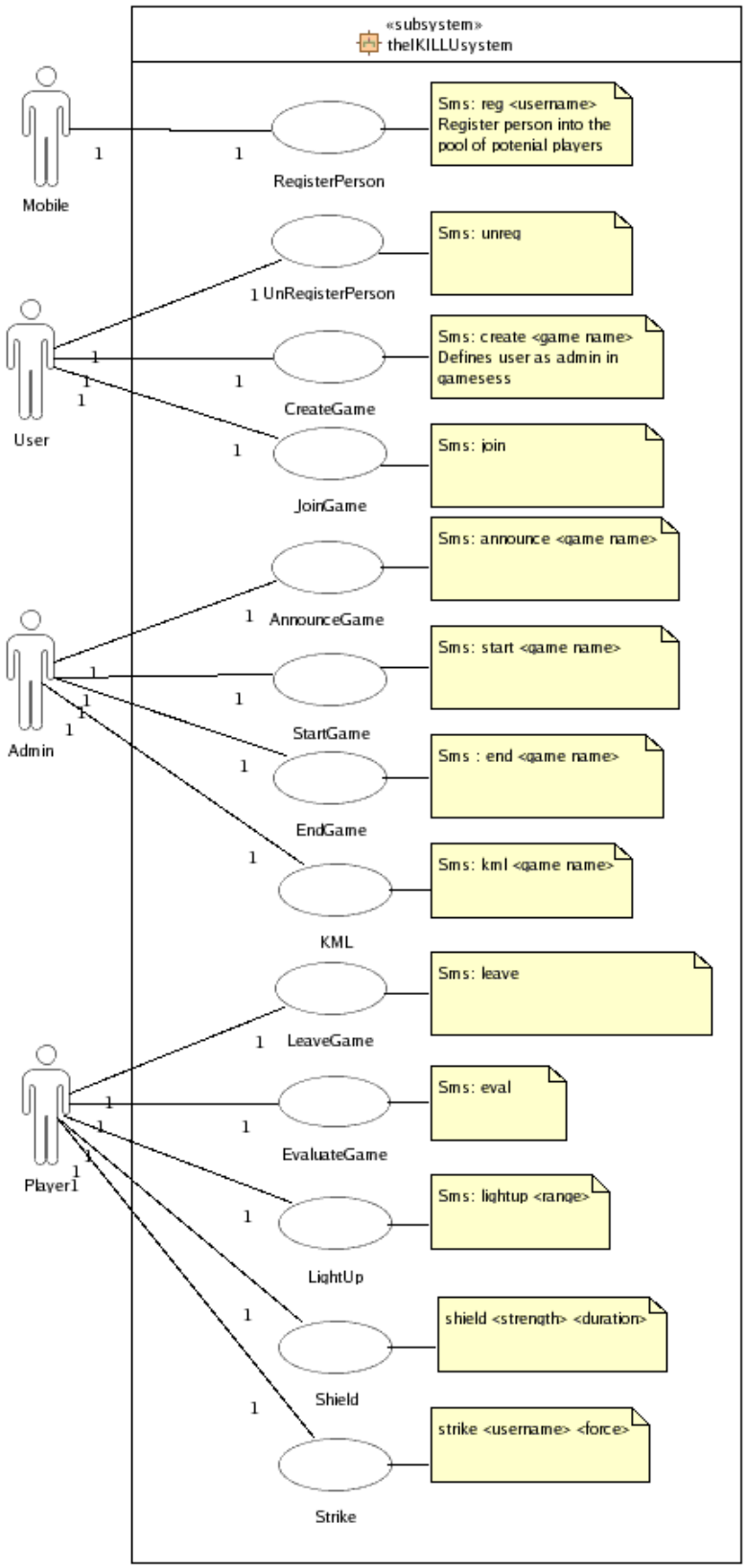
The composite structure illustrates our decomposed version of the iKILLU context. We agreed on three parts after the decomposition, these are: the ikucon, the usersess and the gamesess.

The ikucon is a statemachine that handles all the incoming interactions from the users. When the ikucon has received a sms, it will then forward it further to the correct usersess. Here the will sms be properly processed and then parsed and signals will be sent over to the gamesess for response. In gamesess the user command will be evaluated and executed as intended. Both usersess and the gamesess is able to return messages to the users.



Figur 2 - Composite Structure

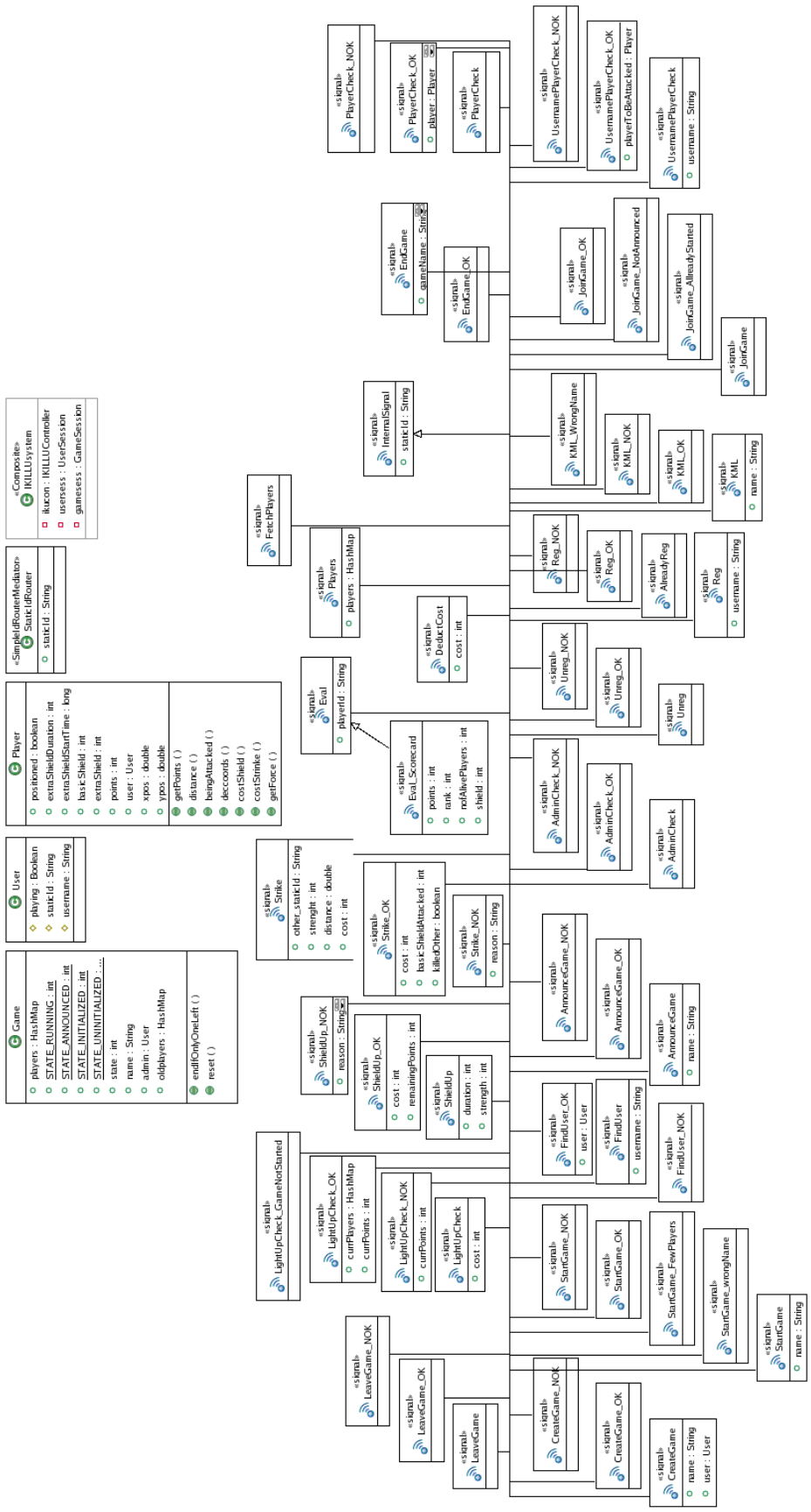
2.4 Use Cases



Figur 3 - Use case diagram

Within the use case diagram we illustrate the possible users that can interact with our system. We have divided the actors into four groups. These are: User, Admin, Player and Mobile. As it is shown in the diagram, we present different things the actors can do. For instance, a User can create a game, join a game and unregister from the system. If a user creates a game, this would makes him an admin of that game. When he's defined as an admin, he will receive new commands. The players of a game can send in different commands, these are: leave and evaluategame and lightup, shield and strike.

2.5 Class Diagram (incl signals)



Here we are displaying all of our signals that are defined in the iKILLU implementation, as well as the different data classes that is used. The different classes that we use were: User, Player, Game and ikillusystem.

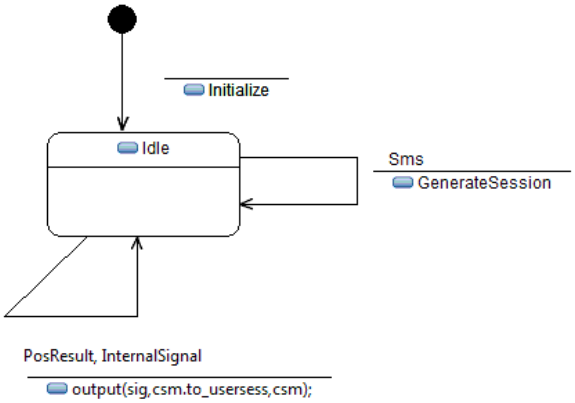
Figur 4 - Class diagram with signals

2.6 High level state machines

By looking at the high level state machines one can see what the components described in the composite structure are able to do.

2.6.1 Controller

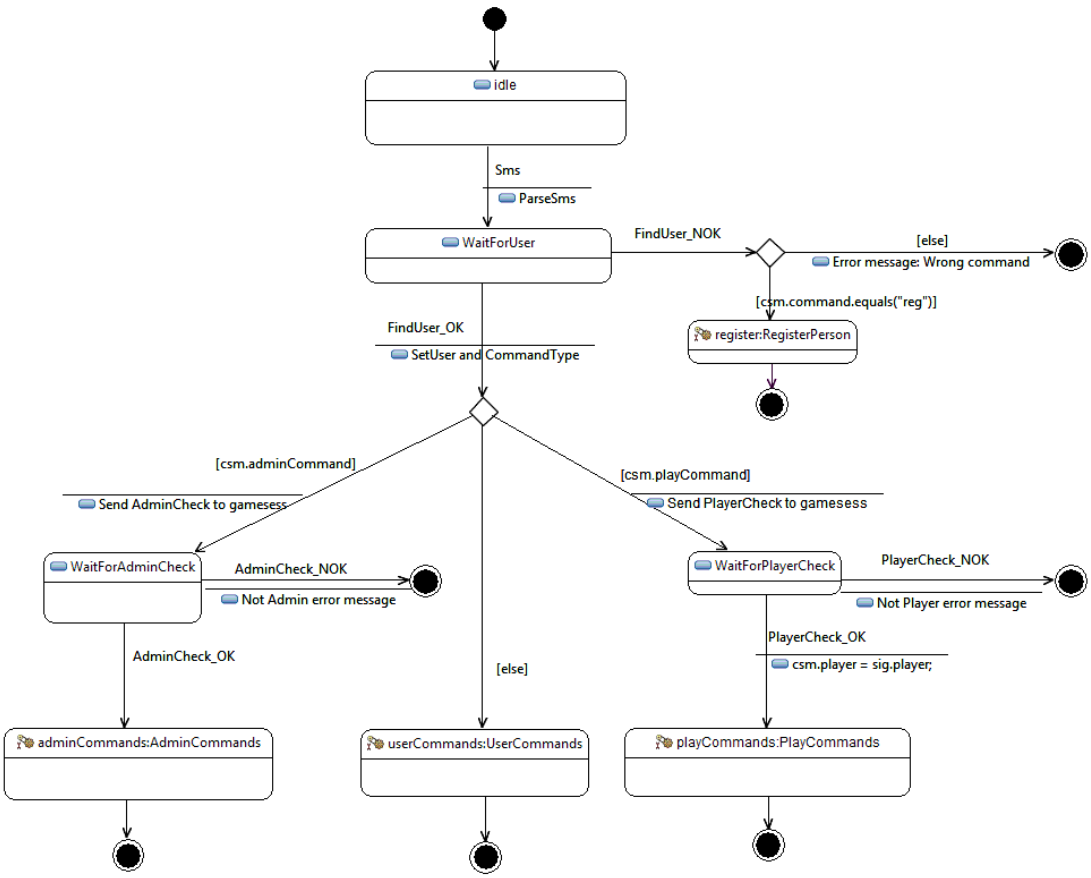
The Controller is the first instance in the system and receives all incoming messages. For each sms it creates a new UserSession and stores it locally with the staticId of the message as a key and forwards the sms signal to the newly created Usersession where it is handled with.The session can only be created if there exists no active UserSession for this user. The controller decides this based on the stored staticId's. If the controller detects an active session, it will try the creating again later by routing the signal in the back of the queue, which means the message is delayed.
The Controller also forwards any other defined signals to the desired UserSession, based on the staticId of the signal. These signals can either come from the GameSession or they are Positioning Results.



2.6.2 UserSession

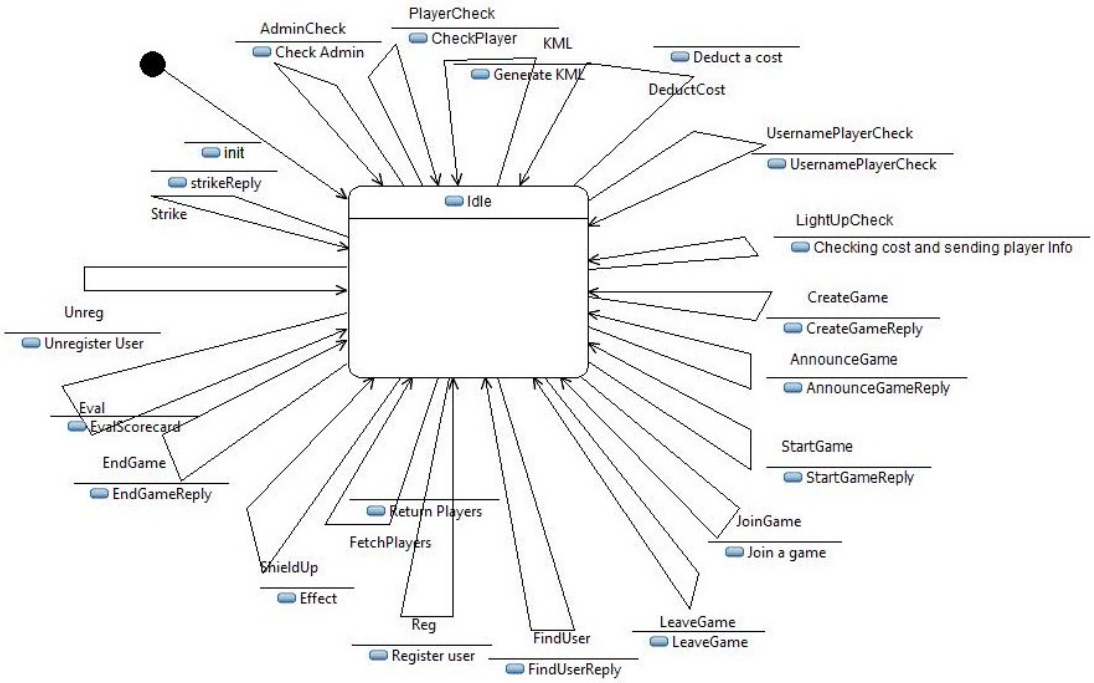
A new UserSession is created for each sms and destroyed after execution of the command is finished. Therefore it has no knowledge of the game information.
After parsing the message, it will execute the command. For every command it checks first if the user is allowed to execute the command due to his role and the state of the game. We have distinguished four kinds of commands, based on the role of the initiator.
The first one is a **register command**. This can only be executed, by a person who is not known to the system.
The second kind are the **user commands**. These are commands, that registered persons can perform, for example create a new game .
A special kind of user commands are **play commands**. For such a command the user has to be part of an active game.
The remaining commands are part of the **admin commands**, which can only be executed by the administrator of the game.

If all needed checks for a command pass in the UserSession, there still might be other checks. Based on the command a given Signal is sent to the GameSession, to get information or perform the data modification to the game information. The UserSession waits for a response signal from the GameSession, tobe able to inform the user about the result of the command, as well as blocking the user from performing another command, before the last one is finished.
After the command is executed (or not executed), the session finalizes.



2.6.3 GameSession

The GameSession takes care of all the game information and hence all data modifications are performed here. Every command that passed the checks in the UserSession accords to one Effect in the GameSession. As the GameSession can handle only one signal at a time, we avoid synchronization/concurrency problems. The GameSession will always deliver the most updated



information about a player or the game.

Due to time requirements, the GameSession has one single game, but with this architecture, it would be possible, to split up the GameSession into a GameController and a GameSession. The GameController would handle the different Games that are played and every GameSession would be able to execute commands for the game it represents. The game controller would be aware of which games a user is playing and forwarding the message to the certain GameSession instance.

2.6.4 Alternative Design of the three main components:

Another possible design of the Controller, that we discussed, was that the controller also parses the incoming sms, forwards it to a GameController and from there the sms would go further to the desired game. We discussed this, due to fairness reasons (first come, first served). The distribution of incoming messages to UserSessions breaks the ordering of the messages, which might be very important, e.g. consider strike messages. We wanted achieve that the messages regarding one game, rare executed in the same order, as they are received by the system. We dissociated from this, because the first instance of the system should be as lightweight as possible. Another reason is that we are dealing with asynchronous messages , and there is no proof, that the messages will reach our system in the same order, as they have been sent. The largest reason for time lag in the game as a whole is outside of our control. (The mobile/SMS/PATS world)

We also discussed the Design of the UserSession. To get the full Power of state machines, we would have made long lasting UserSession's. That means, the UserSession wouldn't be destroyed after the execution of one message. To the advantage of keeping knowledge in terms of states and avoiding additional traffic we would have had the disadvantage of highly storage needs.

2.7 Setup Game Services (Sequence Diagrams and StateMachines)

To avoid unnecessary complicating the document only the decomposed (level2) sequence diagrams are included here. The level 1 sequence diagrams for each service is included in the appendix. Before we arrive to any of the services there will be a check for whether the user is registered or not (FindUser). And if he is, then there is a check on what kind of command the user wishes to execute to send the command to the right substate. If it is a playCommand there also is PlayerCheck and if an adminCommand there will be an AdminCheck. In most of the diagrams shown under, we assume that the FindUser and Admin/PlayerChecks are done successfully. This to not overcomplicate the diagrams.

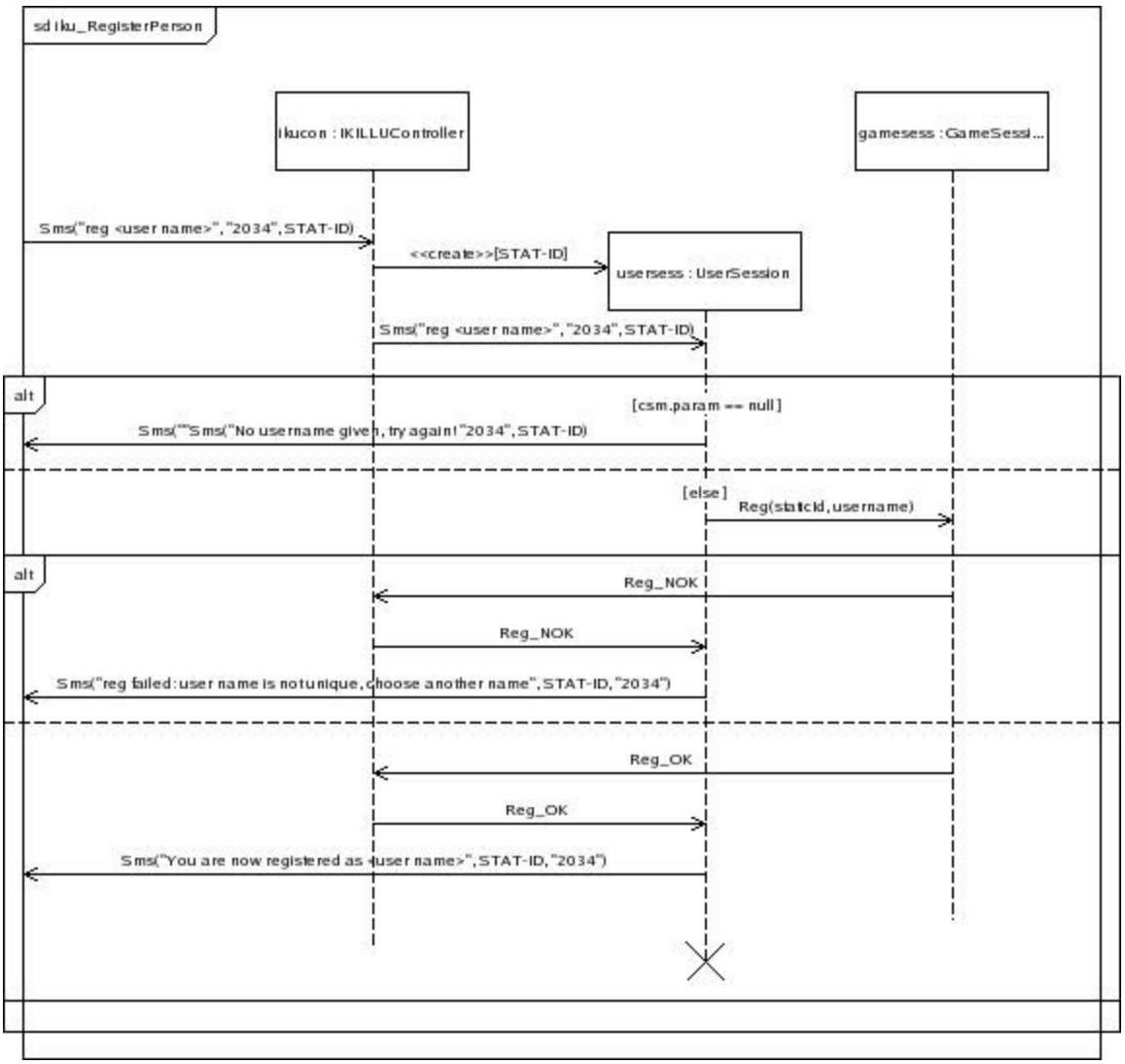
2.7.1 RegisterPerson

The System stores a pool of people who are registered as potential players. To become part of a game, a player has to be in this pool. The outcome of this is that the system knows the staticid of the users. We discussed whether a non-registered user can become part of a game, but decided that this might cause a problem in further communication because of the unknown staticId.

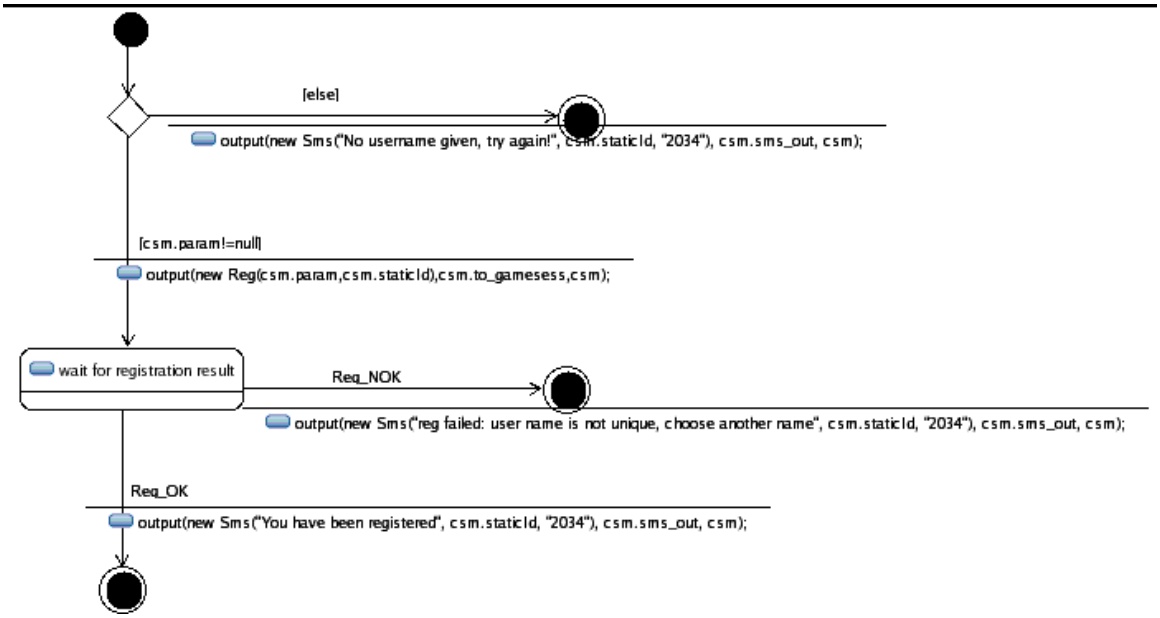
To become part of the pool, the user has to send a SMS with the command **reg** with a user name. This user name must be unique in the system. It will be helpful in the further game, so that the players can identify each other.

After receiving the message, the controller forwards this message to the userSession. In each command the userSession will first send a Signal to the gameSession to check whether the user object exists. For the reg command the userSession must receive a FindUser_NOK Signal, otherwise the command will fail, since the user is already registered in the system. After receiving the signal, the userSession sends a Reg Signal to the gameSession to register the user.

The gameSession first checks if the user name is unique. In that case, it will create a new User and save it in the user–pool, and send a Reg_OK message back. If the name is not unique, the command fails and the userSession gets a Reg_NOK message back. Finally the user session informs the user whether the registration was succesful.



Figur 5 - Decomposed sequence diagram for RegisterPerson



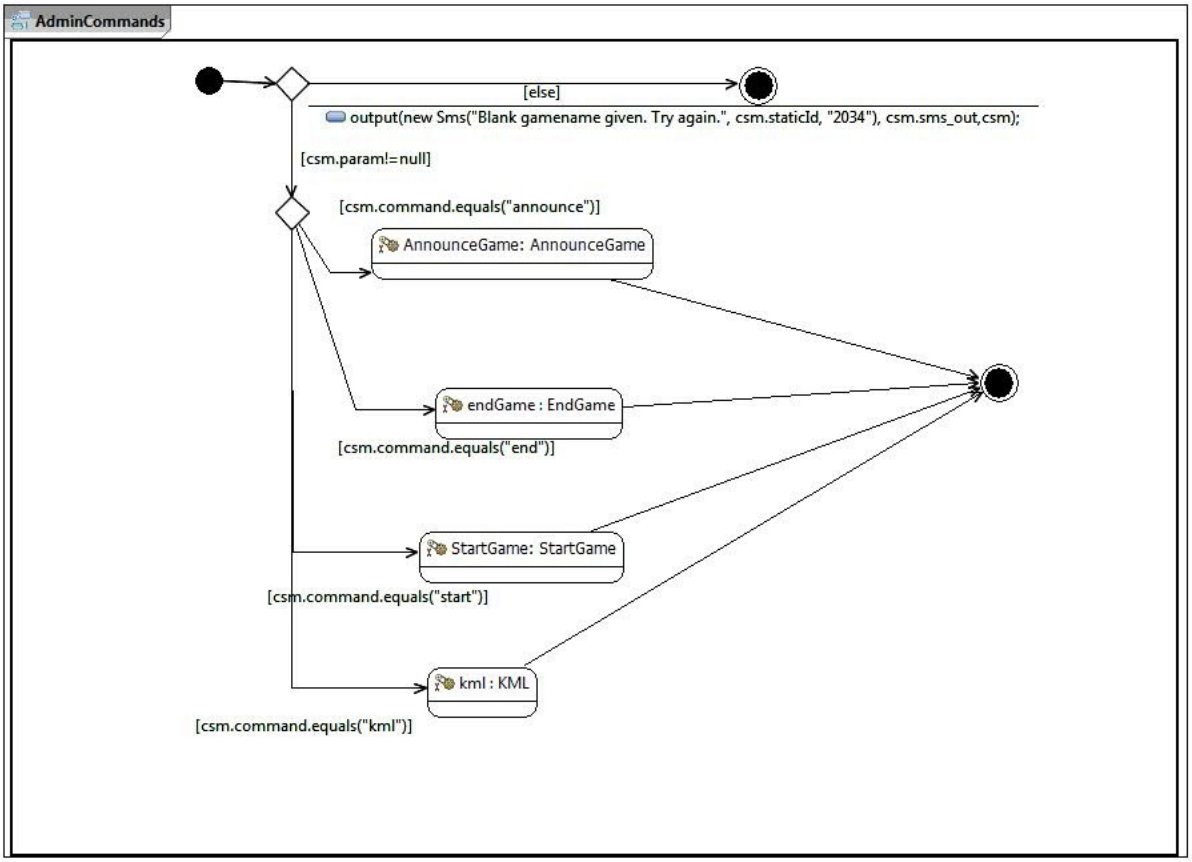
Figur 6 - State machine diagram for RegisterPerson

2.8 Admin Services (Sequence Diagrams and State Machines)

Each game needs an admin, who takes care of the game. The admin himself is not allowed to take part in the game. The game is initialized, with the create-game command, then announced to the players and finally started. The create-game command itself is not an admin command, because every player can perform the command and become admin in that way. The admin can request a KML file during the running game, to get all information about the players. At each time, a game can be ended with the end game command.

Every command has one substatemachine, that it located in the AdminCommands state machine. To enter this state machine, the command has to pass the Admin Check.

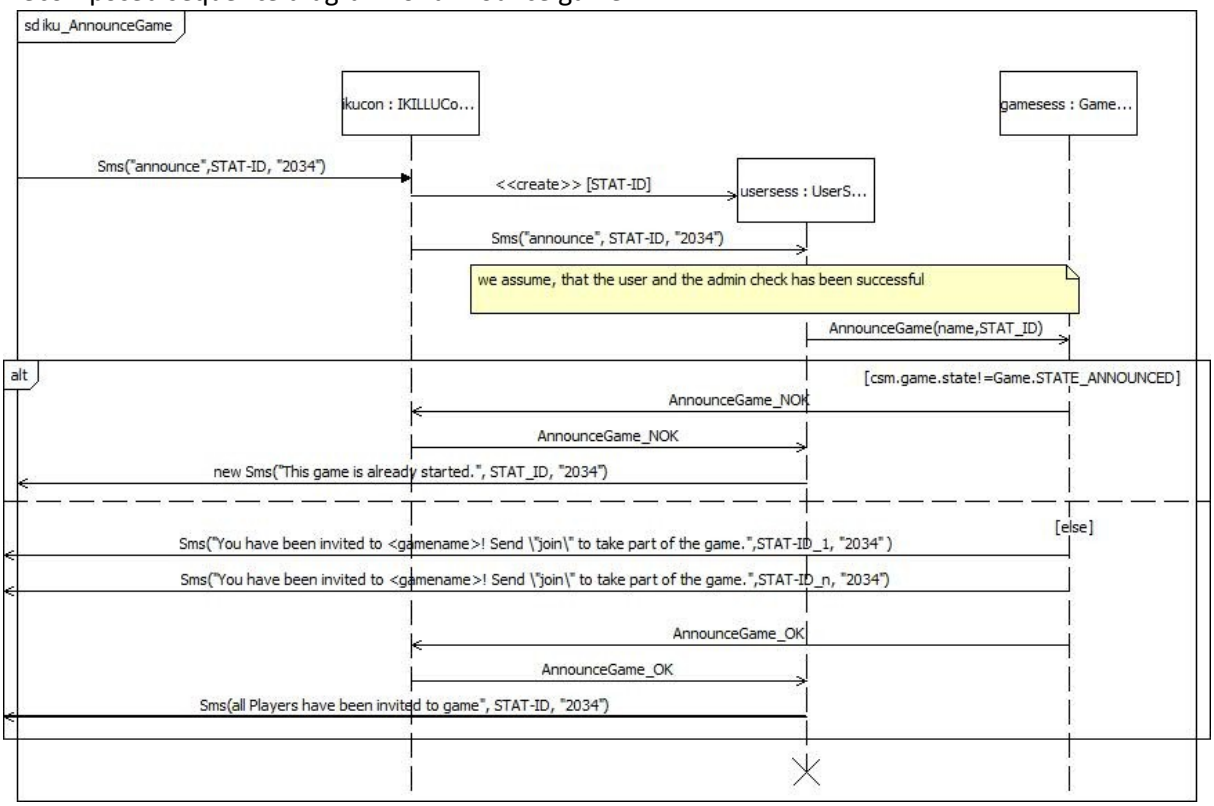
State machine diagram of admin commands:



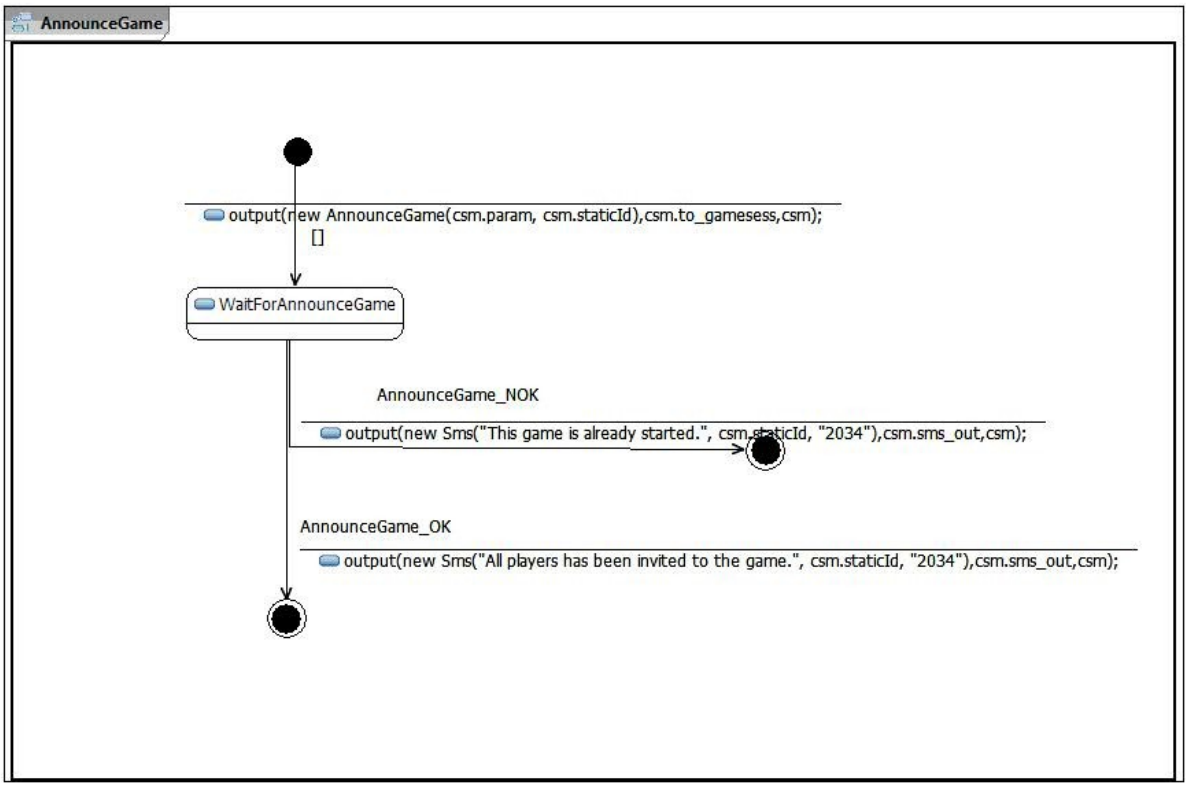
2.8.1 AnnounceGame

In this sequence diagram we try to illustrate how the administrator will announce the game to several members. After a game is announced, every player, that received an invitation can join the game through sending a join message. As it is for now, all the registered players will get an invitation for the game. The announce command checks the state of the game, that has to be “initialized”, and sends the invitations out. The administrator will also be getting a confirmation sms back. This sms indicates that all players have been invited and the state of the game has changed.
If the administrator types the wrong game name in the sms, he/she will receive an error message with the failure

Decomposed sequence diagram for announce game:



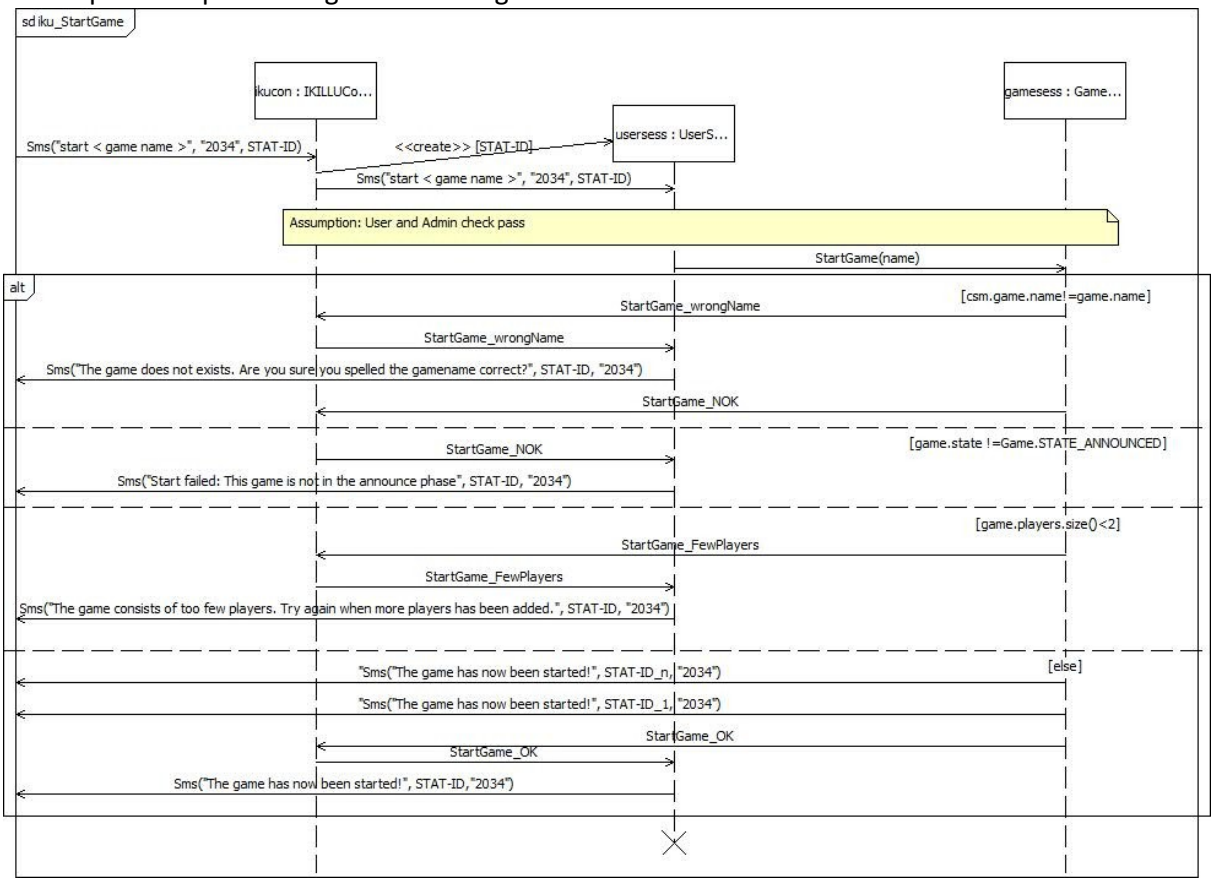
State machine diagram for announce:



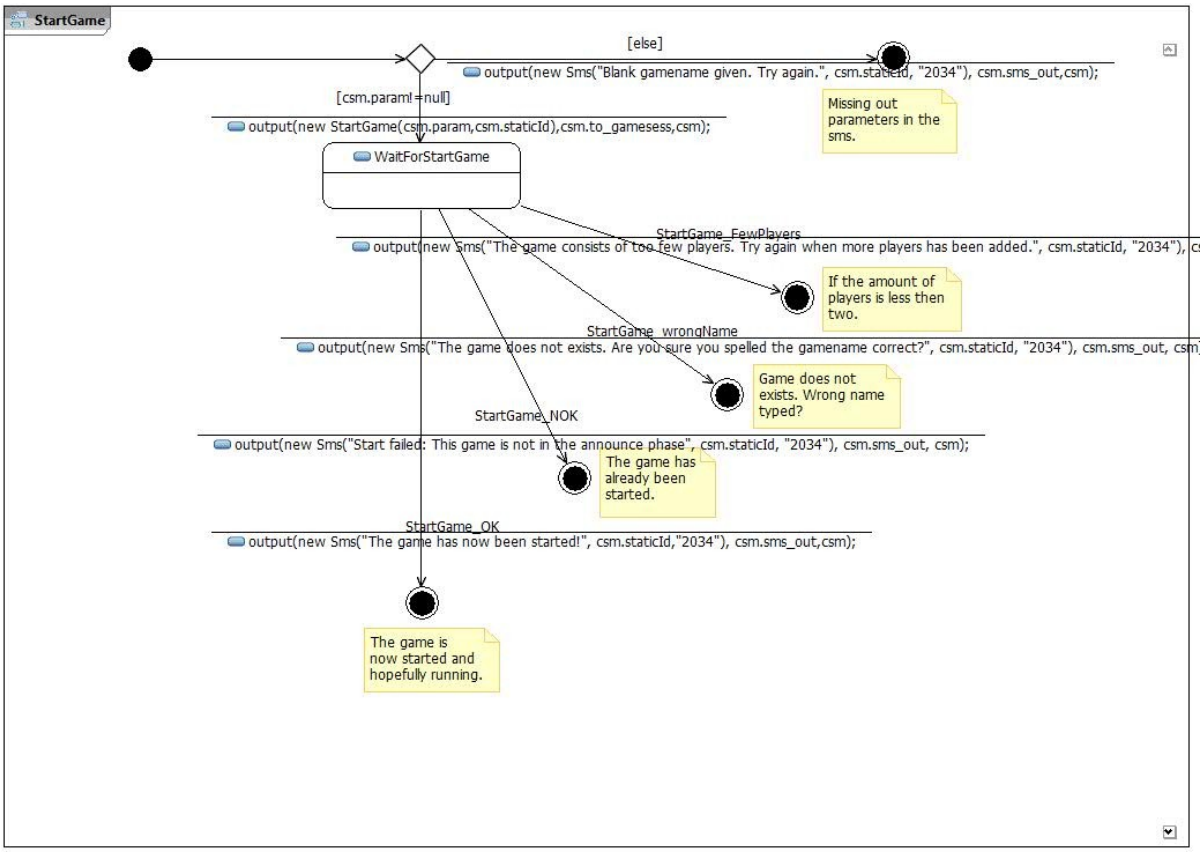
2.8.2 StartGame

When the administrator decides, that enough players have joined the game for playing it, he can start the game with the **start** command. After the game is started, no other user can join the game. The command will be parsed and coded in a StartGame signal. The signal will be preceded by the gamecontroller. The game can only be started, when it is in state “announced”, the right game name is given and the at least two players have joined the game. If all these checks pass, the GameSession will start the game, send this information to all players, and replies with a StartGame_OK singal to the UserSession. If one of the checks fails, it will reply with an error signal. Finally the UserSession informs the admin about the effect of the command.

Decomposed sequence diagram for start game:



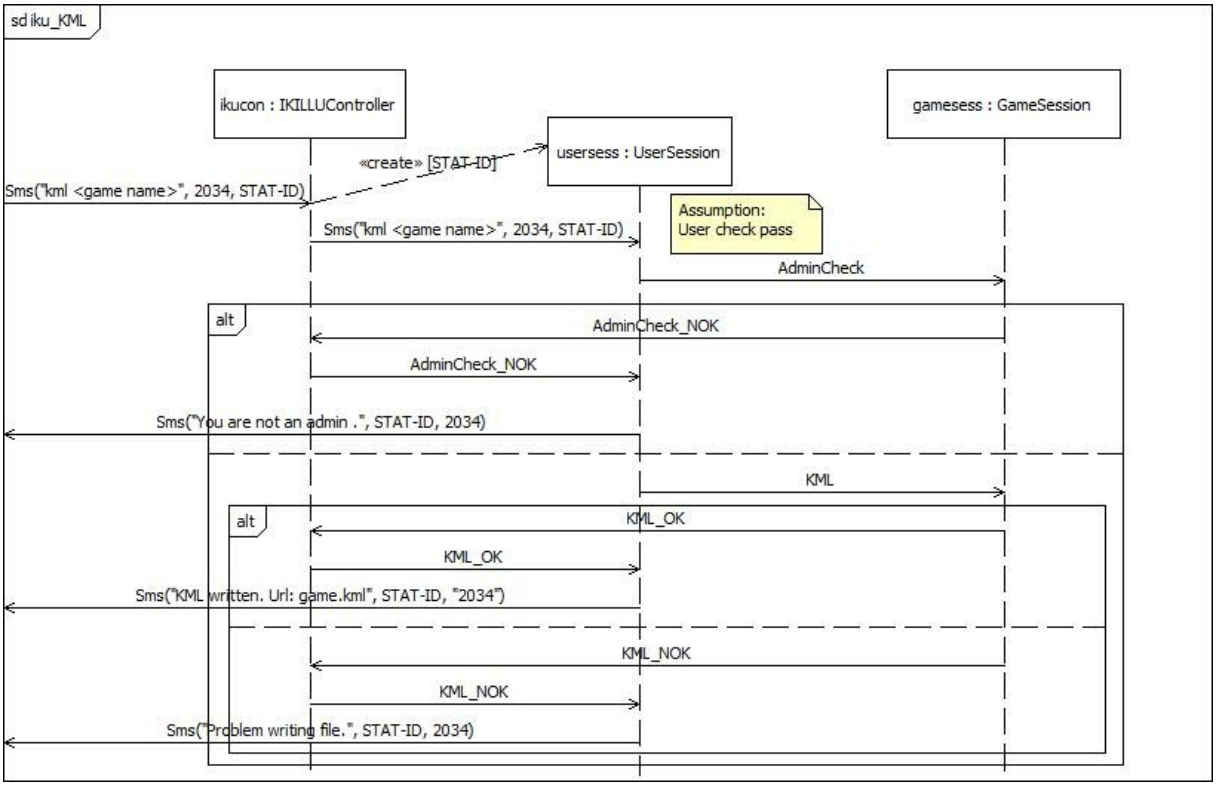
State machine for start game:



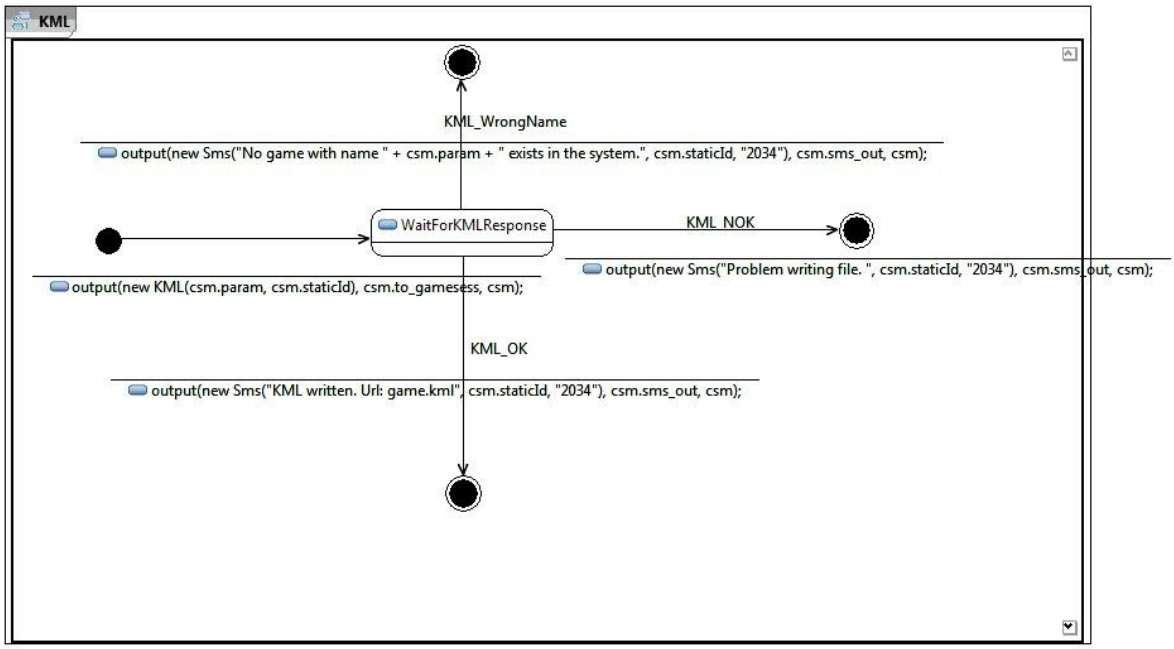
2.8.3 Generate KML

KML is an administrator function where the admin can request the system to write a KML file containing the last user position (from functions as strike, lightup) and with the current scores. If the file fails to write or the game is not in a running state, the function returns an error message. The KML file is opened by navigating to the root of the system files and opening the game.kml in Google Earth. The admin sends the KML command every time he wishes to refresh the file.

Decomposed sequence diagram for generate KML:



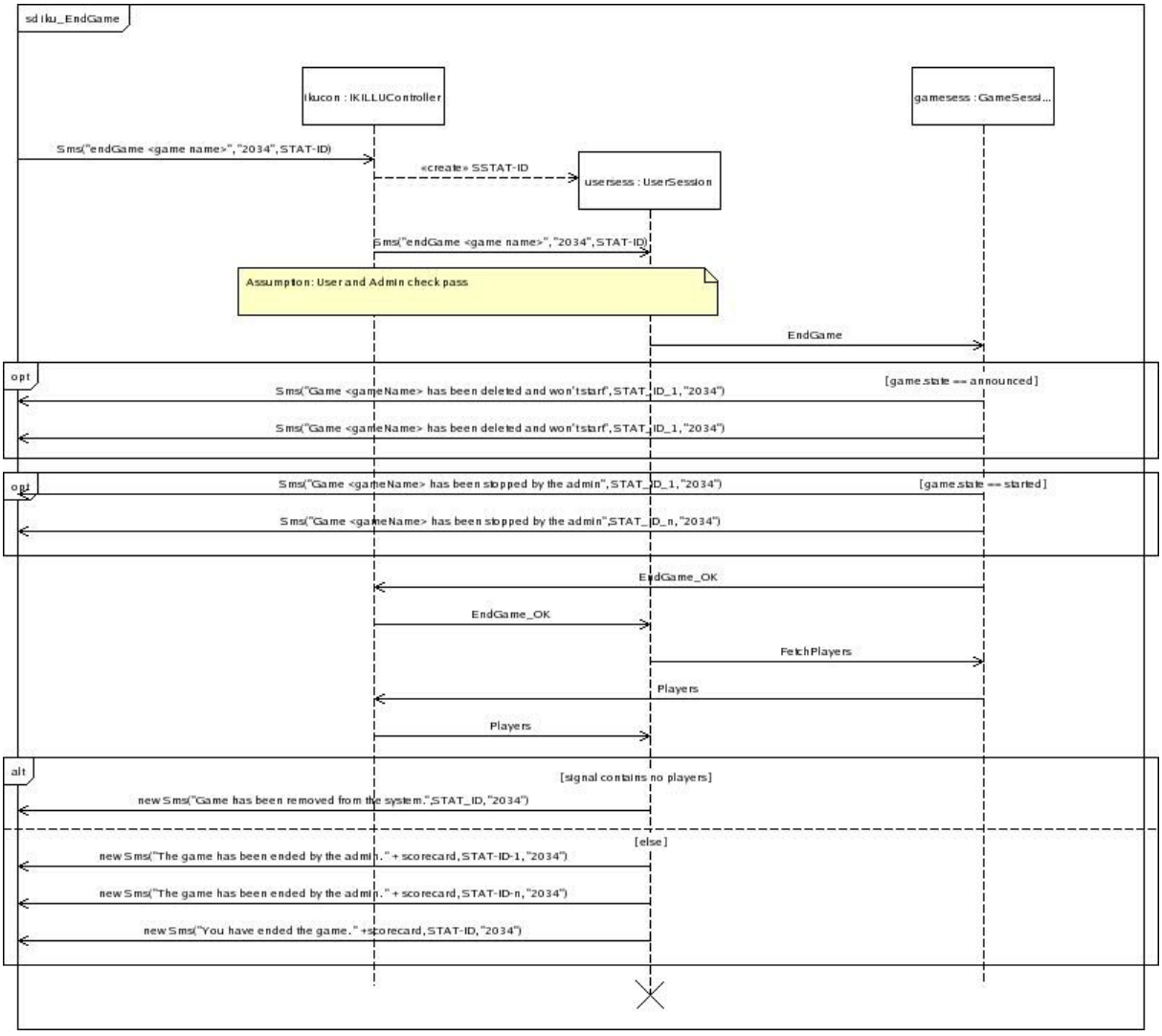
State machine for generate KML:



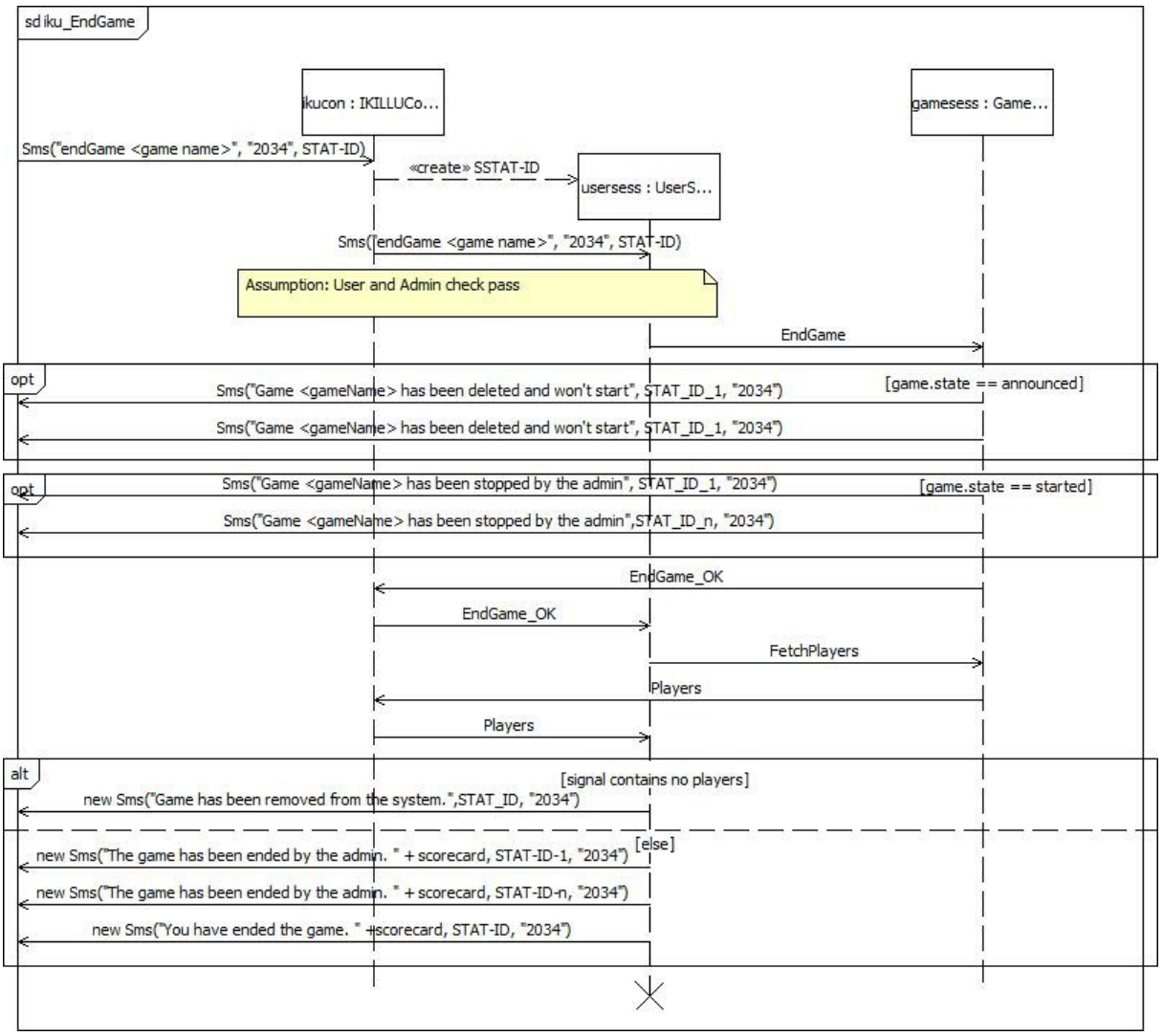
2.8.4 EndGame

The administrator of a game can decide to end his active game at any time. If a running game is stopped by the administrator, the player points are evaluated and all the current players get a message with the score within. Afterwards the game will be deleted.
If a game is stopped before the announcement, it will just be deleted. If it has already been announced, all invited players will be informed, before the game will be deleted.
After the end game command the game will accept the create command again, so that a new game can be started.

Decomposed Sequence diagram for end game:

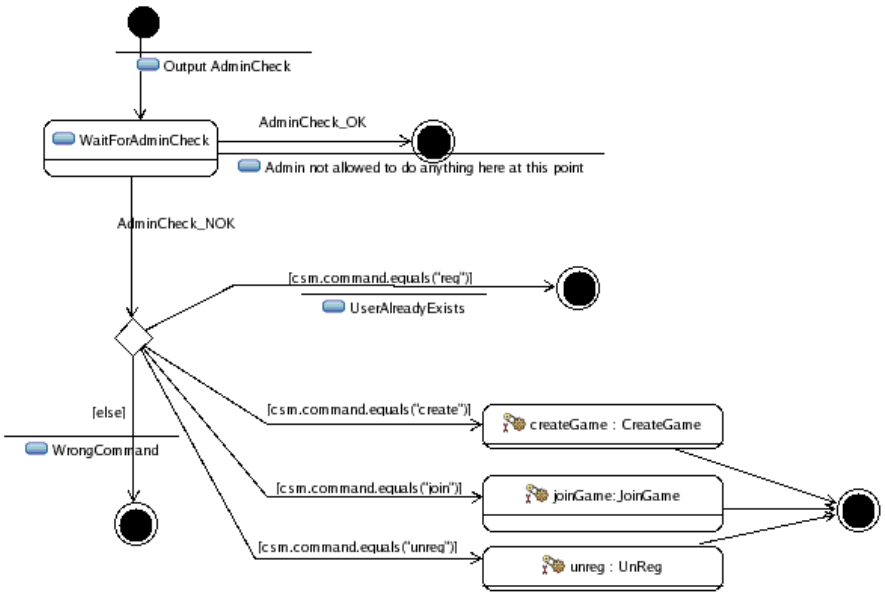


State machine for end game:


sd iku_EndGame
ikucon : IKILLUCo...
gamesess : Game...
Sms("endGame <game name>", "2034", STAT-ID)
«create» SSTAT-ID
usersess : UserS...
Sms("endGame <game name>", "2034", STAT-ID)
Assumption: User and Admin check pass
EndGame
opt
Sms("Game <gameName> has been deleted and won't start", STAT_ID_1, "2034")
[game.state == announced]
Sms("Game <gameName> has been deleted and won't start", STAT_ID_1, "2034")
opt
Sms("Game <gameName> has been stopped by the admin", STAT_ID_1, "2034")
[game.state == started]
Sms("Game <gameName> has been stopped by the admin",STAT_ID_n, "2034")
EndGame_OK
EndGame_OK
FetchPlayers
Players
Players
alt
[signal contains no players]
new Sms("Game has been removed from the system.",STAT_ID, "2034")
[else]
new Sms("The game has been ended by the admin. " + scorecard, STAT-ID-1, "2034")
new Sms("The game has been ended by the admin. " + scorecard, STAT-ID-n, "2034")
new Sms("You have ended the game. " +scorecard, STAT-ID, "2034")

2.9 User Services

The user service additional checks, if the user is the admin of a game. The admin is not allowed to perform these services, for example join a game or unregister. Afterwards the command is forwarded to the right substatemachine.

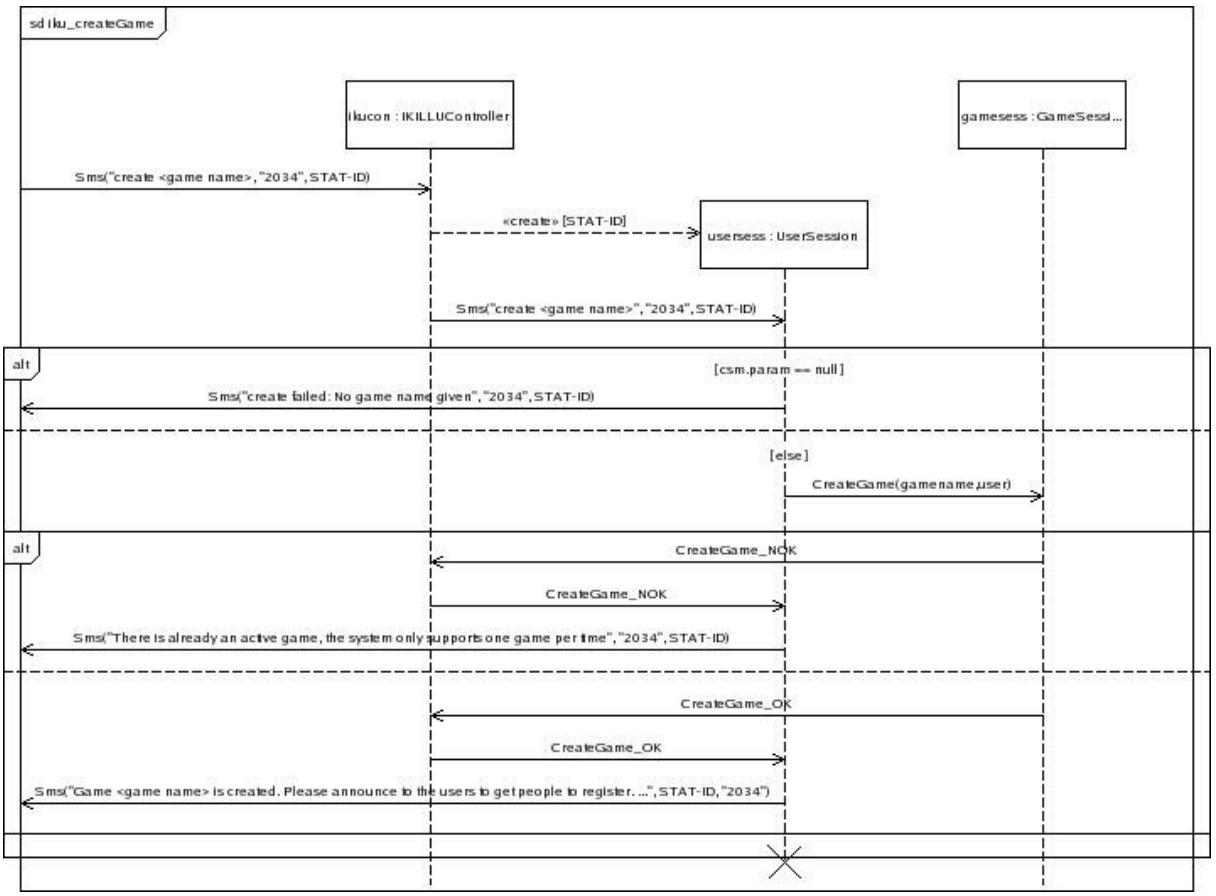


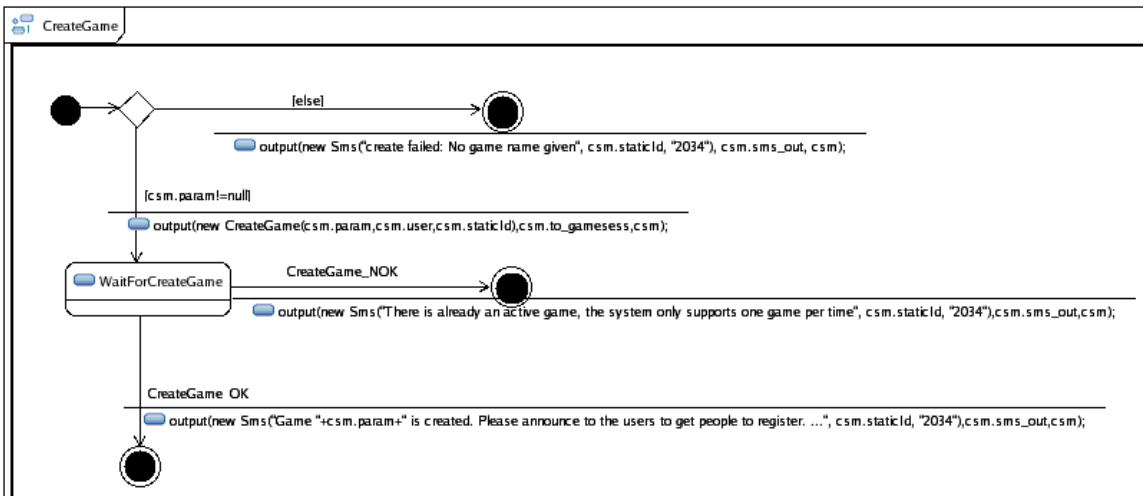
2.9.1 CreateGame

Every user, who is registered in the System, can create the game and become administrator (admin). We decided to divide the initial phase in three steps. This allows the admin to configure the settings of the game. Therefore we have create, configure and announce as initialization steps.

The first step is the **create game** command.
For every command apart from *register*, the first check the userSession does is to get the User Object from the game controller. Every game should be named. So the User Session checks, if the a new game name is given as a parameter to the command. Then the command is sent to the Game Session. As it only possible to have one game in the system for now, it will check, if there is an active game. If there already is an active game, the gameSession sends a CreateGame_NOK Message. Otherwise it create a new Game Object, then initialize the game data and send a CreateGame_OK Message back. The UserSession will inform the user about the result of the command.

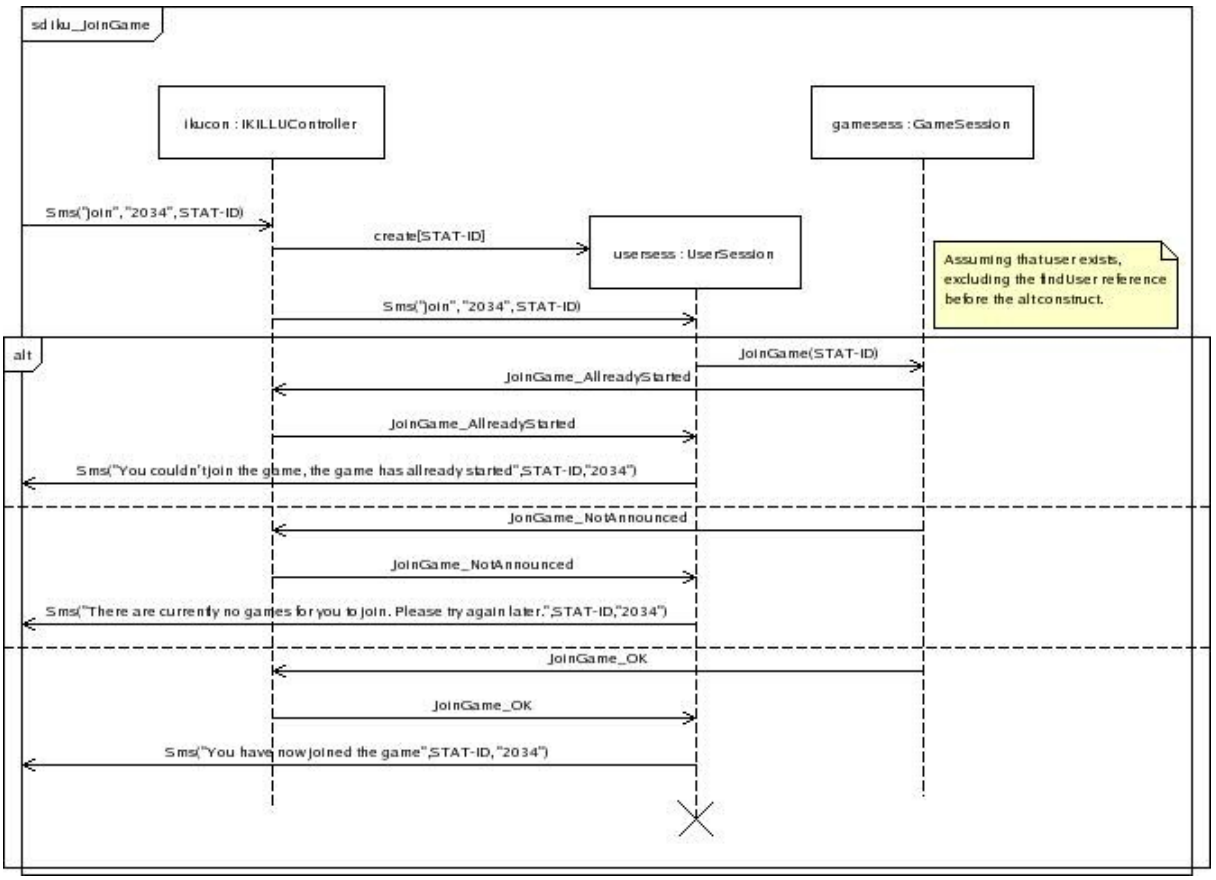
We haven't implemented the **configure** command. That means the game can only be played with the default configuration. The next steps to get the game up and running is therefore the ***announce*** and ***start*** commands which will be described under the section AdminCommands.

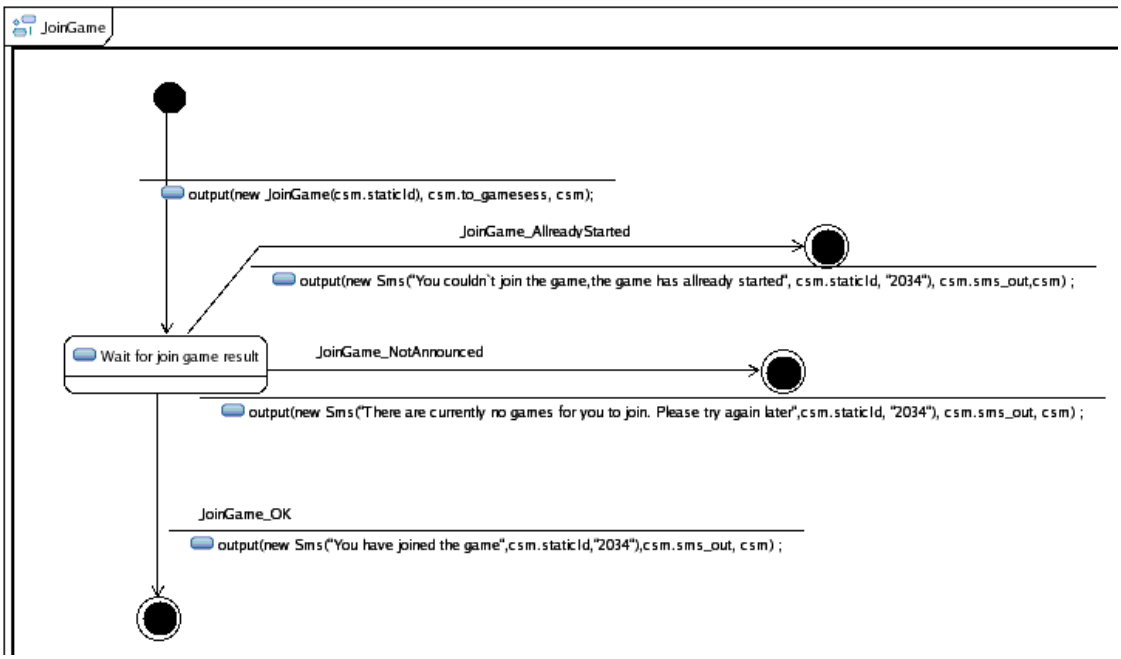




2.9.2 JoinGame

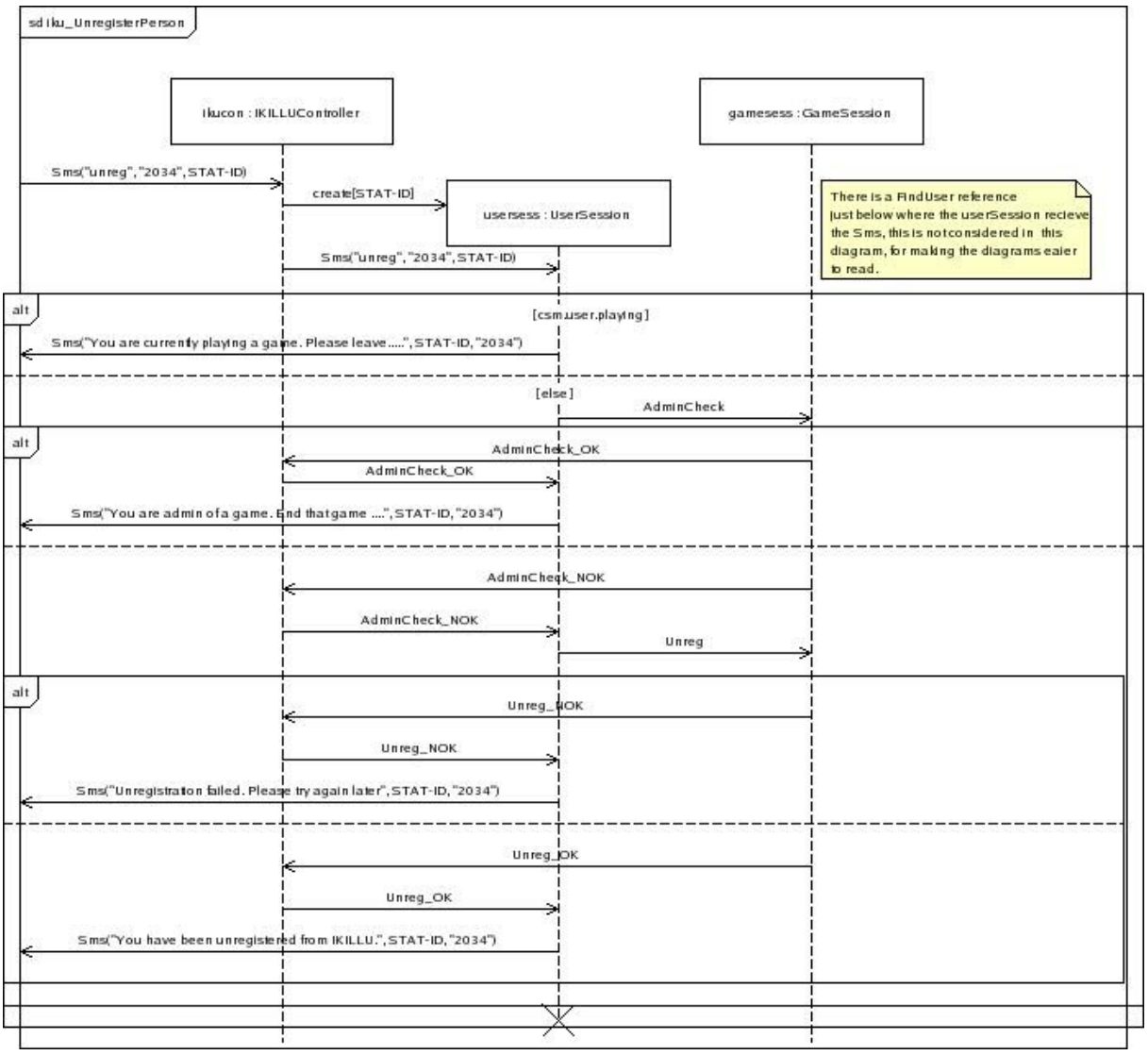
To be able to join a game it must first be announced. The first step is to send a message with the JoinGame command ***join***. There are three possible outcomes. If the game has been announced but has not started yet, the player will then get a message that he/she has joined the game. Otherwise the userSession will send a message to the player with the result that the game could not be joined either because the game has already started or that it has not been announced yet.


sd Iku_JoinGame
Ikucon : IKILLUController
gamesess : GameSession
Sms("join","2034",STAT-ID)
create[STAT-ID]
usersess : UserSession
Assuming that user exists, excluding the findUser reference before the alt construct.
Sms("join","2034",STAT-ID)
alt
JoinGame(STAT-ID)
JoinGame_AllreadyStarted
JoinGame_AllreadyStarted
Sms("You couldn't join the game, the game has allready started",STAT-ID,"2034")
JoinGame_NotAnnounced
JoinGame_NotAnnounced
Sms("There are currently no games for you to join. Please try again later.",STAT-ID,"2034")
JoinGame_OK
JoinGame_OK
Sms("You have now joined the game",STAT-ID,"2034")

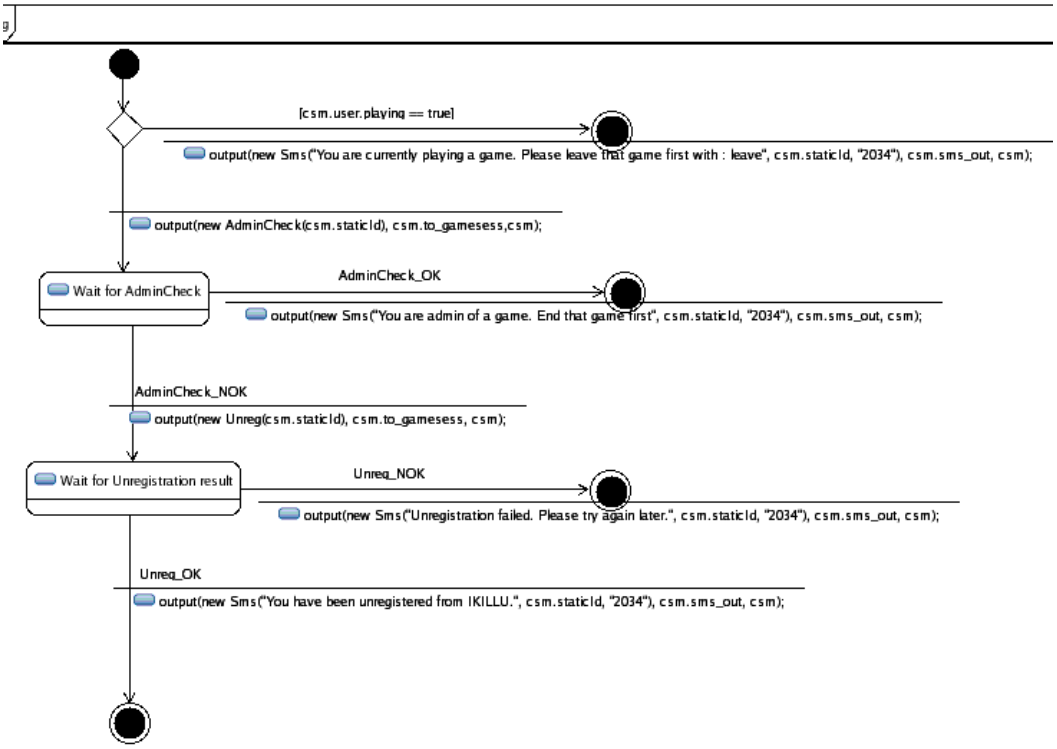

JoinGame
output(new JoinGame(csm.staticId), csm.to_gamesess, csm);
JoinGame_AllreadyStarted
output(new Sms("You couldn`t join the game,the game has allready started", csm.staticId, "2034"), csm.sms_out,csm) ;
Wait for join game result
JoinGame_NotAnnounced
output(new Sms("There are currently no games for you to join. Please try again later",csm.staticId, "2034"), csm.sms_out, csm) ;
JoinGame_OK
output(new Sms("You have joined the game",csm.staticId,"2034"),csm.sms_out, csm) ;

2.9.3 UnregisterPerson

The diagram shows what will happen if a registered user wants to unregister. A situation where the user is *not* registered is not handled in this document.
Various unassailability and game/system integrity checks are performed before the user actually is allowed to unregister. If a user is either a Player of a game or an Admin of a game then he will not be allowed to unregister. The user must then follow the standard procedure for either leaving a game or ending the game depending on if he is a Player of an Admin.



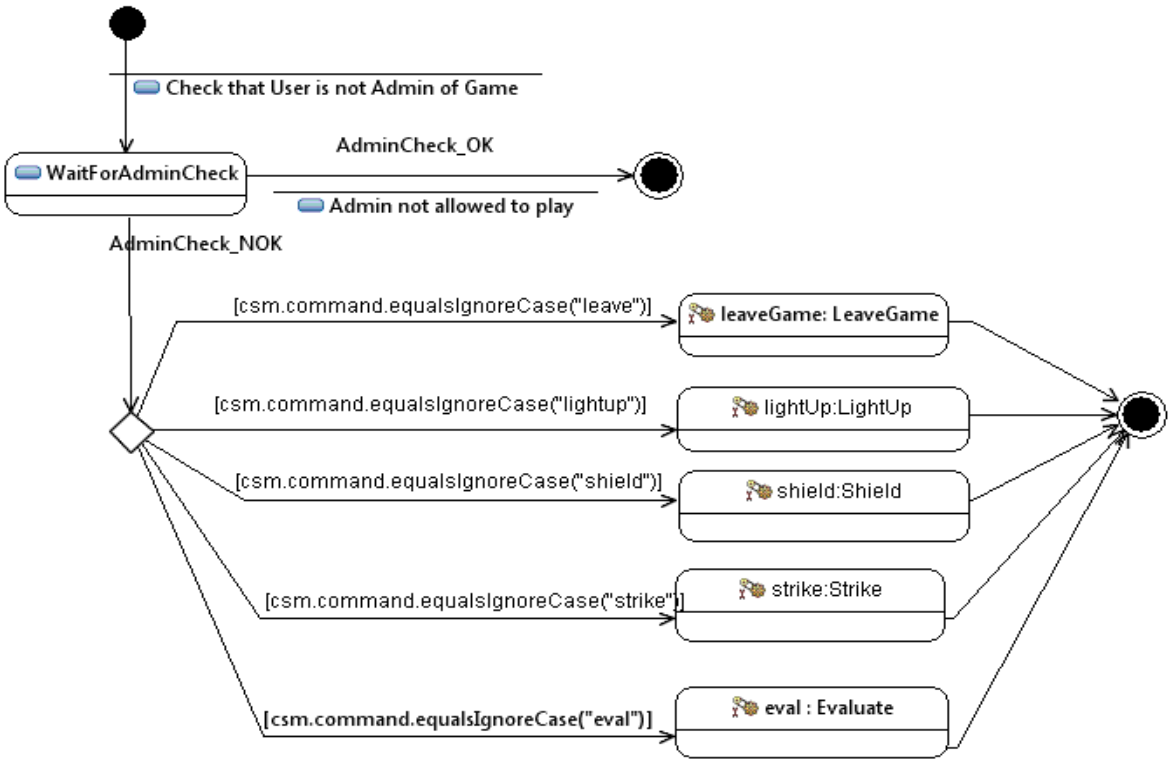
Figur 7 - Decomposed sequence diagram for UnregisterPerson



Figur 8 - Sub state machine for UnregisterPerson

2.10 Game Services (Sequence Diagrams and StateMachines)

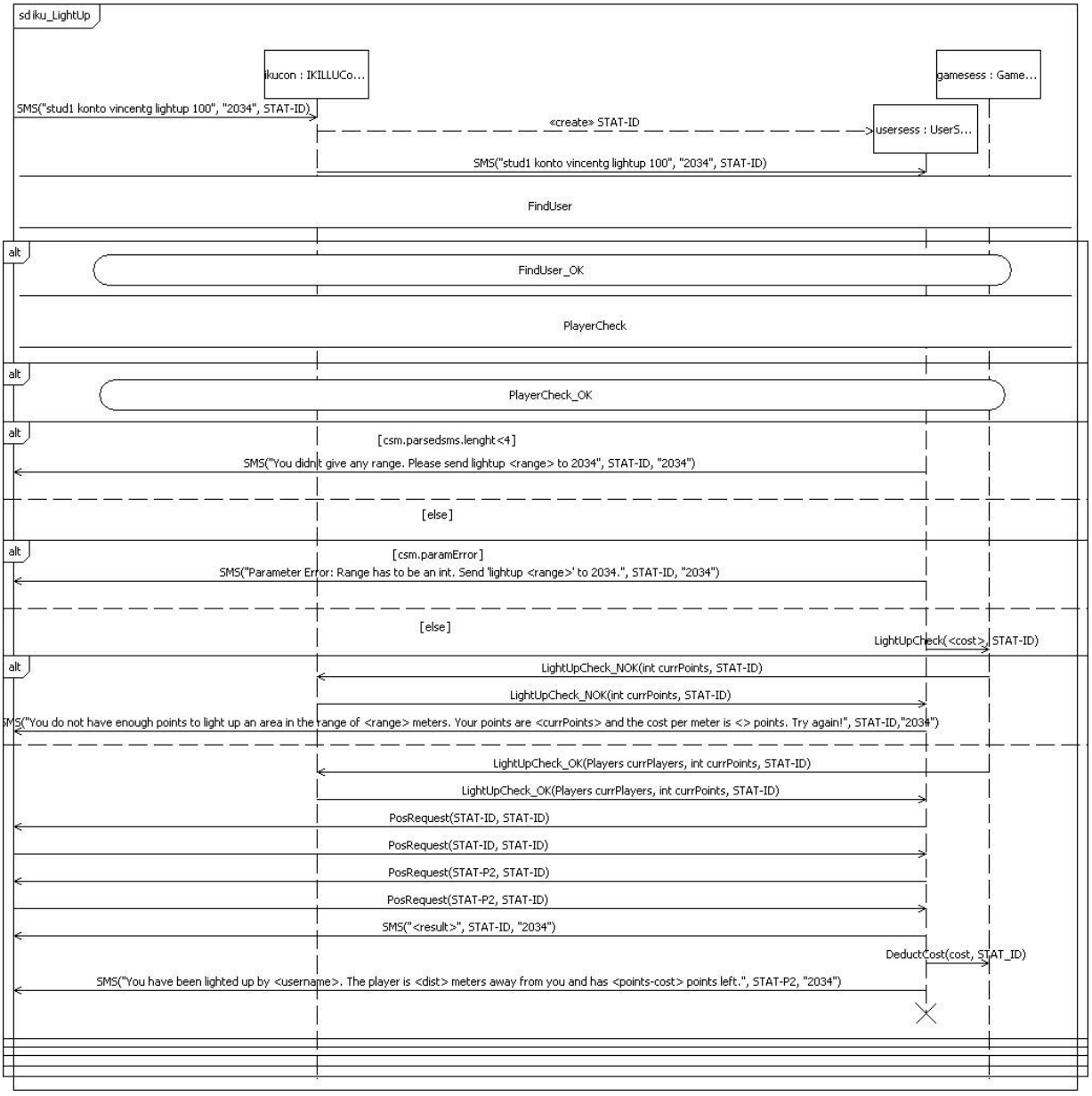
Here are the commands used by a player in a running game.
To enter this state machine the command must be leave, lightup, shield, strike or eval.
An admin cannot use a game command.
Here is the state machine that drives the SMS according to the command.



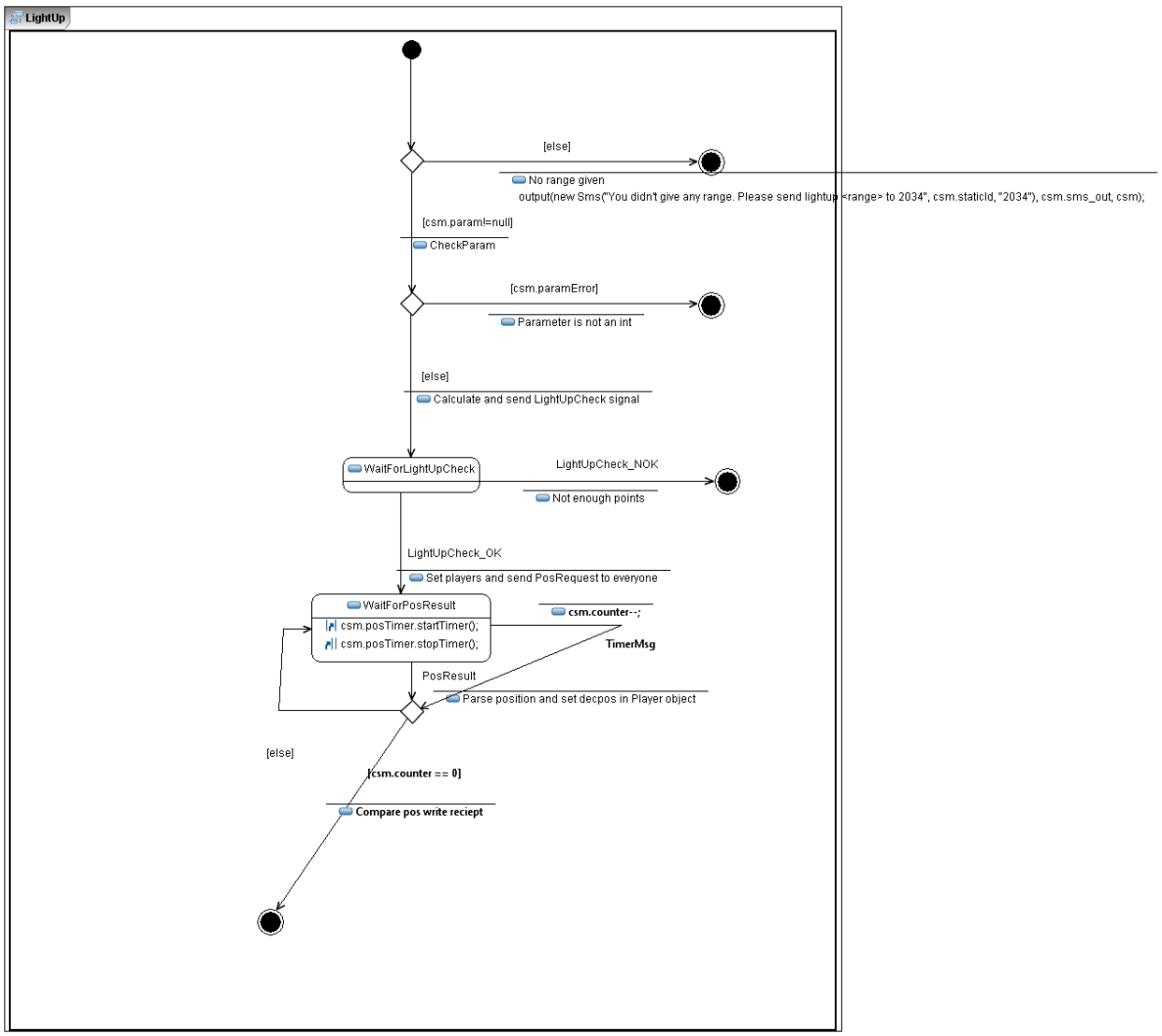
2.10.1 LightUp

Here we have referenced some sequence diagrams. They can be found in Appendix A: Referenced sequence diagrams

Sequence diagram:



State machine:



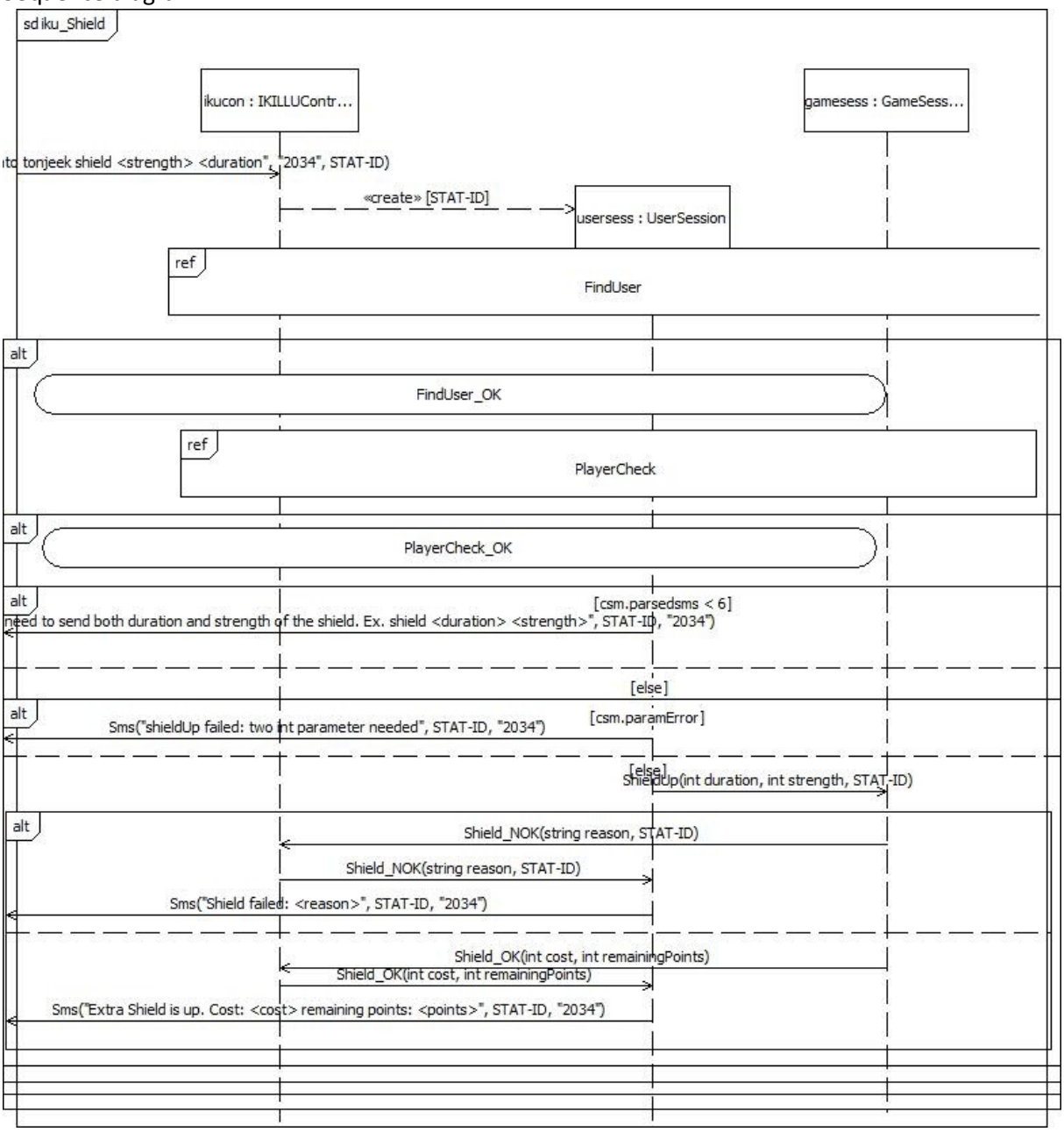
The lightup command has one parameter which is the range of the wanted area (circle around the sender) to light up.
We have to assure that we're in the case of a player playing in a running game (references to FindUser and PlayerCheck).
Then we prevent the process for incomplete/wrong SMSs.
Then the system will check if the player has enough points to perform this light up: the user session sends the internal signal *LightUpCheck* to the game session which will be in charge of according the light up (*LightUpCheck_OK)* or not (*LightUpCheck_NOK)*.
With the *LightUpCheck_OK* signal, the user session gets also the current players in the game (which will be useful to perform the light up) and the points the player has.
We're now ready to perform the light up operation, in order to do so we have to locate every player in the game, do the calculation of distance between the sender and every other player (using the distance method in the player object) and send the answer back with all the players who have been found (means all the player who have a distance less than the range given).
We introduced a timer so that we don't wait infinitely for a PosResult when locating.
All the lighted up player receive a SMS which tells them that they've been lighted up, by who, his distance and his remaining points.

If the command sender himself cannot be positioned, the light up cannot be performed and the sender receive a message that tells him so, the cost is of course not deduced.

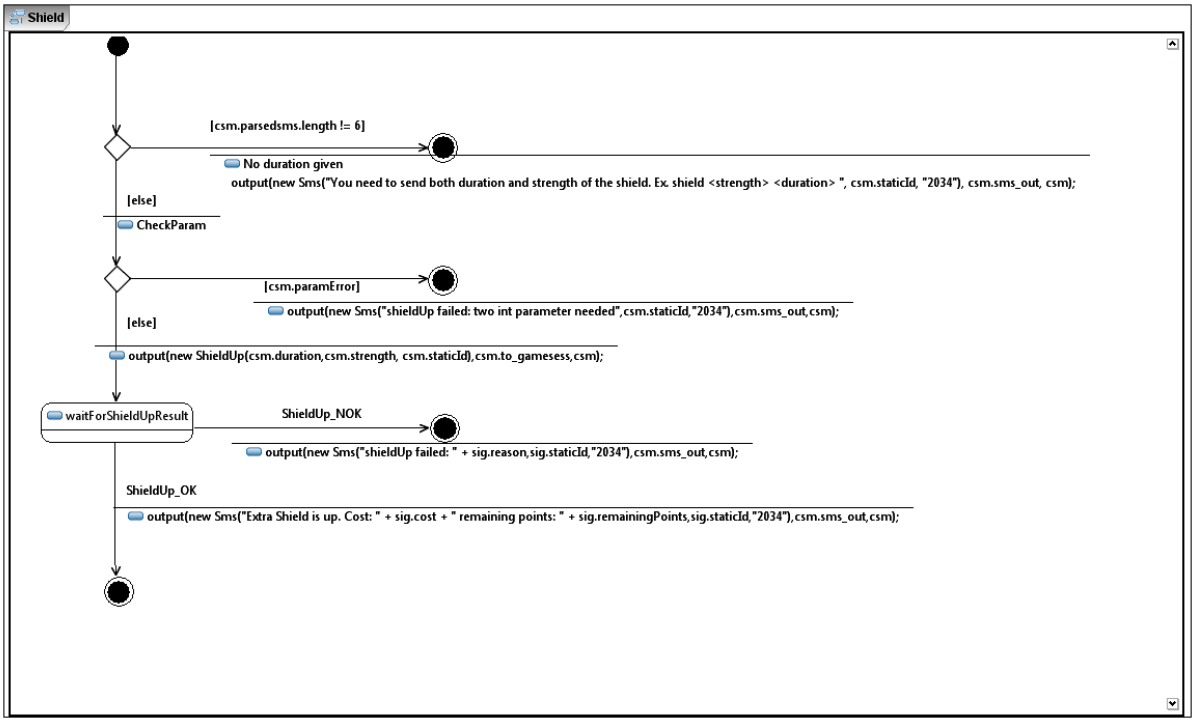
2.10.2 IncreaseShield

Here we have referenced some sequence diagrams. They can be found in Appendix A: Referenced sequence diagrams.

Sequence diagram:



State machine:



The shield command has two parameters: first the strength, then the duration.
First of all we prevent the process to have wrong arguments in the SMS.
The user session sends a *ShieldUp* internal signal with both duration and strength for parameters to the game session.
A *ShieldUp_NOK* can be sent back for two reasons:

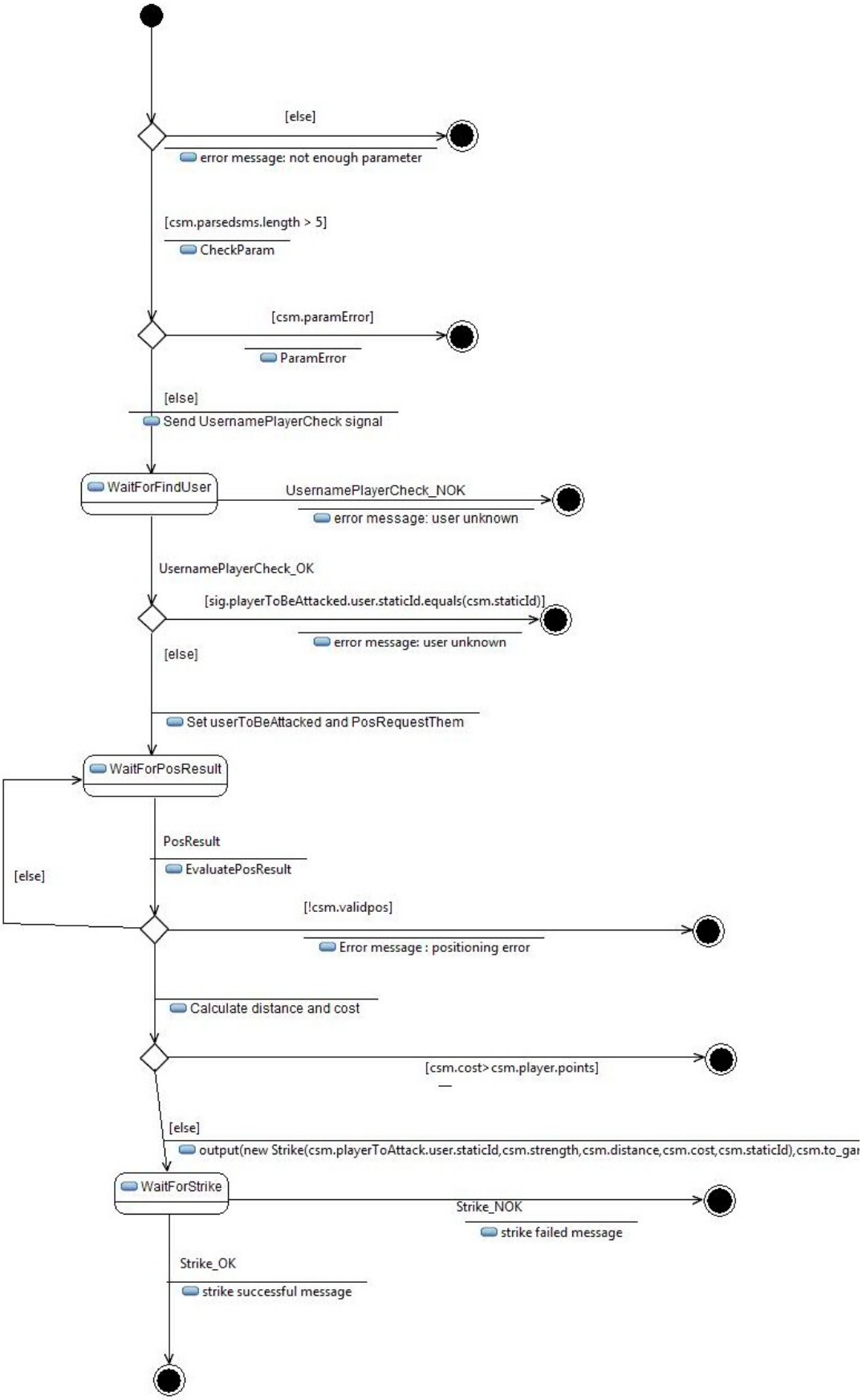
- The player is already dead when he wants to shield up.
- He hasn't enough points to perform this operation

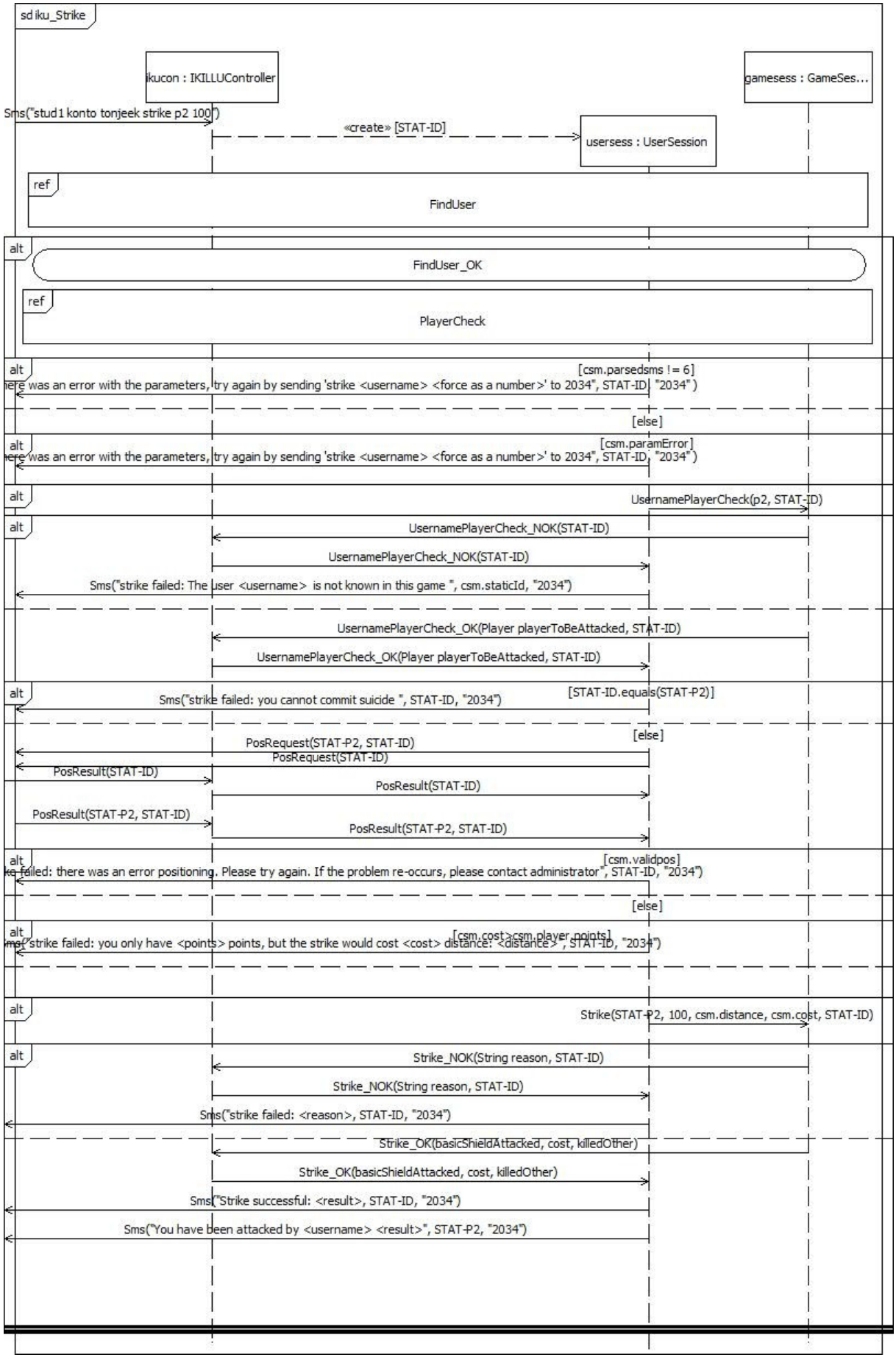
In both cases the reason is sent back in the signal (ikucon then usersess for routing matters) and then proceeds by sending a SMS.

When the shield is accepted, the process is done directly in the game session (by updating the extraShield value in the player object, the extraShieldDuration and by putting the current time in extraShieldStartTime). The cost is also deducted in a breath.
If the shield up is accepted, a *ShieldUp_OK* signal is sent to the usersess (by the ikucon), the user is notified by a SMS that is shield up action has been approved.

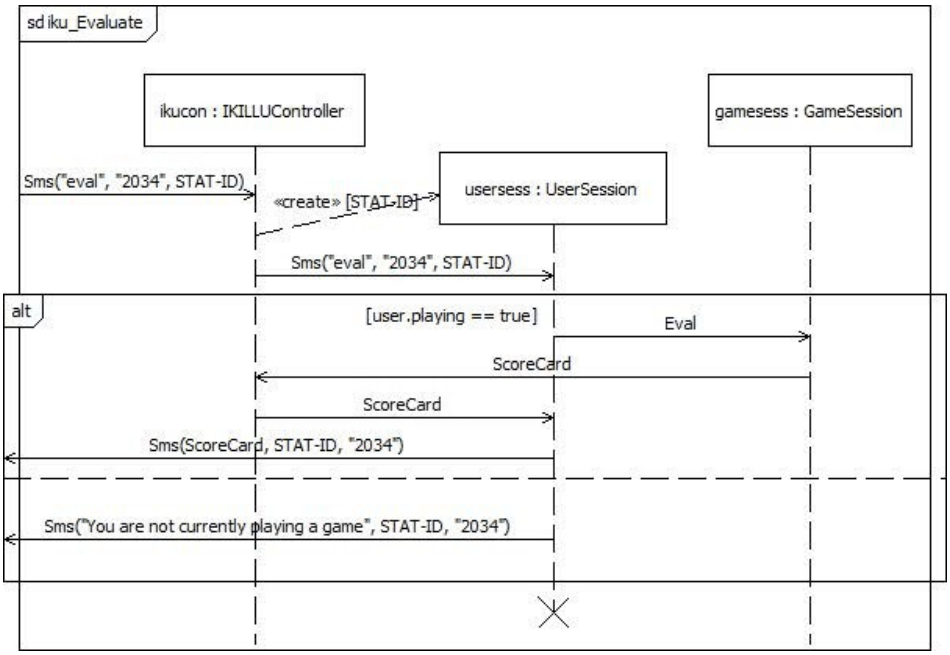
2.10.3 Strike

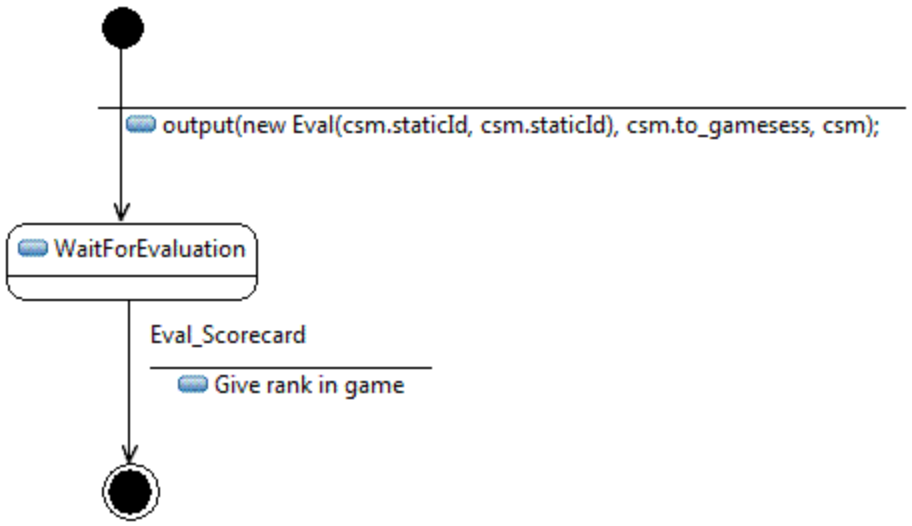
Strike is the central player command. A player, that wants to attack another player send a strike message with the name of the attacked player and the force. First there are a few checks done, regarding the parameters of the message. If the player sends the wrong amount of parameters or the value of the strength is not a positive number the process will stop and deliver an error message to the user.
In case of proceeding, the GameSession is asked for the attacked player. In case the attacked is unknown, or it is identical with the attacker (we also discussed the surprising outcomes of suicide bombers, but finally disqualified them) , the process will again stop after delivering an error message to the user.
After having the information about the players, the UserSession will try to locate both players, through the sending of Positioning Requests. Afterwards it can calculate the needed distance between the players. The last checkis the calculation of the cost, which is based on the distance , to check if the player has enough points for the strike.
The GameSession then receives the command and executes the strike. The force of the attack is multiplied with a random number between zero and two. The attacked player will loose shield points according to this value.
If this was the final strike for the attacked player, that means his basic shield is below zero after the strike, he will drop out the game and the attacker receices points from him. The GameSession checks additional, if there is only one player left and ends the game then.
The last step is to deduct the cost of the strike. A message is sent back to the UserSession and both players are informed about the result of the strike.


[else]
error message: not enough parameter
[csm.parsedsms.length > 5]
CheckParam
[csm.paramError]
ParamError
[else]
Send UsernamePlayerCheck signal
WaitForFindUser
UsernamePlayerCheck_NOK
error message: user unknown
UsernamePlayerCheck_OK
[sig.playerToBeAttacked.user.staticId.equals(csm.staticId)]
error message: user unknown
[else]
Set userToBeAttacked and PosRequestThem
WaitForPosResult
PosResult
EvaluatePosResult
[else]
[!csm.validpos]
Error message : positioning error
Calculate distance and cost
[csm.cost> csm.player.points]
[else]
output(new Strike(csm.playerToAttack.user.staticId,csm.strength,csm.distance,csm.cost,csm.staticId),csm.to_ga
WaitForStrike
Strike_NOK
strike failed message
Strike_OK
strike successful message


sd iku_Strike
ikucon : IKILLUController
gamesess : GameSes...
Sms("stud1 konto tonjeek strike p2 100")
«create» [STAT-ID]
usersess : UserSession
ref
FindUser
alt
FindUser_OK
ref
PlayerCheck
alt
[csm.parsedsms != 6]
ere was an error with the parameters, try again by sending 'strike <username> <force as a number>' to 2034", STAT-ID, "2034")
[else]
alt
[csm.paramError]
ere was an error with the parameters, try again by sending 'strike <username> <force as a number>' to 2034", STAT-ID, "2034")
alt
UsernamePlayerCheck(p2, STAT-ID)
alt
UsernamePlayerCheck_NOK(STAT-ID)
UsernamePlayerCheck_NOK(STAT-ID)
Sms("strike failed: The user <username> is not known in this game ", csm.staticId, "2034")
UsernamePlayerCheck_OK(Player playerToBeAttacked, STAT-ID)
UsernamePlayerCheck_OK(Player playerToBeAttacked, STAT-ID)
alt
[STAT-ID.equals(STAT-P2)]
Sms("strike failed: you cannot commit suicide ", STAT-ID, "2034")
[else]
PosRequest(STAT-P2, STAT-ID)
PosRequest(STAT-ID)
PosResult(STAT-ID)
PosResult(STAT-ID)
PosResult(STAT-P2, STAT-ID)
PosResult(STAT-P2, STAT-ID)
alt
[csm.validpos]
ke failed: there was an error positioning. Please try again. If the problem re-occurs, please contact administrator", STAT-ID, "2034")
[else]
alt
[csm.cost>csm.player.points]
ms("strike failed: you only have <points> points, but the strike would cost <cost> distance: <distance>", STAT-ID, "2034")
alt
Strike(STAT-P2, 100, csm.distance, csm.cost, STAT-ID)
alt
Strike_NOK(String reason, STAT-ID)
Strike_NOK(String reason, STAT-ID)
Sms("strike failed: <reason>, STAT-ID, "2034")
Strike_OK(basicShieldAttacked, cost, killedOther)
Strike_OK(basicShieldAttacked, cost, killedOther)
Sms("Strike successful: <result>, STAT-ID, "2034")
Sms("You have been attacked by <username> <result>", STAT-P2, "2034")

2.10.4 Evaluate

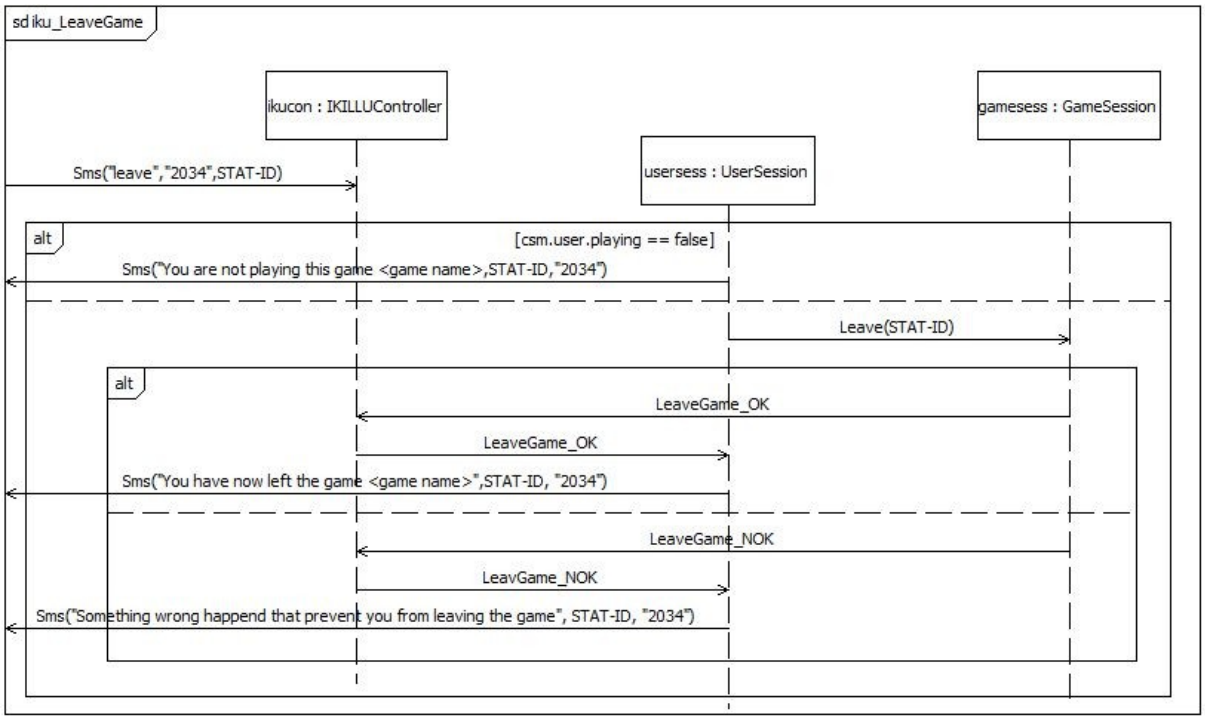


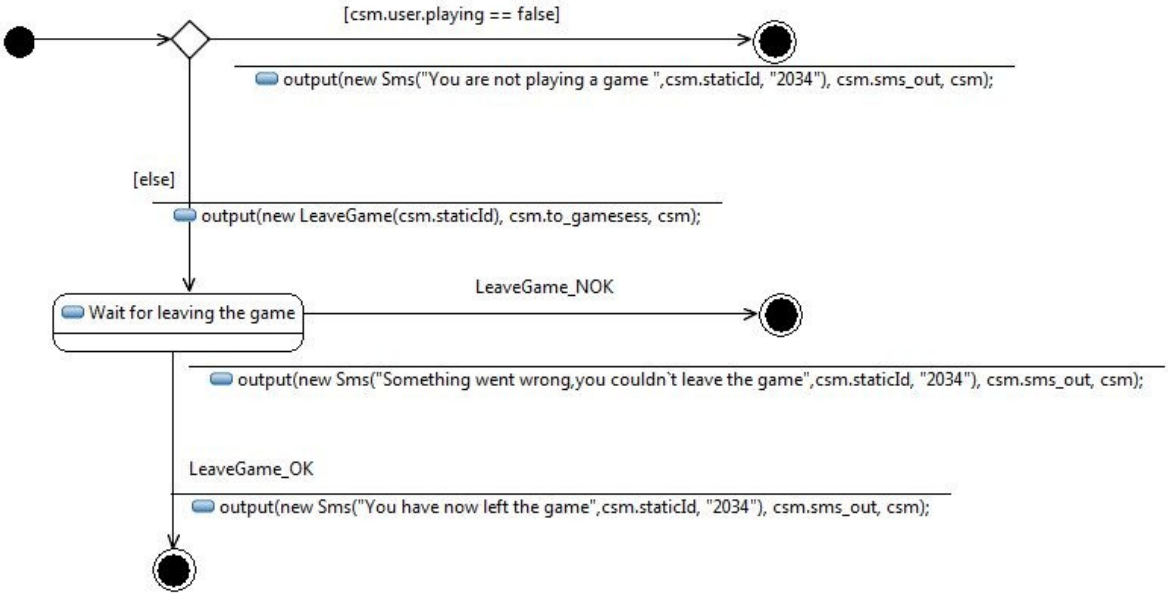


This is a Player command. The player can send a message to see his/hers current rank in the game, current amount of points and the current total shield value. No other type of return other than Eval_Scorecard is possible since to get this far the Player must be a valid Player with a valid Score. The algorithm for checking rank is as follows: All players start the iteration with rank 1. If a another player has more points than the one to be ranked the rank is incremented. Therefore multiple players with the same score can have the same rank (ex. In a game of 4, 3 can have rank 1st while the 1 (if his score is lower than the other 3) then will get rank 4th).
The function also returns the number of alive players. Aliveness is determined by “if points > 0”. A player with shield left but no points is therefore considered irrelevant to the game since he cannot win or strike anyone.

2.10.5 LeaveGame

By sending a message with the leavegame command it is possible to leave the game, if a player does not want to be a part of it anymore. The player will then be removed from the players list, and be set as not playing. If a player tries to leave a game that he/she is not playing or by accident writes the game name wrong, the system will then notify him/her by sending back a message. After leaving the game the player is still registered, and can join a new game when he/she gets a new invite.





3 Assumptions, weaknesses and improvements

Future improvements of the game/system:

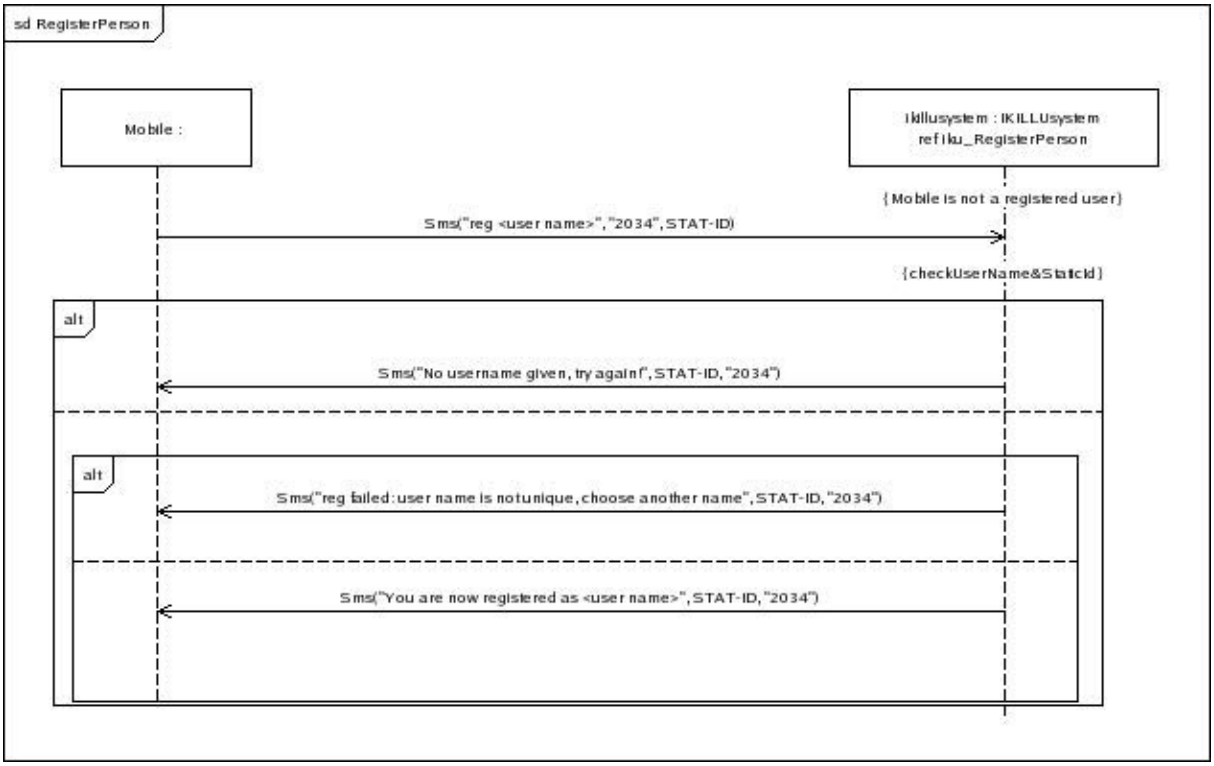
- Multiple games and multiple administrators
- Persistent storage of data
    - As of now, a system reboot will reset all data
    - This would also give the system the possibility of logging more efficiently the progress of the game, and would also enable Players to more thoroughly examine their own session and how the game transpired.
- Web-based access-system for the KML file with authentication.
- Interface for Admins to set individual starting points for the Players, shield values etc.
- Interface for Sysops to reset points and user-information in case of unknown bugs or hacker creating errors, faults in the data. This would provide at least one failsafe to guarantee that customer points and thereby winnings are maintained securely.
- User authentication
    - Password protected sessions with timeouts
- Code/structure efficiency:
    - Timestamps for last positioned would enable the system to determine intelligently if the position needs updating. The system now positions every single player when the Player lights up an area. This is not the most efficient way to solve the Light up command.
    - Timers on Positioning requests, system-wide. In case the third-party supplies of Positions does not respond with Positioning results in due time, all parts of the system should automatically stop waiting and move on.
    - Timers to kick players out of the Game if they do not perform an action in due time. Or a method for the administrator to notify the user that they will be kicked if they do not perform an action soon.
- Game improvements:
    - Analyze and determine the best cost-functions for shield, strike, lightup based on evidence based research and usability analysis.
- Implement the configuration of a game. These are possible parameters, we thought about.
    - The initial points /shield values of the users
    - A fix duration of the game
    - A fix starting time
    - The game starts, when a xed number of players have registered for this
    - game
    - Users that are inactive for a certain amount of time will be dropped out
    - the game
    - If all users are inactive for a certain amount of time, the game will be
    - evaluated
    - only a dened subset of the registered players will be invited to the game
    - set a region for the game (that means, every potential user has to be located)

4 Appendix A

To keep the document as uncluttered as possible, we chose to add all sequence diagrams in an appendix.

4.1 Setup Game Services (Sequence Diagrams)

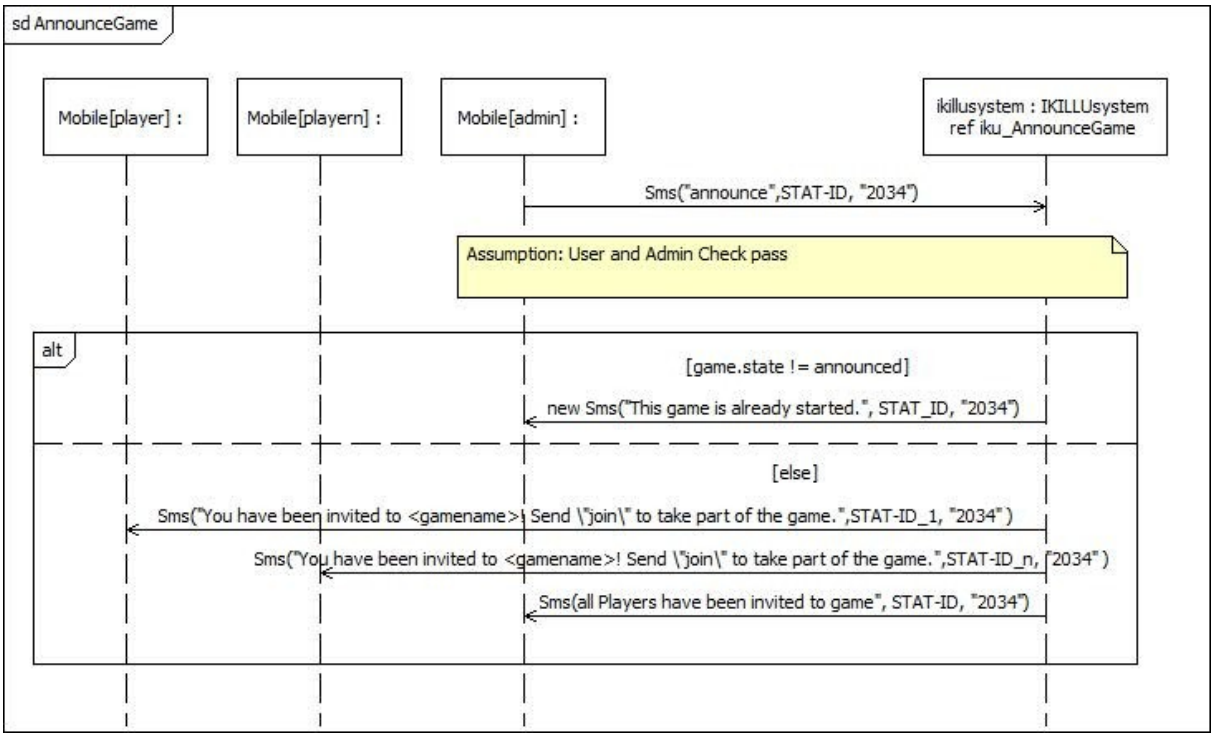
4.1.1 RegisterPerson



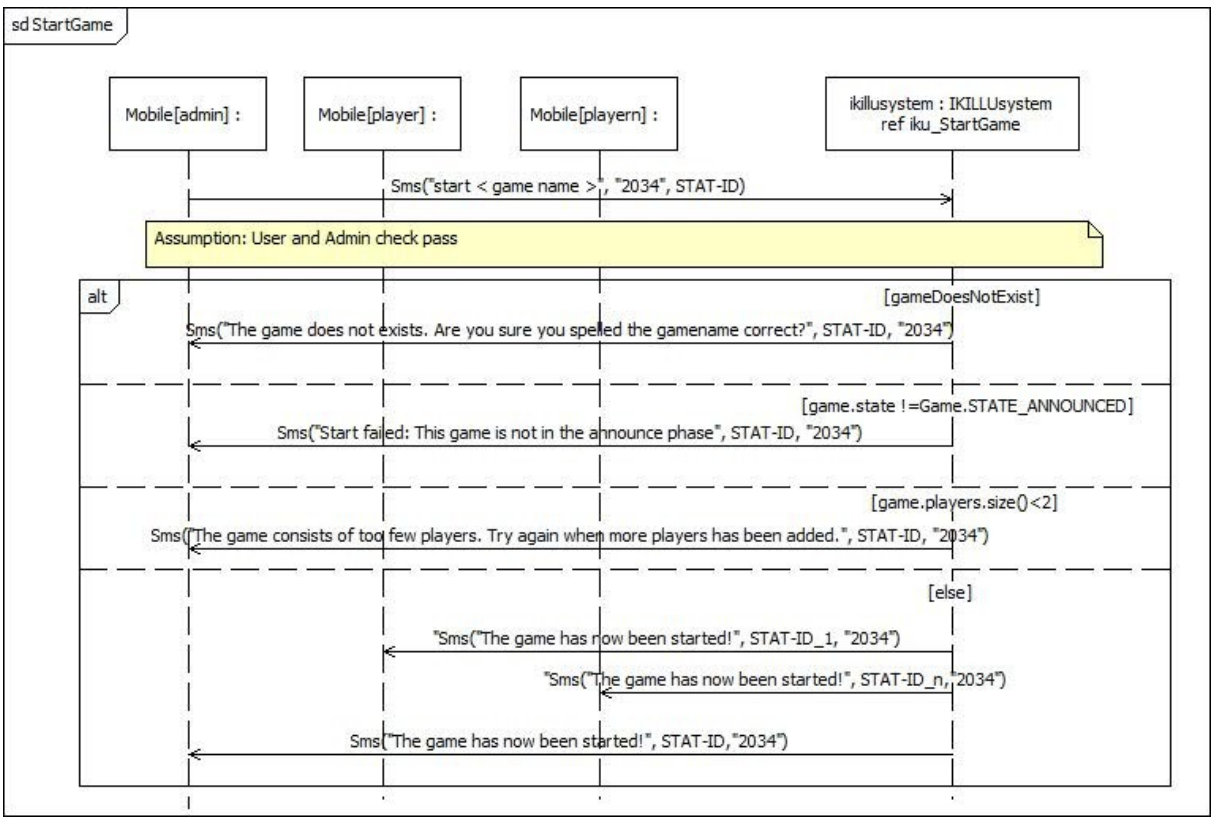
Figur 9 - Sequence diagram for RegisterPerson

4.2 Admin Services

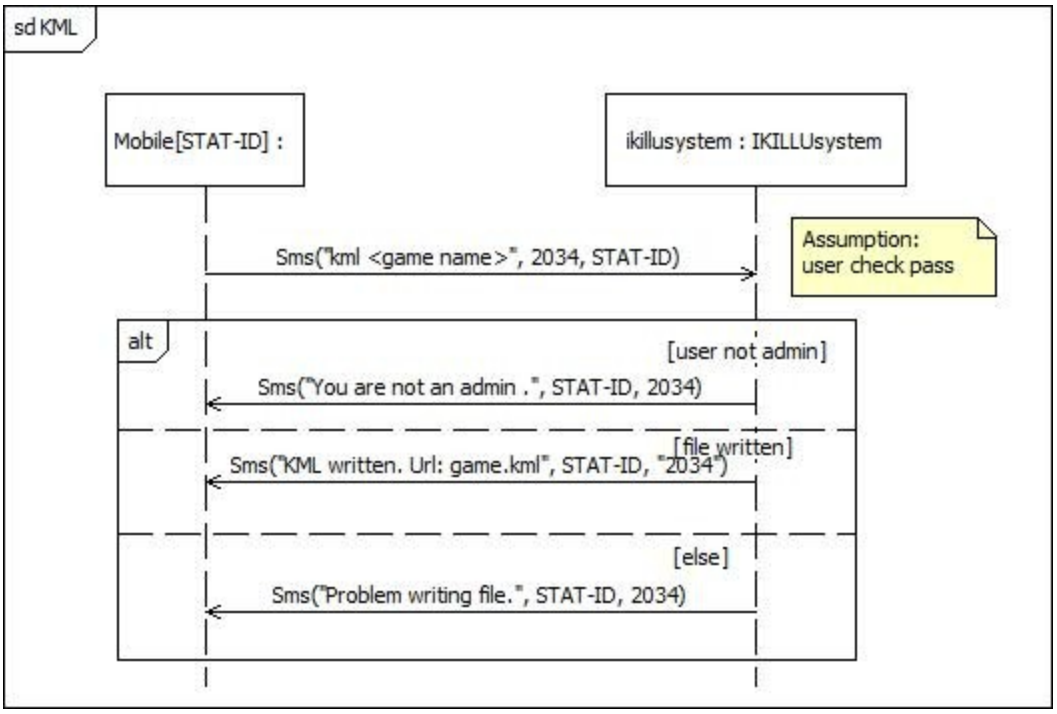
4.2.1 AnnounceGame



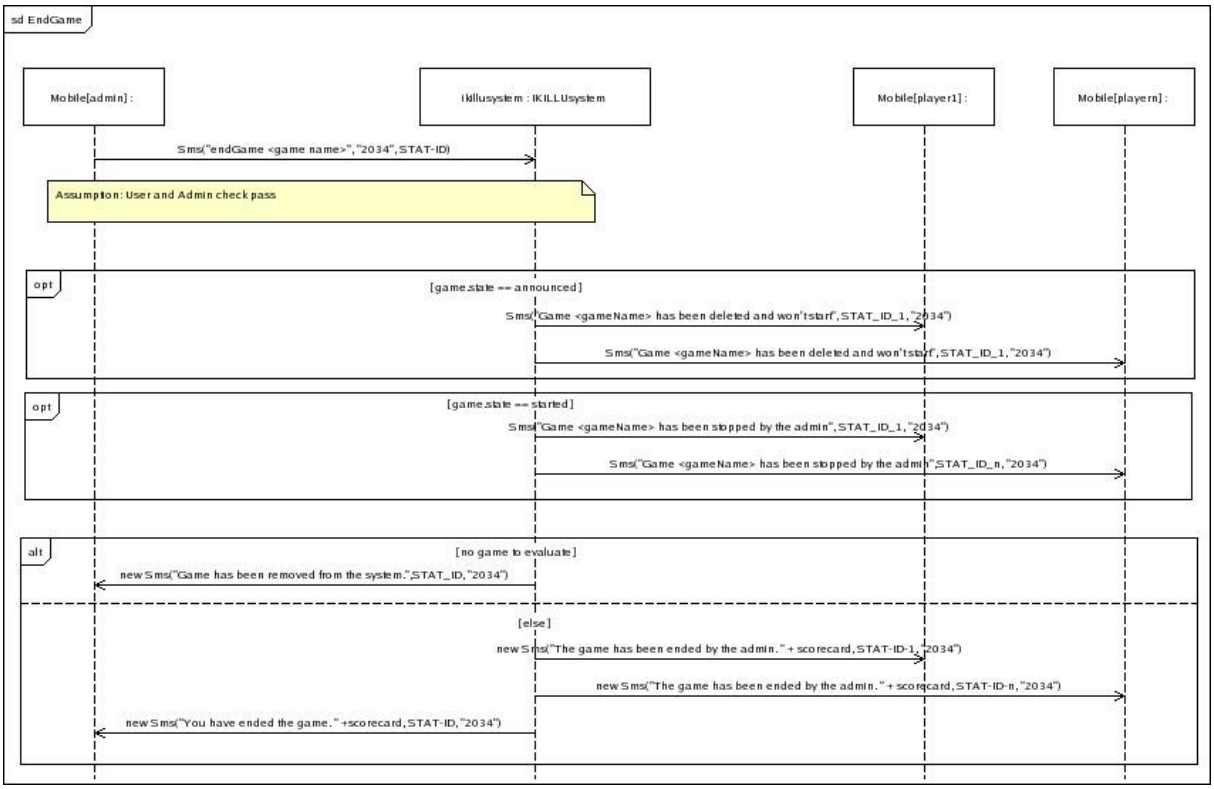
4.2.2 StartGame



4.2.3 GenerateKML

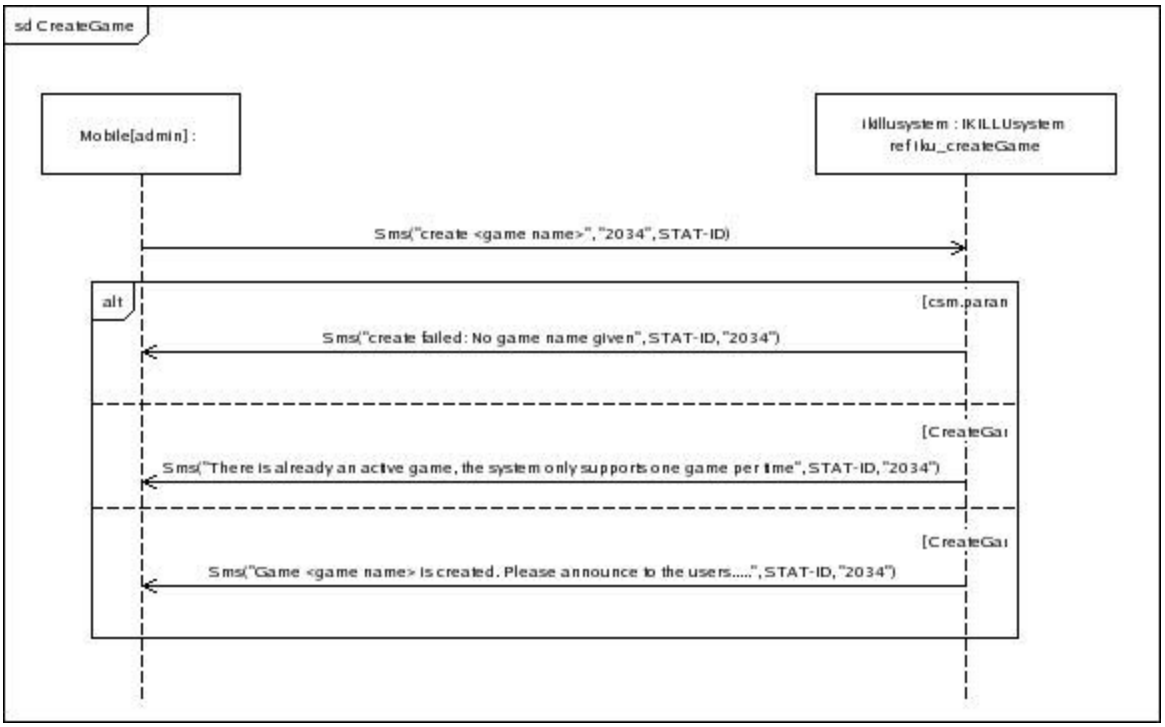


4.2.4 EndGame

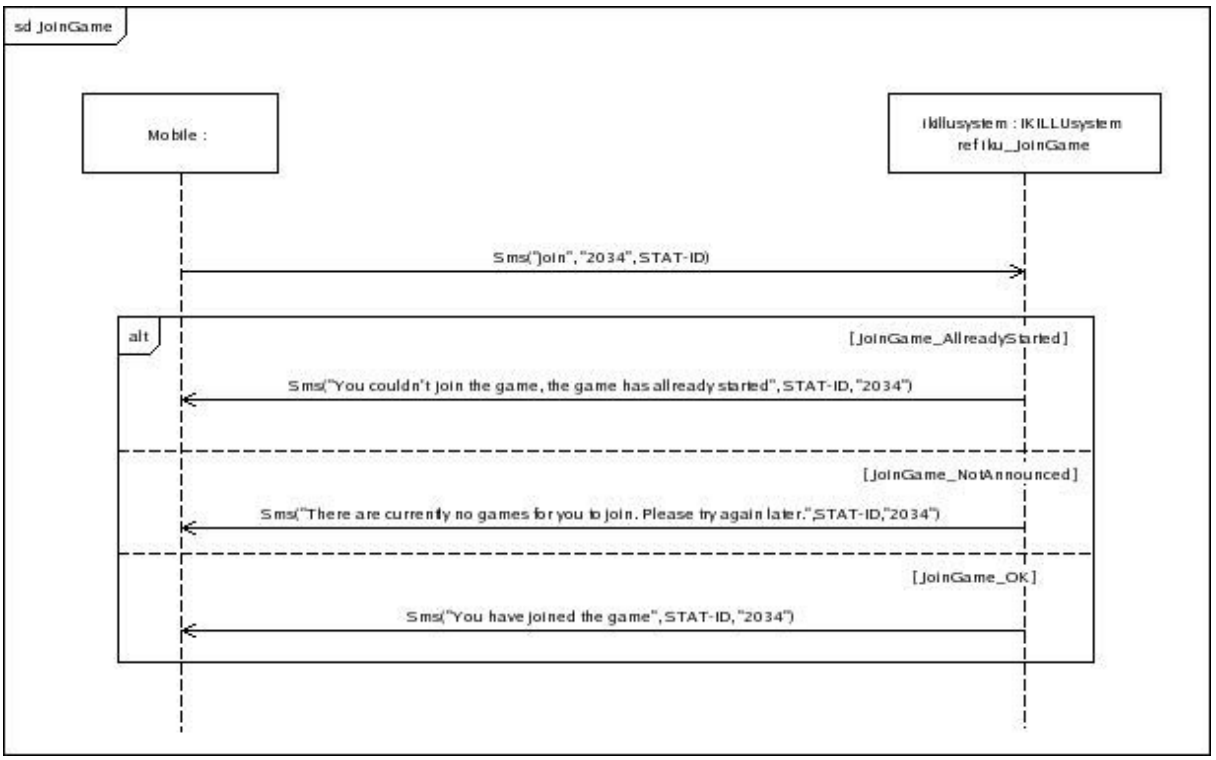


4.3 User Services

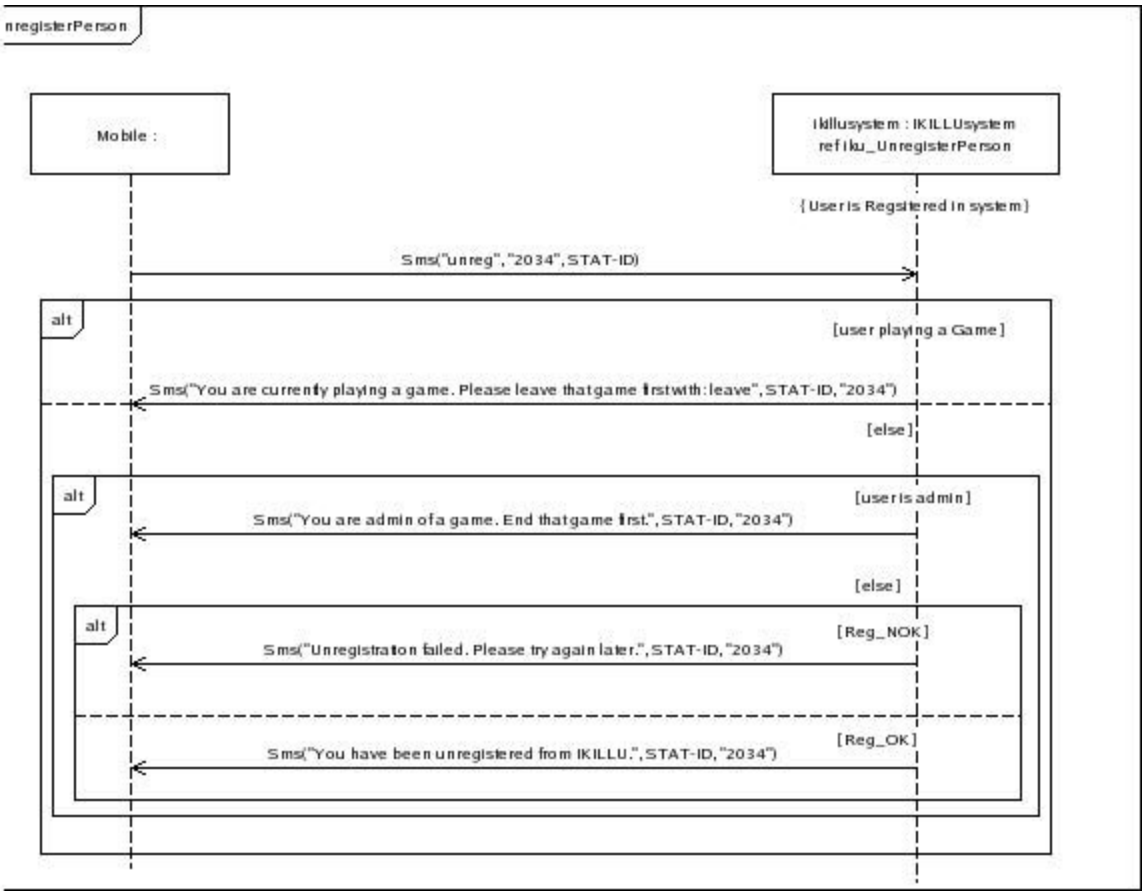
4.3.1 CreateGame



4.3.2 JoinGame



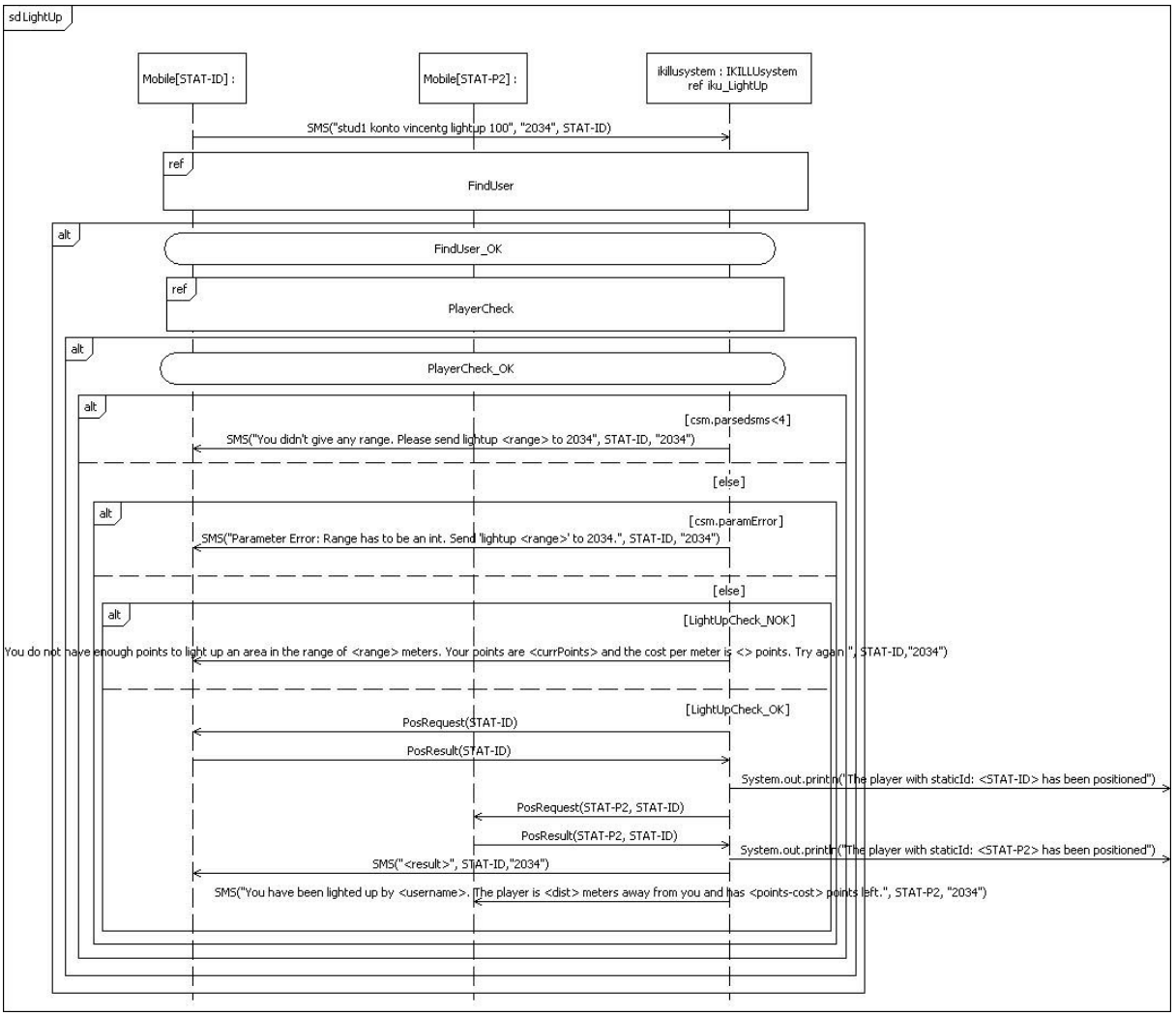
4.3.3 Unregister Person



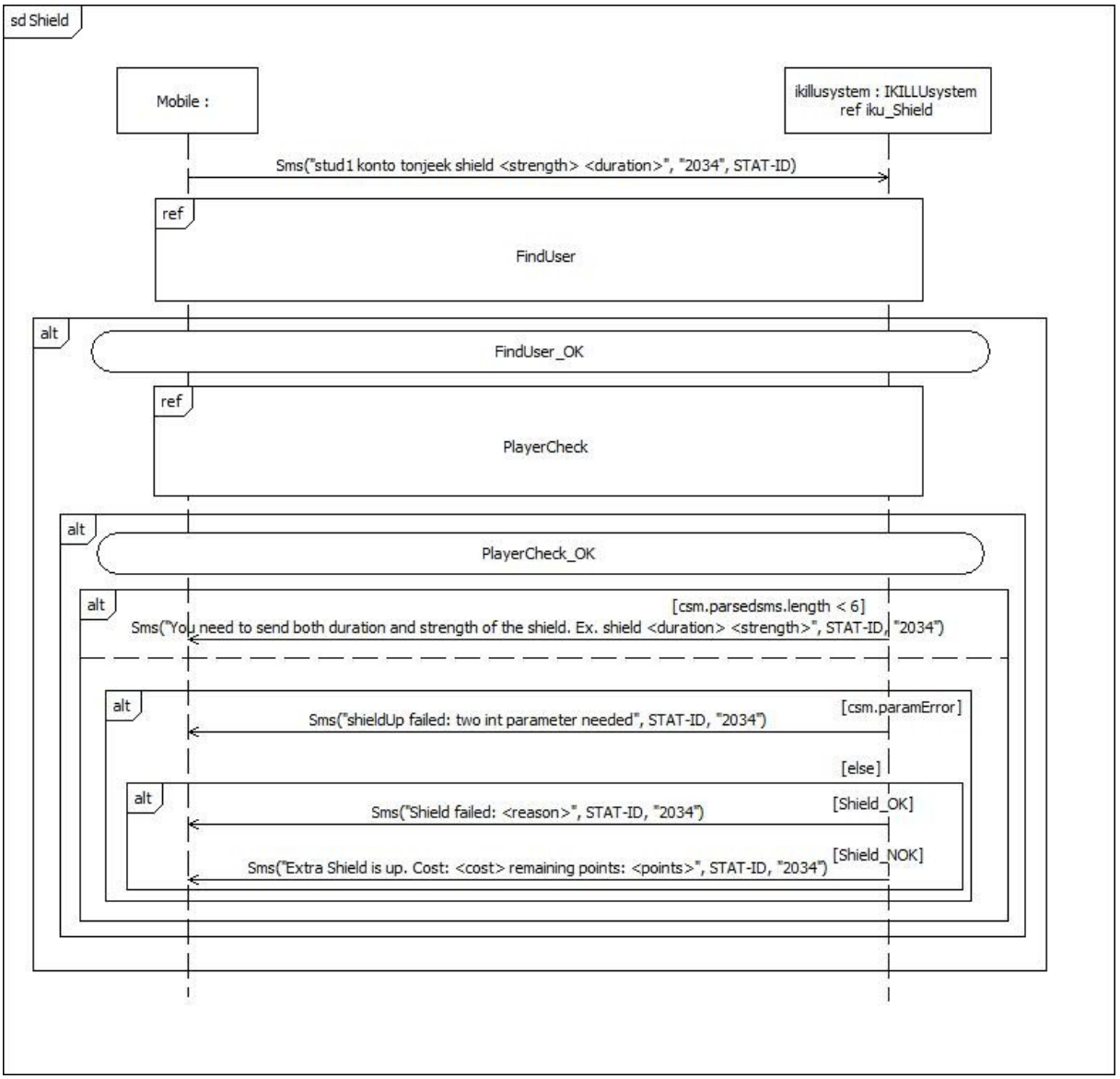
Figur 10 - Sequence diagram for UnregisterPerson

4.4 Game Services

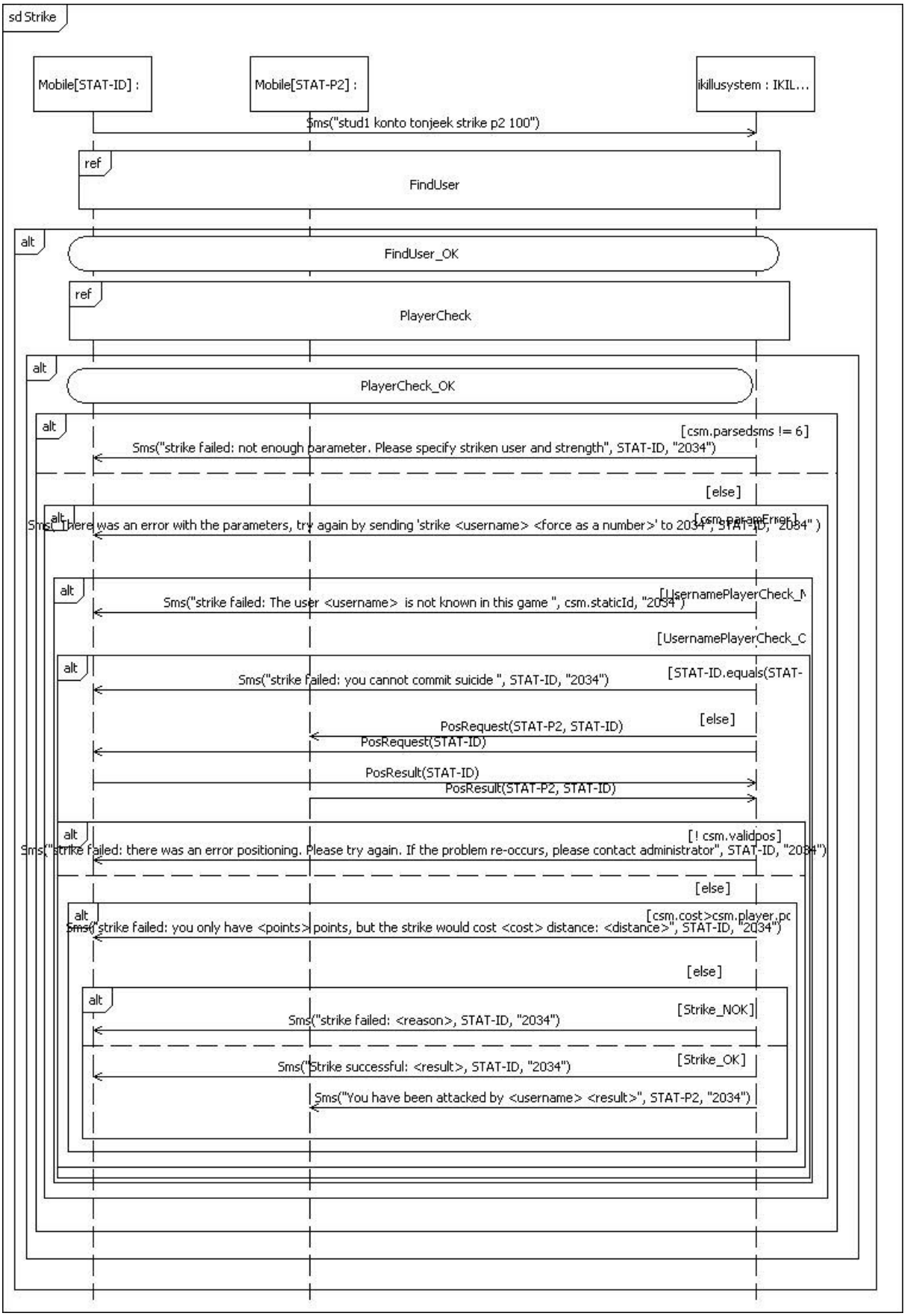
4.4.1 LightUp



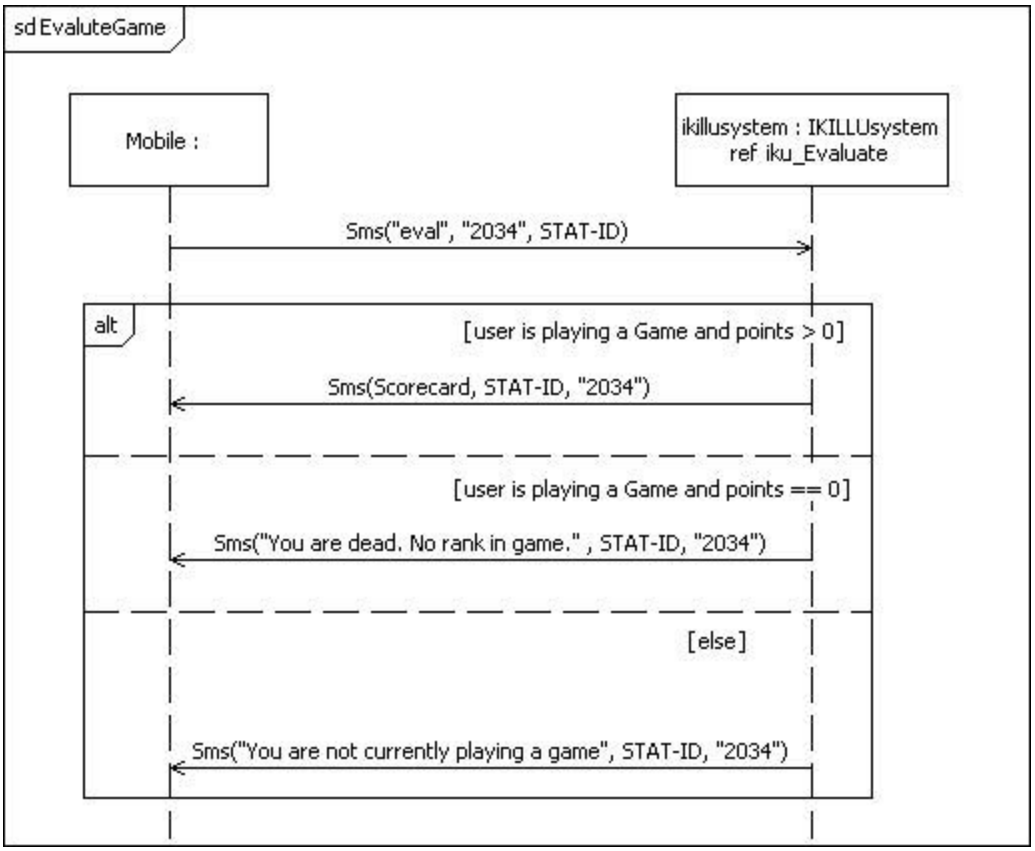
4.4.2 IncreaseShield



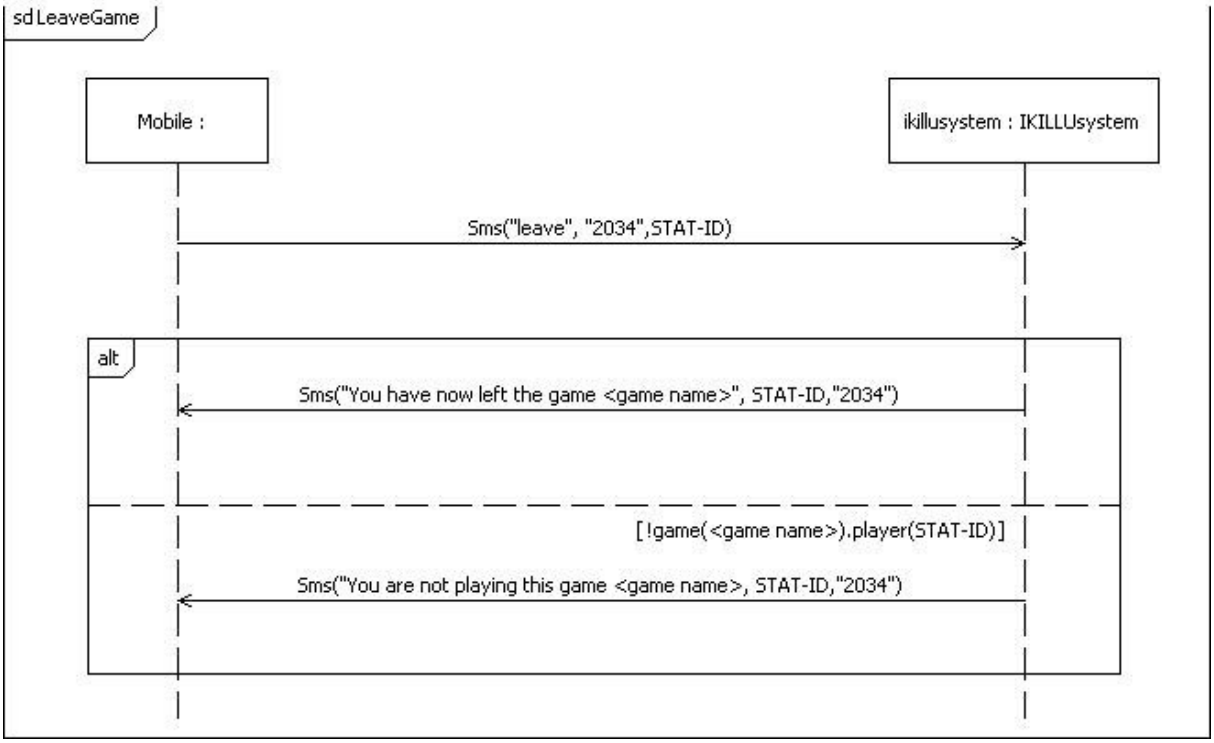
4.4.3 Strike


sd Strike
Mobile[STAT-ID] :
Mobile[STAT-P2] :
ikillusystem : IKIL...
Sms("stud1 konto tonjeek strike p2 100")
ref
FindUser
alt
FindUser_OK
ref
PlayerCheck
alt
PlayerCheck_OK
alt
[csm.parsedsms != 6]
Sms("strike failed: not enough parameter. Please specify striken user and strength", STAT-ID, "2034")
[else]
alt
[csm.paramError]
Sms("There was an error with the parameters, try again by sending 'strike <username> <force as a number>' to 2034", STAT-ID, "2034")
alt
[UsernamePlayerCheck_N
Sms("strike failed: The user <username> is not known in this game ", csm.staticId, "2034")
[UsernamePlayerCheck_C
alt
[STAT-ID.equals(STAT-
Sms("strike failed: you cannot commit suicide ", STAT-ID, "2034")
[else]
PosRequest(STAT-P2, STAT-ID)
PosRequest(STAT-ID)
PosResult(STAT-ID)
PosResult(STAT-P2, STAT-ID)
alt
[! csm.validpos]
Sms("strike failed: there was an error positioning. Please try again. If the problem re-occurs, please contact administrator", STAT-ID, "2034")
[else]
alt
[csm.cost>csm.player.pc
Sms("strike failed: you only have <points> points, but the strike would cost <cost> distance: <distance>", STAT-ID, "2034")
[else]
alt
[Strike_NOK]
Sms("strike failed: <reason>, STAT-ID, "2034")
[Strike_OK]
Sms("Strike successful: <result>, STAT-ID, "2034")
Sms("You have been attacked by <username> <result>", STAT-P2, "2034")

4.4.4 Evaluate



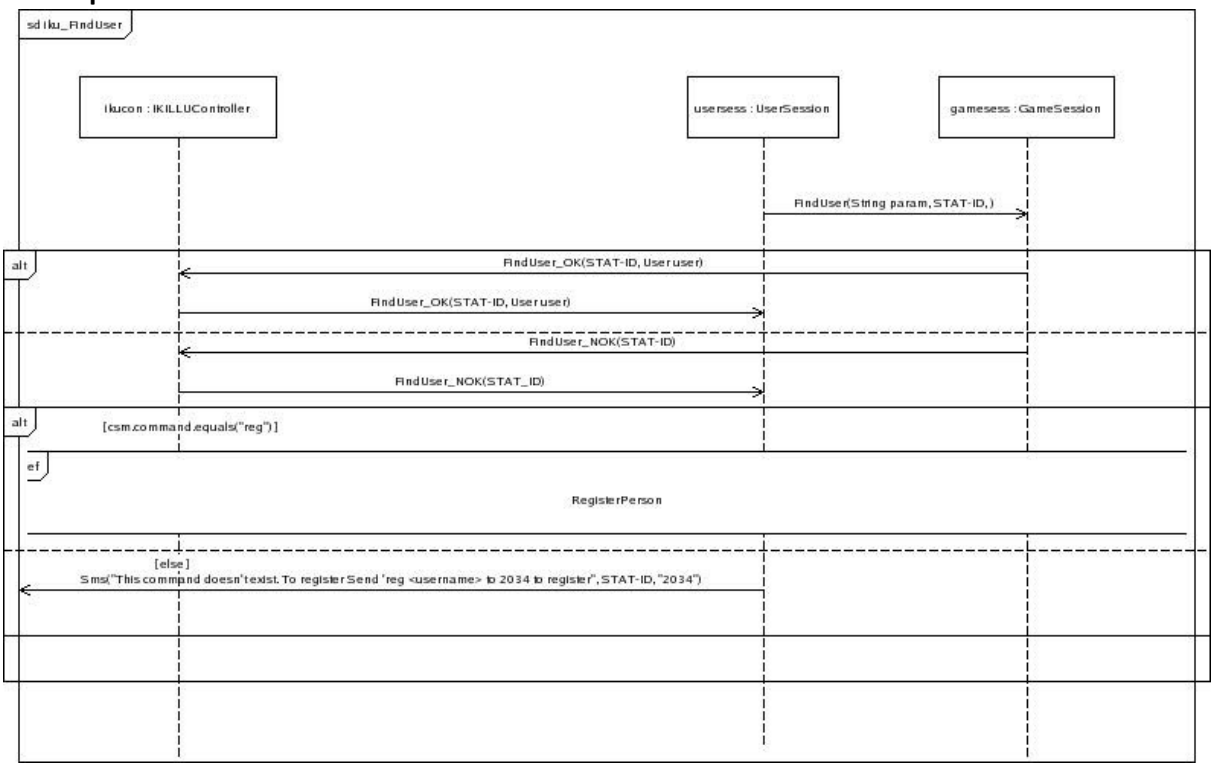
4.4.5 LeaveGame



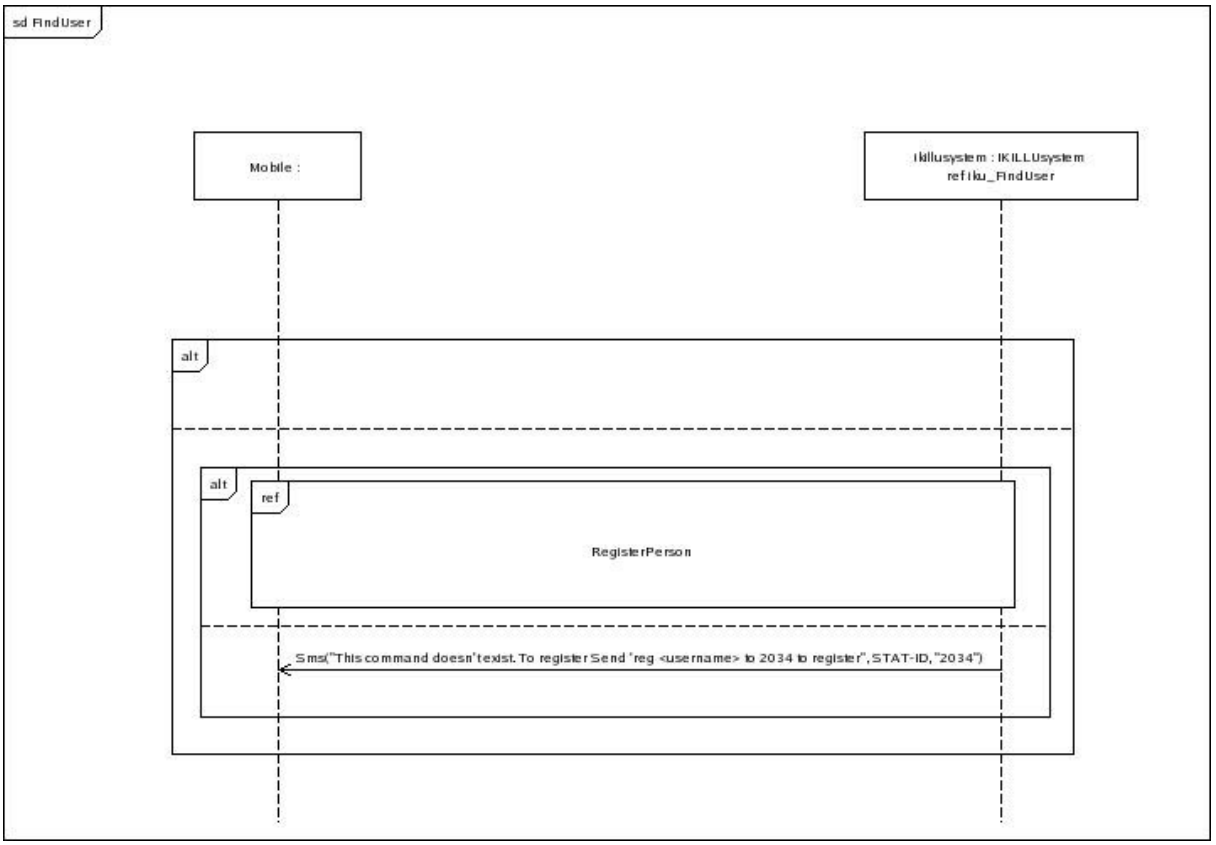
4.5 Referenced sequence diagrams

4.5.1 FindUser

Decomposed

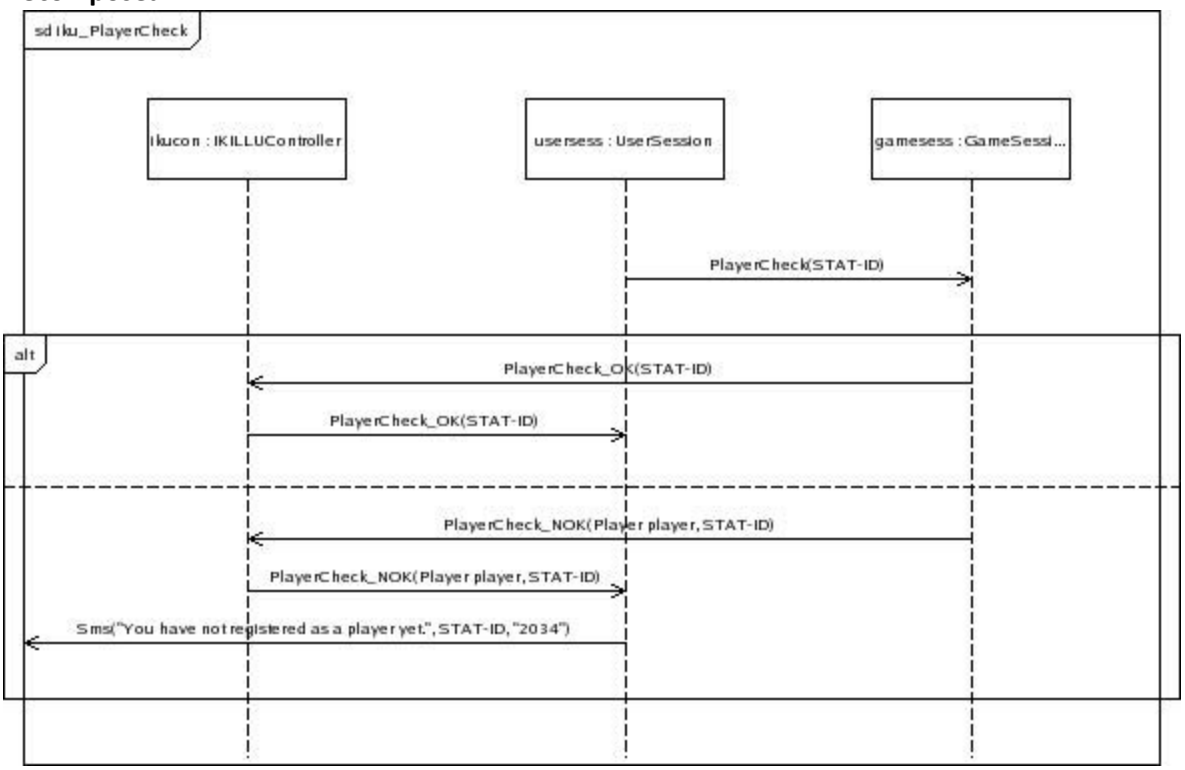


SD

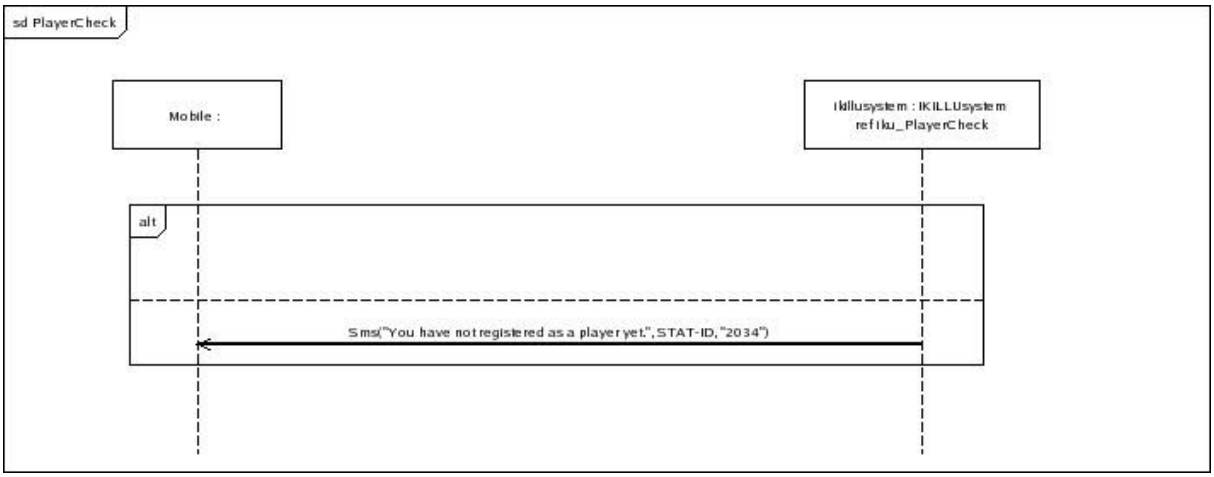


4.5.2 PlayerCheck

Decomposed



SD



Documentation of

CORAS

for the

iKILLUsystem

Autumn 2007

Contents of the CORAS report

Introduction

In this document we will conduct a security risk analysis for the IKILLUsystem. Through the course INF5150 we have been introduced to a method-based security analysis called CORAS. The CORAS method consists of seven steps, which we have tried to follow through out this document. The CORAS method also provides a tool do support documenting, maintaining and reporting analysis result. In our case we have used the tool do make the design of the stakeholder with our defined assets, draw the threat diagrams with different consequences and likelihoods for the assets. We also used it to draw the risk overview diagram and the risk treatments diagrams for the assets; user information, points and availability. By using the 'export as image' feature of the tool we have implemented the diagrams as shown below.

Security analysis

Step1 – Introduction

Goal of the analysis

Find the weaknesses of the target of analysis. Recommend treatments.

Target

Our main target will be focused towards the IKILLUsystem.

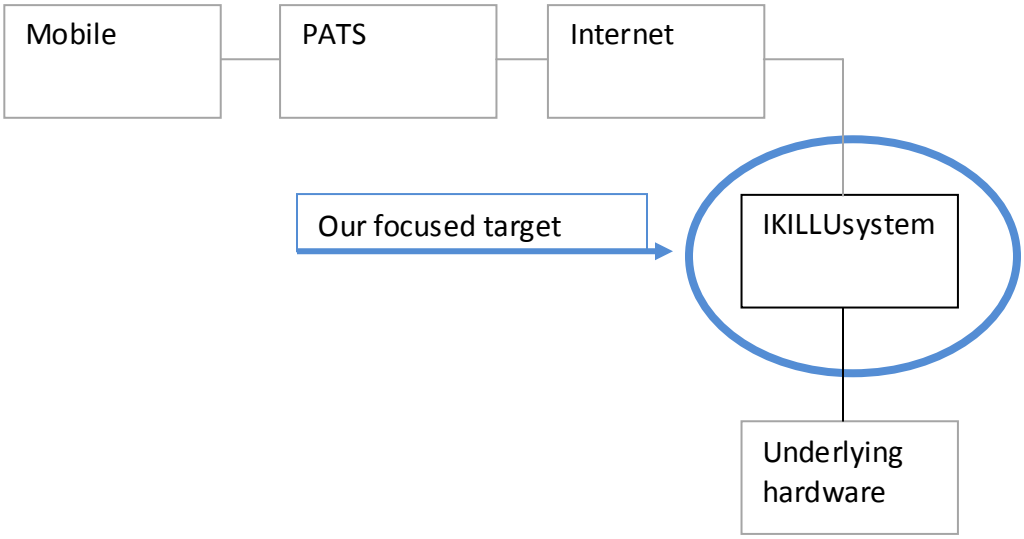


Figure: Illustrates where our main focus of the system will be.

Assumptions

We will in our security analysis concentrate on the IKILLUSystem.
The person who is the admin of the game is also the owner of the system and has a stake in the company profits.

Direct Assets descriptions

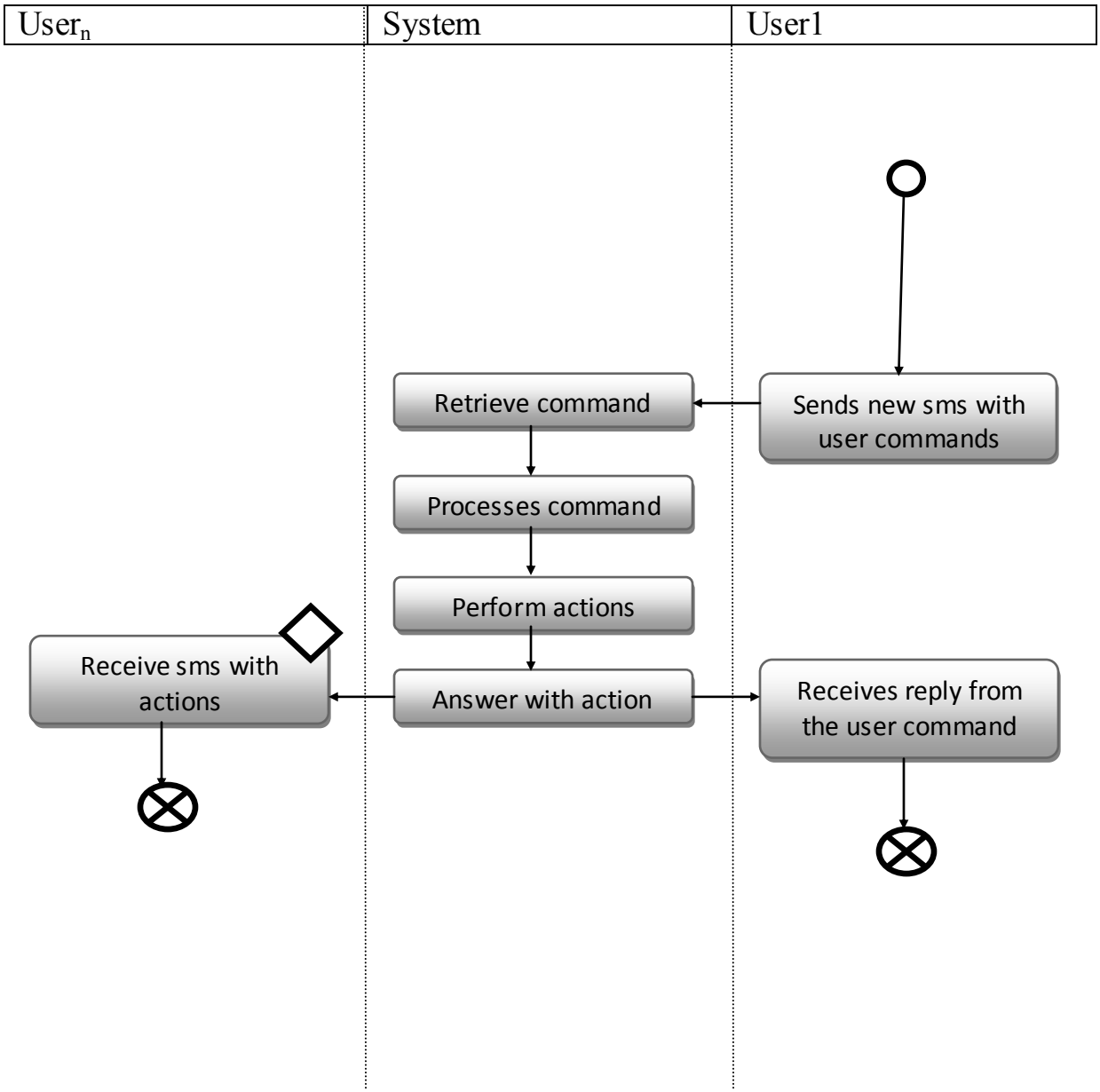
Asset #1: Availability: This asset is all about the system and external factors. It will be important for us considering all the sms's are being handled with the PATS system, witch is connected to the internet and such.
Asset #2: Points: This is a very important asset within our system. For instance, if some users could successfully manipulate other players' points in his/her advance, then this would be a serious issue for us, because the players are using real money.
Asset #3: User information: All the users data is important to maintain at all time. If some other user would get hand on a another players data, then this would also be another important issue for us.

Conventions

A **Person** is a person with a mobile.
A **User** is a registered Person.
A **Player** is a playing User.

Step2 – High level analysis

Activity diagram



Our diagram explained:

Action performed by a user/the system

Process is initiated

Process is terminated

Optional

Assets

Within the system

- User information
- Availability
- Points

Outside of system

- Public trust in system
- Company profit
- Player winnings
- Privacy

Asset Diagram

When we brainstormed through the different possible assets we came up with, we started to form an asset diagram. This diagram is describing the direct assets of the system, and the ones that are indirect. Our diagram ended up looking like this:

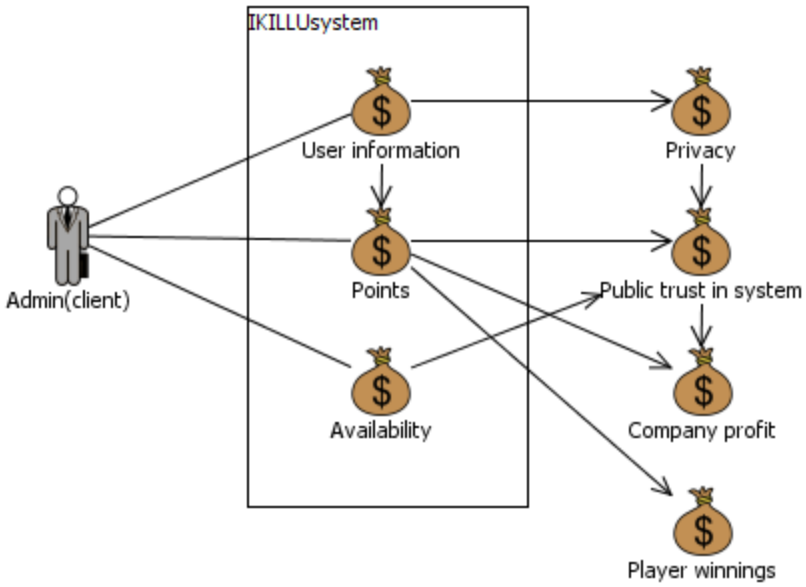
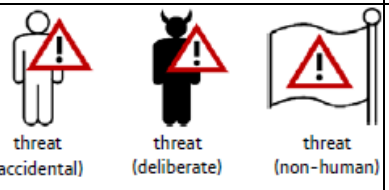


Figure: Shows the stakeholder with our defined assets, both the direct and indirect.

High-level risk table

This table is describing the different possible threats to our system. This was helpful for us, since it help defining what could be the incident, what may be harmed and what makes the incident possible.

Who/what causes it?	How? What is the incident? What does it harm?	What makes it possible?
		
User, Hacker	Someone not playing a game positions another registered user. Privacy is not maintained.	Bug in system.
System failure	System goes down, all game information is lost and user's points are lost.	Lack of persistent storage.
System failure, Network failure, Hacker	System stalls and does not give feedback. Availability is damaged.	System dependent on network availability. Bugs in the system.
Admin	Admin manipulates game. Harms Player winnings.	Admin has full access and control of the system.
External system, Network failure, Hacker	System is fed with wrong information by an external entity. Harms game information, availability.	External system lacks security, integrity or network errors can occur.
External system	External system changes interface. Changes are not compatible with the target system. Availability and trust decreases.	System is dependent on external service and changes are not communicated or acknowledged.
Hacker, Player, Thief	Game is manipulated to another user's benefit. Player accidentally finds bug. Hacker looks for bug. Thief manipulates game by using a player's stolen mobile. Points do not reflect the correct situation.	Bugs exist in the system. System does not authenticate person, only mobile (no password etc.)
System bug, Hacker, Admin	Confidential material is leaked because of system bugs, security breaches, Admin leaks KML-file. Harms privacy.	Bugs exist in system. Authentication is lacking. Admin has full access to the game information.

Step 3 - Approval

Asset table

This table is illustrating how we decided to rank the different assets. We rank them as it is suggested in CORAS with importance from 1 to 5, and the type as direct or indirect asset.

Asset	Importance	Type
Player winnings	1	Indirect
Company profit	1	Indirect
Privacy	1	Indirect
User information	2	Direct
Points	1	Direct
Availability	2	Direct
Public trust in system	(Scoped out)	Indirect

Risk Evaluation Criteria

Here we will try to evaluate different consequences and frequencies concerning the direct assets we've illustrated in the asset table.

Consequence scale: User information

Consequence value:	Description:
Catastrophic	16% - 100 % All information is lost or manipulated
Major	6% - 15% A lot of information is lost or manipulated
Moderate	2% - 5% Some information is manipulated or lost
Minor	0% - 1% A small amount of information is manipulated or lost
Insignificant	No information is affected

Likelihood scale: User information

Likelihood value:	Description:
Often	More than 16 times per year
Sometimes	From 6 – 15 times per year
Seldom	From 1 - 5 times per year
Never	Zero times per year

Risc evaluation matrix: User information

	Insignificant	Minor	Moderate	Major	Catastrophic
Never	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable
Seldom	Acceptable	Acceptable	Evaluation Required	Evaluation Required	Evaluation Required
Sometimes	Acceptable	Evaluation Required	Evaluation Required	Evaluation Required	Evaluation Required
Often	Acceptable	Evaluation Required	Evaluation Required	Evaluation Required	Evaluation Required

Consequence scale: Points

Consequence value:	Description:
Catastrophic	51% - 100 % All of the players current points has been lost
Major	21% - 50% A lot of the players current points has been lost
Moderate	6% - 20% Some of the players current points has been lost
Minor	1% - 5% A small amount of the players current points has been lost
Insignificant	None of the players current points has been lost

Likelihood scale: Points

Likelihood value:	Description:
Often	More than 25 times per year
Sometimes	From 11 – 24 times per year
Seldom	From 5 - 10 times per year
Rarely	From 1 – 4 times per year
Never	Zero times per year

Risk evaluation matrix: Points

	Insignificant	Minor	Moderate	Major	Catastrophic
Never	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable
Rarely	Acceptable	Should be evaluated	Should be evaluated	Evaluation Required	Evaluation Required
Seldom	Acceptable	Should be evaluated	Evaluation Required	Evaluation Required	Evaluation Required
Sometimes	Acceptable	Evaluation Required	Evaluation Required	Evaluation Required	Evaluation Required
Often	Acceptable	Evaluation Required	Evaluation Required	Evaluation Required	Evaluation Required

Consequence scale: Availability

Consequence value:	Description:
Catastrophic	100+ users affected
Major	41 – 99 users affected
Moderate	21 – 40 users affected
Minor	10 – 20 user affected
Insignificant	1 – 9 users affected

Likelihood scale: Availability

Likelihood value:	Description:
Often	More than 100 times per year
Sometimes	From 30 – 99 times per year
Seldom	From 11 - 29 times per year
Rarely	From 1 – 10 times per year
Never	0 times per year

Risk evaluation matrix: Availability

	Insignificant	Minor	Moderate	Major	Catastrophic
Never	Acceptable	Acceptable	Acceptable	Acceptable	Acceptable
Rarely	Acceptable	Should be evaluated	Should be evaluated	Evaluation Required	Evaluation Required
Seldom	Acceptable	Should be evaluated	Evaluation Required	Evaluation Required	Evaluation Required
Sometimes	Should be evaluated	Evaluation Required	Evaluation Required	Evaluation Required	Evaluation Required
Often	Evaluation Required	Evaluation Required	Evaluation Required	Evaluation Required	Evaluation Required

Step 4 – Risk identification

Within this step we will have a short brainstorm on different threat scenarios that may happen. These scenarios will help us create different diagrams that we will create with respect to the different assets and threats we have identified earlier.

Initial threat diagram

These diagrams are modeled like this:
We start with displaying the possible threats, then the vulnerably, scenario, unwanted incidents then the assets(direct/indirect).

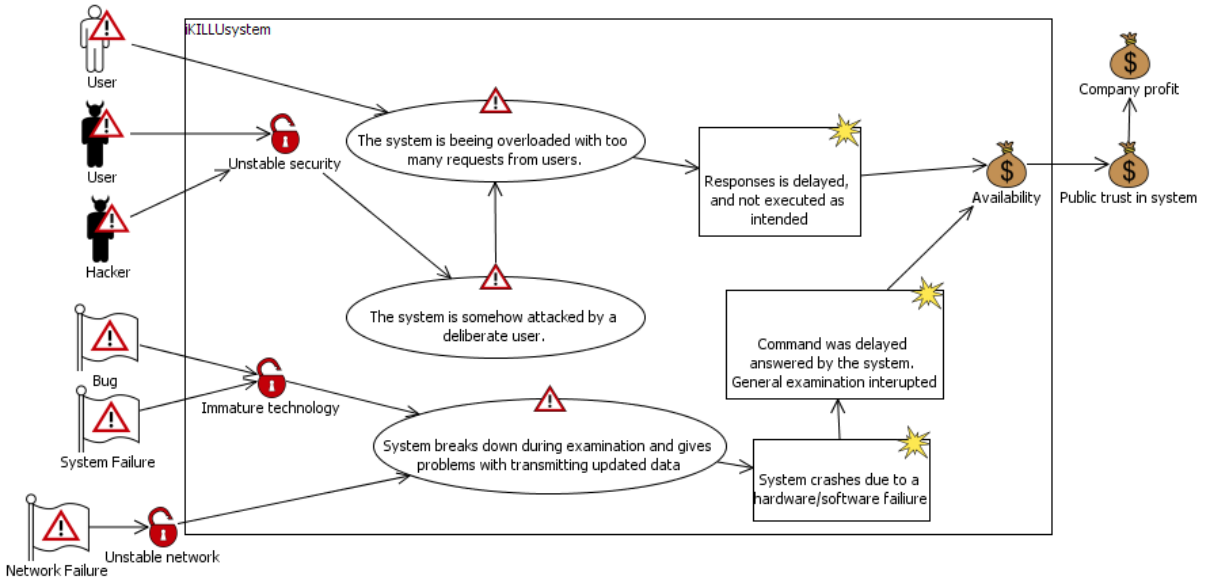


Figure: This diagram illustrates the threat diagram of the asset "*Availability*" with the indirect asset Public trust in system and company profit.

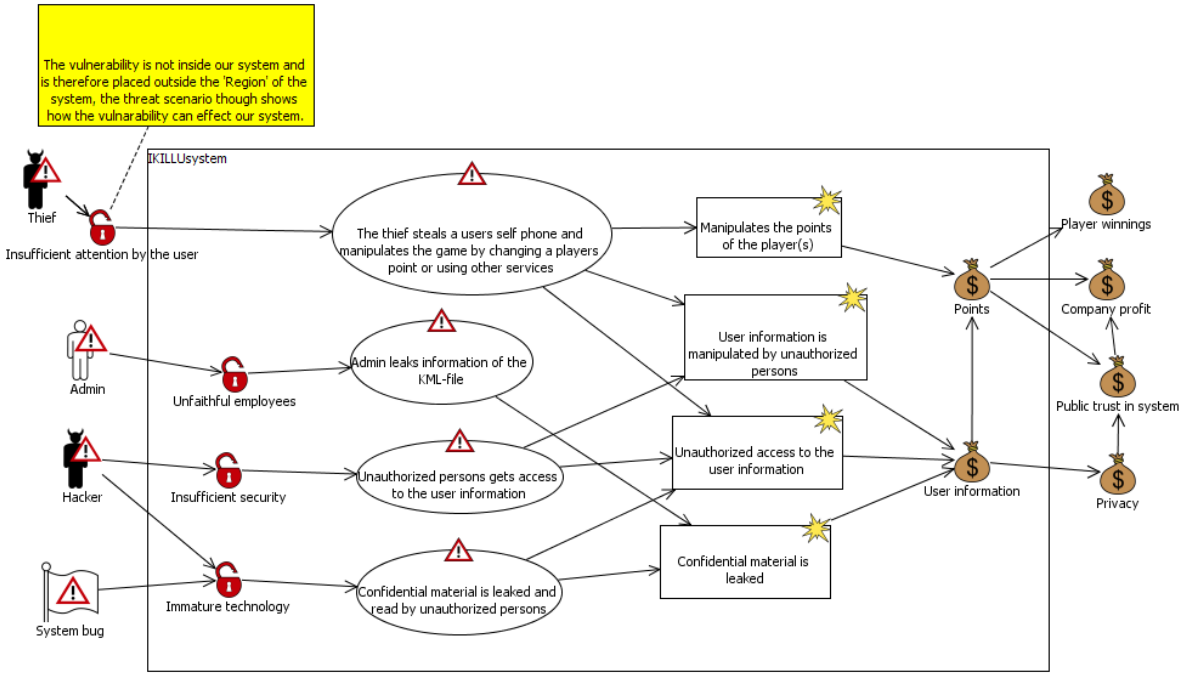


Figure: This diagram illustrates the threat diagram of the asset "*User information*" and "*Points*" with the indirect asset "*Player winnings*", "*Company profit*", "*Public trust in system*" and "*Privacy*".

Step 5 – Risk estimation

When the previous parts are described in different diagrams, it's time to estimate likelihood values and consequences. This will help us to compute risk values, and give us an overview of

the threats that can be considered not important, or threats that we should considered for evaluating.

Threat diagram with likelihood and consequence

Here we will define the consequences and the likelihoods of the threats.

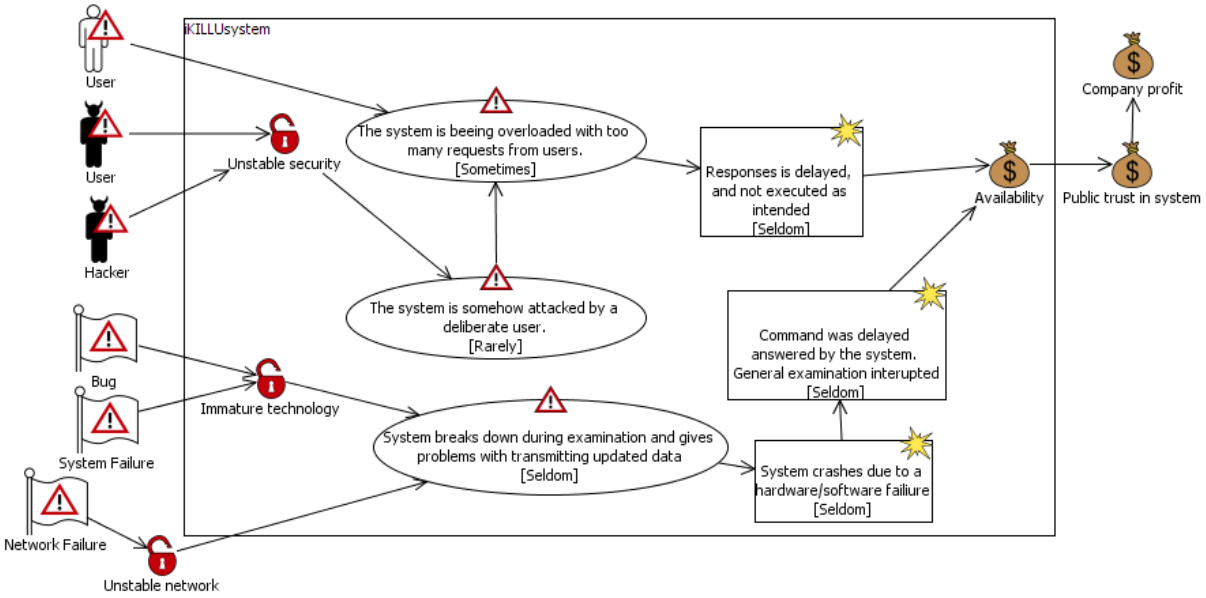


Figure: This threat diagram is now shown with the different consequences and likelihoods for the asset "*Availability*" with the indirect assets "*Public trust in system*" and "*Company profit*".

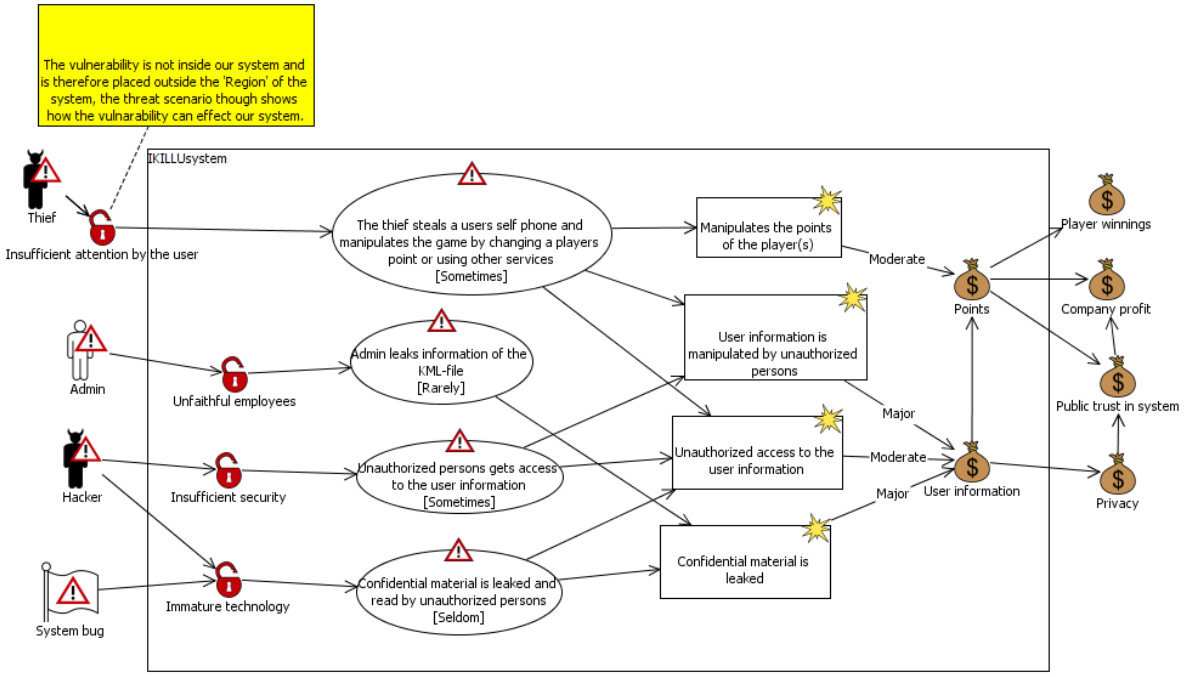


Figure: This threat diagram is now shown with the different consequences and likelihoods for the assets "*User information*" and "*Points*" with the indirect assets "*Player winnings*", "*Company profit*", "*Public trust in system*" and "*Privacy*".

Estimations

Now we will try to estimate the risk. These estimations is grouped in two different categories, likelihood and consequence.

Threat scenario	Likelihood	Unwanted incident	Combine likelihood
System breaks down during examination and gives unwanted problems with transmitting updated data	Seldom	Request is not answered by the system. General examination interrupted. (AR1)	Seldom

Table: Describes the combined likelihood estimates for whether request is not answered.

Threat scenario	Likelihood	Unwanted incident	Combine likelihood
The system is being overloaded with too many requests from the users	Sometimes	Responses is delayed and not executed as intended. (AR2)	Seldom
The system is somehow attacked by a deliberate user	Rarely		

Table: Describes the combined likelihood estimates if the responses are not executed.

Threat scenario	Likelihood	Unwanted incident	Combine likelihood
The thief steals a users self phone and manipulates the game by changing a players point or using other services	Sometimes	Manipulates the points of the player(s) (P1)	Seldom

Table: Describes the combined likelihood estimates for: Manipulates the points of the player(s)

Threat scenario	Likelihood	Unwanted incident	Combine likelihood
The thief steals a users self phone and manipulates the game by changing a players point or using other services	Sometimes	User information is manipulated by unauthorized persons (UI1)	Sometimes
Unauthorized persons gets access to the user information	Sometimes		

Table: Describes the combined likelihood estimates for: User information is manipulated by unauthorized persons

Threat scenario	**Likelihood**	**Unwanted incident**	**Combine likelihood**
Unauthorized persons gets access to the user information	Sometimes	Unauthorized access to the user information (UI2)	Seldom
Confidential material is leaked and read by unauthorized persons	Seldom		
The thief steals a users self phone and manipulates the game by changing a players point or using other services	Sometimes		

Table: Describes the combined likelihood estimates for: Unauthorized access to the user information

Threat scenario	**Likelihood**	**Unwanted incident**	**Combine likelihood**
Confidential material is leaked and read by unauthorized persons	Seldom	Confidential material is leaked (UI3)	Seldom

Table: Describes the combined likelihood estimates for: Confidential material is leaked

Step 6 – Risk evaluation

Within this step we are supposed to make a risk evaluation of the risk estimation. These risks will then be shown in several matrices tables.

Matrices tables

The risk that needs to be evaluated is presented in light blue, while the unpainted ones are acceptable.

User information risks

		Consequence				
		Insignificant	**Minor**	**Moderate**	**Major**	**Catastrophic**
Likelihood	**Never**					
	Rarely					
	Seldom				UI1, UI3	
	Sometimes			UI2		
	Often					

Points risks

		Consequence				
		Insignificant	**Minor**	**Moderate**	**Major**	**Catastrophic**
Likelihood	**Never**					
	Rarely					
	Seldom			P1		
	Sometimes					
	Often					

Availability risks

		Consequence				
		Insignificant	**Minor**	**Moderate**	**Major**	**Catastrophic**
Likelihood	**Never**					
	Rarely					
	Seldom		AR2			
	Sometimes			AR1		
	Often					

Our risk overview diagram

In this diagram we are going to present all the possible threats within the system. This is for helping the costumer to understand witch assets should be taken in consideration for treatment.

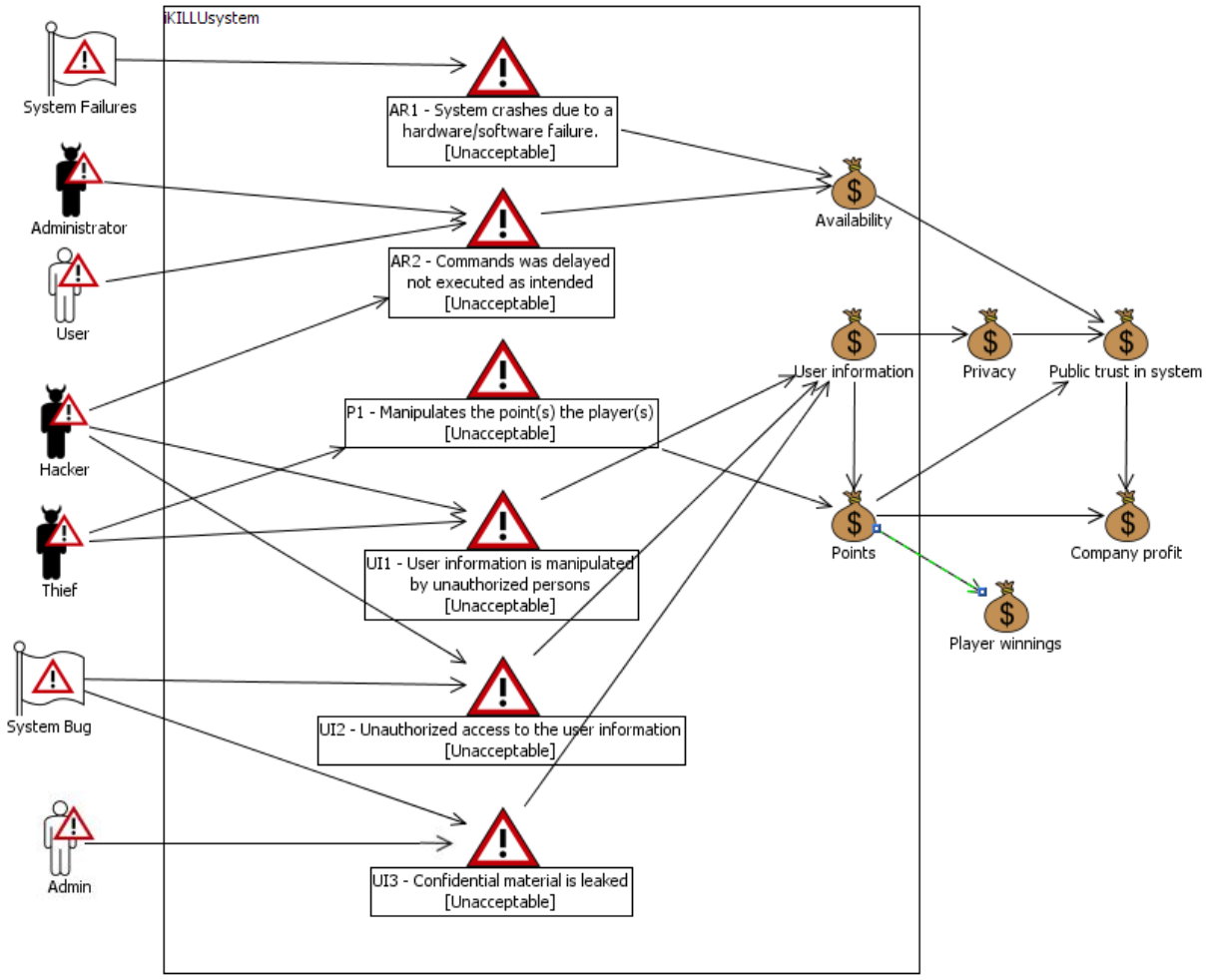


Figure: Risk overview diagram of all the possible threats towards the assets.

Step 7 – Risk treatment

In this final step we are going to suggest a couple of treatments for the threats that is threatening our system assets. In order to decrease these threats, some treatments are needed.

Risk treatment diagrams

At this point we have identified the possible risks, so we can now start working on possible treatments for our system. We would like to make clear that some of the possible treatments cannot be done by us since they are of an external factor. The external factors will then be the ISP, general network problems and similar issues.

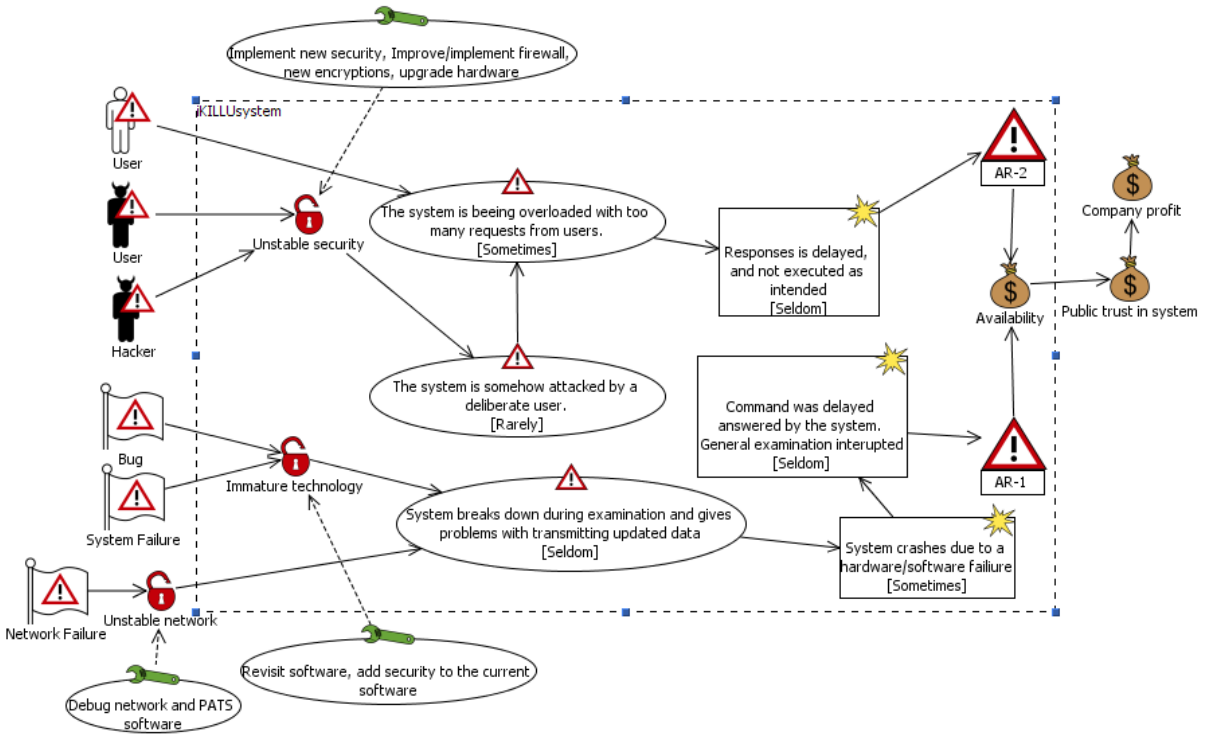


Figure: Our risk treatments for the "*Availability*" threat diagram.

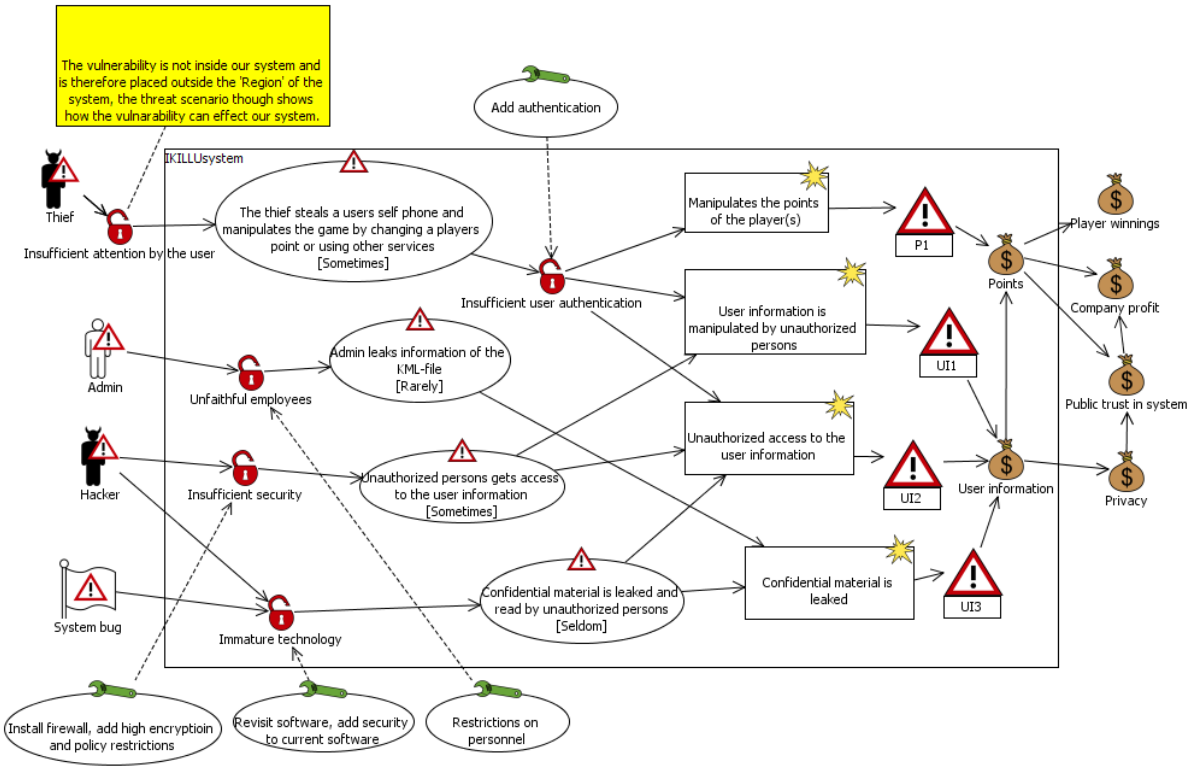


Figure: Our risk treatments for the "*User information*" and "*Points*" threat diagram.

Conclusion

As mentioned in the introduction we have conducted a security analysis for the IKILLUsystem. Through the seven steps in the CORAS method we revealed several vulnerabilities and made suggestions with treatments for the actual vulnerabilities.