INF-5150 Obligatory excercise 2

“Survival of the SMSest”

November 2007

GROUP 2:
Jonas Winje, Aida Omerovic,
Christian Rudfoss, Shaozhi Yang

Table of Contents

INF-5150 Obligatory excercise 2....1
“Survival of the SMSest”....1
1. Instructions for use....3
2. System design: Structure diagrams....5
3. System design: Use case and sequence diagrams....8
3.1 Register for system....9
3.2 Announce game....10
3.3 Registering for a game....11
3.4 Starting a game....13
3.5 Setting up a shield....14
3.6 Light up players....16
3.7 Striking a player....18
3.8 Get status report....21
3.9 Write KML-file....22
3.10 Get position....22
4. System design: State machine diagrams and signals....24
5. “Survival of the SMSest” - Security Risk Analysis....34
5. 1 Target description....34
5.2 Asset identification....34
5.3 Risk identification and estimation....38
5.4 Risk evaluation....40
5.5 Risk overview....42
5.6 Risk treatment....43
References....46

1. Instructions for use

When interacting with the system SMSes are exchanged according to a predefined template, "PXIFI KONTO <phonenumber> <message>" where <message> is the text specifying the commands below.

Players

To register for the system, send "register <username>" where <username> is your prefered nickname. You'll get a reply letting you know if the registration was successful. You can only register once, and you must register for the system before joining a game.

You'll be invited to a game when the admin announces a game. To join the game, you must send "joingame" before the admin starts the game. You will get a reply if you have successfully join the game.

When the admin starts a game you've joined, you will get a message letting you know the game has started along with one with status (points, shield strength and duration).

When you're in a game, you can send the messages "getstatus", "shield", "lightup" and "strike".

Send "getstatus" to get a reply with your current status.

Send "shield <force> <duration>" to attempt to set up a shield. <force> is the strength of the shield and <duration> is the duration of the shield in minutes. The shield strength will decrease whenever another player strikes you and the shield will dissappear after <duration> minutes. The shield costs <force> x <duration> points, and if you cannot afford the shield you're trying to set up, you will recieve a message letting you know how many points you have left. If you already have an active shield, you will get a message with the old shield's strength and duration and you will have to reply with "throw" to throw the old shield away and use the new one or "keep" to keep using the old shield.

Send "lightup <range>" to get a list of players within "a range", where <range> indicates the number of hundreds of meters away. Lightup costs <range>^2 points. Players spotted in a lightup will get a message telling them they've been spotted, but not who has spotted them.

Send "strike <target> <force> <range>" where <target> is the nickname of another player, <force> is the strength of the strike and <range> is the striking distance in hundreds of meters. Strike costs <force> x <range> points. If the player you're trying to strike is within range you will hit them, and you will get a message confirming the hit. That player's shield strength is then reduced by <force>. If this reduces the shield strength to below zero, or the player has no active shield, the player is hit, and you will get a message confirming this. You will then recieve that players points. After attempting to strike a player or recieving a player's points you will get a message with your status.

If you are hit or killed by another player, you will recieve a message letting you know of this and a message with your status. You will not know which player hit or killed you.

Admin

Note that the system has only been tested properly with FakePATS and that the static ID used for the admin is currently «admin» (this can be changed in the model).

To start a new game, the admin must first announce the game by sending "announce". Players registered in the system will then be invited to the game.

After sending "announce" the admin can start the game by sending "start". If more than one player has joined the game the game starts and the players that have joined the game will get a message saying the game has started and a message with their status. If one or no players have joined the game the game will end and the admin will have to announce a new game.

When a game is running the admin can send "kml" to make the system write all the player positions to a file, file "players.kml", that can be viewed in GoogleEarth.

2. System design: Structure diagrams

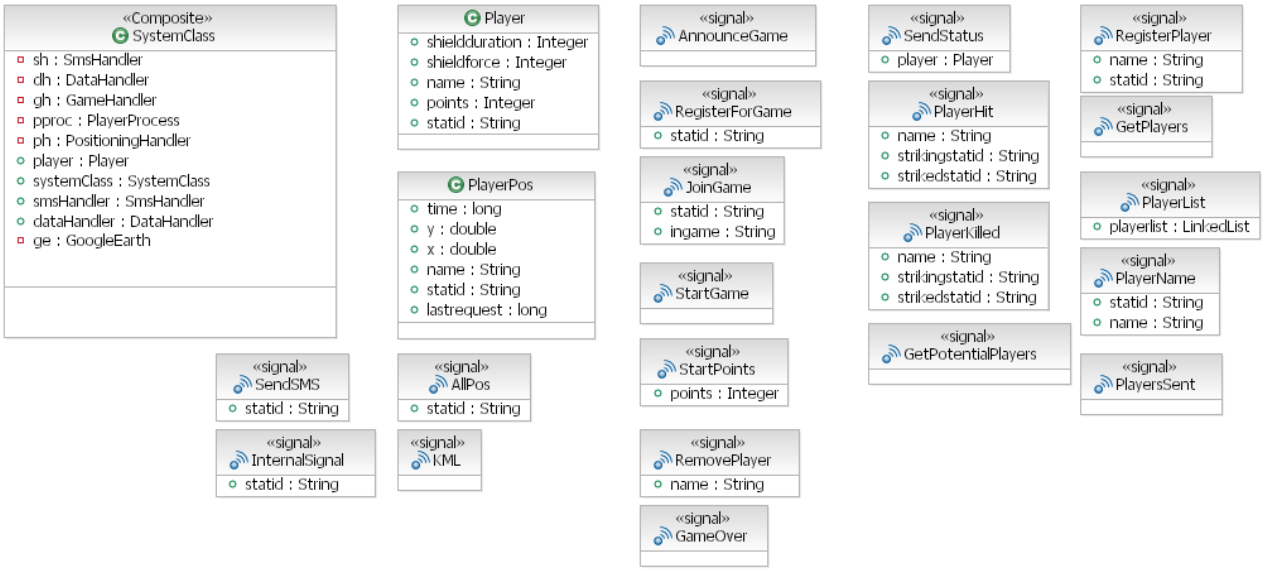


Figure 2.1

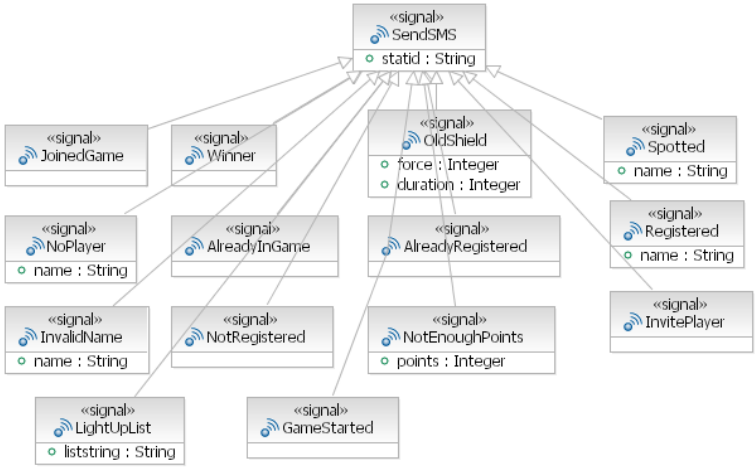


Figure 2.2

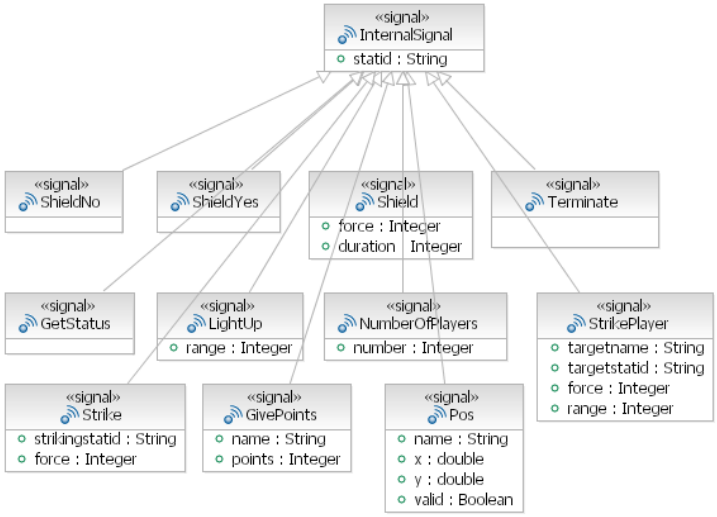


Figure 2.3

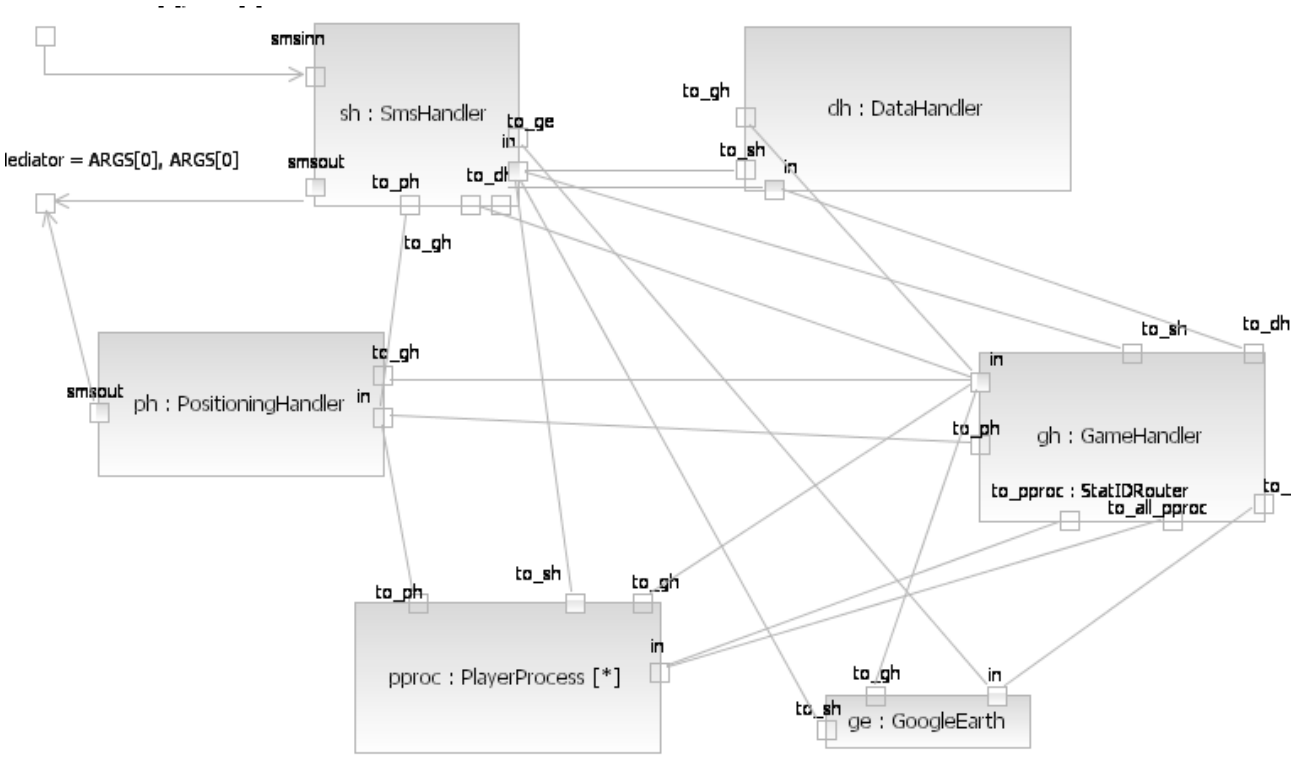


Figure 2.4

```
create table Player (
  statid varchar(8) not null,
  username varchar(20),
  inGame varchar (1),
  primary key (statid)
);
```

Figure 2.5

The system uses a database with a «player» table (as shown in figure 2.5) for a list of players registered for the system, and keeps track of which game, if any, each player is in (the system currently supports only one game at a time). Playerdata for the players in the game (points, and shield strength and duration) is not kept in the database, but in objects in player processes.

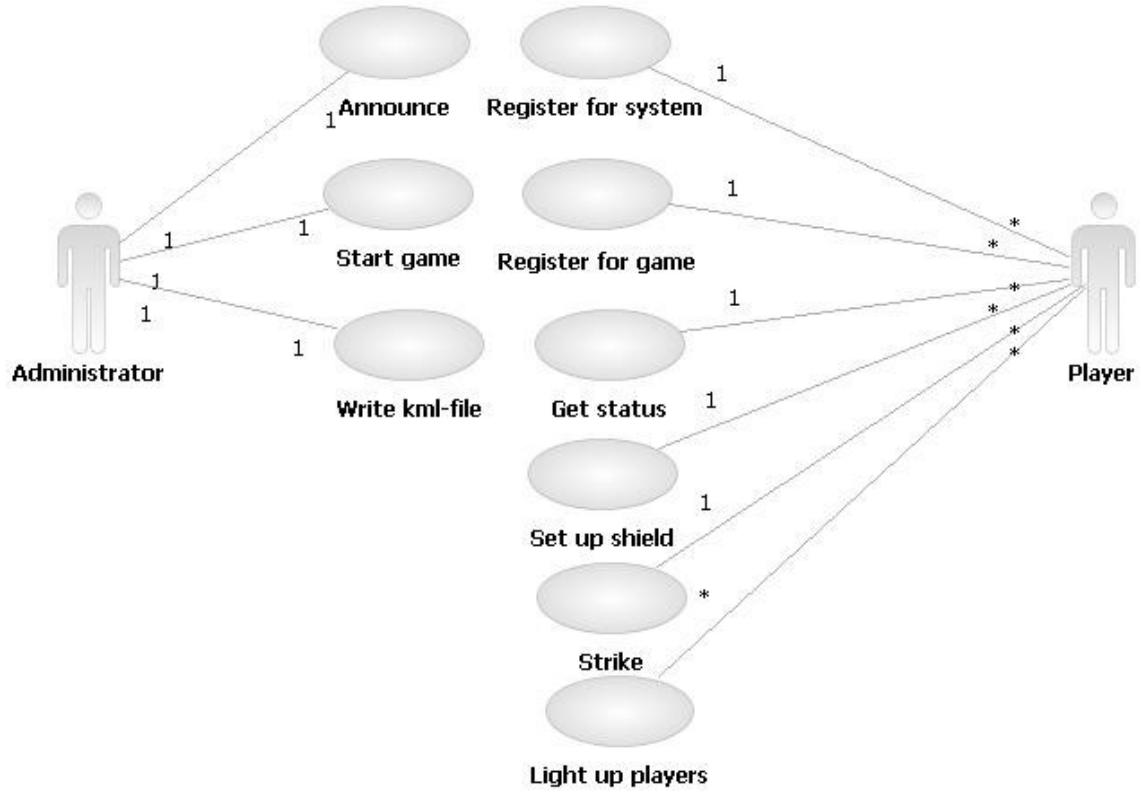
Only registered players are eligible to join a game. Static ID is the unique identifier of a player, and nicknames can always be traced back to the player's static ID. Intergrity is ensured by storing the user date in a databaes which only the system administrator has access to. Messages sent to players never contain static IDs.

The admin functions are initiated by sending SMSes to the system. The system currently supports having one admin, and the static ID of the admin is hardcoded. The admin is not able to register for the system or join games.

The system creates one process for every player in the game when a game starts. Each of these player processes terminate when its player is killed or the game is over. No processes are created while the game is running. This means that each player can only perform one action at a time. If a player does a lightup, and then immediately tries to strike another player, the strike action won't be performed until the lightup is done. The player process always takes care of other players striking its player and the timer signals at once, so a player can't avoid taking damage or delay the shield running out by being in the middle of actions. Since our strike action does not have a duration, no player action will ever take too long to complete, and it shouldn't be a problem that a player can't do everything at the same time. If strike had had a duration, depending on how long durations we supported and how risky we wanted striking to be, it might have been meaningful to support several strikes from one player at once. This could be done by using a new strike statemachine that we could create instances of whenever someone tried to strike.

The system checks incoming messages and lets the sender know when they are not syntactically correct. Messages sent when they shouldn't be ('strike' when there is no game running and the likes) are handled corectly by the system (ignored, that is), but in such cases the senders do not get any messages back.

3. System design: Use case and sequence diagrams



The figure above is a use case diagram, and shows the relation between the roles and actions for the system. An administrator may announce a game, start a game or write a kml-file. A player may register for the system, register for a game, get his/her status in the game, set up a shield while playing, strike another player while playing, and light up other players while playing.

The use case above is more closely described by the following sequence diagrams. Each sequence diagram has two levels, which also means that they are decomposed in the same way. It is the lifeline “SystemClass” that is decomposed into all, or some of these lifelines: “SmsHandler”, GameHandler”, “PlayerProcess”, “PositioningHandler”. For the sake of clarity we will show all the sequence diagrams at both levels.

3.1 Register for system

This sequence diagram shows how to register for the system. A player can not register for a system if he/she already is registered, or if the username is taken. Figure 3.1.1 is a high level representation of this functionality.

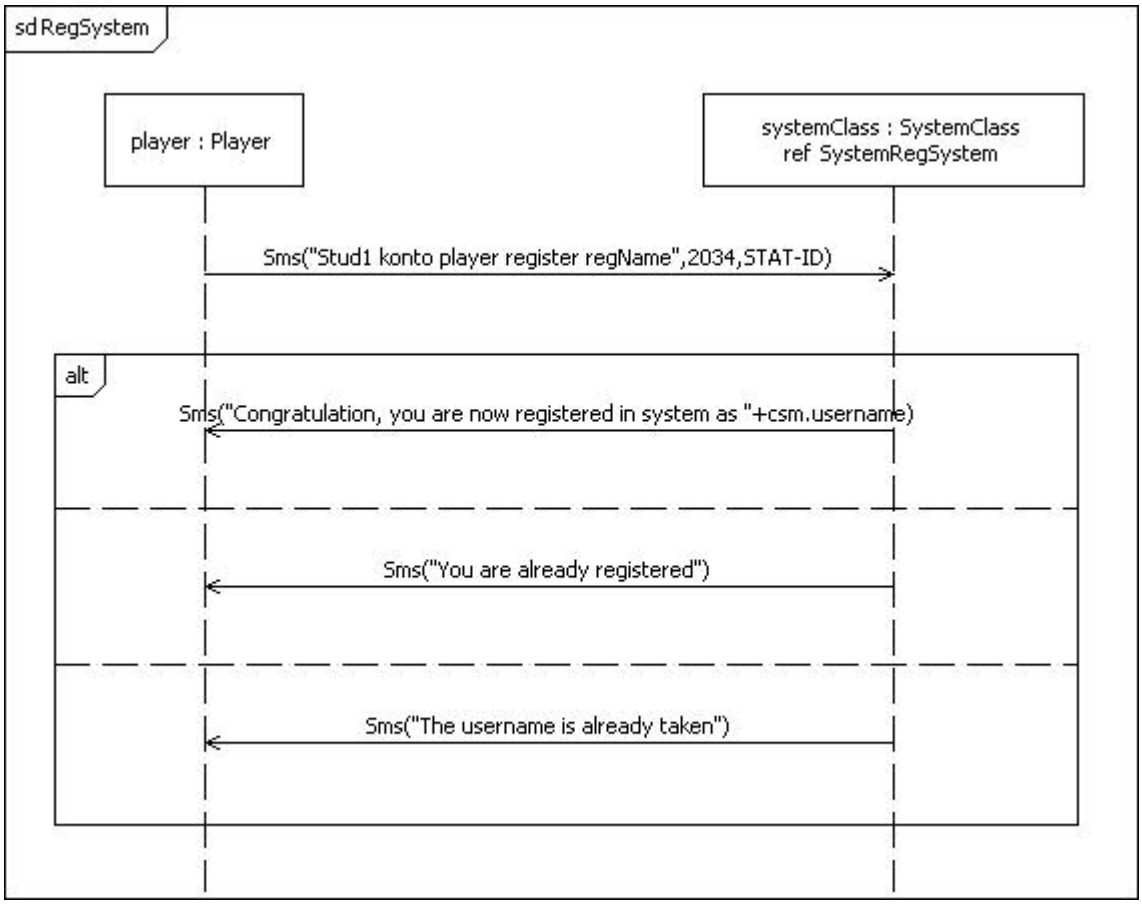


Figure 3.1.1

Lifeline SystemClass refers to SystemRegSystem. Below is the next level of this sequence diagram (figure 3.1.2). It shows what happens inside SystemClass when it gets a message. When a player sends a message to the system, the receiver is SmsHandler. It is always the SmsHandler that is the receiver, which can be seen at the rest of the sequence diagrams. In this case, SmsHandler communicates with Datahandler.

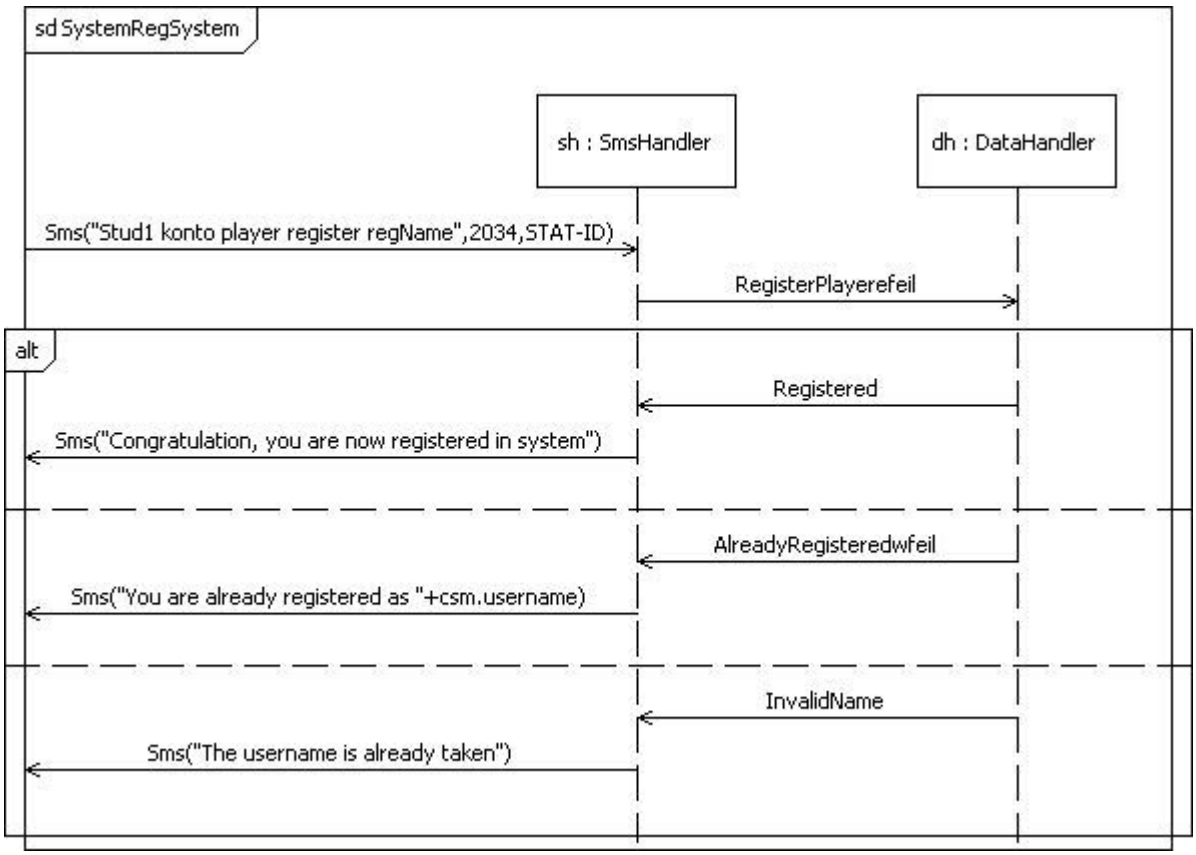


Figure 3.1.2

3.2 Announce game

Administrator sends a command to the system, telling it to send an invitation to all players that are registered for the system containing that they are invited to play a game.

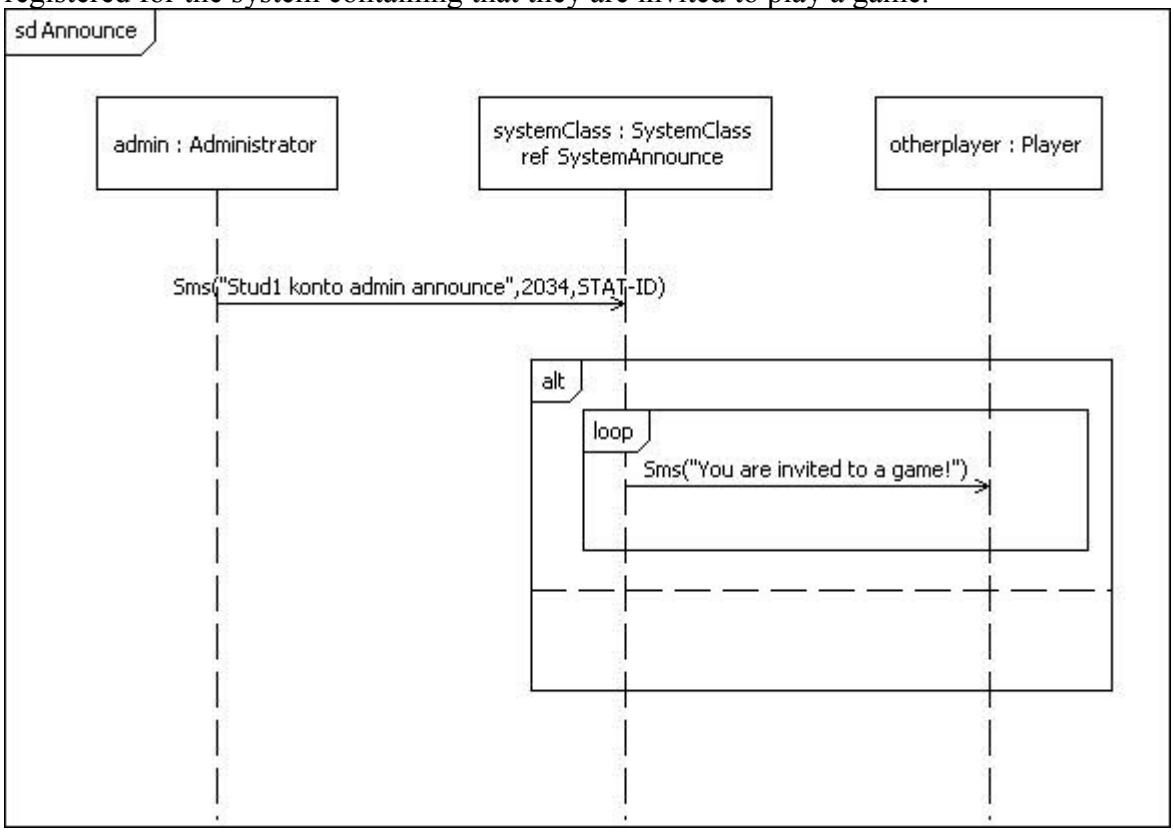


Figure 3.2.1

What happens inside SystemClass is shown by figure 3.2.2 at level 2:

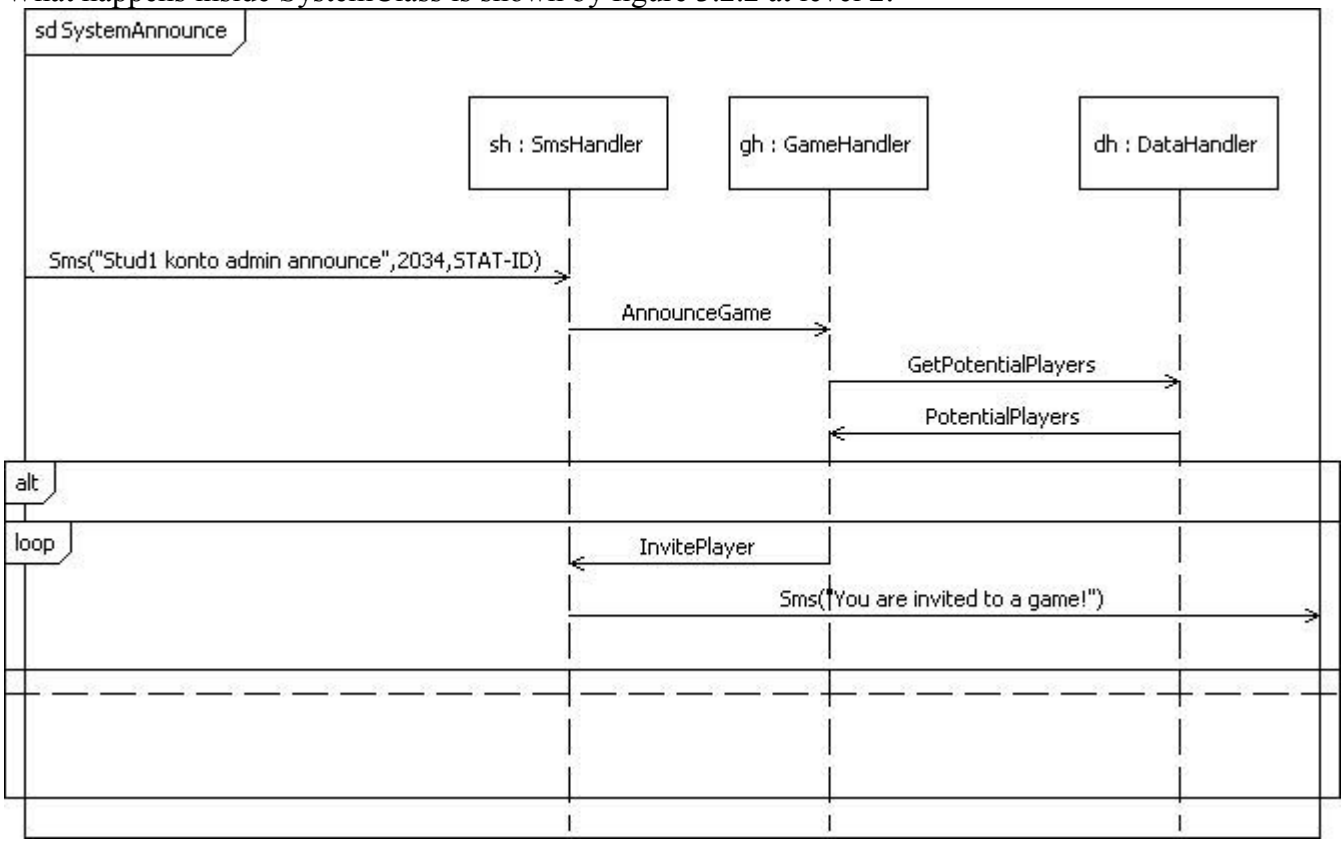


Figure 3.2.2

Gamehandler gets all the potential players from DataHandler. A potential player is a player that is registered for the system.

3.3 Registering for a game

After a player gets an invitation he/she can join a game by registering for it. A player cannot register if he/she already has registered for a game, or if he is not registered for the system. The sequence diagram at high level is shown below in figure 3.3.1

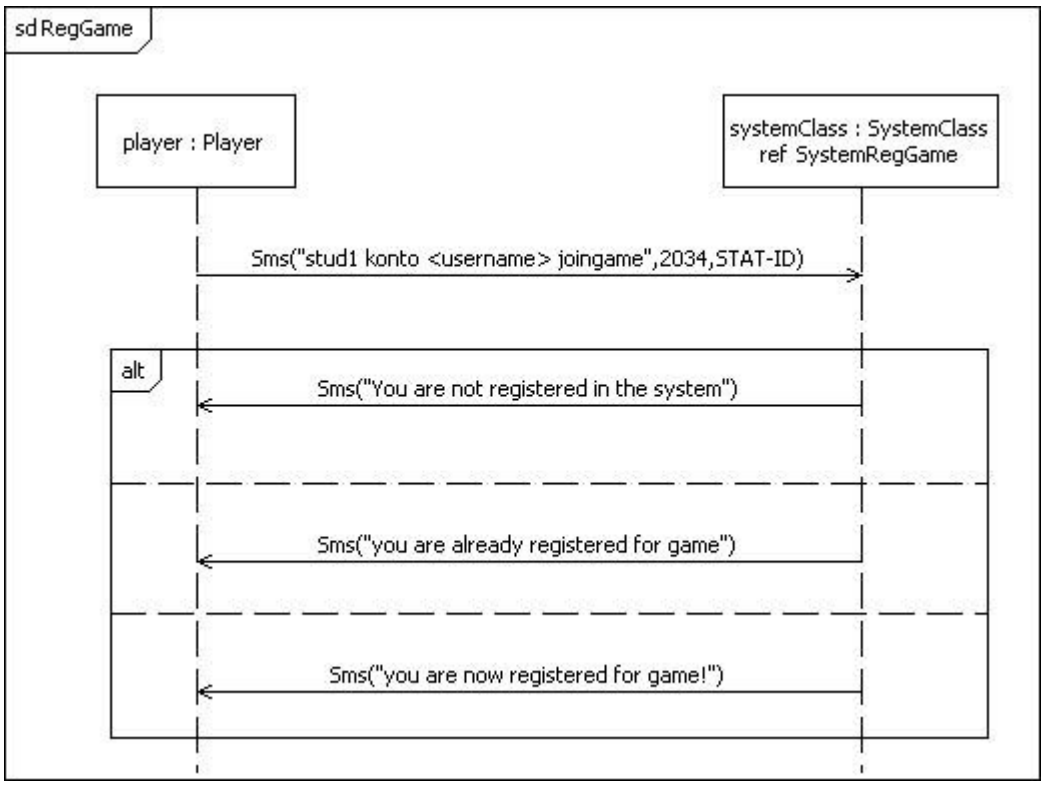


Figure 3.3.1

The decomposition of the lifeline SystemClass is like the one shown by figure 3.3.2:

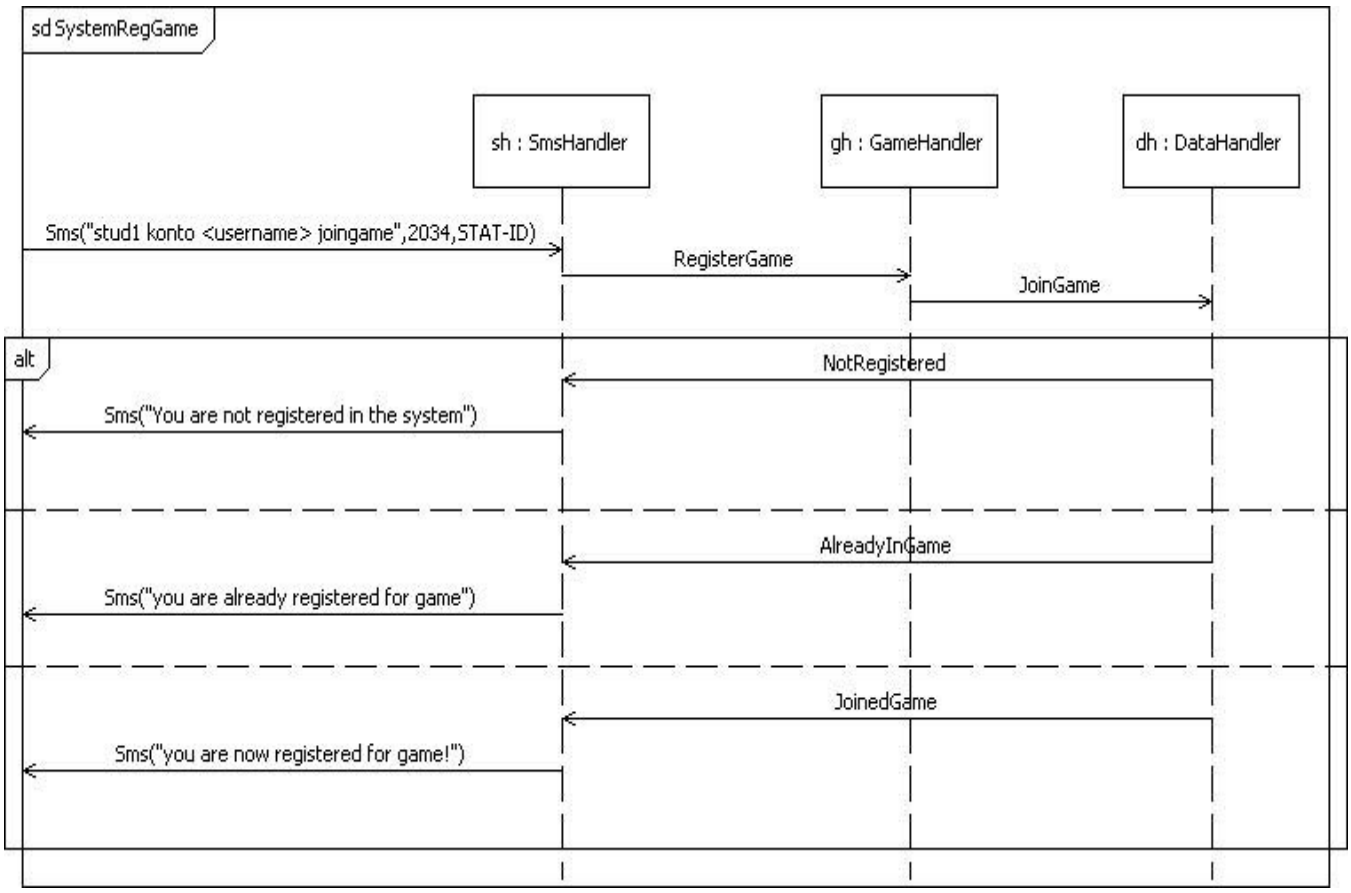


Figure 3.3.2

3.4 Starting a game

To start a game, administrator sends a message to the system containing a message that a game shall start. Thereafrer, every player that has registered for a game will receive a message which tells them that at game has started. See high level diagram in figure 3.4.1.

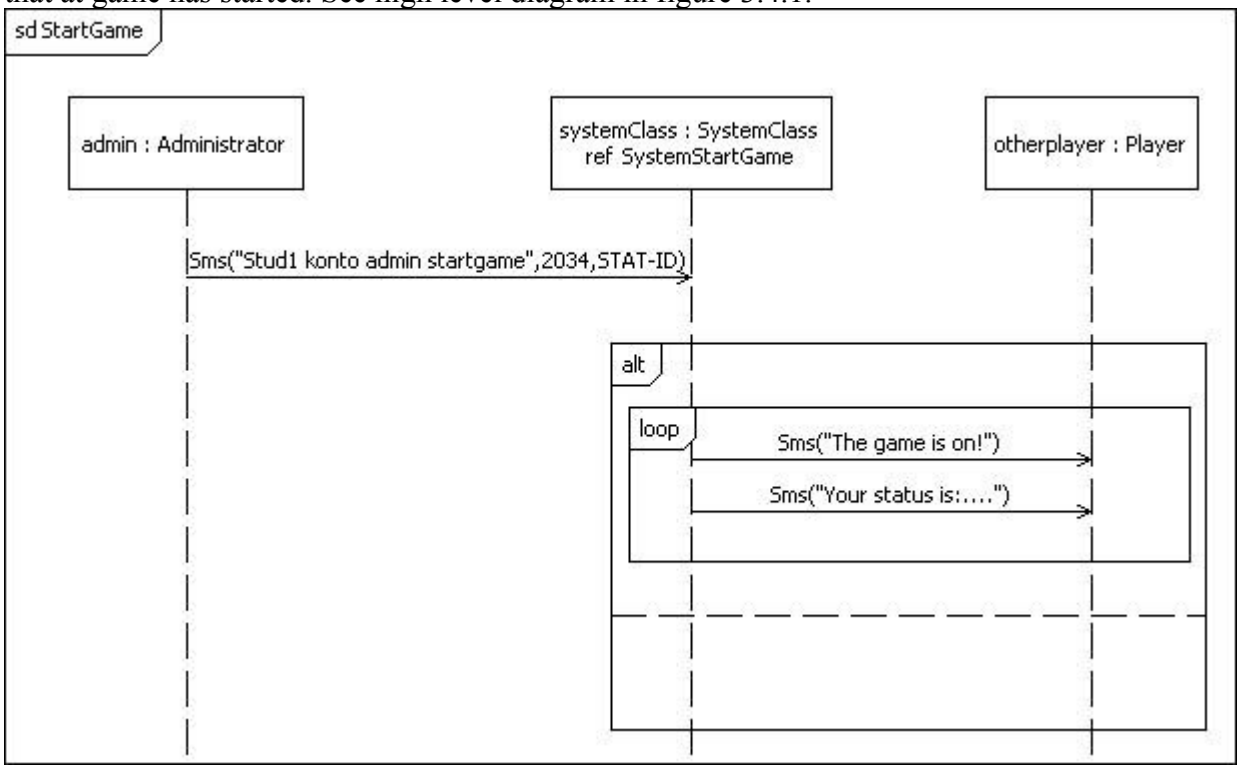


Figure 3.4.1

The sequence diagram SystemStartGame is shown by figure 3.4.2. It shows that when a game is starting, a PlayerProcess is created for each player. It controls everything a player does during a game.

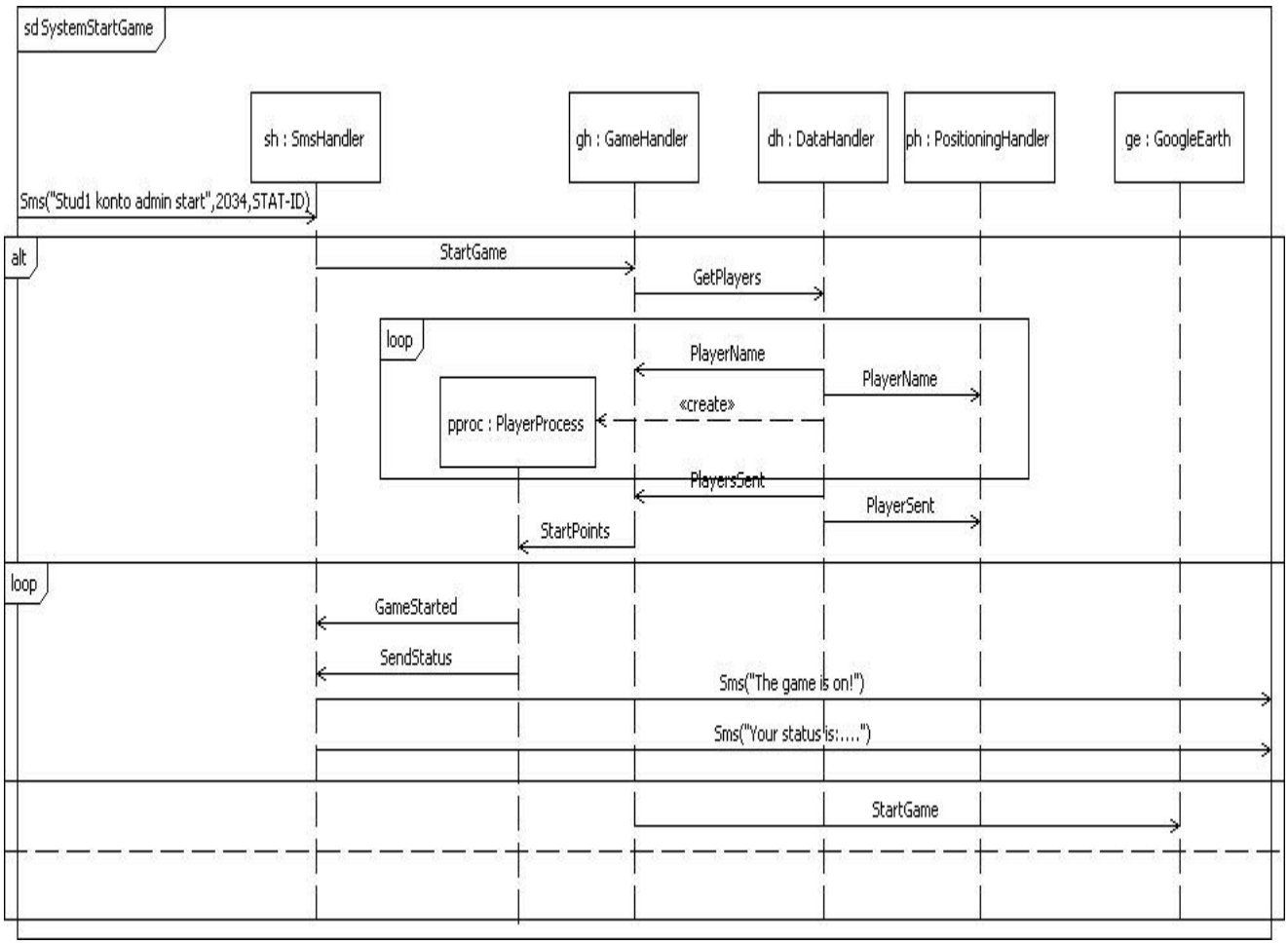


Figure 3.4.2

3.5 Setting up a shield

When a player is playing a game he/she can set up a shield. This shield has strength and duration. You can set up a shield only if you have enough points. If you already are using a shield you have two options. First you get information about the one you are using. Then you choose whether you want to throw it away and use the new one, or keep the old shield. The high level sequence diagram has the name SetShield (see figure 3.5.1). A player always gets a message consisting of what he/her did choose by a status report.

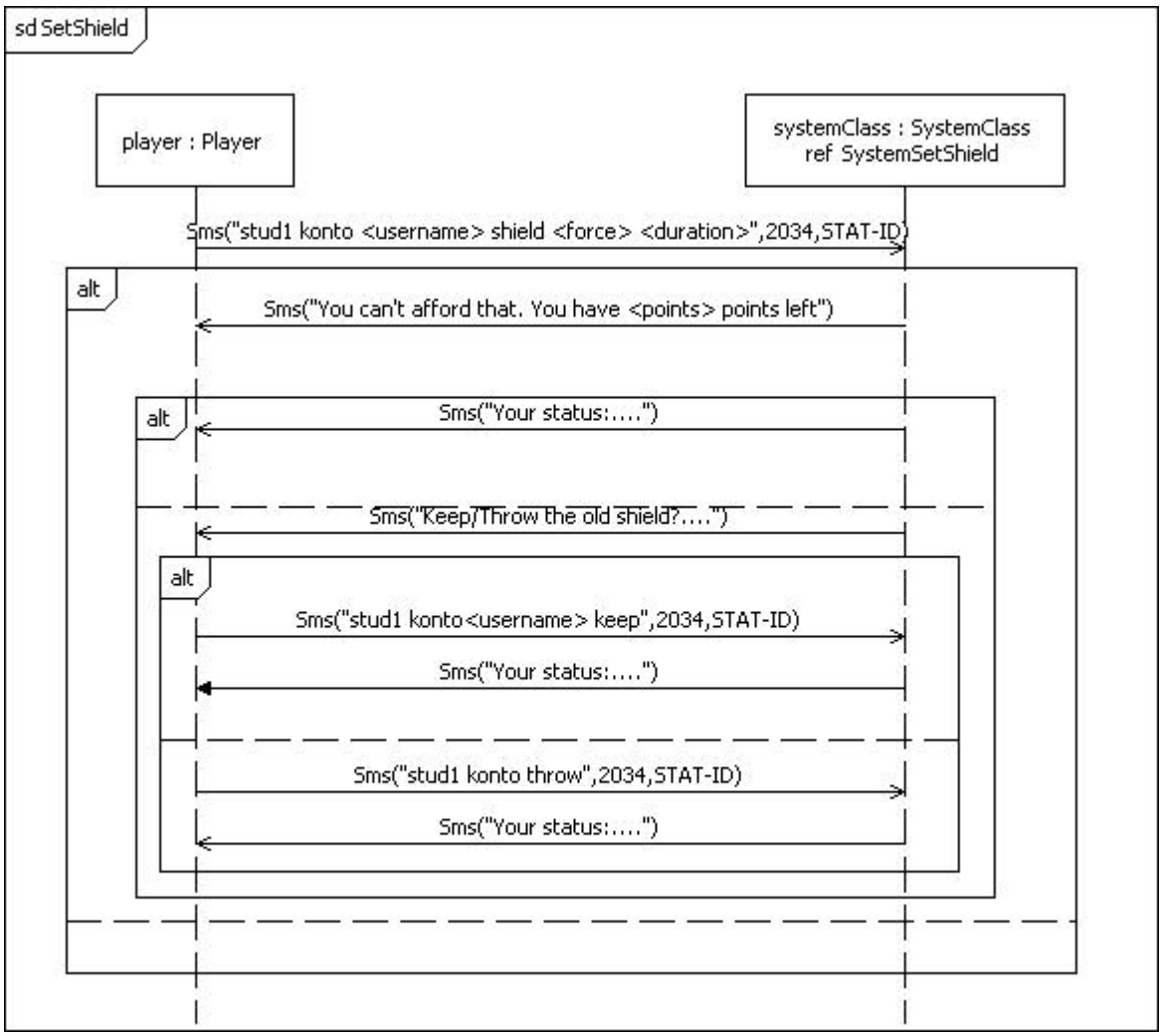


Figure 3.5.1

Figure 3.5.2 shows what is happening inside the lifeline SystemClass, which is decomposed into sequence diagram with the name “SystemSetShield”.

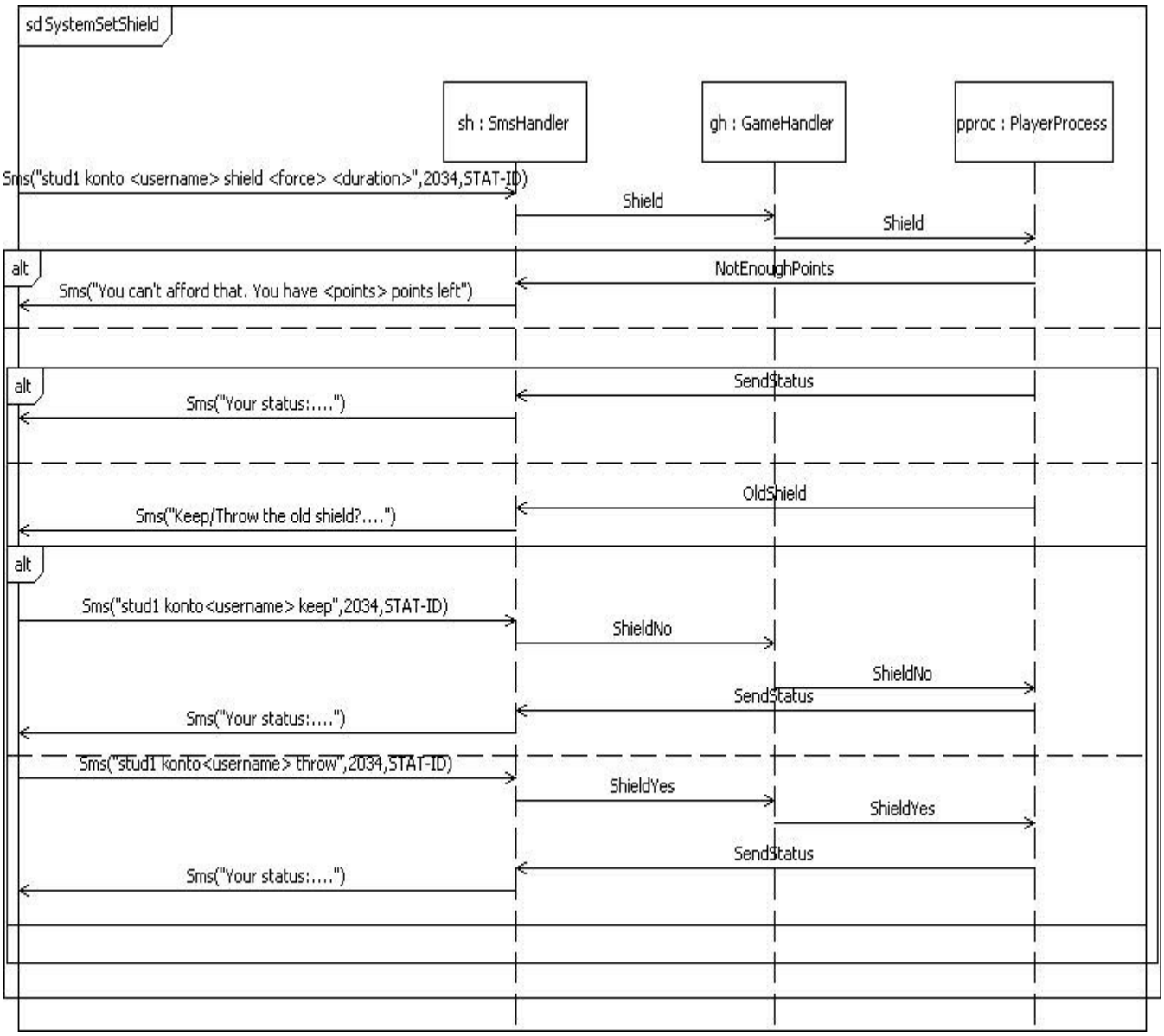


Figure 3.5.2

The lifeline pproc:PlayerProcess is handling every action a player does during the game. Every player has a relation to their own PlayerProcess.

3.6 Light up players

While playing a game you can light up players within a chosen range. You can only do this if you have enough points. If there are other players in this range you will get a list of them. These players will then get a message containing information that they have been spotted, see figure 2.6.1.

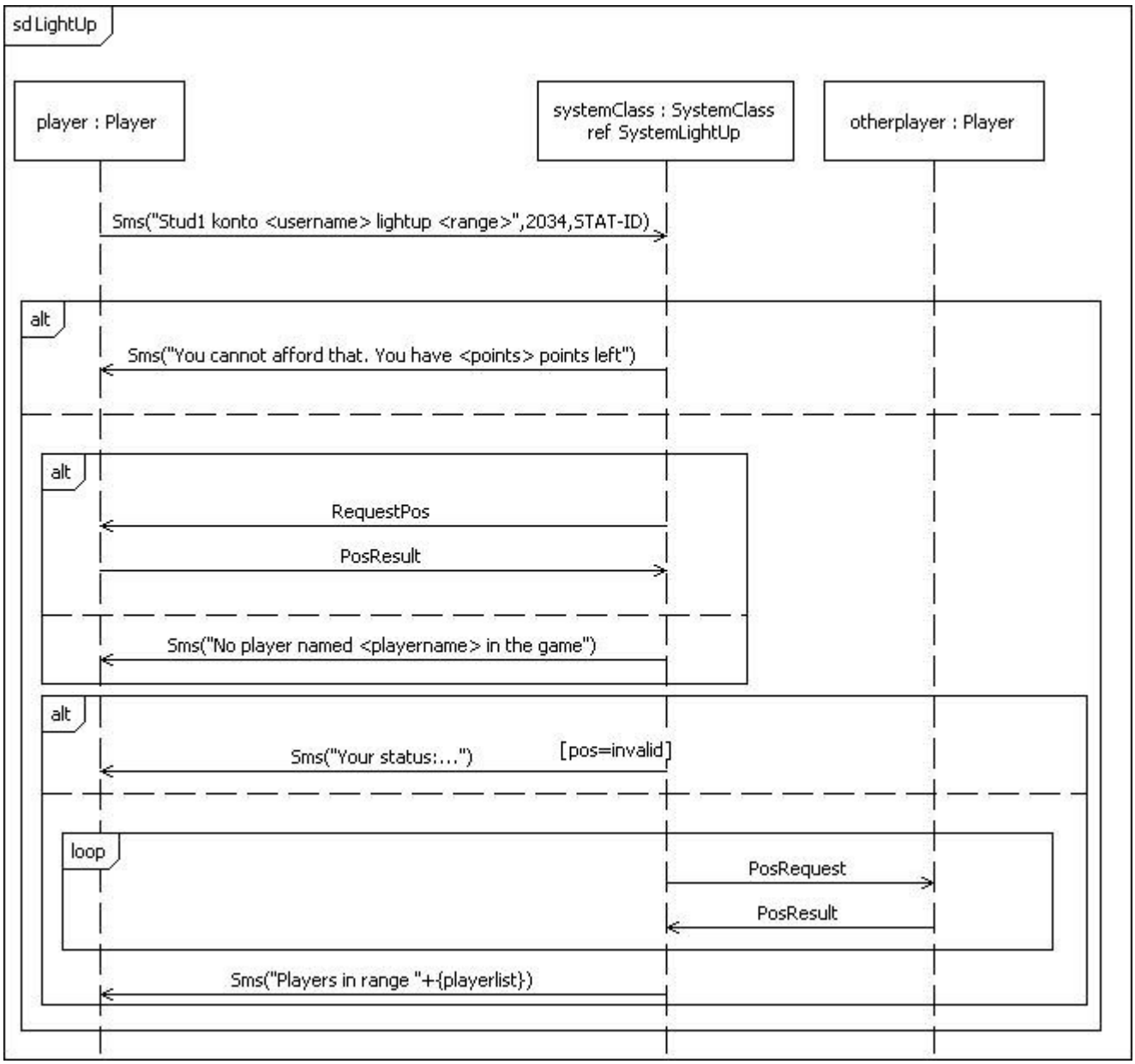


Figure 3.6.1

SystemClass is a decomposed lifeline which contains five other lifelines at level 2. This is shown by figure 3.6.2

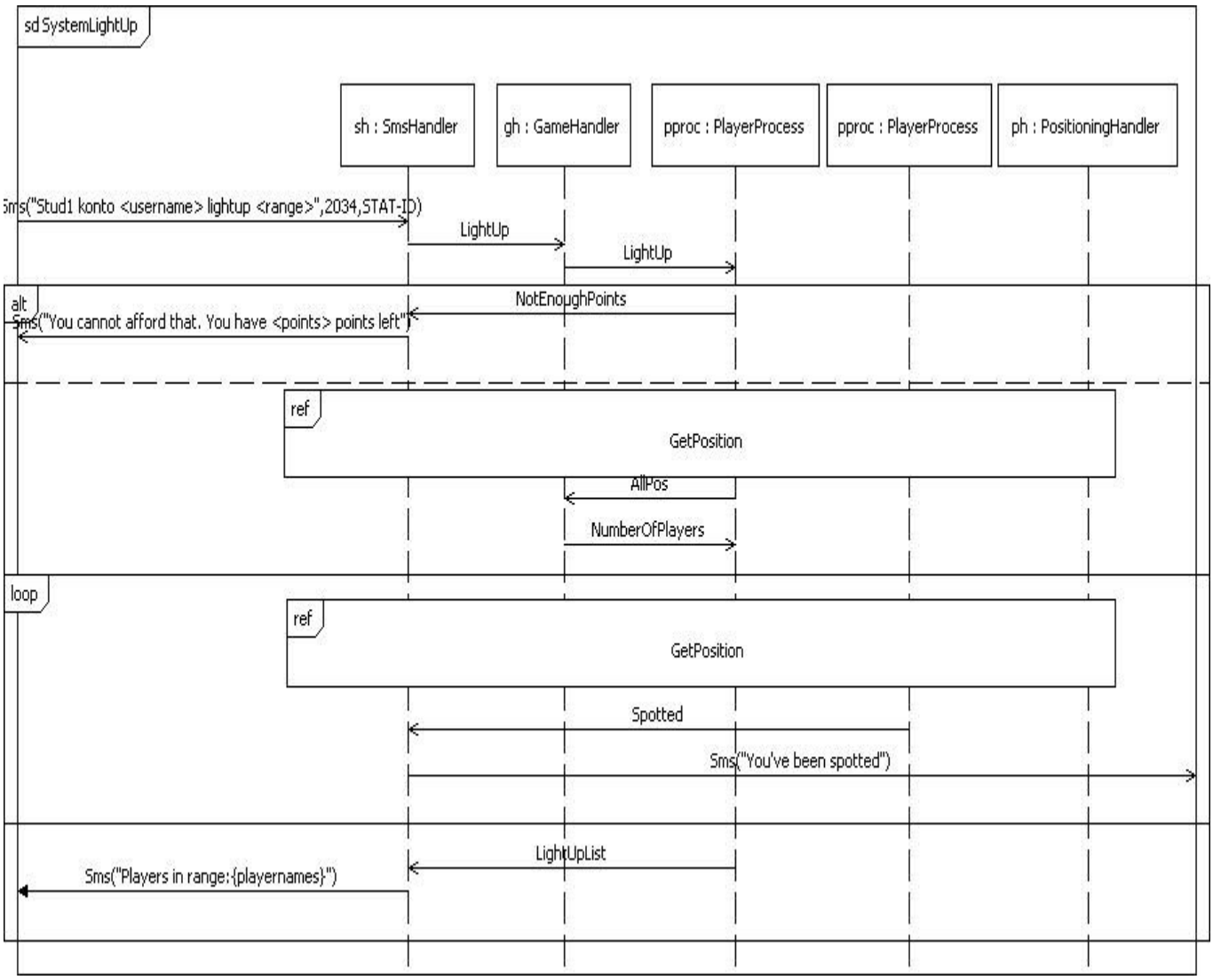


Figure 3.6.2

The figure above contains two PlayerProcess lifelines. The one to the left has a relation to the player who lights up other players. The PlayerProcess to the right has relations to all other players to show where the signals go. GetPosition refers to a sequence diagram that fetches a position for a player. This works as a function, and does the same each time. Figure 3.10.1 in chapter “3.10 Get position” shows the corresponding diagram. The last GetPosition is performed until every player in the game is positioned.

3.7 Striking a player

In a game it is possible to strike another player if you have enough points. You will get an error message if you are trying to strike someone who is not registered for the game. If the one you are striking does not have a shield, he/she will be “killed”. When a player gets status “dead”, he/she will automatically be excluded as a part of the game. Figure 3.7.1 is a high level representation of this functionality.

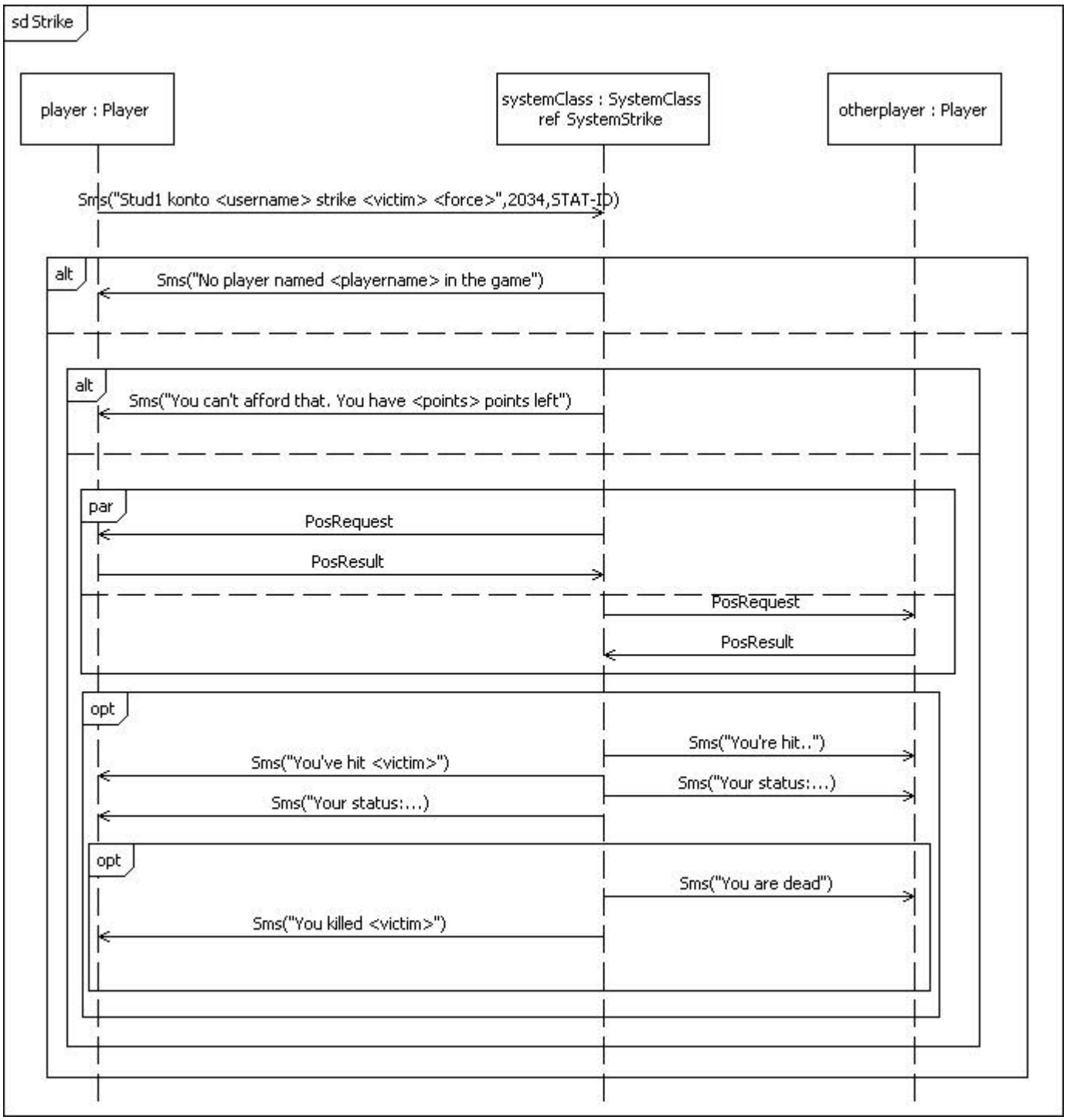
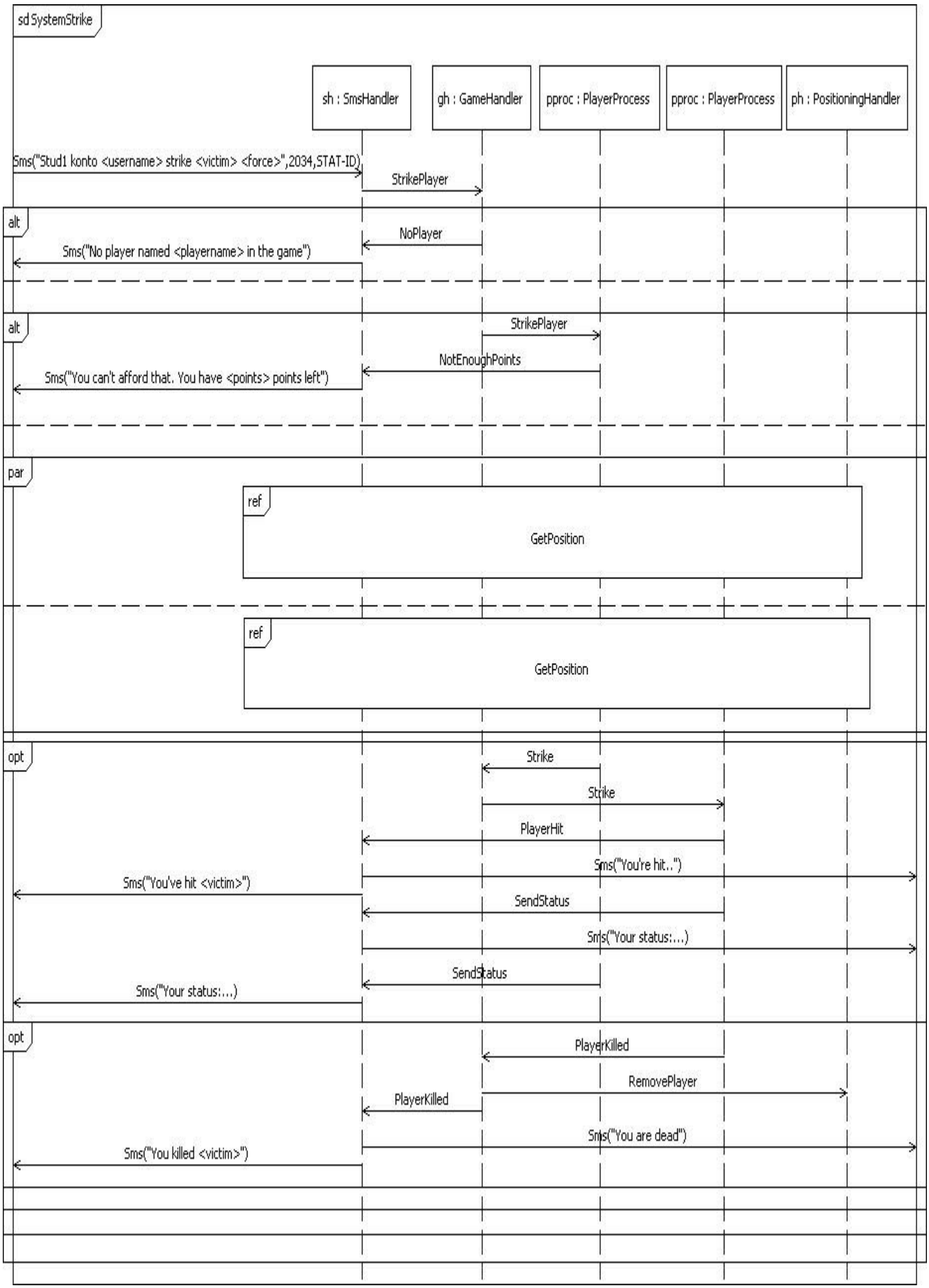


Figure 3.7.1

SystemStrike is decomposed into five lifelines, see figure 3.7.2.


sd SystemStrike
sh : SmsHandler
gh : GameHandler
pproc : PlayerProcess
pproc : PlayerProcess
ph : PositioningHandler
Sms("Stud1 konto <username> strike <victim> <force>",2034,STAT-ID)
StrikePlayer
alt
NoPlayer
Sms("No player named <playername> in the game")
alt
StrikePlayer
NotEnoughPoints
Sms("You can't afford that. You have <points> points left")
par
ref
GetPosition
ref
GetPosition
opt
Strike
Strike
PlayerHit
Sms("You're hit..")
Sms("You've hit <victim>")
SendStatus
Sms("Your status:...)
SendStatus
Sms("Your status:...)
opt
PlayerKilled
RemovePlayer
PlayerKilled
Sms("You are dead")
Sms("You killed <victim>")
Figure 3.7.2

The PlayerProcess to the left represents the player who is striking. The PlayerProcess to the right represents the player who gets struck. If the player who is supposed to get struck is out of range, nothing happens. If you hit a player, both players will get a message whether they hit, or were hit. They will also get a status report. If you at the same time kill him/her, you get a message containing this information.

3.8 Get status report

A player may, whenever he/she wants, get a status report for themselves while playing a game, by sending a message “getstatus” to the system, see figure 3.8.1.

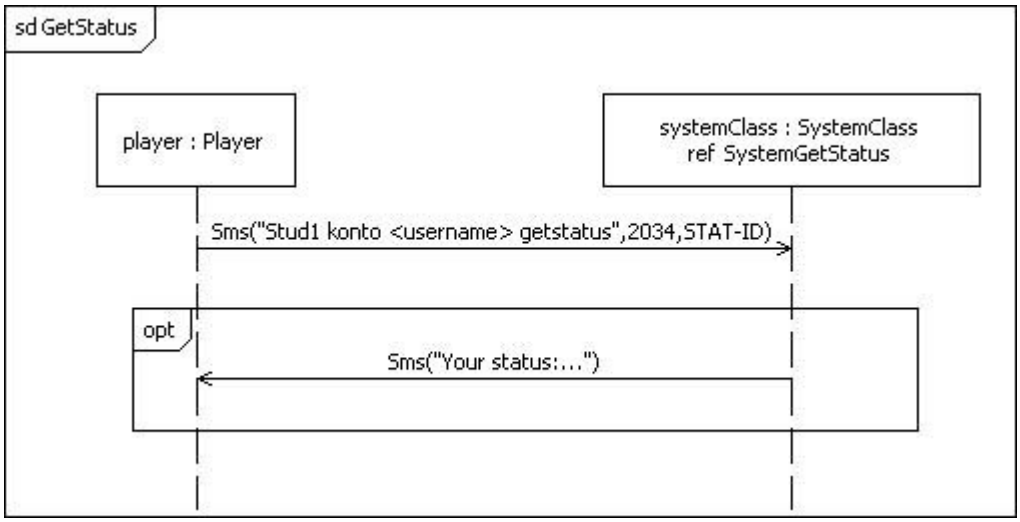


Figure 3.8.1

Figure 3.8.2 shows what is happening inside the decomposed lifeline SystemGetStatus.

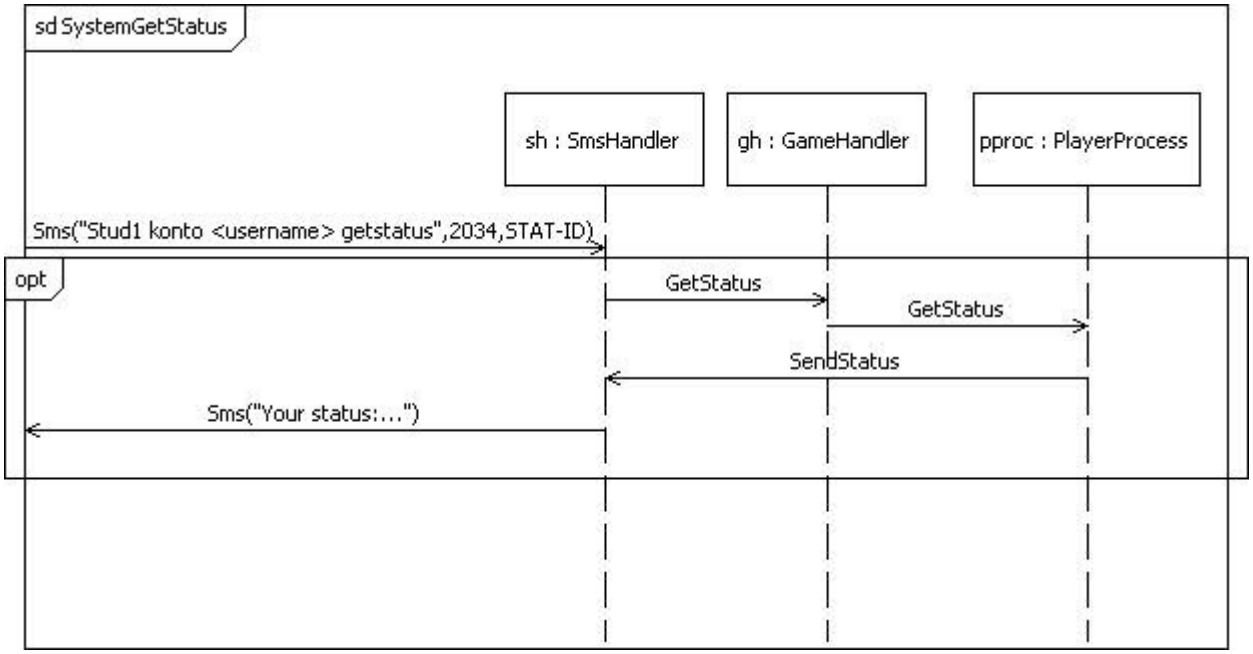


Figure 3.8.2

The PlayerProcess has all information about the related player.

3.9 Write KML-file

The administrator may at any time request the system to write an kml-file, containing positions to all players. The sequence diagram at high level is called “WriteKml” and is shown in figure 3.9.1.

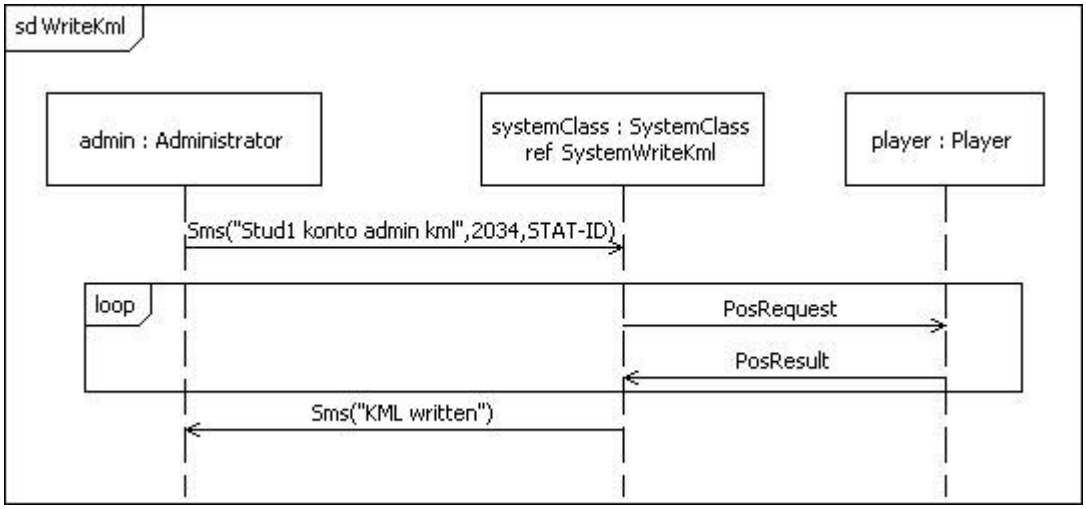


Figure 3.9.1

After the system has performed administrators command, it sends a message to the administrator to confirm that it is done. Figure 3.9.2 shows what happens inside the system.

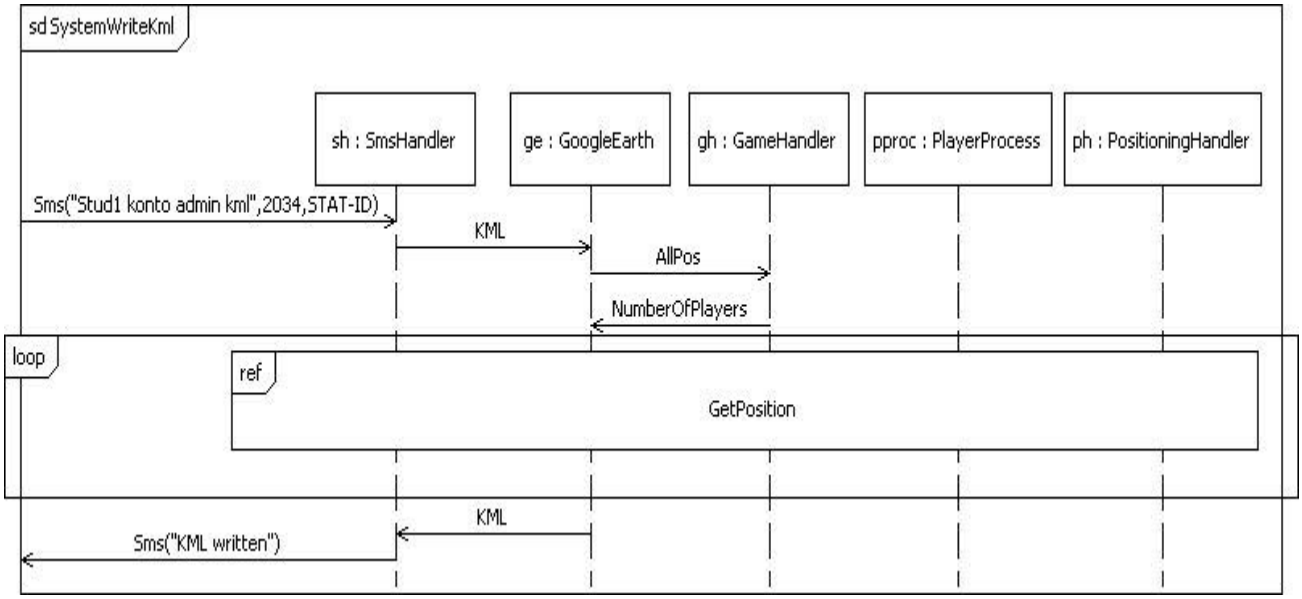


Figure 2.9.2

The kml-file is created by ge:GoogleEarth. GetPosition is performed in a loop until every player in the game is positioned.

3.10 Get position

Figure 3.10.1 shows a sequence diagram which is being referred to in some other sequence diagrams.

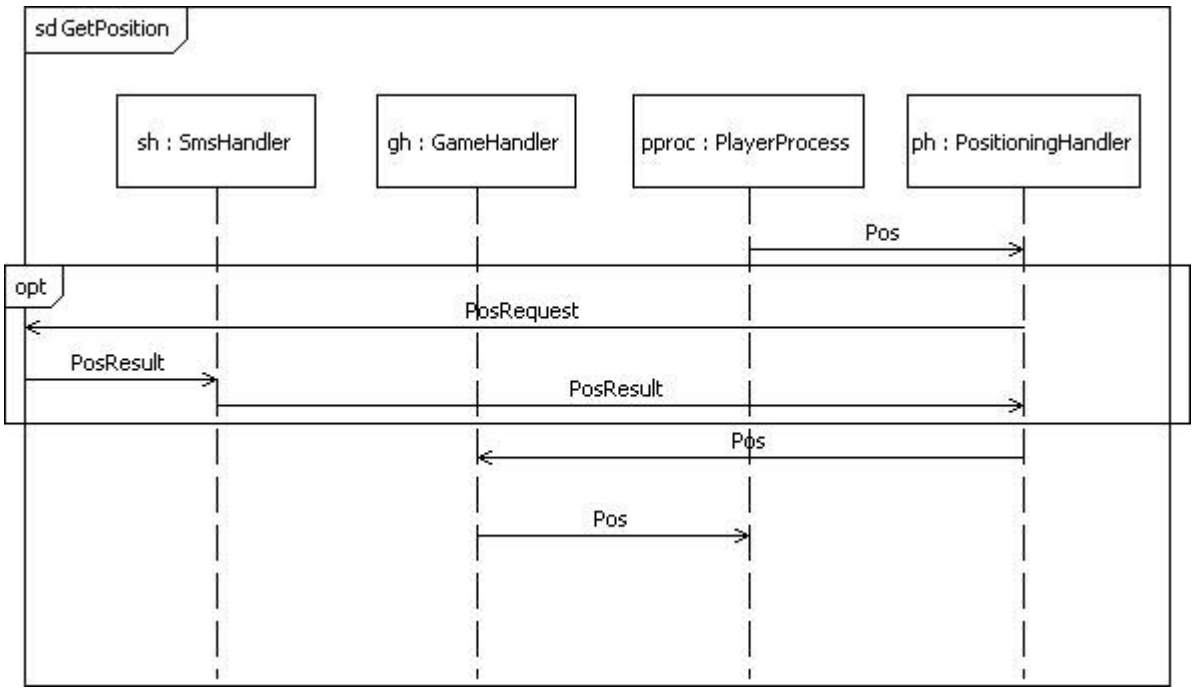


Figure 3.10.1

A PlayerProcess sends a message to PositionHandler. There is an opt-operator which covers the PosRequest and PosResult. This is mainly because we use a list with positions of all the players and the respective timestamps. The positioning is valid for a pre-defined period of time, and the positioning function will not need to be recalled for this player during that time interval.

4. System design: State machine diagrams and signals

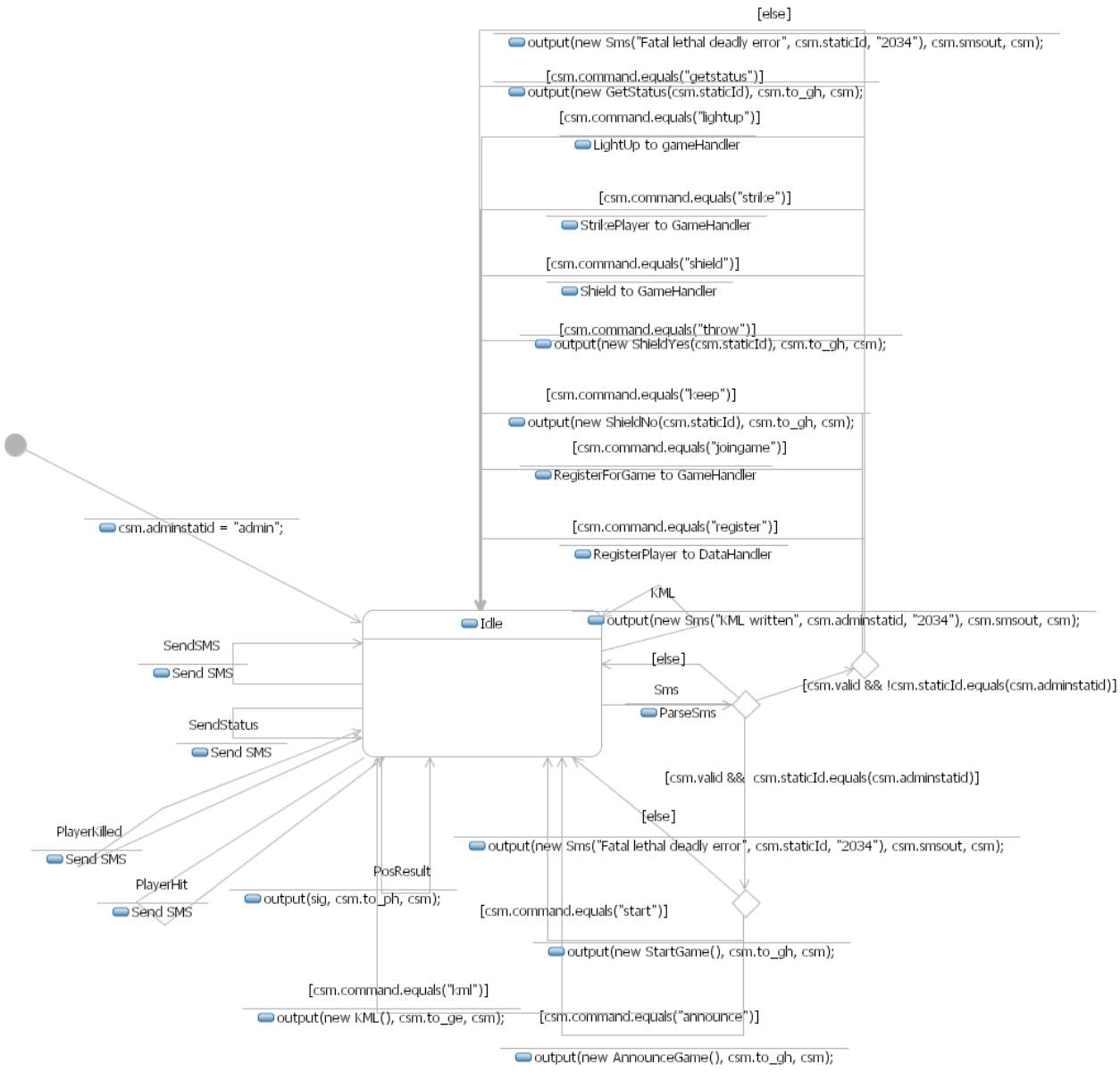


Figure 4.1

The SMSHandler statemachine (shown by figure 4.1) handles incoming SMSes and sends signals to the other StateMachines. It recieves signals from the other statemachines and sends SMSes to users and the admin. It forwards PosResult signals to the PositioningHandler statemachine.

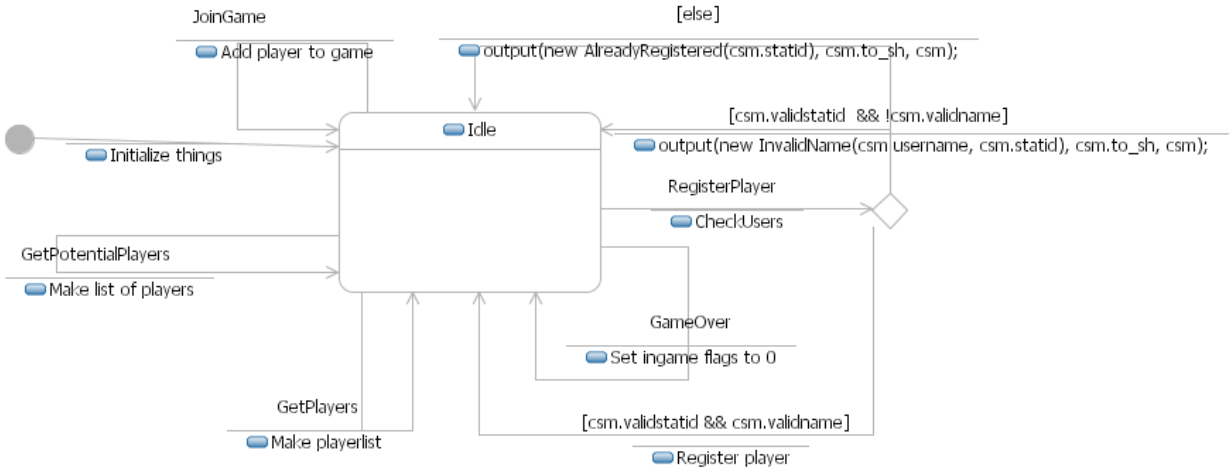


Figure 4.2

The DataHandler statemachine (shown by figure 4.2) handles the database containing the registered users. When users register for the system, their static IDs and usernames are added to the database. The DataHandler sends a list of potential players to GameHandler when a game is announced and sends all the static IDs and usernames of all the players registered for the game when the game starts. The DataHandler is not used for anything once a game is running, except for new users registering for the system.

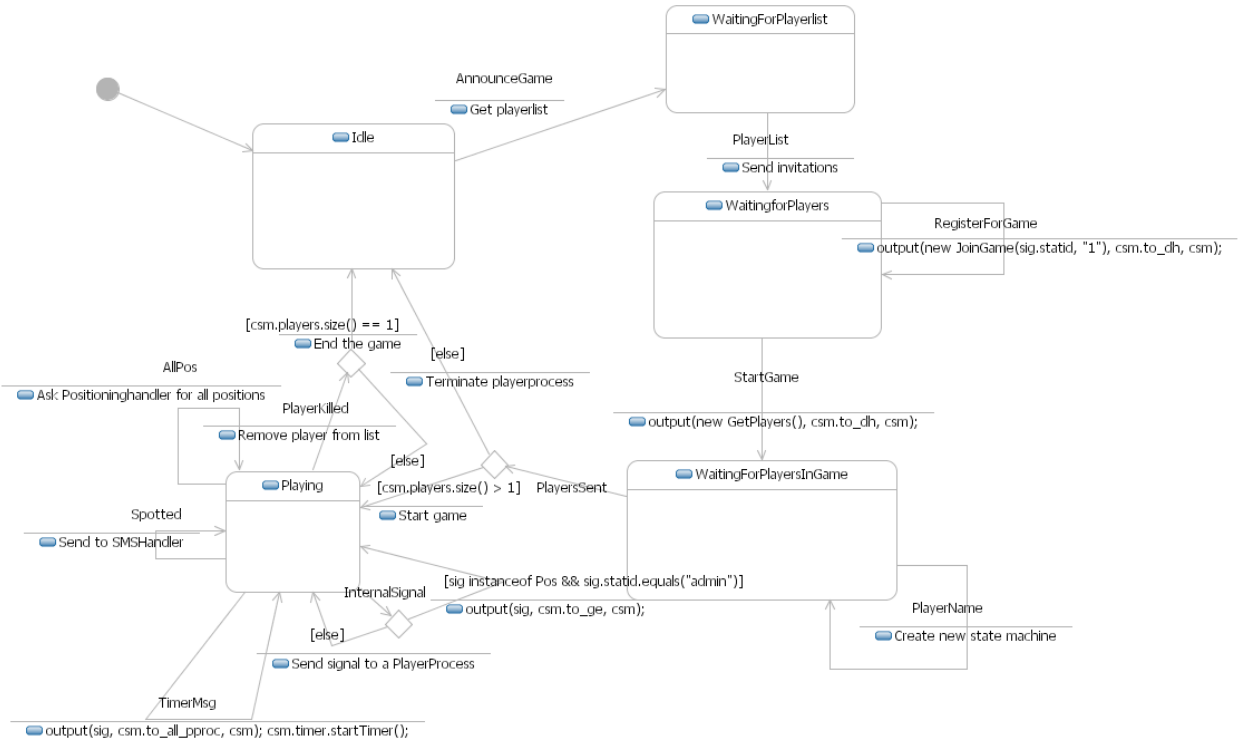


Figure 4.3

The GameHandler statemachine (shown by figure 4.3) keeps track of what stage of the game the system is in. It gets playerdata from GameHandler when the game is announced or started.

When the game is being played GameHandler routes InternalSignals to correct PlayerProcess processes (it sets the targetstatid of the StrikePlayer signal before forwarding it, and certain Pos signals are sent to GoogleEarth instead). It sends Pos signals to PositioningHandler when it recieves the AllPos signal (for lightup or kml). It recieves Spotted signal, it sets its statid attribute and forwards it to SMSHandler. It handler the timer and forwards TimerMsg signals to all the PlayerProcess processes. it keeps track of how many players are left in the game and ends the game when only one player is left.

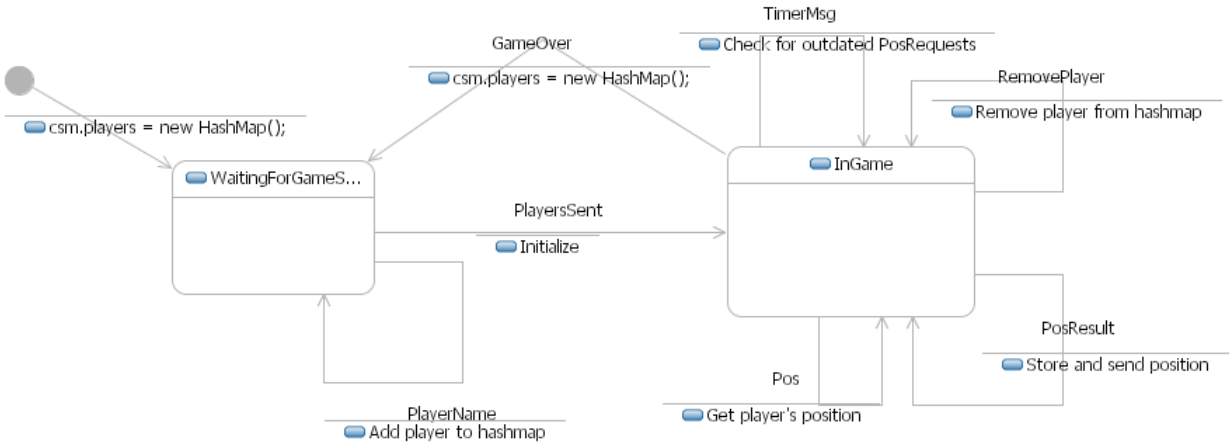


Figure 4.4

The PositioningHandler statemachine (shown by figure 4.4) recieves Pos signals, sets the x and y values of the signal and sends them to GameHandler which will route it to the right player process or the GoogleEarth machine. It keeps a hashmap with player positions so that it doesn't need to send several PosRequest signals when the same position is asked for more than once at quite the same time. PositioningHandler takes care of timeouts on PosRequests, and if a PosRequest times out, the player being positioned is removed from the game or the PosResult signal isn't valid, PositioningHandler won't set the valid attribute of the signal to true, but still send it to GameHandler. It's up to the PlayerProcess or GoogleEarth machine to handle invalid Pos signals. As long as a PosResult is recieved for every PosRequest signal, one Pos signal will be sent from PositioningHandler for every one recieved.

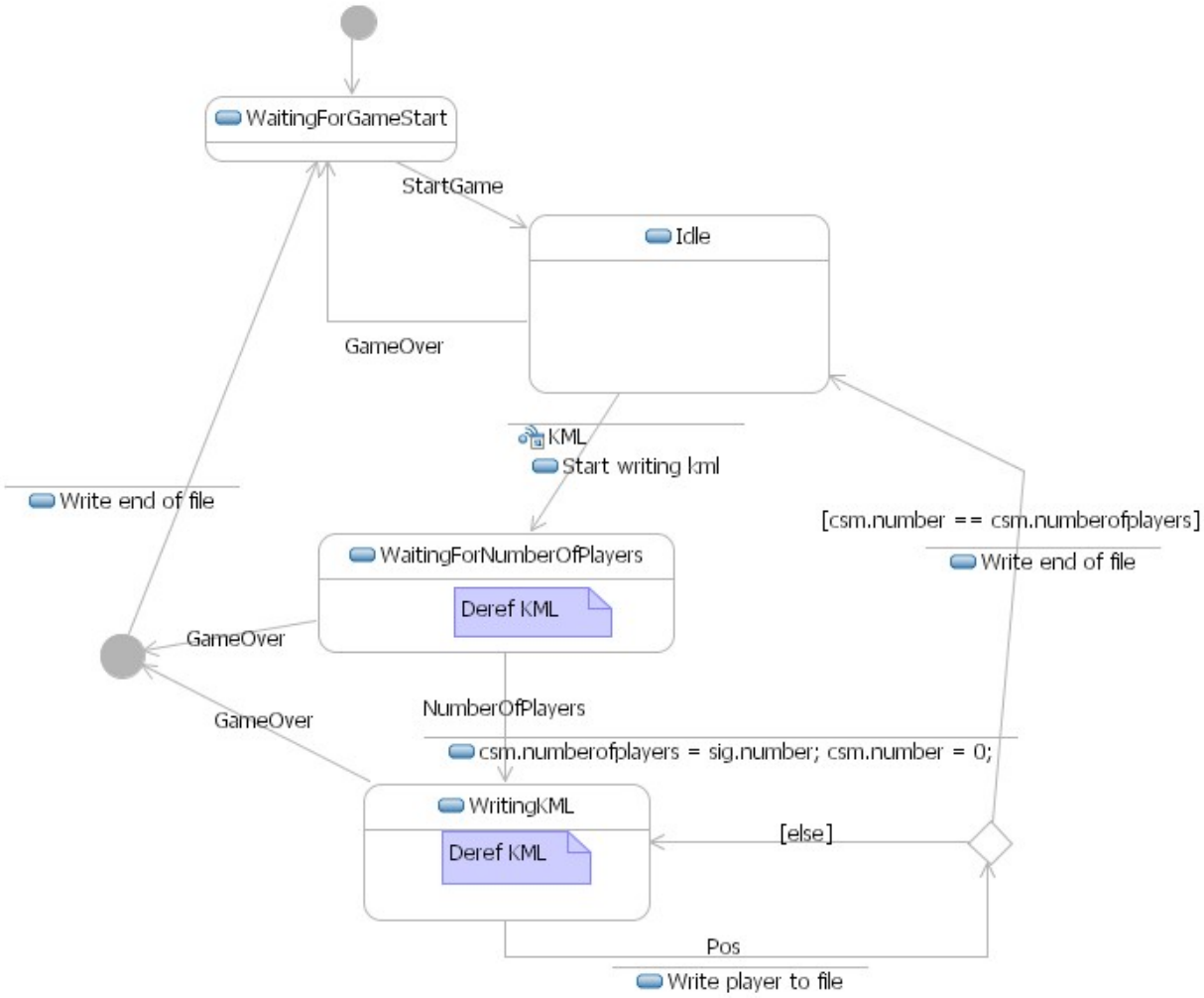


Figure 4.5

The GoogleEarth statemachine (shown by figure 4.5) gets the positions of all the players in the game and writes them to a .kml file to be viewed by GoogleEarth.

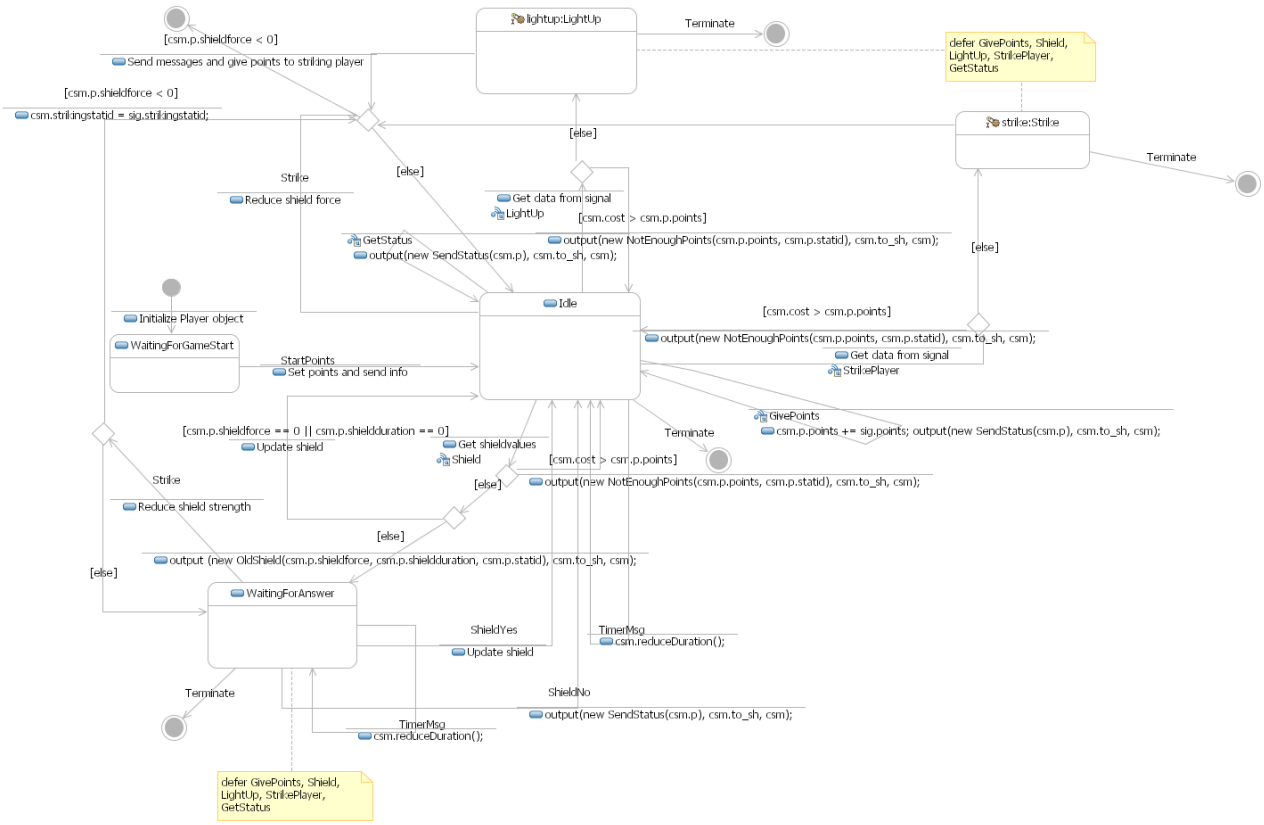


Figure 4.6

Each PlayerProcess statemachine (shown by figures 4.6-4.8) keeps track of a player's stats, and performs the strike, shield and lightup actions for that player. The statemachine's p attribute, a Player object, holds the player's stats while all the other attributes of the statemachine are used for temporary variables. Every state in PlayerProcess handles incoming strikes and timer signals.

A player process handles one command from its user at a time. New commands from the user are only handled in the Idle state (and defered in other states). The timerMsg signal and Strike (that player is struck by another player) are taken care of in all states.

When recieving StrikePlayer, Shield and LightUp signals, the process will check the cost of the action and perform it if its player can afford it. If not it sends a message to its player with the amount of points left.

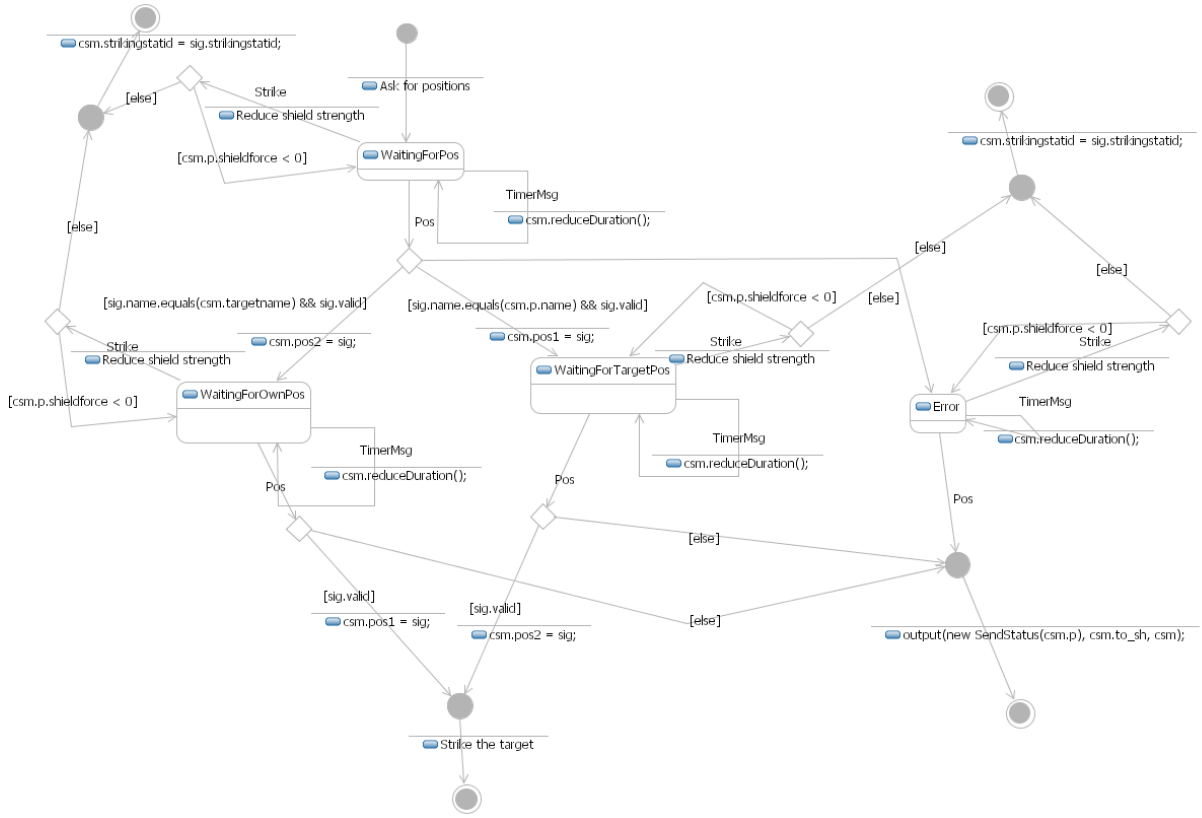


Figure 4.7

In the Strike substatemachine (shown by figure 4.7), the player process gets its player's and the target's positions. If it recieves two valid positions it pays for the action (note that if the first position recieved is invalid it waits for the second Pos signal before leaving the substate, to make sure it won't recieve an additional Pos signal when it's performing some other acion later). It then checks if the target is withing range and sends a Strike signal to that player's process if it is.

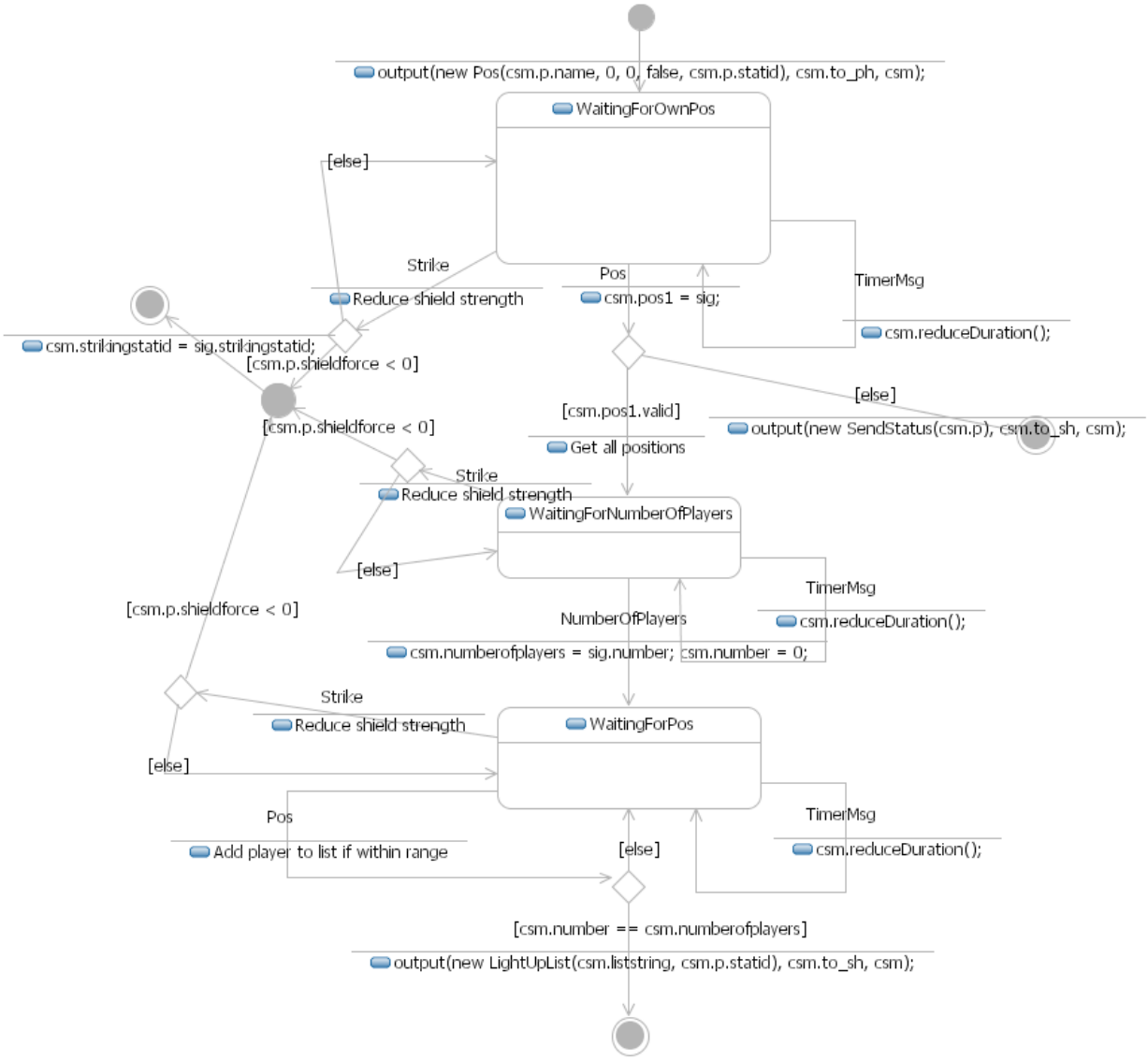


Figure 4.8

In the LightUp substatemachine (shown by figure 4.8) the process first positions its player, before sending an AllPos signal to the GameHandler machine. The GameHandler will send the number of players in the game to the player process and send Pos signals to PositioningHandler for it. The player process will wait for Pos signals until it has recieved as many as the number of players, and for each player positioned add the player's name to a list and send an SMS letting the player know he or she is spotted. Once the machine has recieved all the Pos signals it sends an SMS with the list of playernames to its player and leaves the substate.

Signals (figure 2.1)

- The AllPos signal is sent to GameHandler from a player process or the GoogleEarth machine. GameHandler then sends the number of players back and sends Pos signals for the player process or GoogleEarth machine.
- The AnnounceGame signal is sent from SMSHandler to GameHandler when the admin announces a game. GameHandler then gets a list of potential players from DataHandler and invites everyone to the game.
- The StartGame signal is sent from SMSHandler to GameHandler when the admin starts the game and from GameHandler to the GoogleEarth machine and PositioningHandler when the game starts.
- The StartPoints signal is sent from GameHandler to all player processes when the game starts to give all the players their starting points.
- The GameOver signal is sent from GameHandler to PositioningHandler, DataHandler and GoogleEarth when all but one player in the game has been killed so that those machines can go back to their pre-game states.
- The GetPlayers signal is sent from GameHandler to DataHandler to get the static IDs and names of players that have joined the game.
- The GetPotentialPlayers signal is sent from GameHandler to DataHandler to get a list of players registered for in the system.
- The JoinGame signal is sent from GameHandler to DataHandler when a player joins a game.
- The KML signal is sent from SMSHandler to the GoogleEarth machine when the admin wants a .kml file, and from GoogleEarth to SMSHandler when the file is written.
- PlayerHit and PlayerKilled signals are sent to SMSHandler when a player is hit or goes dead to let the striking and striked player know of it.
- The PlayerList signal is sent from DataHandler to GameHandler when a game is announced. It contains a list of all players registered for the system.
- When the game starts, one PlayerName signal is sent from DataHandler to GameHandler for every player that has joined the game, before a PlayersSent signal is sent.
- The RegisterForGame signal is sent from SMSHandler to GameHandler when a player tries to join a game.
- The RegisterPlayer signal is sent from SMSHandler to DataHandler when a player tries to register for the system.
- RemovePlayer is sent from GameHandler to PositionHandler when a player is killed so that PositioningHandler can update its hashmap of players.
- SendStatus is sent from a player process to the SMSHandler to give a player information about his or her points, shield force and shield duration.

Subclasses of the SendSMS signal (figure 2.2) are all sent to the SMSHandler which will send an SMS to the statid attribute of the signal, depending on the signal type.

- The Registered signal lets a player know he or she has successfully registered for the system.
- The InvalidName signal lets a player know the name he or she is trying to register for the system is already taken.
- The AlreadyRegistered signal lets a player trying to register for the system know that he or she is already registered.
- The InvitePlayer signal invites a player registered for the system to join a game.
- The JoinedGame signal lets a player know he or she successfully joined a game.
- The NotRegistered signal lets a player trying to join a game know that he or she is not registered in the system and how to register.

- The AlreadyInGame signal lets a player trying to register for a game know that he or she has already joined a game.
- The GameStarted signal lets a player know the game he or she has joined has been started.
- The NoPlayer signal lets a player know he or she tried to strike a playername that is not in the game.
- The OldShield signal asks a player trying to set up a new shield if he or she would rather like to keep the old, still alive shield.
- The Spotted signal lets a player know that someone has spotted him or her in a lightup.
- The LightUpList signal lets a player know which players he or she has spotted in a lightup.
- The NotEnoughPoints signal lets a player know that the shield, lightup or strike he or she is trying to perform is a bit too costly.
- The Winner signal lets the winning player know he or she has won.

Subclasses of the InternalSignal signal (figure 2.3) is routed by the GameHandler machine to the player process for the signal's statid attribute.

- The GetStatus signal is sent from SMSHandler when a player wants to know how many points he or she has left and the force and duration of his or her shield.
- Shield signals are sent from SMSHandler when a player is trying to set up a shield, and ShieldNo or ShieldYes signals are sent when a player decides to keep or throw his or her old shield.
- LightUp and StrikePlayer signals are sent from SMSHandler when a player is trying to perform the lightup or strike actions.
- Pos signals are sent to PositioningHandler from the player process trying to position someone, or the GameHandler when positioning all players, and are sent from PositioningHandler to Gamehandler once the player is positioned. The GameHandler routes the signal to the correct player process (or sends it to the GoogleEarth machine).
- The NumberOfPlayers signal is sent from GameHandler to a player process (or the GoogleEarth machine) when that player asks for the positions of all players, so that the player process knows how many Pos signals to wait for.
- The Strike is sent from one player process to another. When recieved the player process reduce its shield force and terminates if shield force is then below zero.
- The GivePoints is sent from a terminating player process to the one that striked it dead, giving the dead player's points to the striking one.
- The Terminate signal is sent from GameHandler when there is only one player left in the game (either because everyone else is killed dead, or because only one player joined the game).

5. “Survival of the SMSest” - Security Risk Analysis

This part of the report presents the results of the risk security analysis of the “survival of the SMSest” system. The analysis is based on the Coras method, see [1] and [2] and conducted using the CORAS tool [3].

5. 1 Target description

For the target description we refer to the system documentation part of the report.

The scope of the analysis is the part of the system developed by the group as a part or the compulsory exercise within the course. Outside the scope are among others PATS, SMS sending/receival by the user phones, the cellular phones of the end users and the network for communication between phones and the parts of the system.

It is assumed that the end users, i.e. that target group is well known by the stakeholder (system administrator), so that their expectations to the service are well defined. Their economical capabilities and the value of the service are assumed to be in accordance to the amount charged. It is also assumed that the hardware supplied is of sufficient quality and capacity. In addition, we assume that the physical security of the hardware is ensured.

5.2 Asset identification

Table 5.1 below shows the assets identified, from a game administrator’s point of view.

Asset	**Asset description**	**Importance**	**Type**
Availability of the system	Availability of the system as a whole	1	Direct asset
Correctness of the system	Correctness of the system’s data presentation	2	Direct asset
Scalability of the system	Scalability of the system in terms of the number of users	3	Direct asset
Security of the system	Security of the system	4	Direct asset
Public trust in the system	Public trust in the system	5	Indirect asset

Table 5.1: Asset table

These assets are identified as the ones having most value for the stakeholder in question, and their importance is assumed to be as shown by the “importance” column. There is one indirect asset, “Public trust in the system” which is not directly a subject to analysis, but is related to the four direct assets.

Figure 5.1 below is the corresponding asset diagram displaying the stakeholder, the scope and the assets. For the purpose of clarity, links from game administrator to the assets are omitted, but normally such links could have been drawn between the stakeholder and all the assets represented.

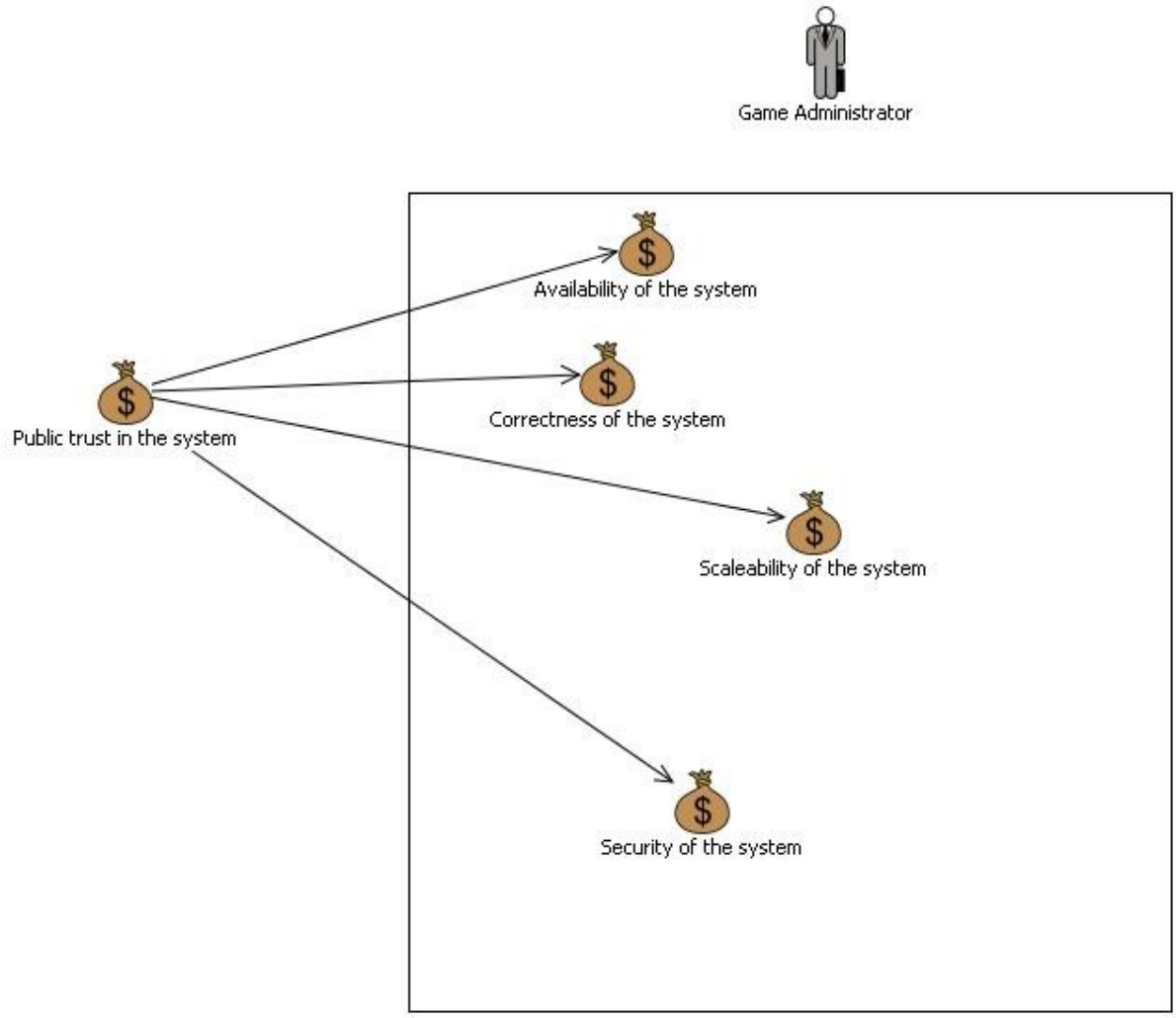


Figure 5.1: asset diagram

Availability is considered to be the most valuable asset, and its consequence scale is proposed to be as shown by table 2 below. The consequences are categorised by the down time of the system.

Consequence value	**Description**
Catastrophic	More than 20 min. per day
Major	More than 10 and up to 20 min. per day
Moderate	More than 4 and up to 10 minutes per day
Minor	More than 1 and up to 4 minutes per day
Insignificant	Up to one min per day.

Table 5.2: Consequence scale for ”availability of the system” asset

Table 5.3 shows the consequence scale for ”Correctness of the system” asset. The consequences are categorised according to the percentage of the service degradation.

Consequence value	**Description**
Catastrophic	80 – 90 % service degradation.
Major	50 - 79 % service degradation
Moderate	20 - 49 % service degradation
Minor	6 - 19 % service degradation
Insignificant	Up to 5 % service degradation

Table 5.3: Consequence scale for ”Correctness of the system” asset

Table 5.4 shows the consequence scale for ”Scalability of the system” asset. The consequences are categorised according to the percentage of the simultaneous users supported at time, by the system.

Consequence value	**Description**
Catastrophic	The system supports less than 70 % of simultaneous users.
Major	The system supports between 70 and 84 % of simultaneous users.
Moderate	The system supports between 85 and 89 % of simultaneous users.
Minor	The system supports between 90 and 98 % of simultaneous users.
Insignificant	The system supports between 99 and 100 % of simultaneous users.

Table 5.4: Consequence scale for ”Scalability of the system” asset

Table 5.5 below shows the consequence scale for ” Security of the system” asset. The consequences are categorised according to the number of intrusions into the system.

Consequence value	**Description**
Catastrophic	10 or more intrusions
Major	6-10 intrusions
Moderate	3-5 intrusions
Minor	2 intrusions
Insignificant	1 intrusion

Table 5.5: Consequence scale for ”Security of the system” asset

Table 5.6 below shows the likelihood scale. The likelihoods are categorised according to the number of occurrences per year.

Likelihood value	**Description**
Certain	More than 10 times per year
Likely	5-10 times per year
Possible	2-4 times per year
Unlikely	Once a year
Rare	Less than once a year

Table 5.6: Likelihood scale

Table 5.7 below shows the risk evaluation matrix for “availability of the system” asset. The acceptance is decided by considering the ratings of likelihood-consequence combinations for the asset in question. The acceptable risks are marked with green and the ones that should be evaluated more closely, are marked with red.

	Consequence				
	Insignificant	*Minor*	*Moderate*	*Major*	*Catastrophic*

Frequency	*Rare*	**Acceptable**	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**
	Unlikely	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**	**Must be evaluated**
	Possible	**Acceptable**	**Acceptable**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**
	Likely	**Acceptable**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**
	Certain	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**

Table 5.7: Risk evaluation matrix for “availability of the system” asset

Table 5.8 below shows the risk evaluation matrix ”Correctness of the system” asset. The acceptance criteria are different from the ones in the previous asset, and this is, mainly, a result of the consequence scale definitions and the importance rating of the asset.

		Consequence				
		Insignificant	*Minor*	*Moderate*	*Major*	*Catastrophic*
Frequency	*Rare*	**Acceptable**	**Acceptable**	**Acceptable**	**Acceptable**	**Acceptable**
	Unlikely	**Acceptable**	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**
	Possible	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**	**Must be evaluated**
	Likely	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**	**Must be evaluated**
	Certain	**Acceptable**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**

Table 5.8: Risk evaluation matrix for ”Correctness of the system” asset

Table 5.9 below shows the risk evaluation matrix “Scalability of the system” asset. Like in the case of the previous asset, the criteria are mainly decided upon likelihood and consequence scale definitions and the importance rating of the asset.

		Consequence				
		Insignificant	*Minor*	*Moderate*	*Major*	*Catastrophic*
Frequency	*Rare*	**Acceptable**	**Acceptable**	**Acceptable**	**Acceptable**	**Acceptable**
	Unlikely	**Acceptable**	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**
	Possible	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**	**Must be evaluated**
	Likely	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**	**Must be evaluated**
	Certain	**Acceptable**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**

Table 5.9: Risk evaluation matrix for ”Scalability of the system” asset

Table 5.10 below shows the risk evaluation matrix ”Security of the system” asset”.

		Consequence				
		Insignificant	*Minor*	*Moderate*	*Major*	*Catastrophic*
Frequency	*Rare*	**Acceptable**	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**
	Unlikely	**Acceptable**	**Acceptable**	**Acceptable**	**Must be evaluated**	**Must be evaluated**
	Possible	**Acceptable**	**Acceptable**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**
	Likely	**Acceptable**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**
	Certain	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**	**Must be evaluated**

Table 5.10: Risk evaluation matrix for ”Security of the system” asset

5.3 Risk identification and estimation

This section covers the risk identification and estimation parts, equivalent to the steps 4 and of the CORAS method, respectively [1]. One diagram is dedicated to each asset, an abstraction level which is found to be most manageable.

Figure 5.2 below shows the threat diagram for Availability asset, presented according to the CORAS graphical notation. Three unwanted incidents, three threat scenarios, two threats and four weaknesses are identified. The relationship between the threats and the weaknesses are deduced and shown by the lines connecting them and pointing to the weaknesses. Depending on how the weaknesses can be utilised, relationship between them and the threat scenarios are drawn. Thereafter, the unwanted incidents are related to the threat scenarios and the asset. The unwanted incidents emphasised in this case ate the ones regarding availability of PATS which the system depends on, access to the database, and communication between the cellular phones and the game server.

Likelihood values are assigned to each threat scenario and unwanted incident, and consequence values (according to the definitions above) are assigned to the links between the unwanted incidents and the asset.

Weaknesses and the threat scenarios presented are the ones assumed to be most likely co cause such incidents. For example, unplanned upgrades or changes of the system parts without taking into consideration their dependencies, or network failure.

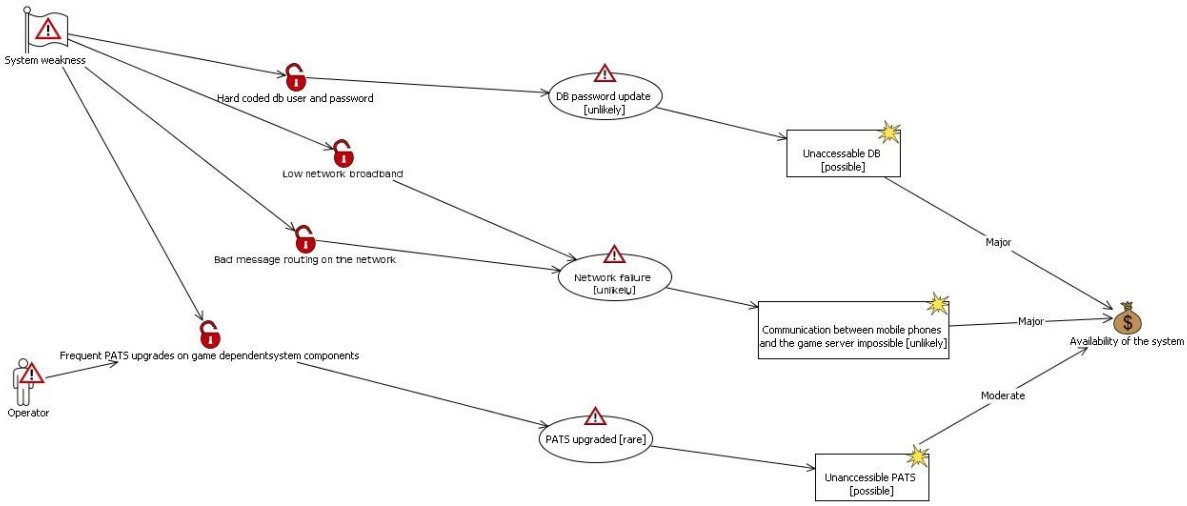


Figure 5.2: threat diagram for Availability asset

Figure 5.3 below shows the threat diagram for Correctness asset, including risk estimation values. As shown, there are three unwanted incidents that can directly harm the correctness of the system, while the weaknesses and the threat scenarios leading to these unwanted incidents can be numerous and with various likelihoods. Incorrect billing and non-traceable billing information are the incidents considered to have highest severity.

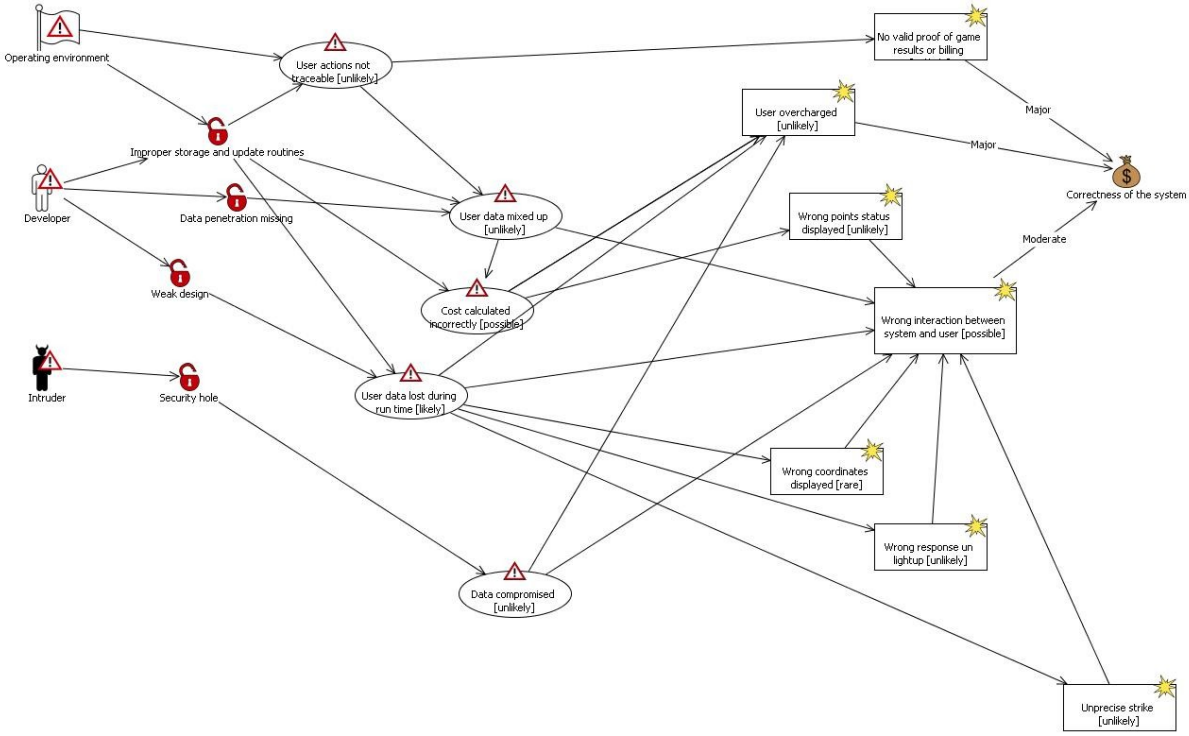


Figure 5.3: threat diagram for Correctness asset

Figure 5.4 below shows the threat diagram and risk estimation related to the Scalability asset.

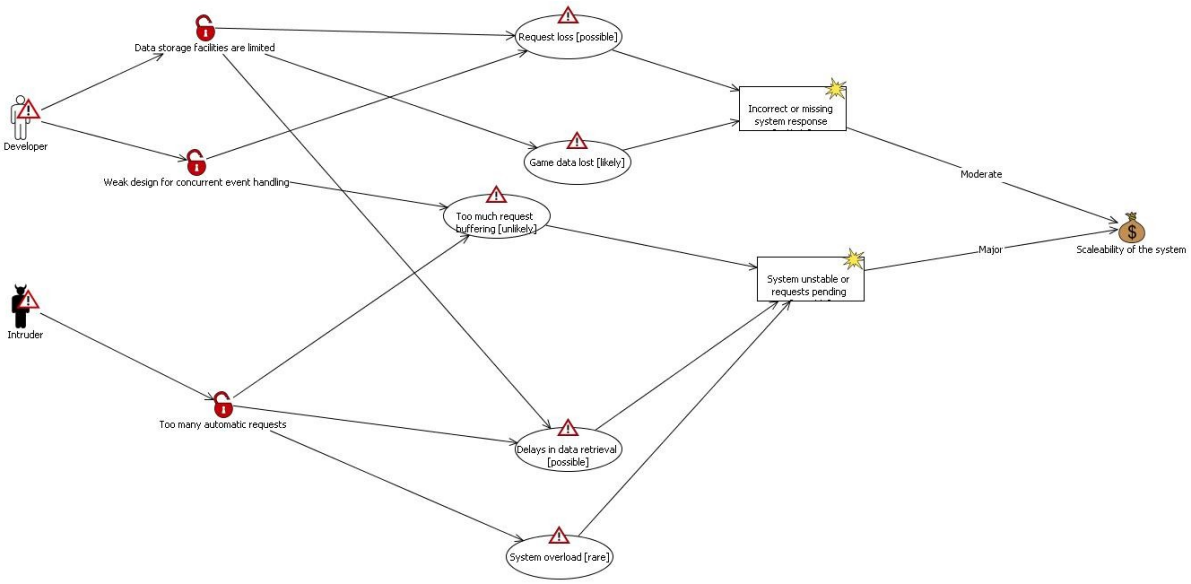


Figure 5.4: threat diagram for Scalability asset

Figure 5.5 below shows the threat diagram and risk estimation related to the Security of the system asset.

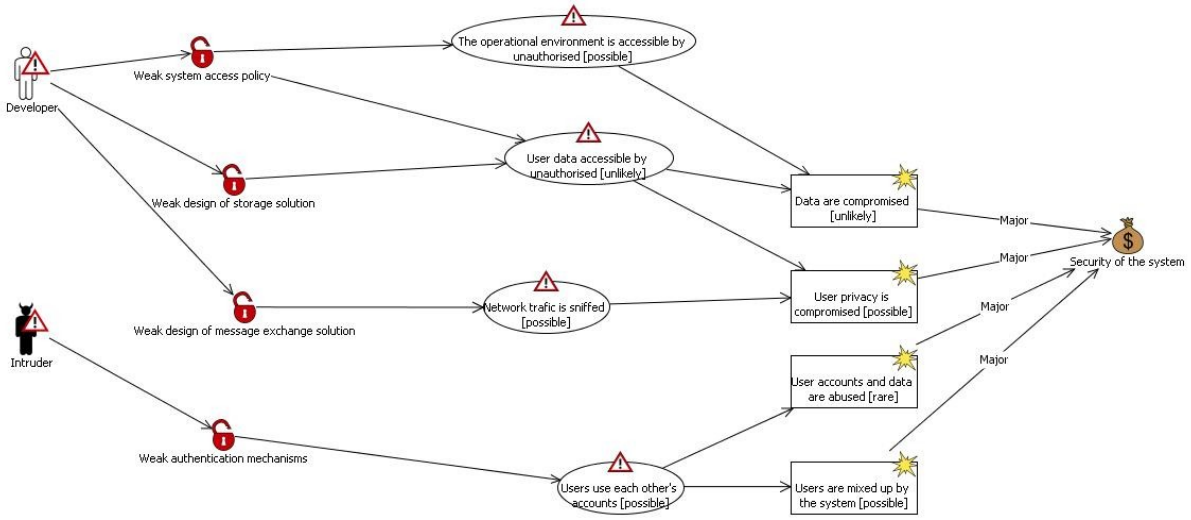


Figure 5.5: threat diagram for Security of the system asset

5.4 Risk evaluation

This section covers the risk evaluation part, equivalent to the step 6 of the CORAS method [1]. One table is dedicated to each asset.

The risks are deduced from the threat diagrams above, by taking the unwanted incidents directly harming the assets. The risks are given the appropriate abbreviations so they can fit in the risk estimation matrixes. Then, the assigned likelihood-consequence combination of each risk is noted and according too this, each risk is placed in the evaluation matrix belonging to the right asset. The colour marking the risk is the one pre-defined for the element location in the matrix, as defined in

the evaluation matrixes in tables 7 through 10.

Accordingly, the risk evaluation concerning the four respective assets result in the risks that either are acceptable (green) or should be evaluated (red), as shown in tables 11 through 14.

		Consequence				
		Insignificant	*Minor*	*Moderate*	*Major*	*Catastrophic*
Frequency	*Rare*					
	Unlikely				**PM1**	
	Possible			**UP1**	**UD1**	
	Likely					
	Certain					

Table 5.11: Risk evaluation matrix for “availability of the system” asset

		Consequence				
		Insignificant	*Minor*	*Moderate*	*Major*	*Catastrophic*
Frequency	*Rare*					
	Unlikely				**VP1, UO1**	
	Possible			**WI1**		
	Likely					
	Certain					

Table 5.12: Risk evaluation matrix for ”Correctness of the system” asset

		Consequence				
		Insignificant	*Minor*	*Moderate*	*Major*	*Catastrophic*
Frequency	*Rare*					
	Unlikely			**IM1**		
	Possible				**SU1**	
	Likely					
	Certain					

Table 5.13: Risk evaluation matrix for ”Scalability of the system” asset

		Consequence				
		Insignificant	*Minor*	*Moderate*	*Major*	*Catastrophic*
Frequency	*Rare*				**UC1**	
	Unlikely				**DC1**	
	Possible				**UPC1, UM1**	
	Likely					
	Certain					

Table 5.14: Risk evaluation matrix for ”Security of the system” asset

5.5 Risk overview

Risk overview diagrams depict the risks identified, both acceptable and the ones that should be evaluated, as identified above. The figures 6 through 9 below display the risks with their relations to the threats and the assets (consistent to the threat diagrams above) with the right abbreviations and the evaluation result (acceptable/must be treated). This representaton is called threat overview diagrams [1] and offers a better overview of the risks identified and their relationship to the threats.

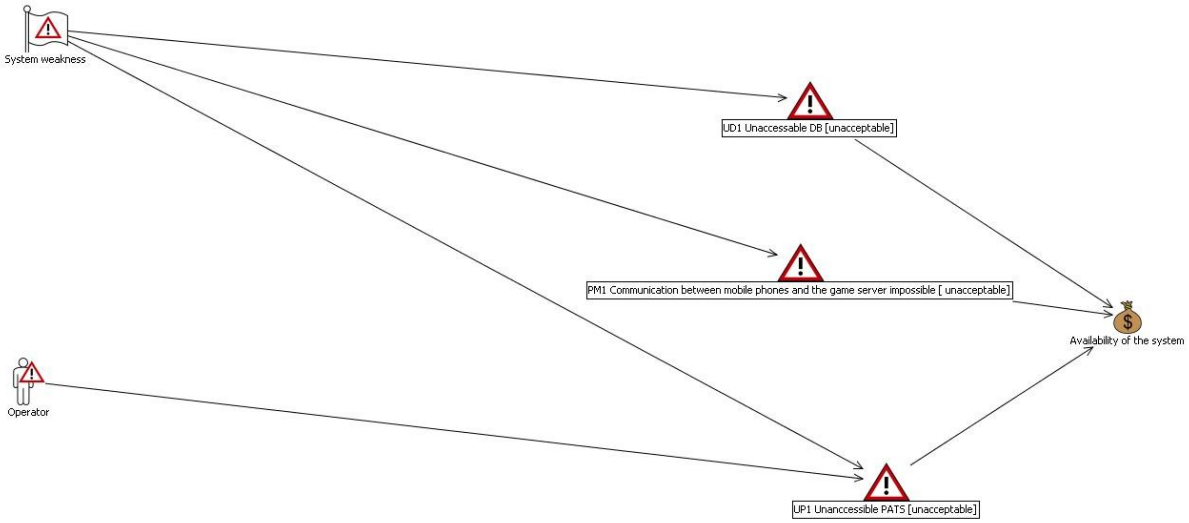


Figure 5.6: risk overview diagram for Availability asset

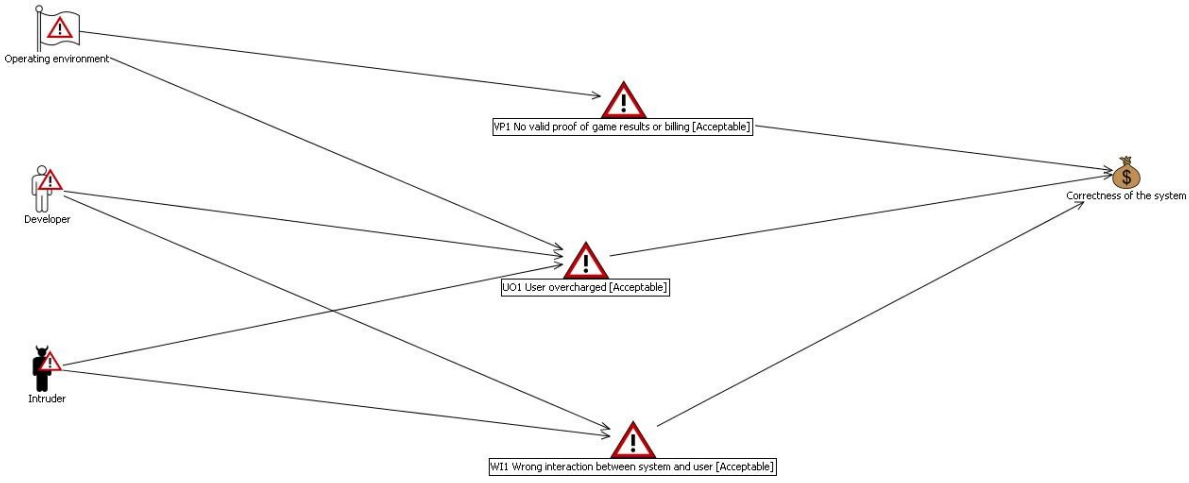


Figure 5.7: risk overview diagram for the Correctness asset

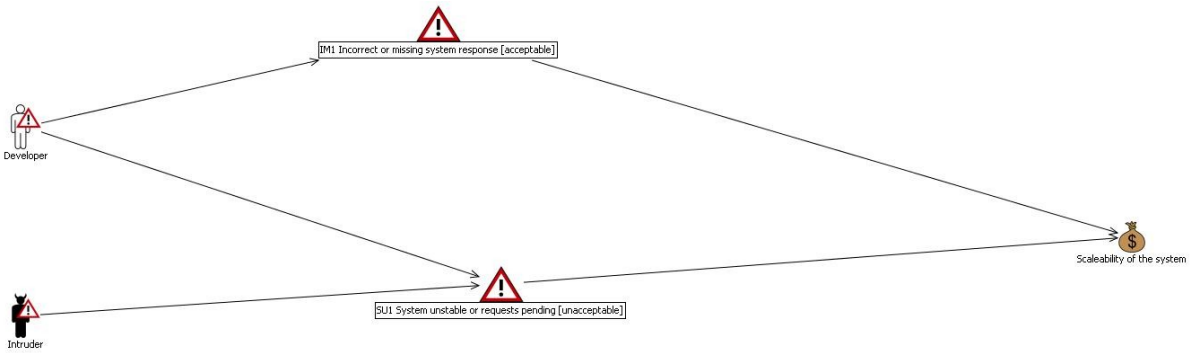


Figure 5.8: risk overview diagram for the Scalability of the system asset

Figure 5.9: risk overview diagram for the Security of the system asset

5.6 Risk treatment

The risks categorised as the ones that must be evaluated (red marked elements in the estimation tables) are further analysed and the possible treatments are proposed wherever appropriate (on weaknesses, threat scenarios, unwanted incidents and the consequences), so that the treatments provide the conditions for the risks to become acceptable.

Treatment diagrams display the risks that must me evaluated only (with their abbreviated names), the suggested treatments on the appropriate parts – otherwise the diagram is similar to the threat diagrams with likelihood estimates.

Followed by the treatment diagrams are the treatment overview diagrams including the risks identified as the ones that must be evaluated, their relationship with the threats and the assets, and the treatments as shown on the corresponding treatment diagrams, but this time directly pointing to the risks they are meant to treat regardless of the stage (voulnerability/threat scenario, consequence or unwanted incident).

Figure 5.10 below presents treatment diagram for Availability asset including the risks evaluated as the ones that must be evaluated.

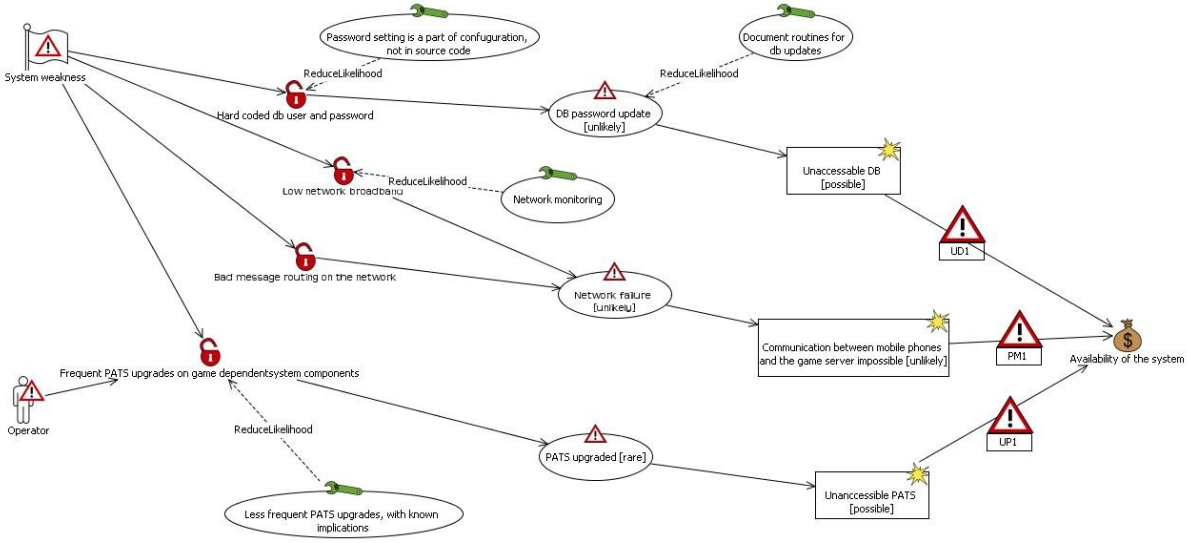


Figure 5.10: treatment diagram for Availability asset

Figure 5.11 below presents treatment overview diagram for Availability asset including the risks identified as the ones that must be evaluated.

Figure 5.11: treatment overview diagram for Availability asset

All the risks on the correctness diagram are acceptable, hence no treatment is proposed for them.

Figure 5.12 below presents treatment overview diagram for Scalability asset. Only the SU1 risk is unacceptable, so that is the only treated risk related to scalability asset.

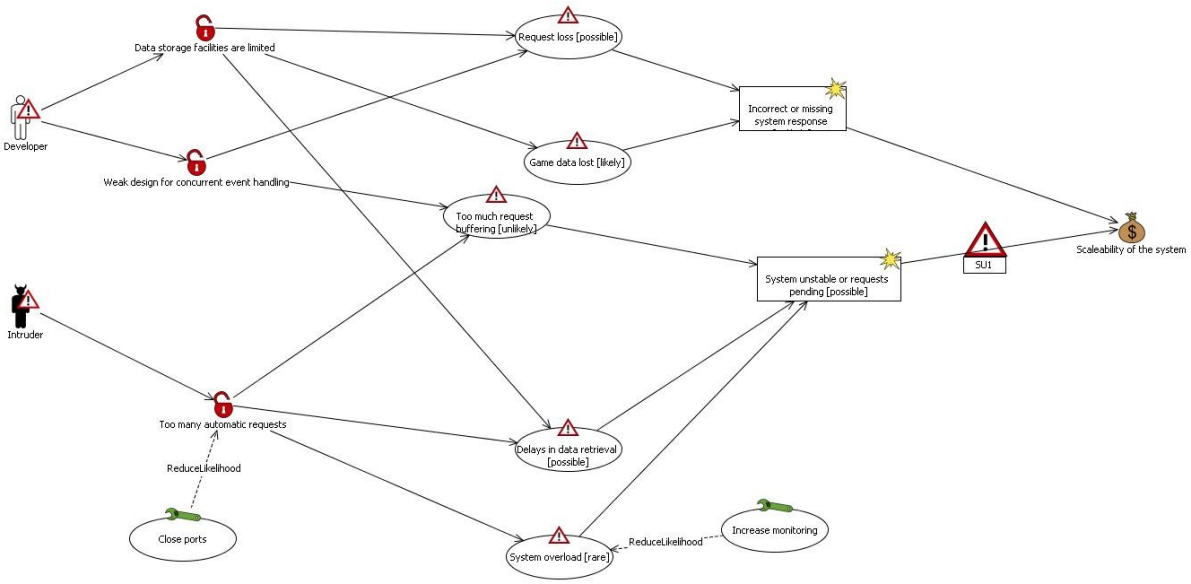


Figure 5.12: treatment diagram for Scalability asset

Figure 5.13 below presents treatment overview diagram for Scalability asset including the risk identified as the one that must be evaluated.

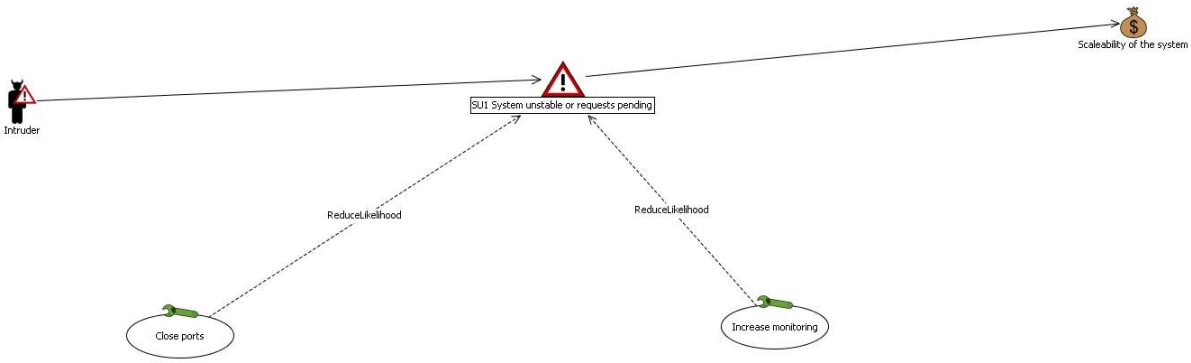


Figure 5.13: treatment overview diagram for Scalability asset

Figure 5.14 below presents treatment overview diagram for Security asset. In this case three out of four risks are unacceptable.

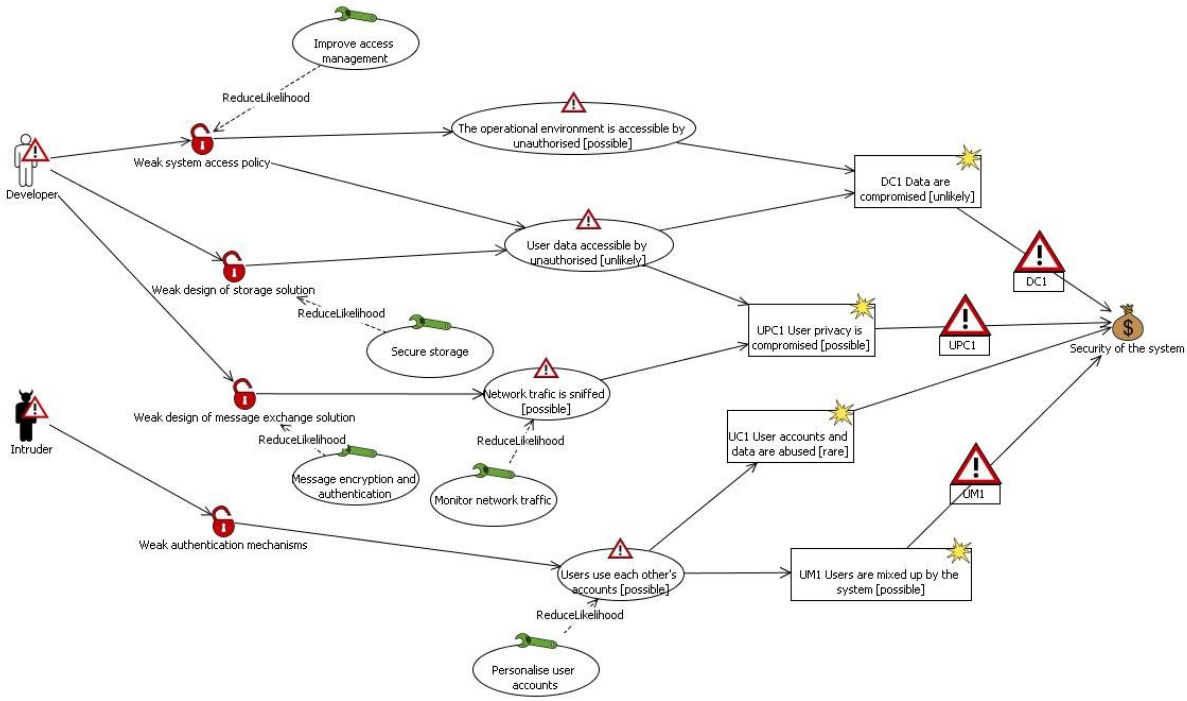


Figure 5.14: treatment diagram for Security asset

Figure 5.15 below presents treatment overview diagram for Security of the system asset including the risks identified as the ones that must be evaluated.

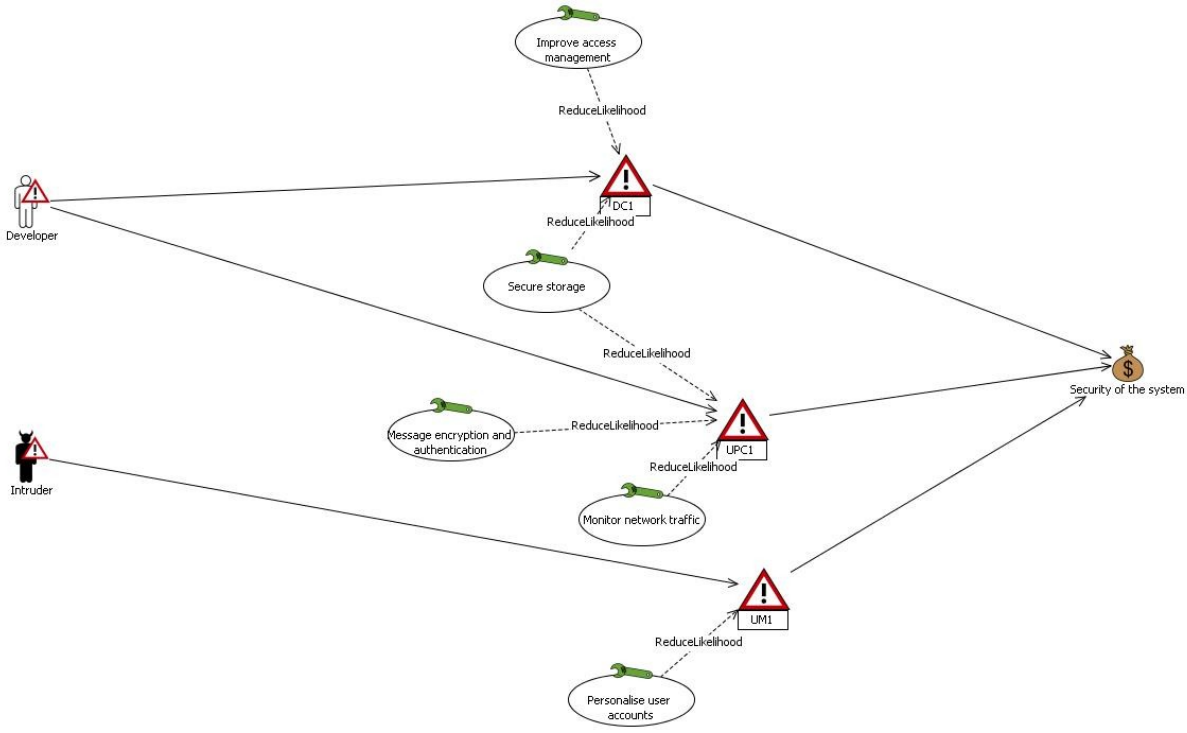


Figure 5.15: treatment overview diagram for Security of the system asset

References

[1] Braber, Hogganvik, Lund, Stølen, Vraalsen “Model-based security analysis in seven steps – a

guided tour to the CORAS method" January 2007

[2] Dahl, Hogganvik, Stølen "Structured semantics for the CORAS security risk modelling language" September 2007-11-22

[3] CORAS modelling language http://coras.sourceforge.net/ (retrieved November 2007)