METU DEPARTMENT OF COMPUTER ENGINEERING
CENG491 Computer Engineering Design I

REQUIREMENT ANALYSIS REPORT
for
Audio-Visual Gaming Network

Prepared By
'Eggs On The Door'

Serap İNCE
Engin ERTAŞ
Duygu ÇELİK

ANKARA
2010

TABLE OF CONTENT

1 Introduction

1.1 Problem Definition

Recently, millions of people have been using the social network sites on the Internet. Facebook and Mynet are popular, since they not only provide people with chatting feature with millions, but also give the opportunity to play multi-player games simultaneously. However, one of the most crucial lacks of these networks is the fact that they do not give users the capability of communicating in an audiovisual manner. On the other hand, there are some software, such as, Windows Live Messenger or Skype providing visual communication; still, they do not support those facilities to more than two users on the same conversation window. Moreover, more than two people cannot play online games using this software at the same game board.

1.2 Purpose

In this project, we propose to develop a video conferencing software in which people can play games that require real time audiovisual interaction between players, such as “taboo”.

The first aim of the project is to provide people with a framework in which two or more people can make audiovisual conferencing in the same window.

Secondly, we aim to develop a flexible gaming platform / console on which various multi-player games written for this software can be executed.

1.3 Scope

Our project will be a .NET based Audiovisual Video Conferencing Gaming Network. Our project will assist the people’s needs to make video conferencing with multiple people and also play games in the same platform. The users can be either clients or host in the system. Any user who wants to create a game table can be the host, and invite people from the friend-list to play the game.In such a case, the host should provide her/his IP address and a password to the clients which is the requirement to join the game. During the game, people will be able to make audio-visual conferencing which is serves a feature differing the software from the present gaming platforms. Our system is aimed to handle 6 people at the same time and respond in a specified time, but due to the bandwidth limits or any unsupported system requirements the performance may decrease. Our aim is to provide a platform that different games can be implemented so that it will be easy to improve the project in the future.

1.4 Market Survey

There are many online gaming networks used by millions of people all around the world. Nowadays, Facebook is one of the most popular gaming networks. Plenty of various kinds of games can be played on those platforms, while people are chatting at the same time. However, audiovisual interactions are not provided in most of them.

In January 2010, Skype developed a new add-on which enables users to make multi-user video chat supporting at most five people simultaneously. The product is expected to attract businessmen's attention. Skype stated its subscriber base was currently growing at a rate of 400,000 new users per day, internationally. The managers of Skype told that its free service had generated 250 billion calling minutes since launching in 2005 and accounted for 12 percent of international call minutes last year. The product is widely used; and as a conclusion, we can say that adding online game ability to the audiovisual conference software will even make it more popular.

"ooVoo" is another option we found for video conferencing, which is Windows-based software. It offers two kinds of licenses: free and paid. Whereas free version supports up to three users video-chat, paid version supports up to six.

Another application is TokBox which is a web-based flash application letting the user to host conference chats, and invite as many people as they like. The users send the URL to the people wanted to be invited; and guests may get into the chat, just by going to the URL.

One of the notable applications in market is MeBeam. It is used with the Skype software by adding the MeBeam in the contact list. With the help of this facility, the users can make multi-user video conferencing whose user number is expandable up to 16 people.

Recently, the audiovisual conferencing products are used for business issues; for example, to make meetings with people who are far away from meeting area. The newly developed products also plan to appeal small companies' interests. We can say that the most of present audiovisual conferencing software do not address the demands of people who desire using the video conferencing application for entertainment and games.

In our approach, we intend to answer the need of video conferencing for entertainment purposes which is not satisfied by the current products.

1.5 Definitions and Abbreviations

SRS : Software Requirement Specification
CLI: Command Line Interfaced
GUI: Graphical User Interface

1.6 References

IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications

1.7 Overview

Readers get a clear understanding of the project and project requirements by examining this requirements analysis report. With the help of the sequence, behaviour, and use-case diagrams, the behaviour of the product and the functions of the system are defined clearly to the user. The system requirements for both the development and usage stages are stated. The user interfaces and the information transmission between the states of the product is made clear to the user.

2 Overall Description

2.1 Product Perspective

Our Audiovisual Gaming Software is divided into two main modules: *Network* and *GamePlay* modules. While *Network* module mainly focuses on the network communications between server and clients, video streaming, compression and filtering, *GamePlay* module supports various games with the help of designed interface.

- *Network* is the key module of the system separated into four sub-modules: *Streaming*, *Filter, Compression and NetworkSettings*.

- *GamePlay* module is responsible for connecting with game executables designed for this software. There are going to be many games supported by our product. Therefore, in order to link these games to the software, we need a powerful interface in this module. There are two sub-modules of *GamePlay*: *GamePlayInterface* and *GamePlaySettings*.

- One of the most crucial parts of our software is *compiled Java games* supported by our software. By the help of those, people may play enjoyable games during audiovisual conversation. Any game communicating with *GamePlayInterface* module with correct interface messages are supported by our software. As a default game in our software, we include Taboo game to be played.

The overall scope of the project is given on *Figure1*, and the relevant data-flow diagrams are shown on Figure 2 and 3.

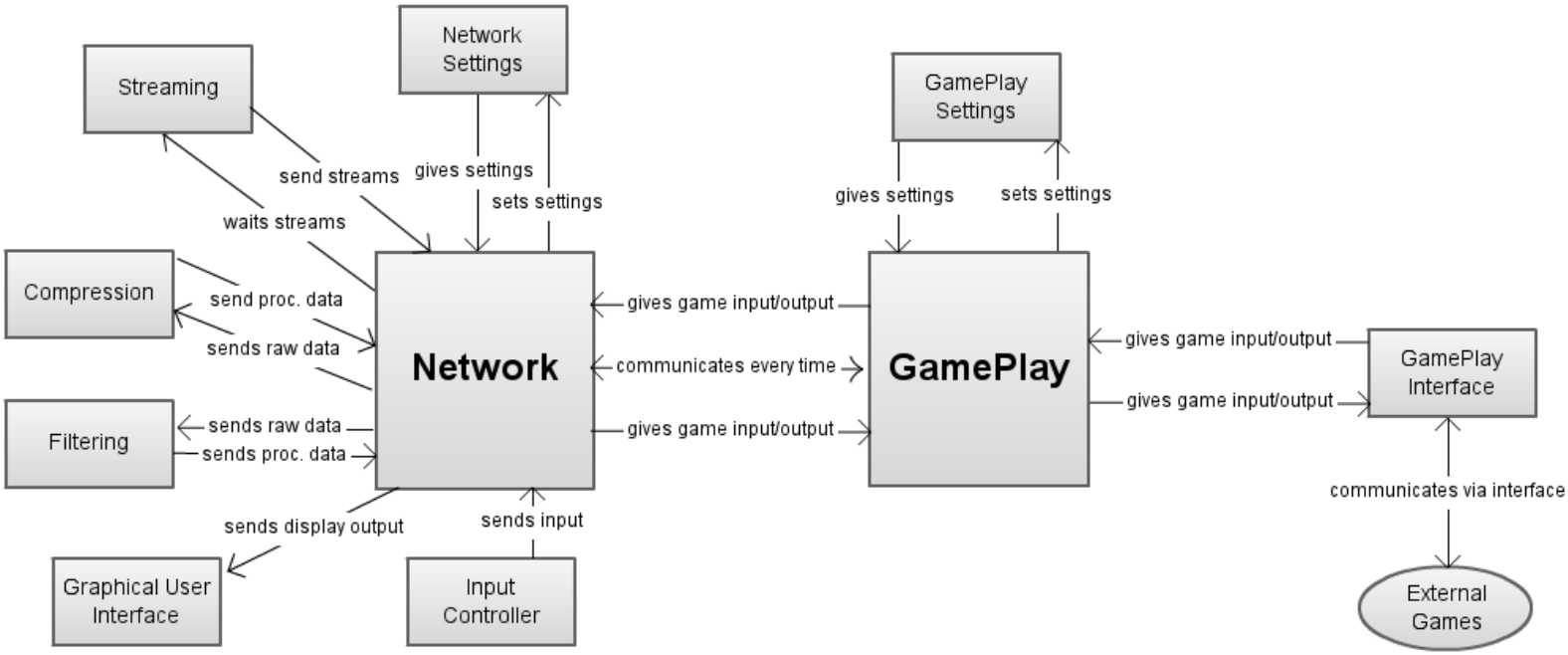


***Figure 1**: Overall description of the project*

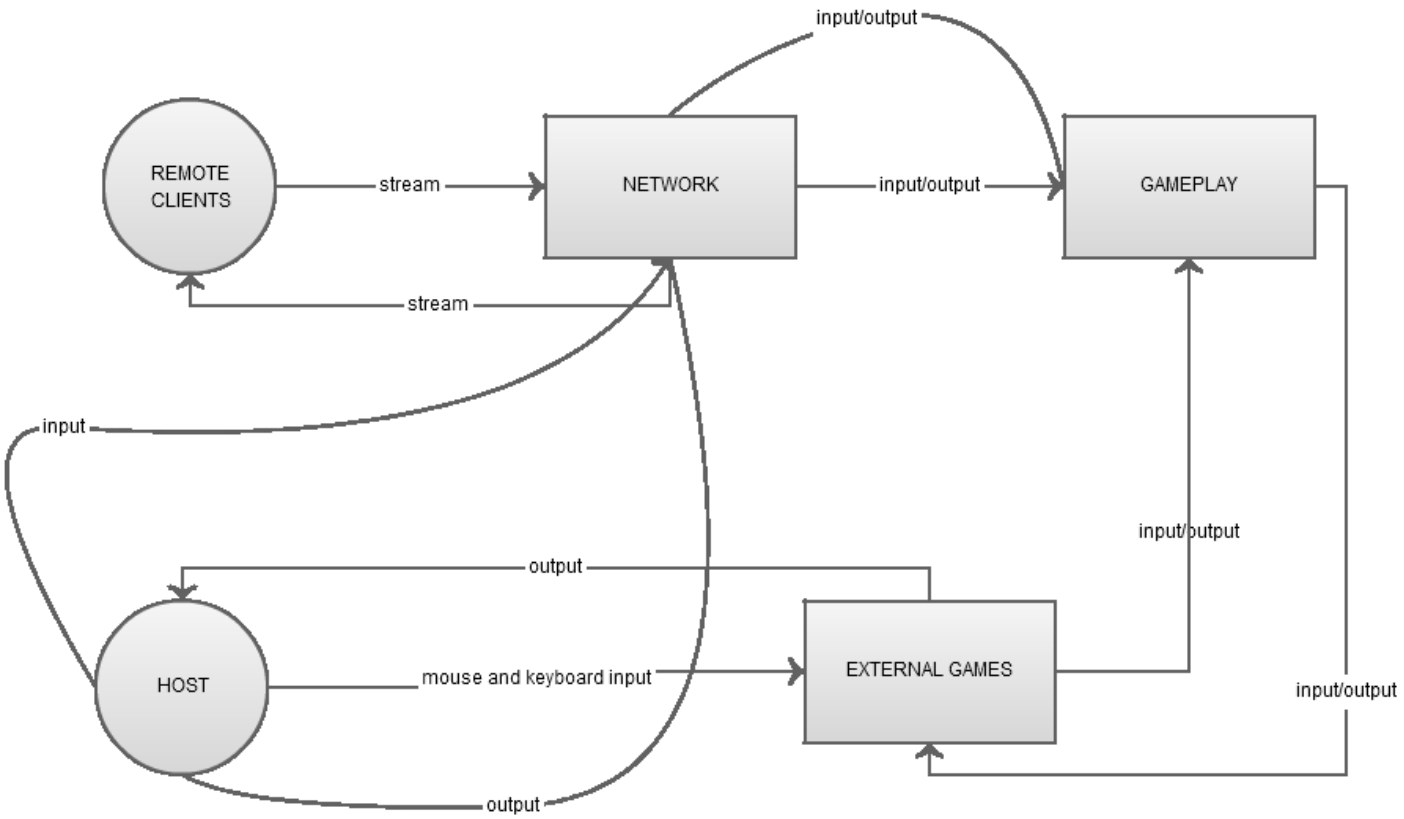


Figure 2: *Level 1 Data Flow Diagram of Software*

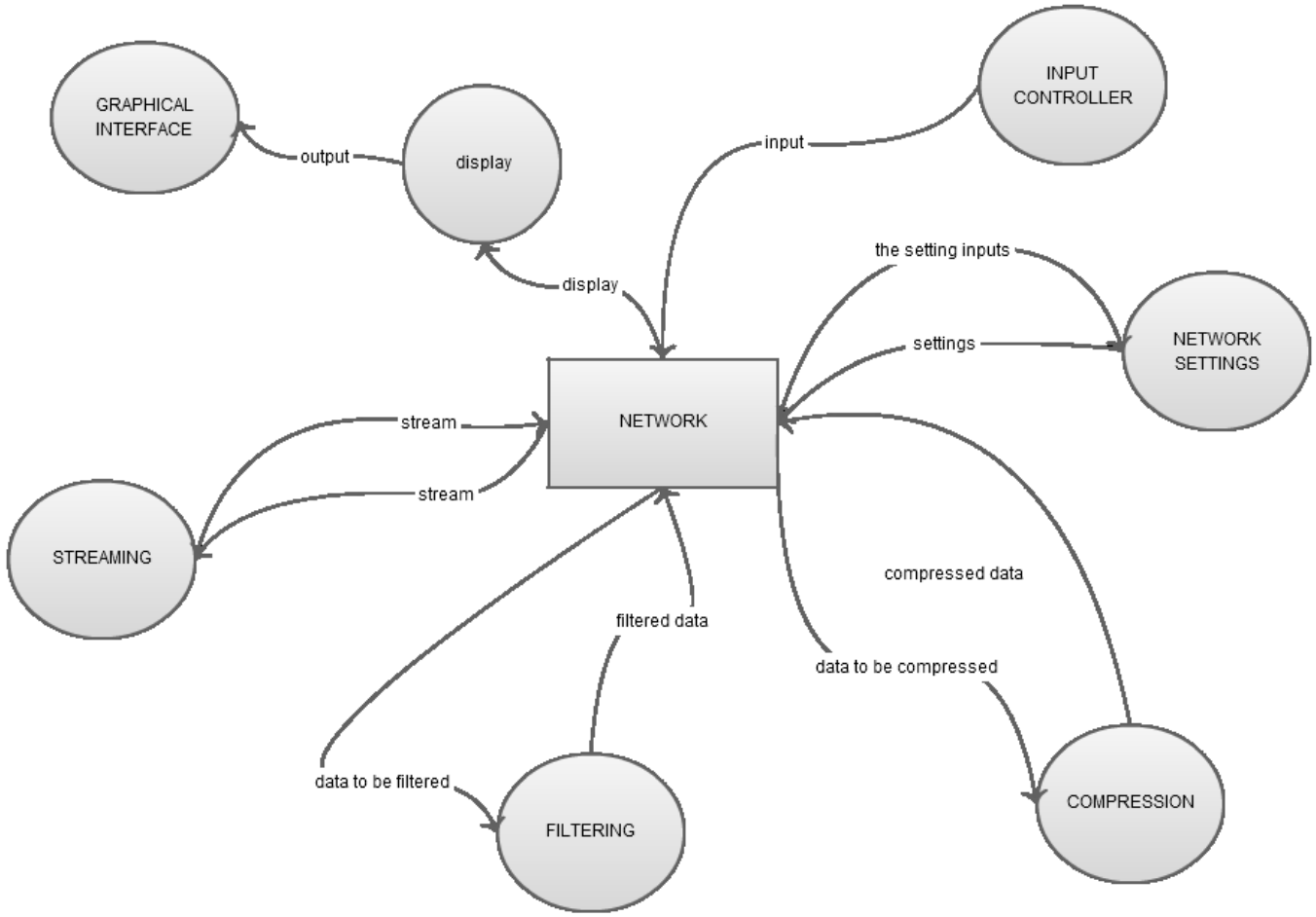


Figure 3: *Level 2 Data Flow Diagram of Network Module*

2.2 Product Functions

System functions that answers to all requirements are stated below. They are grouped into six parts, each of which serves different main purposes that are explained in details in the *3.2 - Functional Requirements* part.

2.2.1 Conversation

By the help of these functions, a user can start a new conversation or join existing conversation, leave the session that he or she logged in. Remaining ones are related to visibility settings via changing camera state.

- *Start Conversation*
- *Join*
- *Invite*
- *Respond to Invitation*
- *Leave Conversation*

- *End Conversation*
- *Respond to End Conversation Demand*
- *Disable Camera*
- *Enable Camera*

2.2.2 Activity Functions for Users

Users perform variety of operations depending on the state. Tasks about games and chats called *"activities"* are carried out by the following system functions:

- *Start Game*
- *End Game*
- *Respond to Start Game Demand*
- *Respond to End Game Demand*
- *Send Chat Message*
- *Send Personal Message*
- *Find in Text*

2.2.3 User/Friend List

Users may either add their friends on user list or delete them from list. Following system functions perform issues related to user lists.

- *Add User to User List*
- *Delete User from User List*
- *Update User Information on List*
- *Sort Users By Username*
- *Sort Users By IP*

2.2.4 Settings

The settings of two main modules, namely *Network* and *GamePlay* are contained in their sub-modules *NetworkSettings* and *GamePlaySettings*. These modules are *set* by the following system functions. In case of requiring information about the settings modules, *get* functions are used by the system.

- *Get Network Settings*
- *Set Network Settings*
- *Get GamePlay Settings*
- *Set GamePlay Settings*
- *Get View Preferences*
- *Set View Preferences*

2.2.5 Logs & Diaries

In our system, data related to conversations and past connections are kept. Following system functions deal with these issues.

- *View GamePlay Diary*
- *Show Game Result*
- *Delete Game Result*
- *View Chat Diary*
- *Show Chat Details*
- *Delete Chat*

2.2.6 Miscellaneous

Additional features of the system are carried out by the following system functions. This part includes functionalities independent of our two main modules: *Network* and *GamePlay.*

- *View About Us*
- *View Help Contents*
- *Download Game*
- *Communicate with Us*
- *Learn Self IP*

2.3 Constraints, Assumptions and Dependencies

2.3.1 Functionality State Constraints

There are three functionality states of our software determining which system functions are going to be available for users. These states are described on ***Figure 4*****.**

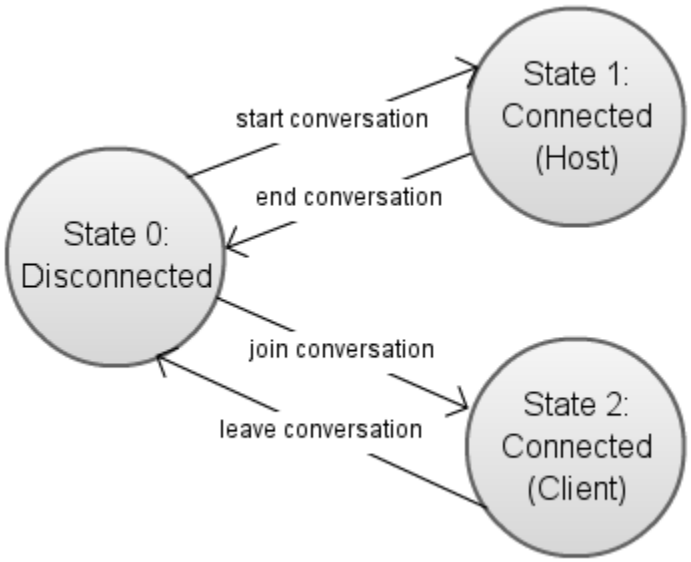


Figure4: *Functionality States*

2.3.2 User Constraints and Characteristics

There are two user types of the system: *Host* and *Client*. Which system functionalities that system users can access is classified by these user types. However, the system users in state 0 have not a type; therefore, a user can reach to any functionality belonging to state 0.

The users are classified according to this rule:

- If the user starts a conversation, he/she turns out to become *Host*.
- If the user joins a conversation, he/she turns out to become *Client*.

Whereas state 0 user functionalities are described on ***Figure 5,*** state 1-2 user functionalities are described on ***Figure 6 .***

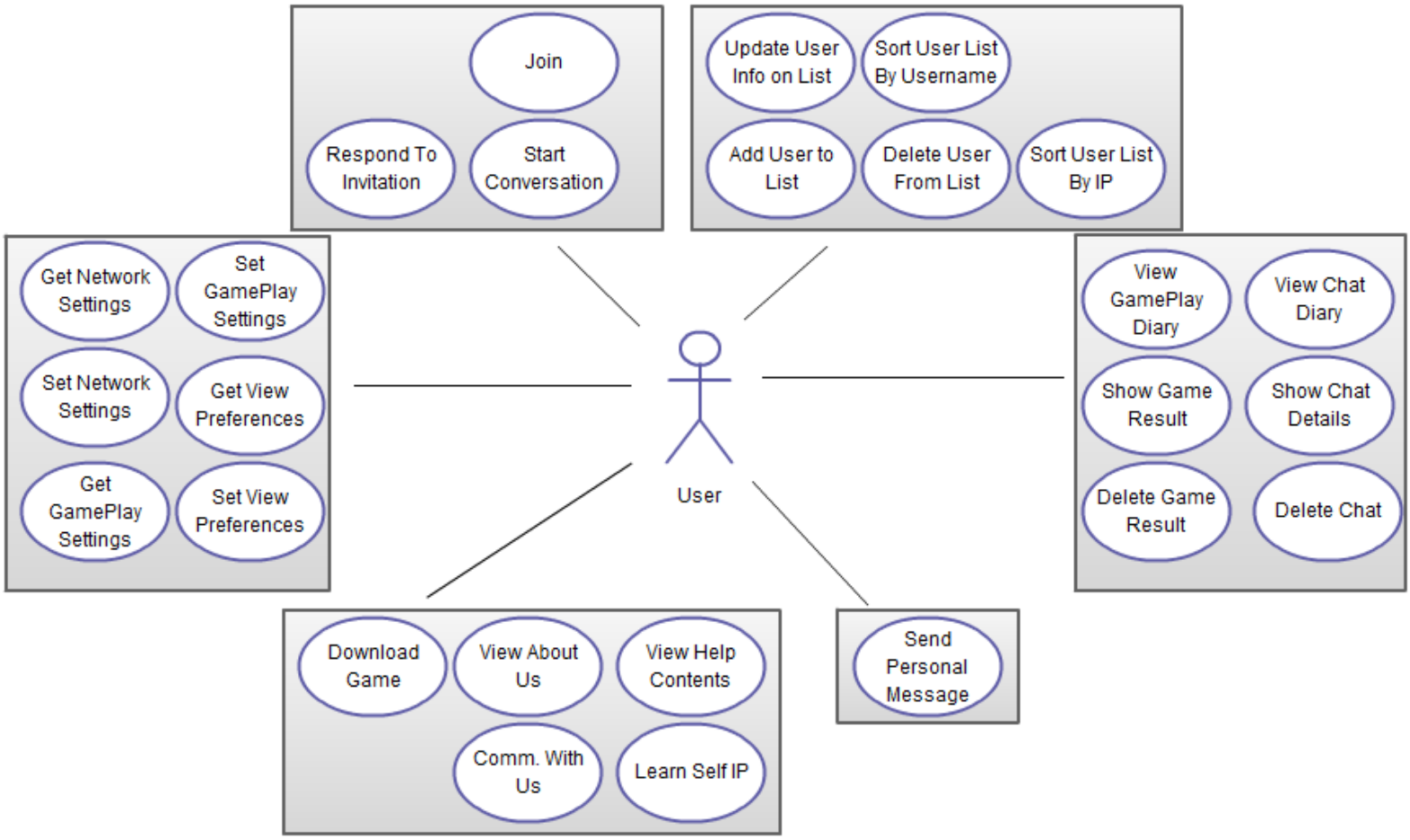


Figure 5: *Use-case diagram for users in State 0.*

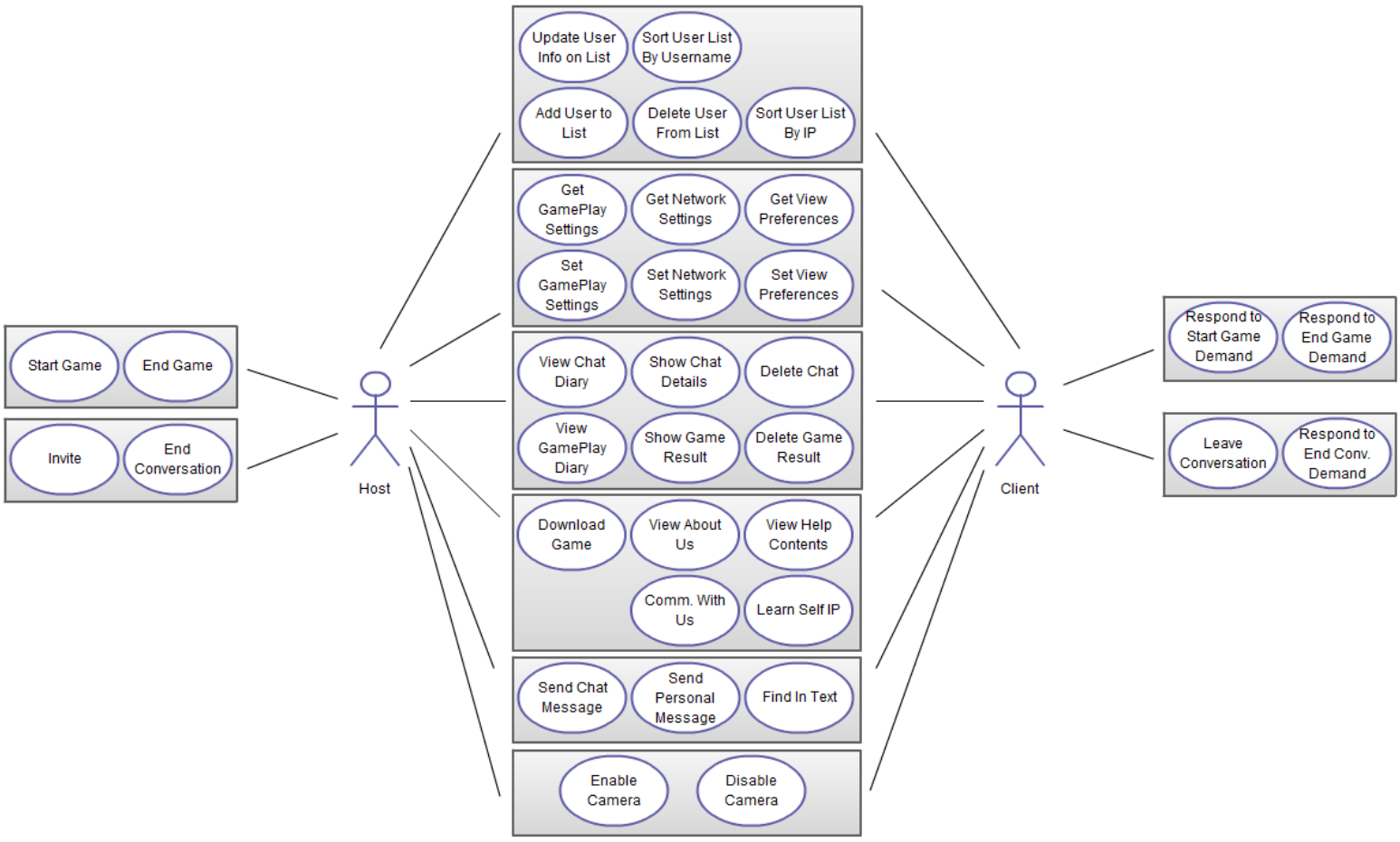


Figure 6: *Use-case diagram for users in States 1 and 2.*

3 Specific Requirements

3.1 Interface Requirements

3.1.1 External Game Interface

One of the most crucial parts of our software is *External Game Interface*. By the help of this interface, our software could communicate with the supported Java games correctly, which enables users to play games during their audiovisual conversation.

Java games must communicate via interface with the help of interface commands in order to be supported (As illustrated on *Figure 1*). Java Games need remote inputs to work properly. Therefore, we need to determine limited set of commands which covers all the input/output needs.

In our project, this interface and interface commands are contained in *GamePlayInterface* sub-module of *GamePlay* module. External games sends these commands and gets relevant results with the help of *command line interface (CLI)* of *GameInterface* module. This CLI cannot be seen by users during game execution. Our default game Taboo that we will code also must use this CLI in order to work correctly.

Here are the CLI commands that an external Java game may use:

- getPlayerCount
    - returns current number of players
    - return type: integer
    - return range: [1, 10]

- getPlayerOrder
    - returns the order of current player on table
    - return type: integer
    - return range: [1,10]

- setPlayerPositions
  <player order> <x coordinate> <y coordinate>
  <player order> <x coordinate> <y coordinate>
  <player order> <x coordinate> <y coordinate>
  …..
  ….. // n more lines seperated by newline
  …..
  endSet
    - makes the software adjust the player positions on the table
    - Use:
        - On the first line, “setPlayerPositions” must be written

        - ■On the last line, “endSet” must be written
        - ■On the lines between the first and last lines,
            “<player order> a <x coordinate> b <y coordinate>”
            must be written
        - ■every argument in < > must be written as integer
        - ■a and b are either *space* or *tab* characters.
    - returns “true” if command is accepted
    - return type: string
    - return range: [“true”, “false”]

- setGameWindow <x coordinate> <y coordinate> <width> <height>
    - makes the software clear the game area in order to locate game window
    - Use:
        - ■“setGameWindow” must be written
        - ■At the same line,
            “<x coordinate> a <y coordinate> b <width> c <height>”
            must be written
        - ■<x coordinate> and <y coordinate> are the coordinates of *bottom-left* corner of game window.
        - ■<width> is the coordinate difference along *x-axis* and <height> is along *y-axis.*
        - ■every argument in < > must be written as integer
        - ■a and b are either *space* or *tab* characters.
    - returns “true” if command is accepted
    - return type: string
    - return range: [“true”, “false”]

- start
    - makes the software start the game
    - returns “true” if command is accepted
    - return type: string
    - return range: [“true”, “false”]

- cancelStart
    - makes the software cancel the start of the game
    - returns “true” if command is accepted
    - return type: string
    - return range: [“true”, “false”]

- getRemoteActivity
    - returns what remote player does in his turn
    - does not return until message from remote computer comes
    - return type: string

- return range: -

- sendActivity
  ……
  ……. //<message>
  ……
  endSend
  - sends what the player in current computer does in his turn
  - Use:
    - sendActivity must be written first of all,
    - Any text to be delivered must be written next,
    - In order to make the command completed, endSend must be written at the end of command.
  - returns “true” if command is accepted
  - return type: string
  - return range: [“true”, “false”]

- writeGameResult
  - makes the software write diary.log file to the *Game Result Diary* in the software
  - Use:
    - diary.log file must be written by the external game before executing this command
    - after writing is complete, only “writeGameResult” command must be written.
  - returns “true” if command is accepted
  - return type: string
  - return range: [“true”, “false”]

- end
  - makes the software end the game
  - returns “true” if command is accepted
  - return type: string
  - return range: [“true”, “false”]

3.2.2 Graphical User Interface

Network, one of the main modules of our software needs an effective user interface in order to increase the usability of our software. Therefore, we decided to create a GUI to satisfy that need. In order to create a GUI, we are going to use MS Silverlight software.

Here is one example of the main window of our software belonging to *Network* module.

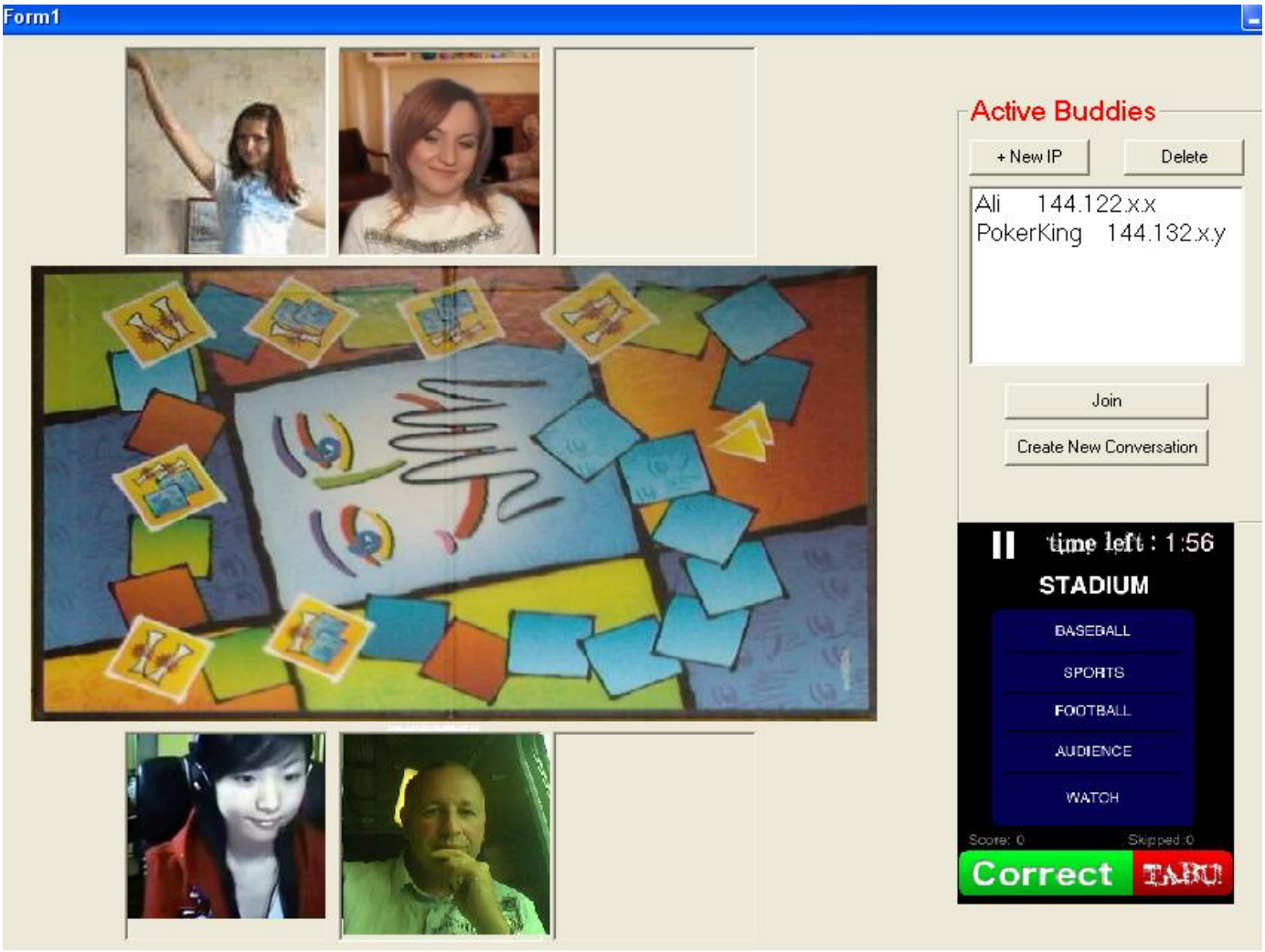


Figure 7*:* An example user interface*.*

3.2 Functional Requirements

3.2.1 Function Descriptions

- ***Start Conversation***
  It is used to start a conversation. User using this functionality turns out to be *Host.*
- ***Join***
  User may prefer to join an existing game instead of creating ones on her/his computer. *Join* system function handles that case during system progress.
- ***Invite***
  Additional users may be invited to an existing conversation. The invitation can be sent only by the host to the users who have not joined any game session yet.

- ***Respond to Invitation***
  Invited user may or may not attend to the game to which she/he has been invited. *Respond to Invitation* system function returns the information whether or not the invited user will join the game.
- ***Leave Conversation***
  Users, except the host, can leave the session they have been joint. Hosts are not allowed to leave the session unless they are not received the approval from the other participants. More detailed explanations given in the related parts.
- ***End Conversation Demand***
  In case of the user who wants to leave the conversation is the host for the current conversation, he/she has to make the others know that the session will end. The *End Conversation Demand* system function operates together with another system function, namely *Respond to End Conversation Demand.*
- ***Respond to End Conversation Demand***
  Respond to End Conversation Demand system function returns the information whether or not participants approved the host's demand to end the conversation.
- ***Start Game***
  After the conversation is established between all participants by joining to existing conversation or accepting the invitations, users may start to play. *Start Game* system function can only be used by the *Host* of conservation.
- ***Respond to Start Game Demand***
  Before the game is started by the *host*, the *clients* respond him/her to give the information whether they will join or not. Depending on the result of this system function, a new game is started or not.
- ***End Game***
  End Game system function enables the host to terminate the current game. This system function requires the approval of the end game demand by the clients.
- ***Respond to End Game Demand***
  Like the *Respond to Start Game Demand* case, this system function also operates to inform the participants, in case the host demands to end the current game. The current game is ended according to the return value of this function.
- ***Send Chat Message***
  There will be an area visible to all users, via which all the players can chat each other. *Send Chat Message* function provides group based communication in written case.
- ***Send Personal Message***
  Users are capable of sending messages only to the ones that they choose. A call to *Send Personal Message* function prompts the chat feature for only the chosen users.
- ***Add User to User List***
  Add User to User List system function enables all the users to add their friends on their user l lists.

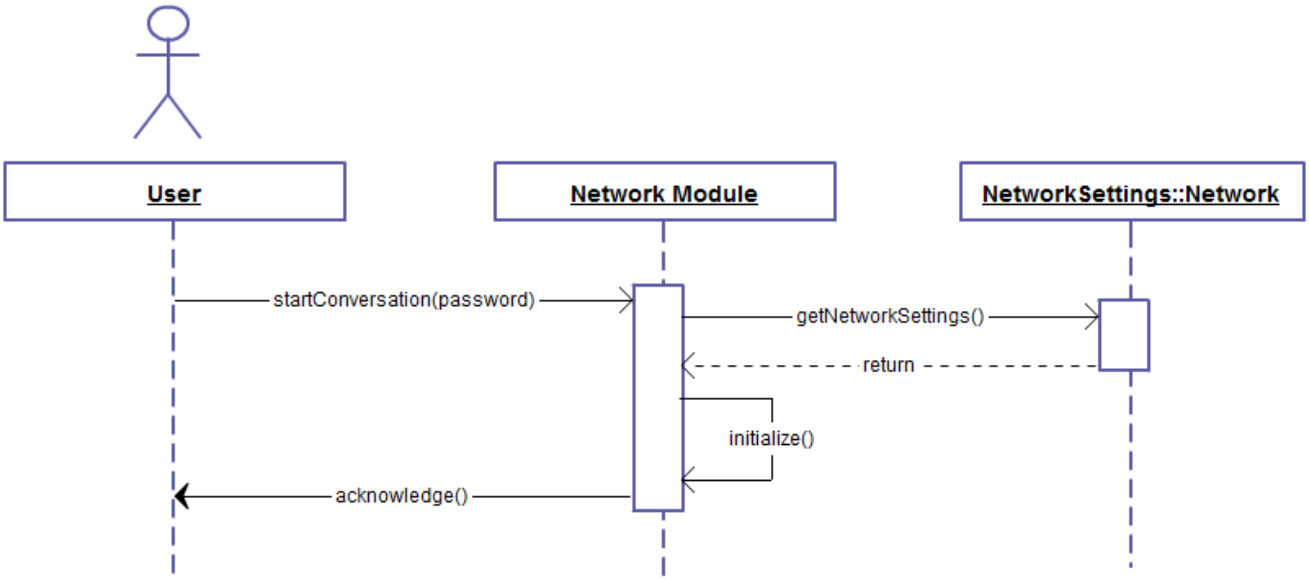
- ***Delete User from User List***
  Delete User from User List system function enables all the users to delete a user from their user lists.
- ***Update User Information on List***
  Update User from User List system function enables all the users to update the properties of a user on their user lists.
- ***Sort Users by Username***
  As the name serves the meaning, the function call sorts the player list alphabetically.
- ***Sort Users By IP***
  As the name serves the meaning, the function call sorts the player list according to IPs.
- ***Set Network Settings***
  Settings including bandwidth, maximum number of connections, etc. are done via *Set Network Settings* function.
- ***Get Network Settings***
  The values of previously set components of network settings are achieved using this system function.
- ***Set GamePlay Settings***
  Settings related to *GamePlay* module are done via the system call *Set GamePlay Settings.*
- ***Get GamePlay Settings***
  The settings of *GamePlay* module are reached via *Get GamePlay Settings* function call.
- ***Set View Preferences***
  The project offers a set of options for the users to customize their program's appearance. *Set View Preferences* system function handles these preference issues.
- ***Get View Preferences***
  Previously set appearance preferences are gotten via the system function *Get View Preferences.*
- ***View GamePlay Diary***
  The logs of the actions related to *GamePlay* module are kept by the system. Users, either the host or the clients, view the logs via *View GamePlay Diary* system function.
- ***Show Game Result***
  The data related to the previously played game are kept and *Show Game Result* function call is used to display this data by the users.
- ***Delete Game Result***
  The users are allowed to delete the results of previous games via the *Delete Game Result* function.
- ***View Chat Diary***
  Logs of chats are kept by the system. The public logs can be accessed by all the users of a certain session, however, the personal chat logs are made visible only to the participants of that conversation, as it is supposed to be.
- ***Show Chat Details***
  Some details, such as, when the conversation was done, who were the participants,etc, are " kept by the system and these data are obtained via the *Show Chat Details* function call.

- ***Delete Chat***
  The logs of chats are deleted when demanded via the *Delete Chat* system function.
- ***View About Us***
  This parts includes information about our group **'Eggs on the Door'**, members, and the project.
- ***View Help Contents***
  A manual will be made available to users to help them while using the software.
- ***Download Game***
  This system function exists for the users to get the executable of the software.
- ***Communicate with Us***
  Via this system function, the users communicate with us, the members of Eggs on the Door.
- ***Learn Self IP***
  Undertaking the task, the host may need to know his/her IP to supply some users with this data and make them connect to his/her conversation. The function call *Learn Self IP* gives this opportunity to the users in some cases, if they demand this.
- ***Find in Text***
  Users can search their logs and chats via *Find in Text* system call.

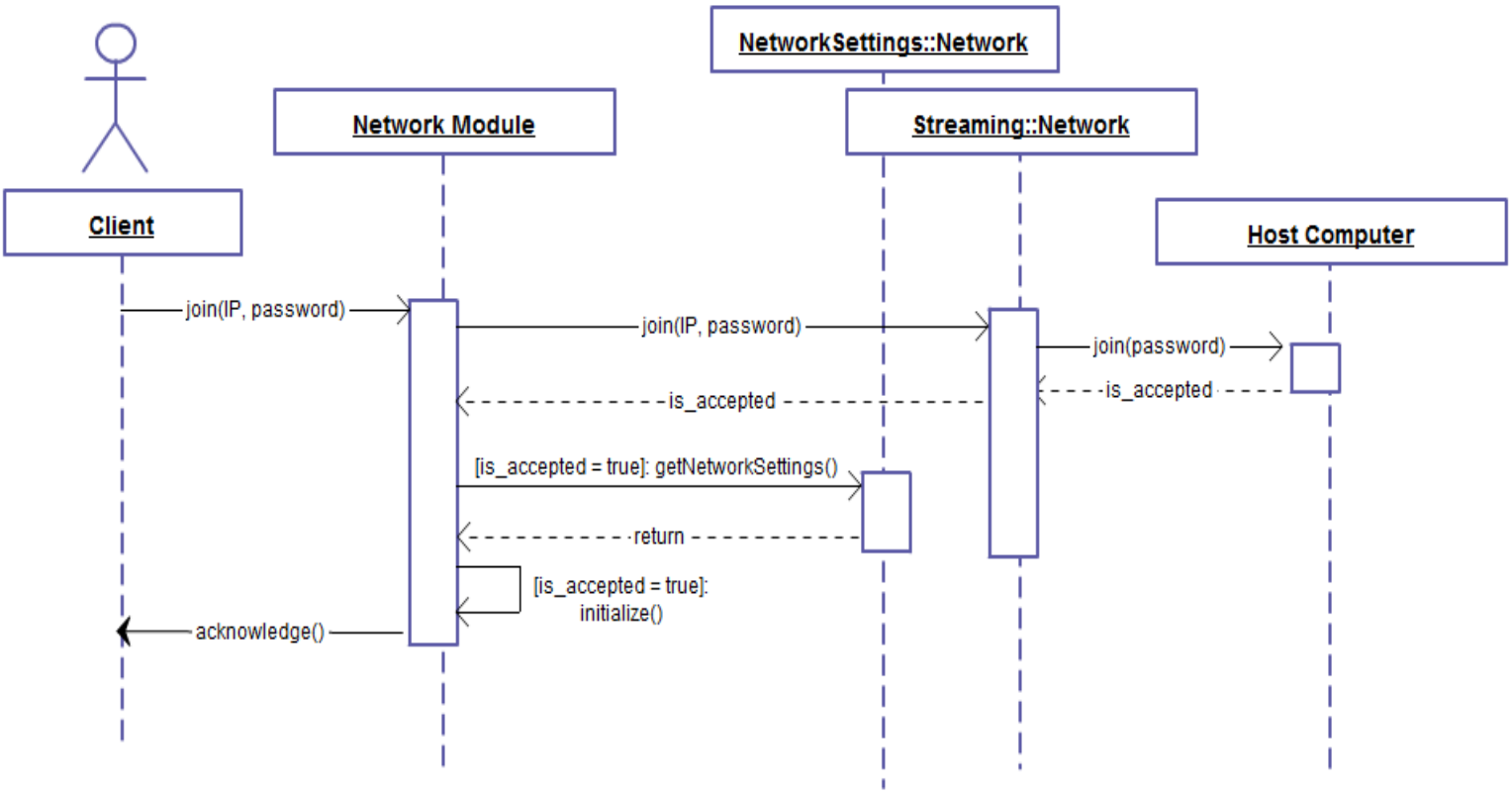
3.2.2 Details of Network Functions

In this section, we describe the working details of complex network functions (see the descriptions of these functions in Section 3.2.1) in order to give better understanding about our complex functions.

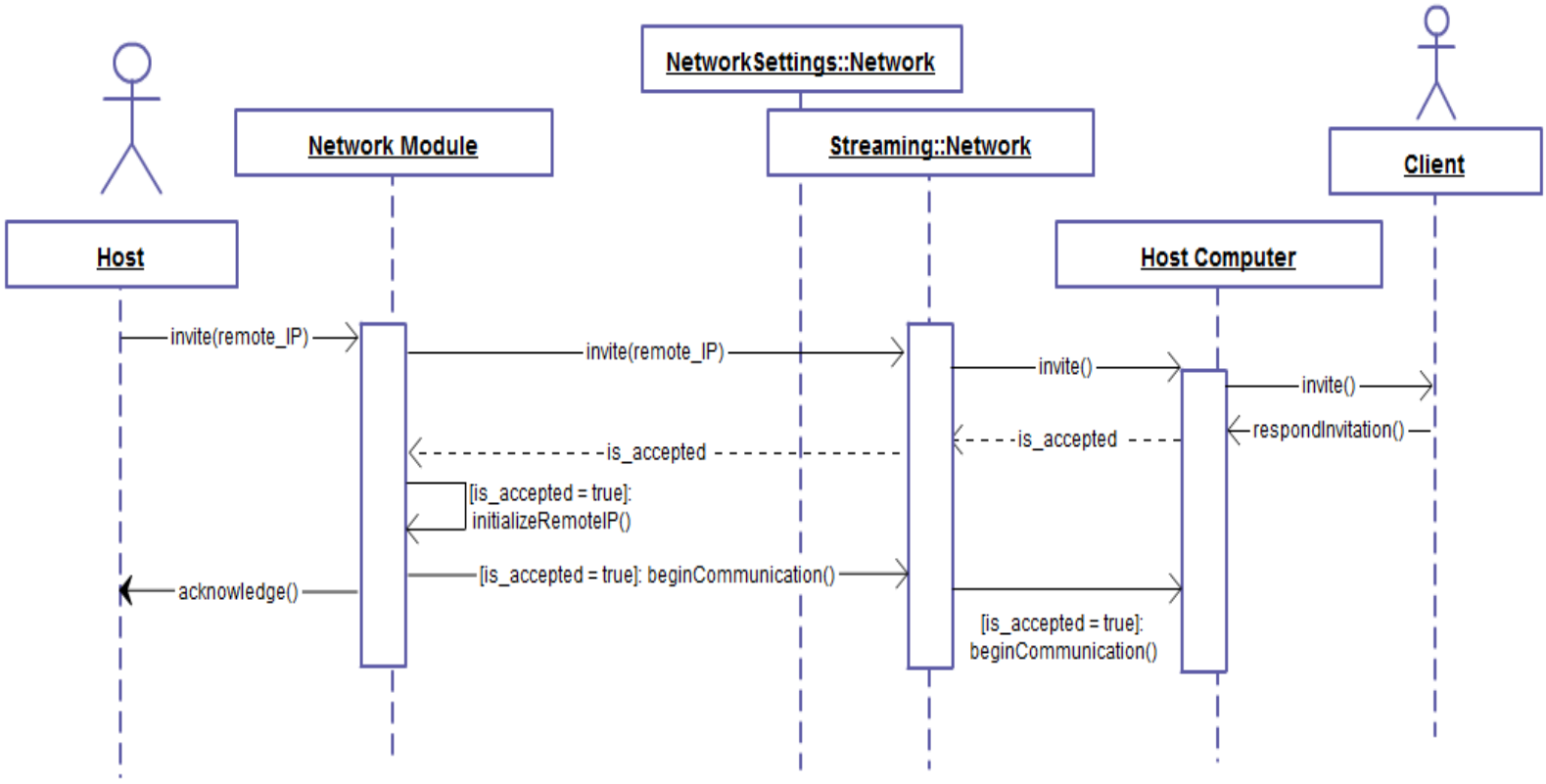
3.2.2.1 Start Conversation



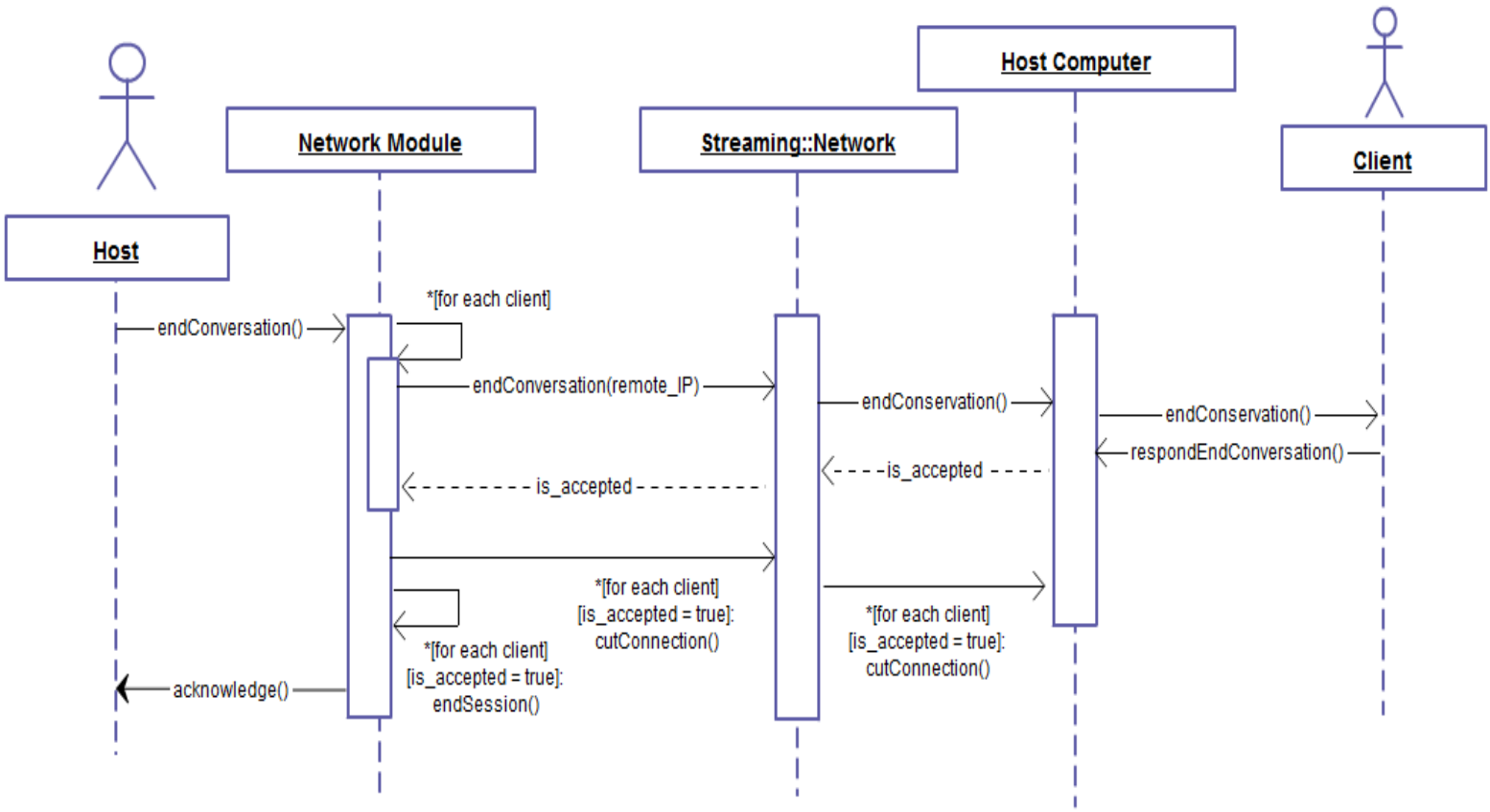
3.2.2.2 Join


NetworkSettings::Network
Network Module
Streaming::Network
Client
Host Computer
join(IP, password)
join(IP, password)
join(password)
is_accepted
is_accepted
[is_accepted = true]: getNetworkSettings()
return
[is_accepted = true]:
initialize()
acknowledge()

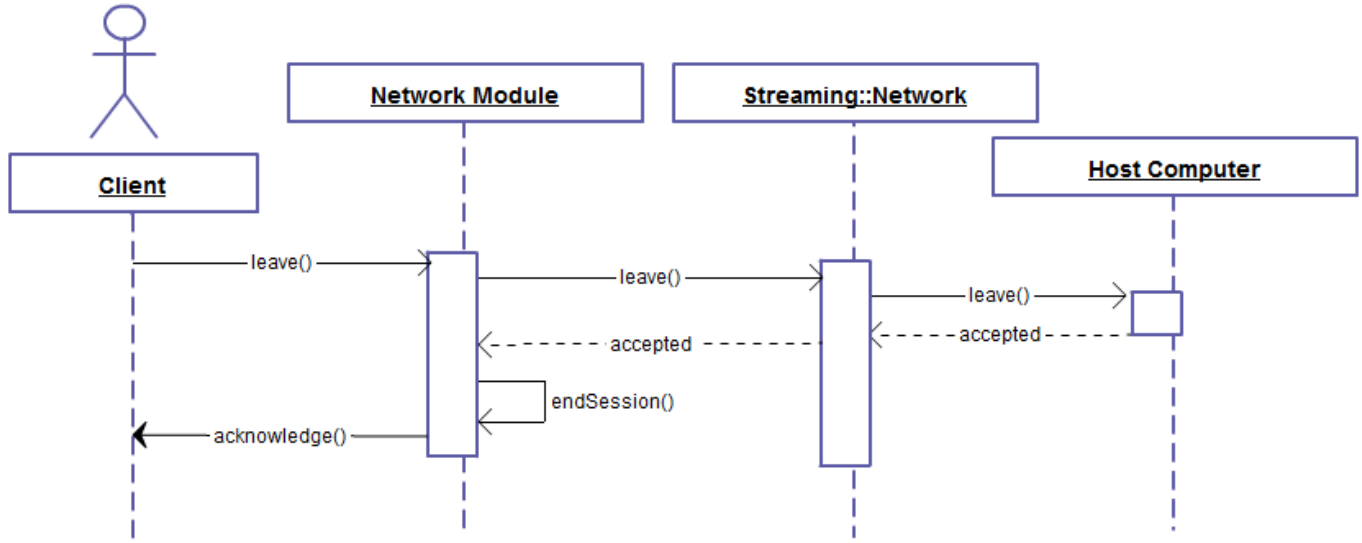
3.2.2.3 Invite - Respond To Invitation Couple



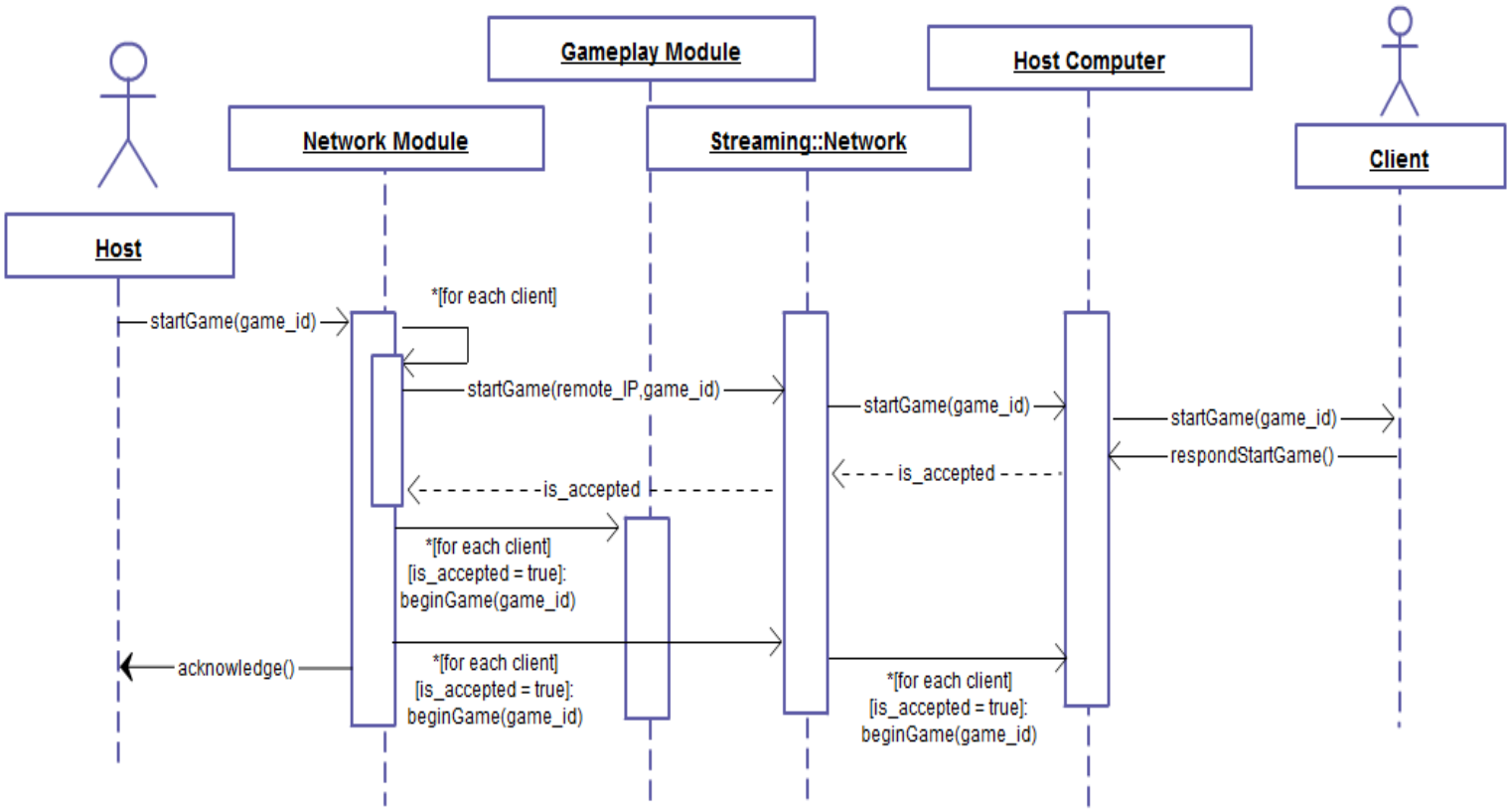
3.2.2.4 End Conversation - Respond To End Conversation Demand Couple



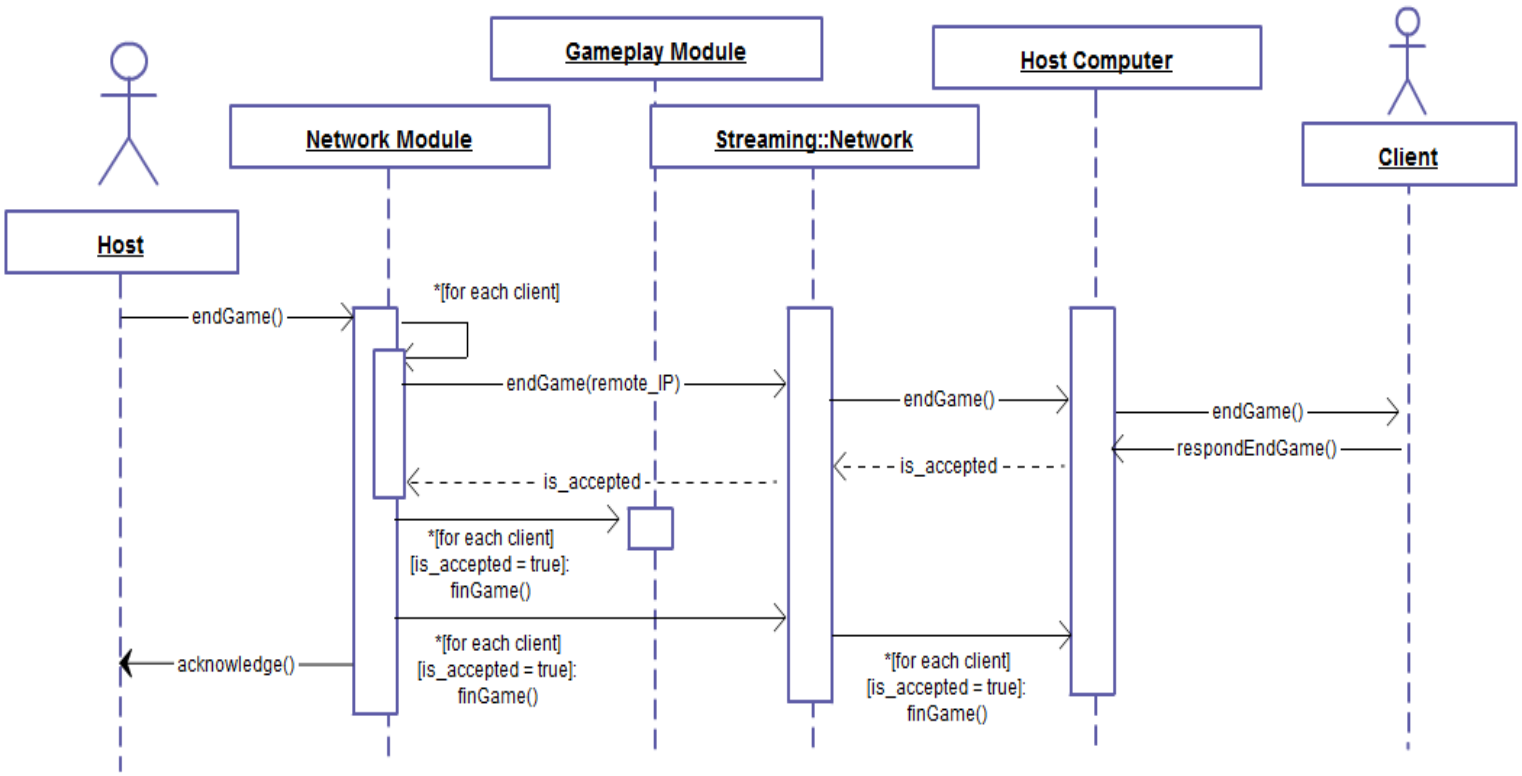
3.2.2.5 Leave Game



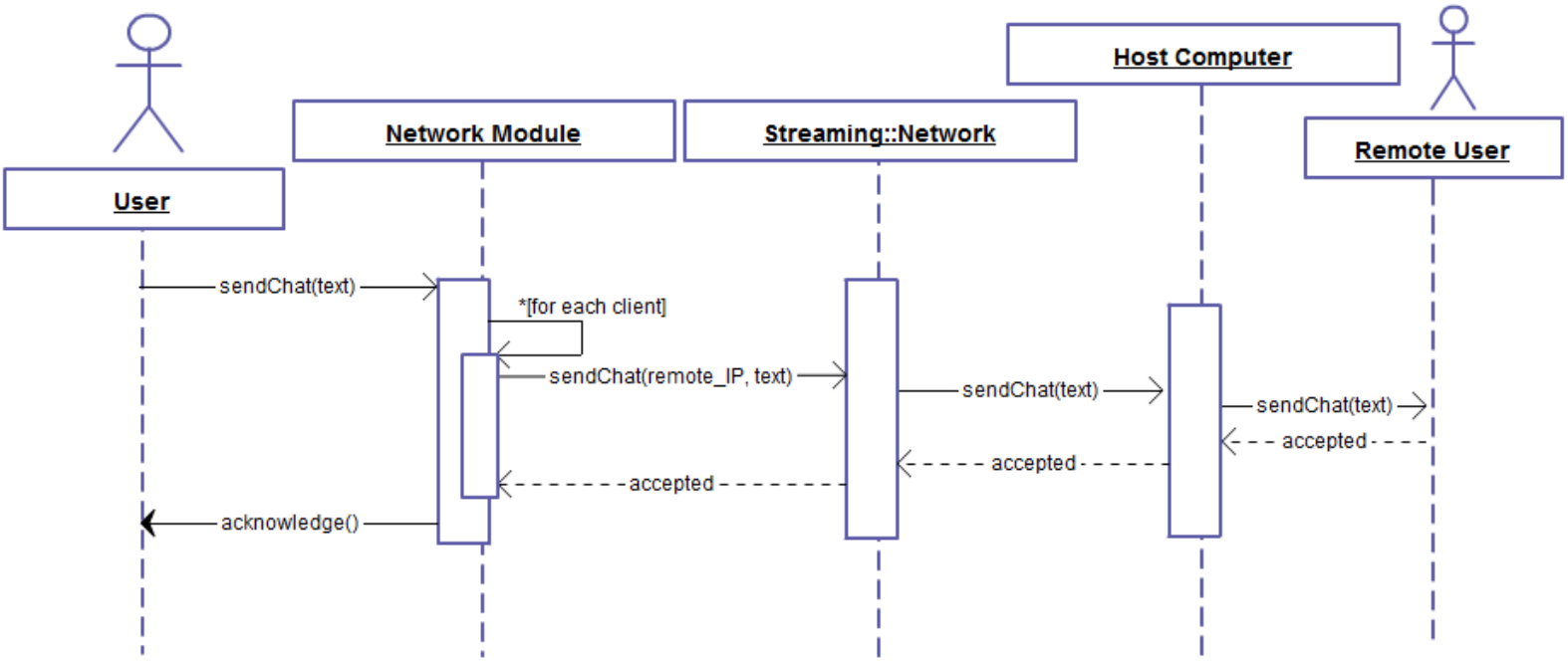
3.2.2.6 Start Game- Respond to Start Game Demand



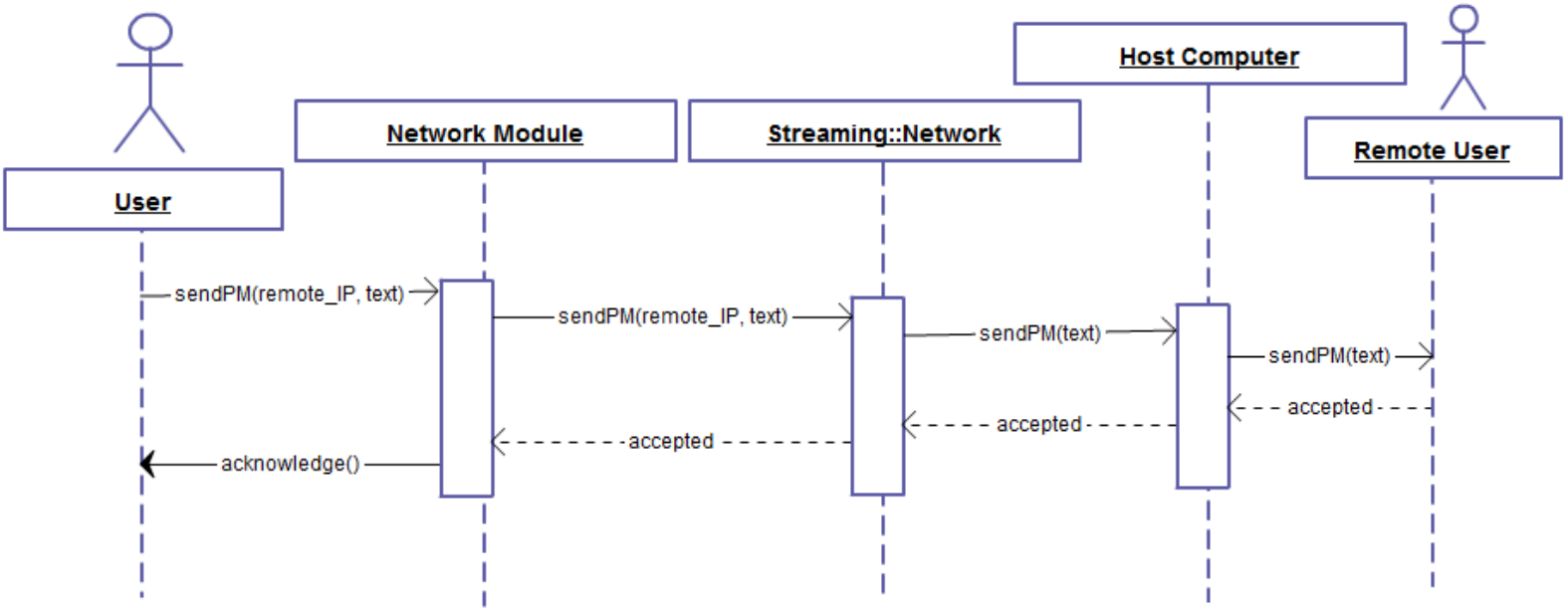
3.2.2.7 End Game - Respond to End Game Demand



3.2.2.8 Send Chat Message



3.2.2.9 Send Personal Message



3.3 Non-functional Requirements

3.3.1 Performance requirements

The performance is the most important part of our project. As our main aim is to entertain people, they should not be bothered by the late responses of the network. We intend to design a system that can respond to 6 people in a specified time; however, if the bandwidth limitations may not enough to support the system requirements, the performance may decrease.

3.3.2 Design constraints

3.3.2.1 Hardware requirements

The requirements for development

- 2 GHz processor
- 2 GB Ram
- 3 GB HDD Space
- Internet connection

The requirements for users

- 2 GHz processor
- 1 GB Ram
- 100 MB HDD Space
- Internet connection
- Bandwidth 1Mbps

3.3.2.2 Software Requirements

Our software requirements are stated in 3 groups. First one is our software requirements for developing process; second one is software requirements for server side, last one is software requirements for client side.

The requirements for development:

- Windows operating system
- Asp.net
- Microsoft Office or OpenOffice or Google Docs
- Microsoft Project
- MS Visual Studio
- MS Silverlight

The requirements for users:

- Windows operating system
- .NET Framework

3.3.2.3 Usability

The Audiovisual Gaming Network Project will be an easy to use game application with audiovisual features. As our main purpose is to provide people an entertainment environment, it is important for us to build a user-friendly application. The users will be able to add or delete contacts to their contact list just by clicking on a button. Opening a game table and hosting people will also be easy tasks which could be done just clicking on the buttons which make the system easy to use for everyone.

3.3.2.4 Reliability

The profiles of the users will be kept in the system and they will be able to edit it when necessary. The communication between the host and the client side will be done using the ip and password that the host will provide to the clients. The systems will be able to handle about 6 people. The consistency of the video and chatting feature is the most important issue of the project. We should keep the systems synchronized in all client sides and prevent deadlocks. The system should be capable of handling 6 people and respond in a specified time.

3.3.2.5 Security

Our application will communicate using the Ip addresses of the users and a password that the host provides to the clients. The clients will not be able to connect to the host without an invitation from the host.

3.3.2.6 Portability

The Audiovisual Gaming Network could be played on the computers which have windows operating system .Net framework is also required to be installed on the computer.

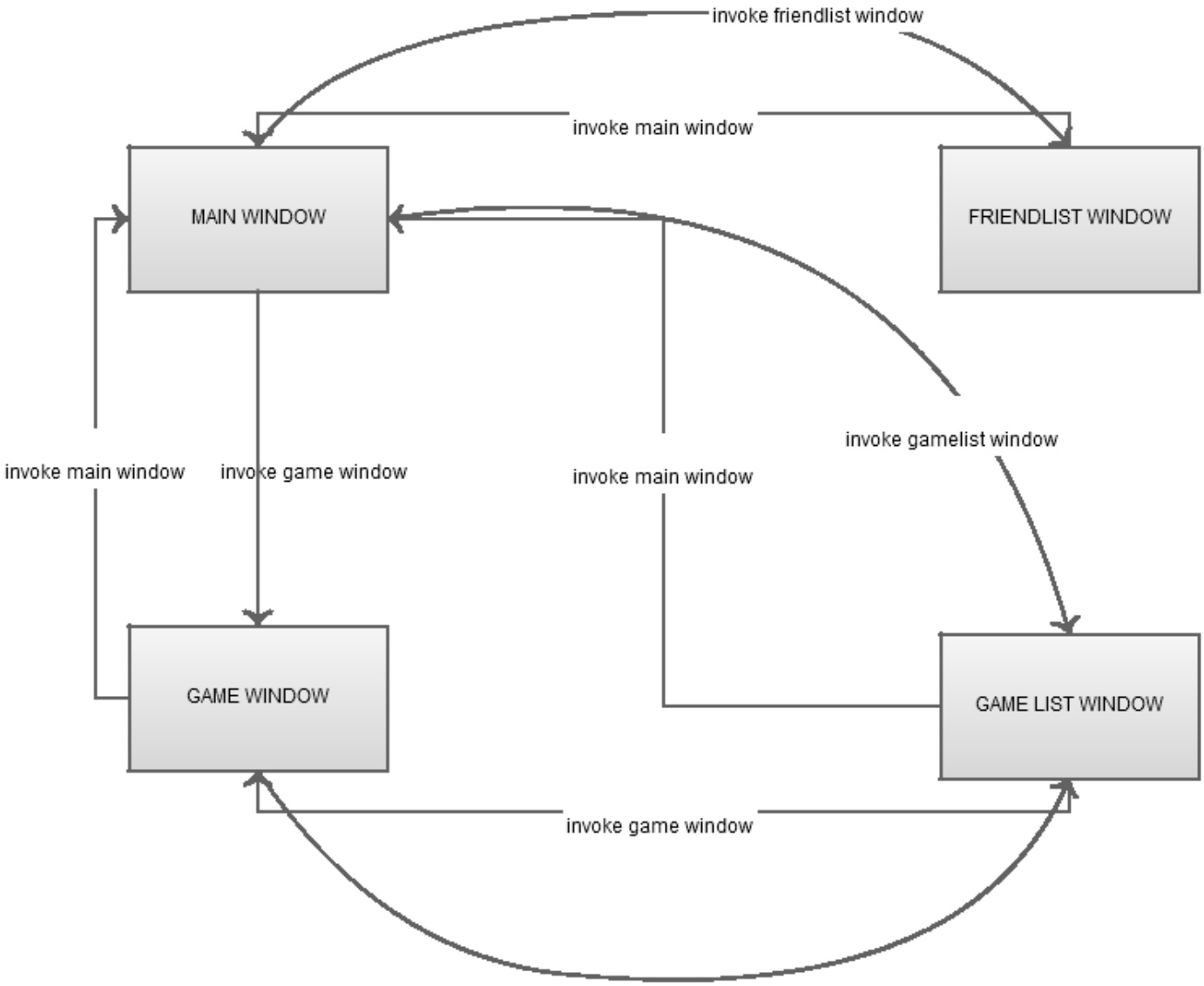
4 Behavioral Model and Description

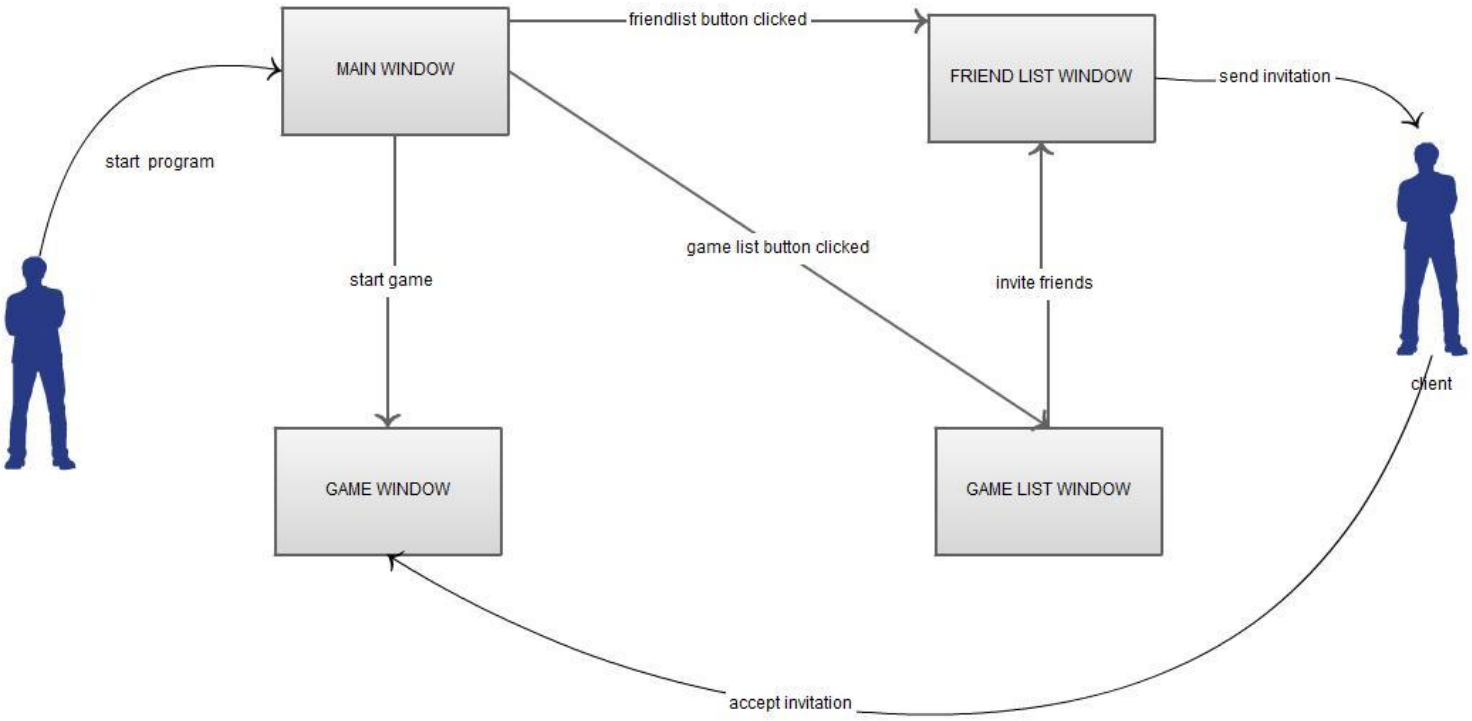
4.1 Description for software behavior

Our software consists of different windows and the links providing the transitions between them. The main window will be present during the execution of the program. The other windows will be reached through the main window, and they will be seen on the screen when necessary. When the program starts, the main window that is going to be shown is linked to the *friend-lis*t, *game-list* and game windows. If the user clicks on the button to open the *friend-list* window showing the list of friends will appear on the screen. Then, if desired, the invitations for playing a game could be sent to any of the friends.

Another window is the *game-list* window which could also be reached via the buttons on the main window. If the related button is clicked, the list of the games downloaded on the computer and ready to be played are shown on the screen. After choosing the game, the host user sends a game start request to the other users to get them to join the game. When all the users respond, the game players are directed to the game screen which is the graphical user interface of the chosen game.

4.2 State Transition Diagrams





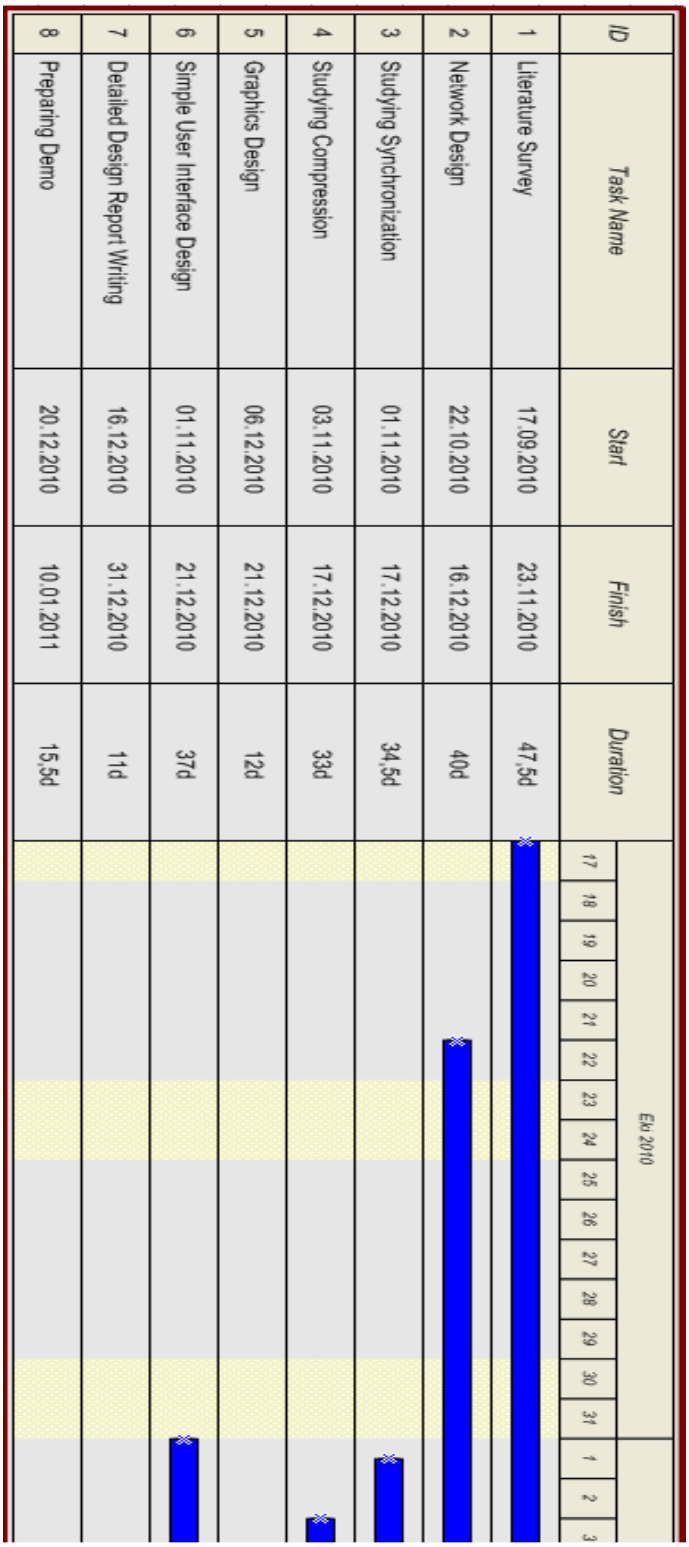
5 Planning

5.1 Team Structure

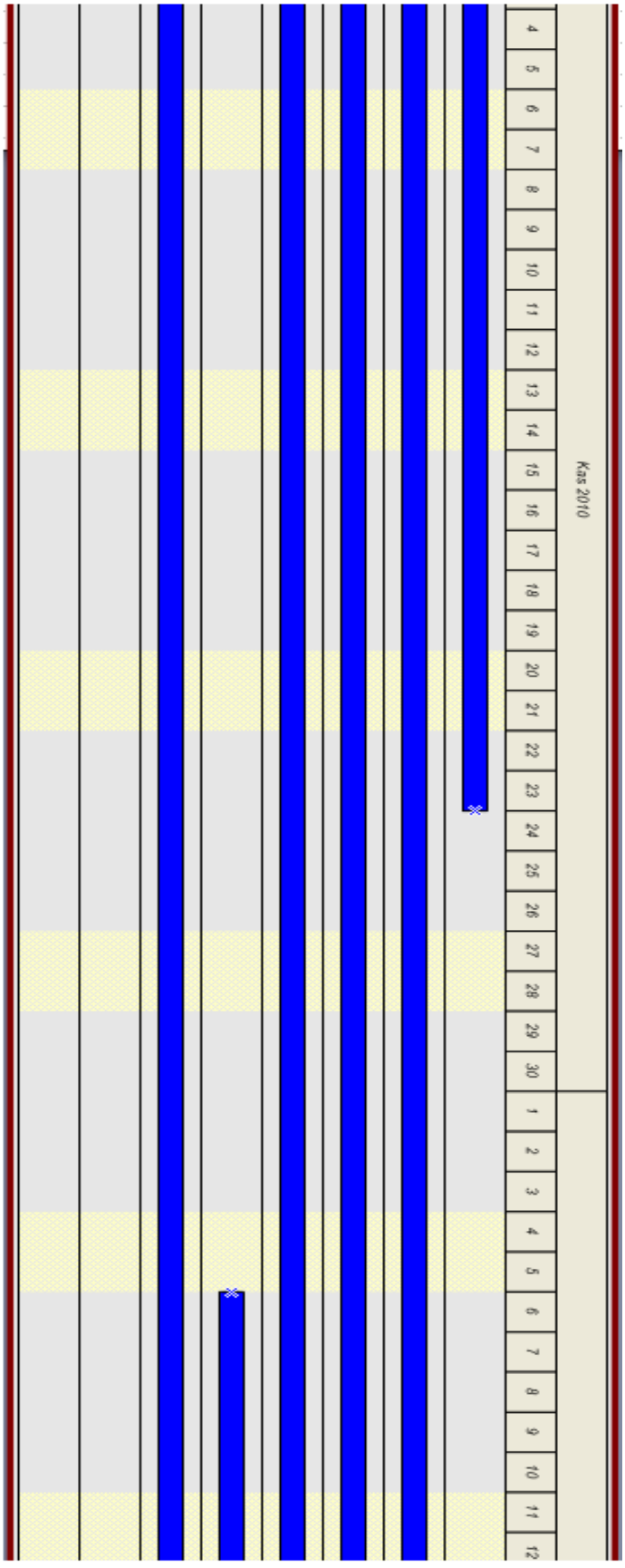
Our project is mainly consists of two parts. First part is the web service part is the audio-visual conferencing part that which is more important and requires more work and energy that decided to work in collaboration for that part. The audiovisual part consists of some sub-parts like streaming compression and synchronization. In our division of labor, Engin Ertaş is going to work on streaming and synchronization, Duygu Çelik and Serap İnce are going to work on the filter & compression algorithms.

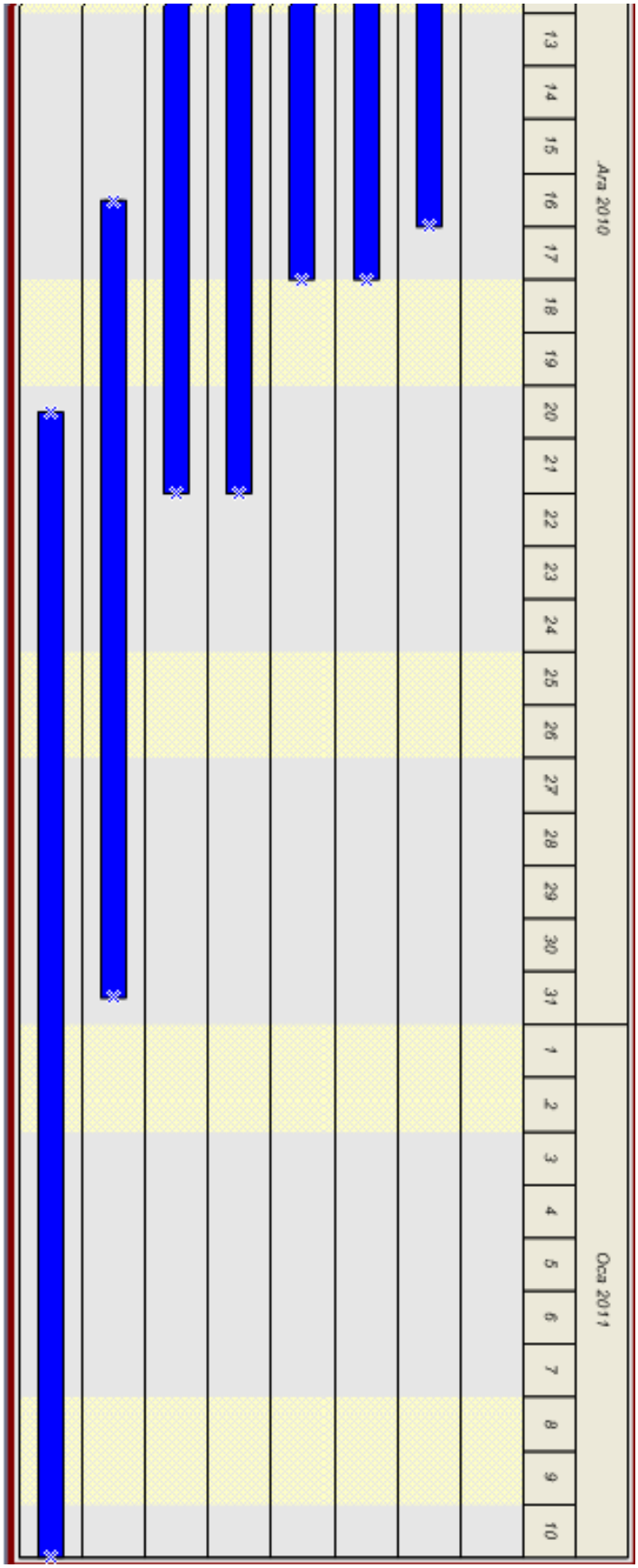
The second part consists of the user interfaces, *GamePlay Interface* design and Taboo game creation. Serap İnce and Duygu Çelik are going to work on the Taboo game which will be the default game in our software, in addition to user interfaces. Our communication coordinator is Engin Ertaş. We do not have a team leader, so the decisions are made in collaboration, as it is supposed to be. We meet with our teaching assistant once a week (more than once if needed) to decide on the details of weekly progress and make a schedule to work together.

5.2 Estimation (Basic schedule)



ID	Task Name	Start	Finish	Duration
1	Literature Survey	17.09.2010	23.11.2010	47,5d
2	Network Design	22.10.2010	16.12.2010	40d
3	Studying Synchronization	01.11.2010	17.12.2010	34,5d
4	Studying Compression	03.11.2010	17.12.2010	33d
5	Graphics Design	06.12.2010	21.12.2010	12d
6	Simple User Interface Design	01.11.2010	21.12.2010	37d
7	Detailed Design Report Writing	16.12.2010	31.12.2010	11d
8	Preparing Demo	20.12.2010	10.01.2011	15,5d


Kas 2010
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
1
2
3
4
5
6
7
8
9
10
11
12


Ara 2010
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
Oca 2011
1
2
3
4
5
6
7
8
9
10

5.3 Process Model

Our process model will be based on creating a basic prototype and then developing it. First we will try to make communications between 2 or maybe 3 users and then develop our system by incrementing the client number connecting the host and implement the gaming modules in the system. We choose the Spiral Model as our process model which suits our process plan.

6 Conclusion

In order to accomplish the project, software requirement analysis is the crucial part of the progress for us. The first reason for this is the fact that the analysis embodies the ideas in our minds. In other words, writing functional and nonfunctional requirements and drawing related use cases, or data flow diagrams, etc. provides us with not only shaping our ideas but also thinking in a more detailed manner, leading us to notice the potential problems to be encountered. Another essential task is the scheduling. While writing the requirements analysis report, we made an estimation on what we will do in which interval.

To conclude, we believe that this requirement analysis report we wrote will guide us through this semester.