Chris Klassen – Melvin Loho – Jonathan Chu – Joel Mabbott

BCIT  3700 Willingdon Avenue

Windows assignment 4 design document

Table of Contents

[Requirements 2](#_Toc413260487)

[User Interface Mocks 2](#_Toc413260488)

[Overview 2](#_Toc413260489)

[State Flow Diagrams 4](#_Toc413260490)

[Overview 4](#_Toc413260491)

[Server 4](#_Toc413260492)

[Client 5](#_Toc413260493)

[Server Pseudo-Code 6](#_Toc413260494)

[Initialization 6](#_Toc413260495)

[Multicast 9](#_Toc413260496)

[Unicast 10](#_Toc413260497)

[Teardown 10](#_Toc413260498)

[Client Pseudo-Code 11](#_Toc413260499)

[Initialization 11](#_Toc413260500)

[Idle 11](#_Toc413260501)

[Connect 12](#_Toc413260502)

[Music 13](#_Toc413260503)

[Broadcast 14](#_Toc413260504)

[Unicast 15](#_Toc413260505)

[Microphone 16](#_Toc413260506)

[Teardown 17](#_Toc413260507)

[Control Message Protocol 17](#_Toc413260508)

[Overview 17](#_Toc413260509)

[Message Structure 17](#_Toc413260510)

[Types of Messages 18](#_Toc413260511)

# Requirements

* Create a Windows application
* Use the TCP/IP protocol suite to transfer audio from a server to clients
  + Must use some completion ports
* Play audio from the clients
* Client must be able to connect to a remote server
* Client must be able to save and retrieve sound files from the server
* The .wav format must be supported
* Two-way microphone support must be provided between clients
* The server must use multicasting and unicasting

# User Interface Mocks

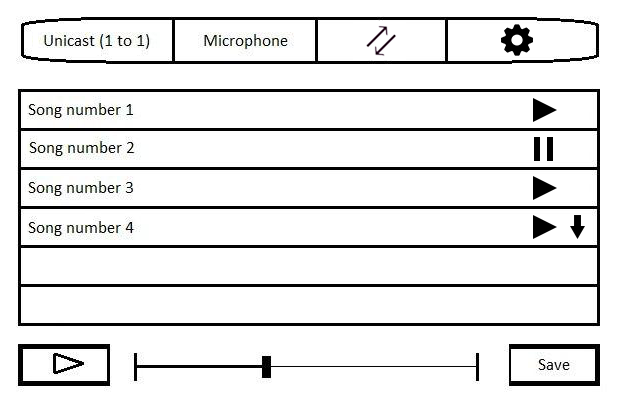
## Overview

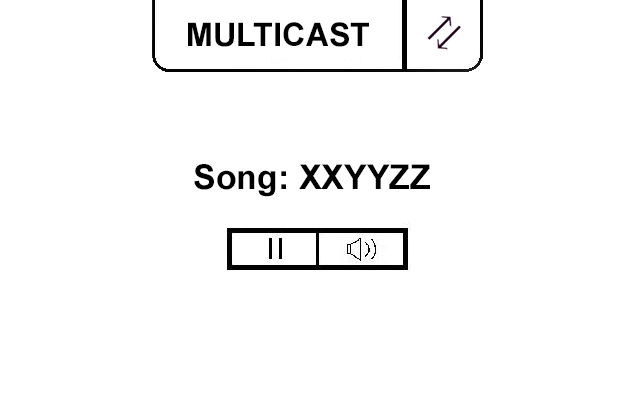
Of the two applications that will be developed for this project, only the Client will have a GUI. The Server will be developed for the command line, for the sake of simplicity and efficiency.

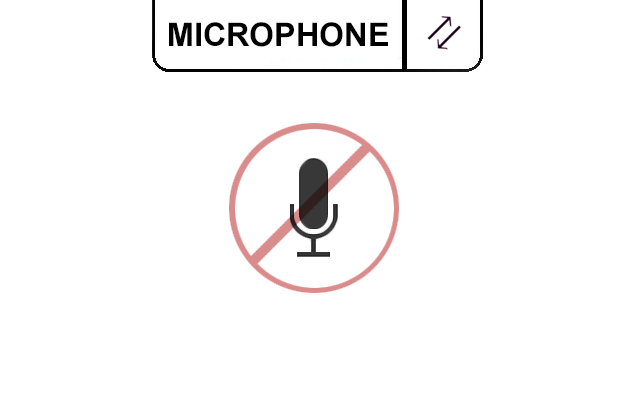
The Client’s user interface is divided into two main screens: the **Music** screen and the **Microphone** screen.

The Music screen switches context between Multicast and Unicast, depending on the state of the server. In Multicast mode, the client simply displays the details of the song currently being played. In Unicast mode, it displays a list of songs, from which the user can stream or download one of them.

The Microphone screen simply displays the details of the Client that the user is currently connected to.







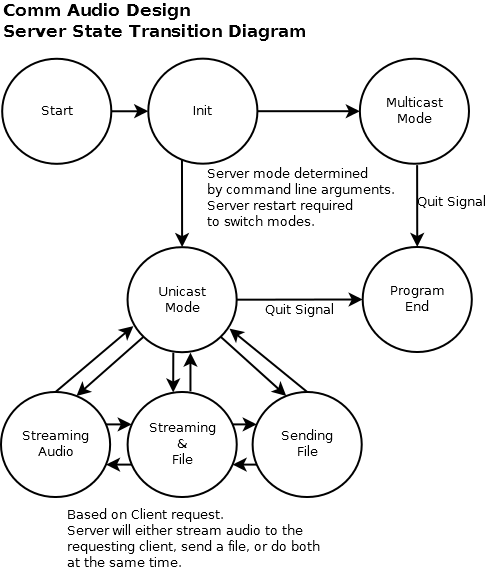
# State Flow Diagrams

## Overview

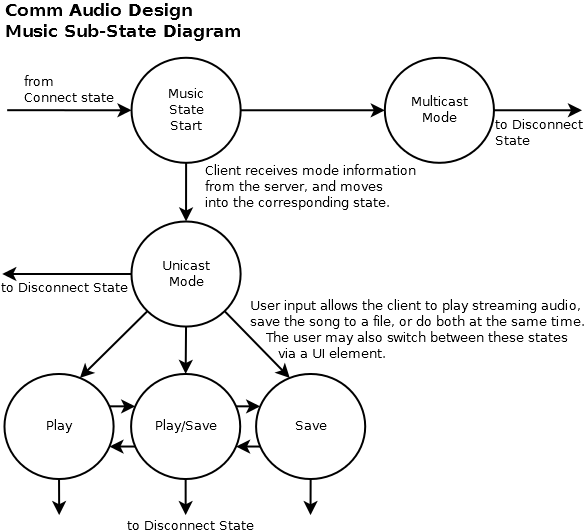
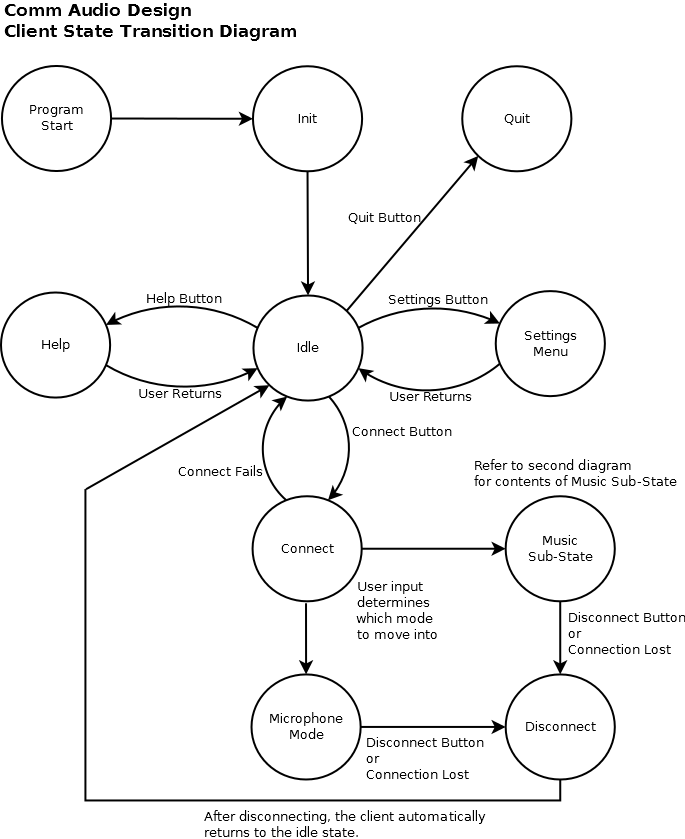
The following state flow diagrams exist to identify the major paths that the user can take through each application. Each state in these diagrams contains corresponding pseudo code below.

The one exception to this is the Control Message Handler Functions. These functions are responsible for accepting control messages from both a server and a client, and redirect interpreted tasks to various parts of the application.

## Server



## Client



# Server Pseudo-Code

## Initialization

**Main Function**

{

Call the Setup Function

Call the Load Tracklist Function

Call the Open Socket Listener Function

If the mode is multicast

{

Call the Multicast Function

}

Else

{

Call the Unicast Function

}

Call the Teardown Function

If the Teardown Function fails

{

Print error message

exit

}

exit

}

**Setup Function**

{

Parse the command line arguments

If the command line arguments are invalid

{

Print the usage string

Return false

}

Set the mode to argument 1

Set the port to argument 2

Start Winsock

Return true

}

**Load Tracklist Function**

{

If the music folder does not exist

{

Print error message

Return false

}

If the number of audio tracks is zero

{

Print error message

Return false

}

For each audio track in the music folder

{

Add the file name to a music list

}

Return true

}

**Open Socket Listener Function**

{

Open a TCP listener socket for the Accept New Client Function

If the socket fails to open

{

Print error message

Return false

}

Bind the listener socket to an available port

If the bind fails

{

Print error message

Return false

}

Return true

}

**Accept New Client Function**

{

accept the client request

Create a new client TCP socket for the Control Message Handler

Function

send the client a Start Connection message

add the client to a list of connected clients

}

**Control Message Handler Function**

{

switch on the message type

{

case End Connection:

{

Close the connection with the specified client

remove the specified client from the client list

}

case Play Song:

{

call the Play Song Function

}

case Save Song:

{

call the Save Song Function

}

}

}

## Multicast

**Multicast Function**

{

Open a multicast socket

Connect the multicast socket to a multicast group

Call the Play Music Function

}

**Play Music Function**

{

While the done signal has not been received

{

Pick a random song from the tracklist

Open the song file

Create a new thread on the Send Current Song Function

While we have not reached the end of the file and

the done signal has not been received

{

Send 512 bytes of the song to the multicast socket

}

If the done signal has not been received

{

Call the Load Tracklist Function

}

}

}

**Send Current Song Function**

{

for each client in the client list

{

send a control message with the current song information to

the client

}

}

## Unicast

**Unicast Function**

{

While the done signal has not been received

{

Do nothing

}

}

**Play Song Function**

{

open the requested file

Open a UDP socket to the client

While we have not reached the end of the file and

the done signal has not been received

{

Send 512 bytes of the song to the client via UDP

}

Close the UDP socket

Send the End Song message to the client

}

**Save Song Function**

{

open the requested file

While we have not reached the end of the file and

the done signal has not been received

{

Send 512 bytes of the song to the client via control

channel

}

Send End Song message to the client

}

## Teardown

**Teardown Function**

{

for each client in the client list

{

send an end connection message to the client

}

Close the control listener

If the control listener fails to close

{

Return false

}

Stop Winsock

Return true

}

# Client Pseudo-Code

## Initialization

**Main Function**

{

Start WinSock

Create the window

while there are messages left to receive

{

handle a message

}

exit

}

## Idle

**Settings Function**

{

Create Settings popup

Retrieve IP and port from user

return IP and port

}

**Help Button Function**

{

Display help information

}

**Mic Button Function**

{

set the mode to mic mode

open a UDP socket for the Receive Microphone Data Function

}

**Music Button Function**

{

set the mode to music mode

if the UDP microphone socket is open

{

Close the microphone socket

}

}

**Connect Button Function**

{

call the Connect Function

}

**Disconnect Button Function**

{

call the Disconnect Function

if mode is music

{

send the server an End Connection message

close the server control socket

if server mode is unicast

{

close the server stream socket

close the server file saving socket

}

else

{

unsubscribe from the server's multicast session

close the server multicast socket

}

}

else

{

send the client an End Connection message

close the client communication socket

close the client control socket

}

}

## Connect

**Connect Function**

{

Open a TCP socket for the Control Message Handler Function

Connect to the client/server

if connection fails

{

return false

}

if the mode is music

{

create a thread on the Play Music Function

}

else

{

create a thread on the Microphone Function

}

set connection mode to connected

return true

}

## Music

**Control Message Handler Function**

{

switch on the message type

{

case End Connection:

{

call the Disconnect Function

}

case Start Connection:

{

set server mode to message data

if the server mode is multicast

{

set the application view to multicast mode

}

else

{

set the application view to unicast mode

}

}

case Track List:

{

call Set Track List Function with the message data

}

case Now Playing:

{

call the Set Now Playing Function with the message

data

}

case Song Piece:

{

add the message data to the save to file buffer

}

case End Song:

{

if the server mode is multicast

{

set the song done flag to true

}

else

{

save the contents of the save to file buffer

}

}

}

}

### Broadcast

**Play Music Function**

{

while server mode is nothing

{

Do nothing

}

if server mode is unicast

{

return

}

else

{

create a multicast socket for the Add To Music Buffer

Function

subscribe to the server's multicast session

while connected

{

while the music stream buffer has less than 2 messages

worth of data

{

do nothing

}

while the song done flag has not been set and we are

still connected

{

play music from the buffer

}

set the song done flag to false

}

}

}

**Add To Music Buffer Function**

{

add the data to the music stream buffer

}

**Set Now Playing Function**

{

set the song name to the song name portion of the message

set the artist to the artist portion of the message

set the album to the album portion of the message

set the length to the length portion of the message

}

### Unicast

**Set Track List Function**

{

clear the existing track list

loop through each song in the song list

{

add it to the track list

}

}

**Stream Button Function**

{

Pull the selected song from the list

Send the server a Play Song control message

create a thread on Stream Music Function

}

**Stream Music Function**

{

Open a UDP listener on the Add To Music Buffer Function

while connected

{

while the music stream buffer has less than 2 messages

worth of data

{

do nothing

}

while the song done flag has not been set and we are still

connected

{

play music from the buffer

}

set the song done flag to false

}

}

**Mute Button Function**

{

if the volume is 0

{

set the volume to max volume

}

else

{

set the volume to 0

}

}

**Save Button Function**

{

pull the selected song from the list

send the Save Song message to the server

}

## Microphone

**Microphone Function**

{

create a UDP socket to the client

create a thread on the Listen To Microphone Function

while we are connected

{

if the microphone buffer has more than 2 messages of data

{

while there is data left in the microphone buffer and

we are still connected

{

play audio from the microphone buffer

}

}

}

}

**Listen To Microphone Function**

{

Read data from the microphone

Send it to the client

}

**Receive Microphone Data Function**

{

add the data to the microphone buffer

}

## Teardown

**Disconnect Function**

{

close the connection with the server / client

set connection mode to disconnected

set server mode to nothing

clear the music stream buffer

clear the save to file buffer

}

# Control Message Protocol

## Overview

The server and its clients communicate their intents and status changes using a TCP control channel. This control channel is designed in such a way that any number of new events can be added with relative ease.

The control channel works via message strings that can be parsed into a message structure. From this point, the structure can be interpreted and the data can be handled appropriately.

## Message Structure

Every message in this environment has a specific structure. This structure is as follows:

Message

{

Message Type

Message Data (vector of strings)

}

This structure is converted into a string to send through the control channel. The string is defined as follows:

“Message Type~Message Data 1`Message Data 2`…”

This string is delimited by ~ characters. Spaces are considered to be part of the data, but should only appear in the Message Data. Message data can be segmented using ` characters. These will be separated in the vector.

All control messages are variable length and should be read until there is nothing left.

## Types of Messages

Messages sent between the server and its clients can exist in a finite number of message types. These types are defined and described below:

|  |  |  |
| --- | --- | --- |
| MESSAGE\_TYPE | MESSAGE\_DATA | Description |
| END\_CONNECTION | Empty | This message is sent by either the server or client to indicate that the channel should be torn down. |
| START\_CONNECTION | Server Mode | This message is sent by the server to a new client to indicate what mode the server is in. |
| TRACK\_LIST | Song 1`Song 2`… | This message is sent by the server to clients when a client connects, and when a song ends. It contains a list of all server songs. |
| PLAY\_SONG | Song Name | This message is sent by the client to the server to request a song to be played. |
| SAVE\_SONG | Song Name | This message is sent by the client to the server to request a song to be saved. |
| NOW\_PLAYING | Song Name`Artist`Album`Length | This message is sent by the server to clients to indicate the next song to be played (unicast and multicast) |
| SONG\_PIECE | Song Data | This message is sent by the server to a client. It contains some data to be saved to a file. |
| END\_SONG | Empty | This message is sent by the server to a client when a song has finished streaming or being sent. |
| MIC\_PIECE | Microphone Data | This message is sent by the client to another client. It contains some data to be played aloud. |