WiBand

Wireless Music Band

Final Report

Group 11

Kadir Melih Albeyoğlu
Sinem Koçkan
İrem Özbek
Serkan Şahin

May 25, 2010
TABLE OF CONTENTS:

1. DESCRIPTION OF THE GAME: 3
2. SYSTEM ARCHITECTURE: 4
3. IMPLEMENTATION: 5
   3.1. Establishing an Ad Hoc Network: 5
   3.2. Software: 5
      3.2.1. GameEngine.java: 6
      3.2.2. UDPReceiver.java: 6
      3.2.3. UDPTransmitter.java: 7
      3.2.4. ChordSequencer.java: 7
      3.2.5. Player.java: 8
      3.2.6. ChordPlayer.java: 8
      3.2.7. GUIManager.java: 8
      3.2.8. GamePanel.java: 9
      3.2.9. LogPanel.java: 9
4. HOW TO SETUP AND PLAY THE GAME: 9
5. SNAPSHOT OF THE GAME: 10
6. CONCLUSION AND POSSIBLE IMPROVEMENTS: 11
1. DESCRIPTION OF THE GAME:

The game WiBand is based on generating music over an ad-hoc network by playing pre-defined and recorded midi file tracks of a song with 3 players, corresponding to lead guitar, bass guitar and drums. The song is divided into time slots of one-second duration. When the program is running, a specified song is represented by the flow of its time slots. The game assigns these time slots to three keys by a randomly created flow sequence. A player is regarded as successful if he presses the key he is supposed to, correctly and on time. In case of success at a time slot, the corresponding track of a user is played on all laptops for that time slot (in other words for 1 second).

To make the game user friendly, the users are able to see the upcoming 3 buttons in the flow at any instant. The key sequence will be determined randomly on all laptops and it does not need to be known by other users. The other players only need to know if the player plays the game correctly. However, no matter whether the users play correctly or not, there is always background music different than the guitar and drum sounds. This background music is mixed with players’ own tracks as long as the game is played correctly.

There is no hierarchy among the users. After the IPs are broadcasted to indicate that the players are ready for the game, one of the users should start the game. However, that user does not act as a server.
2. SYSTEM ARCHITECTURE:

In our software architecture we aimed to make the classes to be independent of each other as possibly as it can be. We designed our classes in two sub packages; one is called GUI Layer and the other is Model Layer. These two packages, as it can be guessed easily, handle the user interface of the system and the backbone of the system. We made GUI Manager and the GameEngine to be the main classes of their packages and these two packages will be the classes that provide interactions between the packages.

In GUI Layer, GUIManager is the main class and it consist of the GamePanel and GamePanel includes a LogPanel. These three classes are the classes that we provide.
Mostly, GUIManager is responsible with the user interface and the other two classes provide services to it.

In Model Layer, Game Engine is the main class and it uses UDPReceiver, UDPTransmitter and also it consists of ChordSequencer and Player. Moreover, Player class includes ChordPlayer. UDPReceiver and UDPTransmitter classes handle the sending and receiving of packages between the users. Player and the ChordPlayer are handling the sounds of the system and the ChordSequencer plays the sounds of the system.

3. IMPLEMENTATION

The implementation of WiBand could be divided into two parts:

3.1. Establishing an Ad Hoc Network

Our program is implemented to run on Windows. For Windows XP and Windows Vista, using the “Properties” menu of “Network Connections” an ad hoc network could be created under the “Wireless Networks” tab. We set up an ad hoc network called “WiBand” with an authentication key for security purposes. Once the network is established, the laptop that created the network must always be connected to the WiBand network for the other laptops to remain connected to each other. Although, our program is not based on a server-client hierarchy, for the network to be active, the creator laptop needs to act like a server initially.

3.2. Software

WiBand is coded completely on Java. Multiple classes of Java API were made use of for different parts of the program. To mention a few, Java.Sound is used for the musical aspect of the program, Java.Net is used for network communications and several other classes are used for creating our GUI.

There are multiple classes created for the implementation of WiBand. They are called GameEngine, UDPReceiver, UDPTransmitter, ChordSequencer, Player, ChordPlayer, GUIManager, LogPanel and GamePanel. Below, the functions of these classes are given:
3.2.1. GameEngine.java:

This class is used for initialization process of the game. UDPReceiver.java creates an object of this class. The sampling period of the game is determined here as 1 second. There are also two arrays created here; one is for the button sequence that is created randomly and separately for each user at the beginning of the game, and the other is a two dimensional array for players. The elements of the player array are updated when the success messages of the players are received. The music tracks of the users and the background music are initialized here in Sequence class objects and they are opened via Sequencer class objects.

This class creates an object of UDPTransmitter.java and that object is used for sending success messages of the user. There is also a method called inheritIPs(String[] s), that is used to transfer the IP numbers of the players that are received by the UDPReceiver class to the UDPTransmitter object.

When the game is started by one of the players, startGame() method of this class is called and the timeslot starts to be incremented. For this purpose, this class uses Timer class of Java.

3.2.2. UDPReceiver.java:

The main purpose of this class is to handle the reception of all packets over the network. There are three methods created in this class. One is used for IP reception, the other is used for start symbol reception and the last one is for data packet reception. When the program is started, first a GameEngine object is created to display the GUI and to control the program. After that, IP receiver method is called and is run until two IP numbers are received. When the IP numbers are received, the program waits for the start sign that is going to be sent by one of the users. In the mean time, all users are able to send their IP numbers and one of them could send the start signal when IP exchange is done. Although WiBand uses data broadcasting for all network communications, IP numbers of the users are important for identification purpose when a data packet is received. The user that sends his or her IP in
the first place is assigned to be the first external user. Thus, depending on the IP sending sequence, a user could be in the first or second place for different users.

When the start sign is received, the game starts. After all these initialization steps, the data packet receiver method is called and the program remains in this method indefinitely. All of these three methods create different UDP sockets by using DatagramSocket class of Java. IP exchange is done over port 4000, start sign is sent over port 3000 and data packet exchange is done over port 6000.

3.2.3. UDPTransmitter.java:

An object of this class is created in GameEngine.java. Data transmittance is handled in this class by using broadcasting. As mentioned in UDPReceiver.java, first, IP number of the user is broadcasted to the network. When the IP numbers of other users are received, the start sign could be sent using startGame() method. After the game start, data packets are broadcasted using the sendData(String message) method.

3.2.4. ChordSequencer.java:

As mentioned before, there are pre-defined and recorded midi file tracks of a song for 3 users and each track consists of a unique chord sequence. However, the users do not need to play these different chords of the chord-sequence with a different key of the keyboard. Instead, 3 keys are given to each user and these 3 keys randomly represent a part of the chord sequence for 1 second. Thus, the ChordSequence.java Class object is responsible of the association of each timeslot of a chord sequence with a key of the keyboard and responsible of the representation of the chord to the user visually.

GameEngine.java class creates the instance of the ChordSequencer.java class where ChordSequencer.java class object arranges the current chord to be played and uses GamePanel class to represent the current chord visually (with an ImageIcon) to the user. In addition, the instance of the class is also
responsible for displaying the coming chords to the user visually again using GUI components of our program.

3.2.5. Player.java:

The Player.java class stands for the players of the song (users of the program), which are also chord players. Thus, the Player.java class consists of a ChordPlayer.java class instance and uses its object to play the songs for the players. Furthermore, The Player.java class extends the TimerTask class so that its run() method is called regularly by the GameEngine.java class to play the midi file tracks of a song on time.

3.2.6. ChordPlayer.java:

Playing operation is realized only if the user presses the correct key on time (conceptually plays the chord on time). With this purpose, ChordPlayer.java class is responsible for checking whether the chords of the 3 users are to be played or not and is responsible with the synthesizing these 3 chords at the same time to play a monaural.

Player.java creates an instance of the ChordPlayer.java Class where ChordPlayer.java Class controls an already filled user input arrays to decide playing the chords of 3 users and starts the sequencer of the corresponding user accordingly. Note that, we arranged the sequencers (conceptually players of the users) so that the whole sequencers start/stop at the right places of the midi file tracks, which give the sense that, the music never stops.

3.2.7. GUIManager.java:

This class acts as a connection between the GUI and the model layers. It creates LogPanel GamePanel and GameEngine objects. First two objects are for GUI and the third one is for the model layer.
3.2.8. GamePanel.java:

This class is used for creating the GameEngine class and associating the user interface. The visual buttons are created within this class with the corresponding images. The typing area and the game start button are also implemented here. We are also making the layout of the program in this class using the GridLayout. The join button and the start button sends messages to the other users in the network using the GameEngine’s object UDPTransmitter.

The game uses buttons to implement the flow of the future buttons to be pressed. The pictures are sent when the time to hit a key has come. In addition, this class has a key listener, which listens the inputs of the user and processes if the player is right or wrong. If the button is pressed correctly and on time, then a message is sent via GameEngine object, to the other users in the network so that they can listen the sound.

3.2.9. LogPanel.java:

LogPlayer.java Class creates part of the GUI. It adds a text area on the GUI where the users can track the success of all users including themselves and the time slot that is playing. As mentioned, the song is divided into one-second time slots. The time slot number represents the current time slot. By checking the time slot numbers on all users, we checked whether there is a delay or not in the system.

4. HOW TO SETUP AND PLAY THE GAME:

Firstly, the folder called WiBand, which has four chord sequences in our CD should be copied under “C:\”. Then one of the players needs to setup an ad hoc network as described above. After that, make sure that all of the players are connected to this network and then run the program. When the program is run, all of the players need to send IP numbers to other players by using the “Request to Join” button initially. After the IP exchange is done, one of the players should press on “Start the Game!” button. The game starts immediately after this, thus all of the
players need to click on the text field first for the key listener to be activated. When a button turns red, play its corresponding key on the keyboard. ("A","S" or "D" – from left to right) Enjoy!

5. SNAPSHOT OF THE GAME:

This is the screen that the players see when the program is run. The buttons for the initial handshaking could be seen above. The text field under the start button displays the IP numbers are received and the timeslot of the game at an instant. If the players are successful, it is also indicated on the field.

The black area above the three buttons displays the upcoming three buttons. The upcoming buttons flow downwards, and when the time comes, the corresponding button turns red.
5. CONCLUSION AND POSSIBLE IMPROVEMENTS:

With this project, we learned a lot about handling data exchange over an ad hoc network by using Java. Our project required special solutions, since it was a real-time network application. The most important problem to overcome was the synchronization problem. We handled this issue by sending a start signal at the beginning of the game to the connected laptops so that the timers of the laptops are started almost at the same time. (network delays are disregarded) However, the synchronization needs to be maintained throughout the game, as well. The solution of sampling a song at every one-second, in other words, playing the chord sequences of the players for 1 second if they are successful and then stopping them, was also beneficial for our synchronization problem. During every sampling period, the program waits until all the packets arrive and then, at the end of the duration, they are all mixed and played. The network delays were transparent to the players thanks to this solution. The result was successful after our tests.

Furthermore, we learned a lot about the sound and network classes of Java. We are now fully capable of maintaining a UDP connection with Java’s corresponding network classes. We also learned a lot about the sound classes of Java. Although, that is not a part of wireless connection maintenance, our capabilities with Java were improved.

In the future, the number of the songs could be increased and the players could be given the chance to choose the song they want. Also, by changing the sampling period of the game, the difficulty level of the game can be changed and this choice could be given to the players, as well.