CSIT 6910A Report

iBand - Musical Instrument App on Mobile Devices

Student: QIAN Li

Supervisor: Prof. David Rossiter

Table of Contents

I. Introduction

1.1 Overview

iOS (previously iPhone OS) is a mobile operating system developed and distributed by Apple Inc. Originally released in 2007 for the iPhone and iPod Touch platforms, it has been extended to support other Apple devices such as the iPad and Apple TV. Unlike Microsoft's Windows Phone and Google's Android, Apple does not license iOS for installation on non-Apple hardware. As of September 12, 2012, Apple's App Store contained more than 700,000 iOS applications, which have collectively been downloaded more than 40 billion times. It had a 21% share of the smartphone mobile operating system units shipped in the fourth quarter of 2012, behind only Google's Android. In June 2012, it accounted for 65% of mobile web data consumption (including use on both the iPod Touch and the iPad). At the half of 2012, there were 410 million devices activated. According to the special media event held by Apple on September 12, 2012, 400 million devices have been sold through June 2012.

On the other hand, the needs for playing music on smart phones have been rising. It becomes very convenient if we can use our phones to play music, anywhere, anytime.

This project develops an iOS App to let people play musical instrument right on their smart phones. Moreover, the App makes use of Wireless LAN to let users play musical instrument together.

1.2 Objective

The objective of this project is to develop an iOS App, which enables users to play musical instrument together through WLAN. In order to fully realize the required function, the following techniques are needed.

- Wireless LAN data transmission.
- Playing sound in iOS.
- Coordinate instrument input with network.

II. Preparation

2.1 iOS SDK & Xcode IDE

First of all, the iOS SDK is a must for developing iOS Application. The iOS SDK is integrated with the Xcode, which is the official development tools developed by Apple Inc. for developers. Xcode is Apple's powerful integrated development environment for creating great apps for Mac, iPhone, and iPad. Xcode includes the Instruments analysis tool, iOS Simulator, and the latest Mac OS X and iOS SDKs.

The Xcode interface seamlessly integrates code editing, UI design with Interface Builder, testing, and debugging, all within a single window. The embedded Apple LLVM compiler underlines coding mistakes as you type, and is even smart enough to fix the problems for you automatically.

Xcode can be downloaded freely from Mac App Store.

2.2 Wireless LAN Network Communication

This project requires the network communication between devices in the same LAN. Under this requirement, Bonjour is the perfect technique.

Bonjour is Apple’s implementation of a suite of zero-configuration networking protocols. Bonjour is designed to make network configuration easier for users.

For example, Bonjour lets you connect a printer to your network without the need to assign it a specific IP address or manually enter that address into each computer. With zero-configuration networking, nearby computers can discover its existence and automatically determine the printer’s IP address. And if that address is a dynamically assigned address that changes, they can automatically discover the new address in the future.

Apps can also leverage Bonjour to automatically detect other instances of the app (or other services) on the network. For example, two users running an iOS photo sharing app could share photos over a Bluetooth personal area network without the need to manually configure IP addresses on either device.

With this technology, the implementation can be found in iOS in many ways. Although the iOS SDK has already contained APIs which can be used to support Bonjour, I choose to use **DTBonjour** (https://github.com/Cocoanetics/DTBonjour), a open-source library that makes Bonjour more easy to use.

2.3 Sound Generation in iOS

There are many ways to generate sound in iOS, among which there is a very easy and straightforward way to satisfy the requirement for this project. That is to play WAV sound file directly. Many instrument sound can be recorded and then presented by a single WAV file, such as piano and drum, which are the two instruments used in the project.

Comparing to play WAV sound file, sound synthesis and MIDI can be the alternatives. However, sound synthesis of piano is extremely difficult and MIDI sound is not as realistic as the recorded sound of a piano. If there are more instruments added to the project, those approach may be used.

In this project, the sound of the piano are 88 single WAV files. Each of them is associated with a note of the piano. When a certain piano key is pressed, the corresponding WAV file is played. Similarly, there are 3 WAV files for the drum. Each of them represent a certain type of drum. A drum machine is built based on those type of sound.

In order to play WAV file efficiently, which means the time interval between user interaction and actual sound is heard should be near to zero, those WAV file should be loaded to a buffer. The buffer is in the low level of iOS and the sound can be played instantly. Consequently, an open-source library called **ObjectAL** (https://github.com/kstenerud/ObjectAL-for-iPhone) is used in the project. **ObjectAL** is the easy Objective-C interface to OpenAL, AVAudioPlayer, and audio session management. It can pre-load WAV file and play a certain WAV file whenever needed very efficiently.

III. Design

3.1 Overall

The design pattern used by iOS application development is Model-View-Controller. Model–view–controller (MVC) is a software architecture pattern which separates the representation of information from the user's interaction with it. The model consists of application data, business rules, logic, and functions. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as a bar chart for management and a tabular view for accountants. The controller mediates input, converting it to commands for the model or view. The central ideas behind MVC are code reusability and separation of concerns.

The following picture is an example that shows the interaction between Model-View-Controller that are adopted in iOS application development.

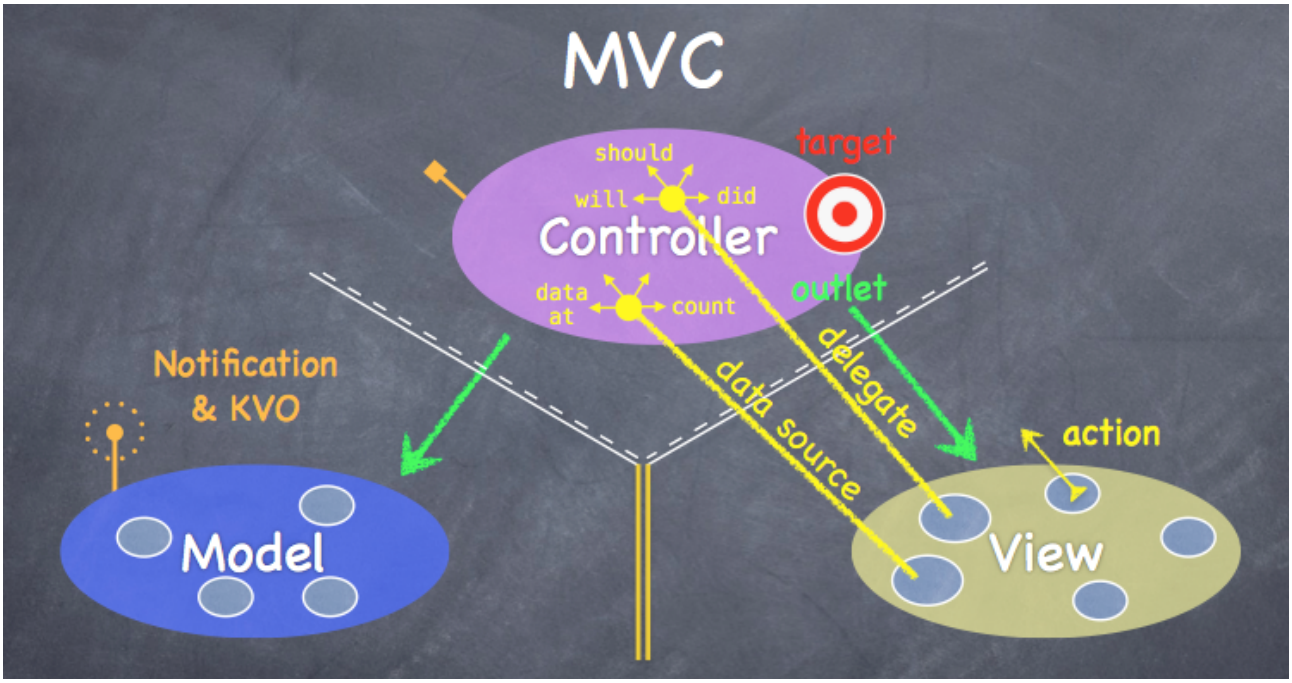

Figure 1. MVC in iOS

3.2 Model

The model of this app are the internal structure and sound of those instruments. It should provide public APIs such as load sound, play sound, stop playing, update instrument details, etc. Meanwhile, in order to follow the convention of Object-Oriented Programming, every instrument is inherited from the same super class, which provide common APIs that are used by every instrument.

Piano

The model of the piano is relatively simple. It should be responsible for pre-loading all the WAV files and playing a WAV file of certain key.

Drum

The model of the drum needs more consideration because the drum is a drum machine, instead of a normal drum.

The drum machine contains three kind of sound and 16 beats in total, as default. The Public APIs of the model should provide functions such as pre-loading sound, start/stop drum machine, turn on/off a certain slot, which means a certain beat of a type of sound, in the drum machine and change the tempo of the drum. More importantly, the drum machine should behave properly. A drum sound should be played whenever the slot of that drum is set and the slot is on the beat. There will be more details later.

3.3 View

Two types of view are apparently needed, the view of piano and the view of drum machine.

The view of piano is relatively straightforward. There should be a keyboard, showing the piano keys. And there should be a view to select octave range, in order to change the keys that are currently displaying. Meanwhile, the octave range size should be changeable, in order to make the piano keys adaptable for different finger size of different users.

The view of drum machine requires a view to show the drum machine and a view to show the current beat. For the simplicity, the view of the drum machine is just a table of size 3 by 16. Every row represents a single type of sound, and every column in a row represents whether the corresponding beat of that sound is turn on. Some other components such as the start/stop button, the slider to change the tempo and the text to show the current tempo are also needed.

3.4 Controller

The job of a controller is coordinate the model with the view. It should update views for any model changes or update models for any user interactions with the views. As a result, each instrument needs a controller.

Piano

The controller of the piano should responsible for playing the correct audio file when the user touches the piano keys, and update current displaying piano keys to the correct keys when user moves the octave range to a new range.

Drum

The controller of drum should responsible for turning on/off the correct slot in the model of the drum machine when user touches that table cell. Moreover, it should deal with starting/stopping the drum machine when user click on the button, updating tempo value in model when user slides the tempo slider and updating the view which indicates the current beat on every beat of the drum machine.

3.5 Other Useful Things

As mentioned before, network communications are required throughout the project. In order to simplify the works and follow the guideline of Object-Oriented Programming, it is a good idea to put all the network communications into one place (that is, one class) and only expose some useful method as the public APIs. This class is responsible for all general purpose network communications such as sending and receiving information. I call it Network Helper class.

On the other hand, there are different instruments so that different information exchanging through network is needed. However, the helper class of network is for general purpose, which means it does not understand the needs of every instrument. One easy way to solve this problem is to create a subclass of the helper class for each instrument. This approach imposes the restriction on extensibility of the application since a new class is created for every newly added instrument and differences between those subclasses are small. My approach is to create a interface between the instruments and Network Helper. The interface is like a translator. The instrument tell the interface its needs of network communication, and then the interface translate those needs to the helper class.

IV. Implementation

4.1 Model Implementation

Preparation

Before implementing models of instruments, the Network Helper, the interface between Network Helper and instruments model, and the library to generate sound should be implemented and added to the project.

As introduced previously, the WLAN network communication makes use of **DTBonjour**. Two classes are created as the server and client in Bonjour connection. They are the subclasses of `DTBonjourServer` and `DTBonjourDataConnection`, and they will be used in Network Helper.

Network Helper class

The Network Helper class, called `IBNetworkHelper`, has a delegate method, two public APIs, and a property.

The delegate method is

```
-(void)didUpdateBonjourDeviceList;
```

It is implemented in the delegate of `IBNetworkHelper`, and will be called whenever there is a new device in the same WLAN opens this App.

The public APIs are

```
- (void)startBonjour;
- (void)updateInstrument:(IBInstrumentType)instrument
                withInfo:(NSDictionary *)dic;
```

They are the method to start WLAN network listening, and send update of a certain instrument with the details in the `NSDictionary`, which is key/value pairs.

The properties are

```
@property (nonatomic, readonly, strong) NSArray *deviceList;
@property (nonatomic, weak) id <IBNetworkHelperDelegate> delegate;
```

The first one is the list of all available devices in the same WLAN. The second is the pointer to the delegate of `IBNetworkHelper`. It should implement the delegate method.

The Interface Between Network and Instrument

On the other hand, the interface, called `IBNetworkInstrumentInterface`, provides many class method (begin with +) for the instruments to send update through network, such as:

```
+ (void)updateInstrumentWithDic:(NSDictionary *)updateDic;
+ (void)sendDrumUpdateWithDrumType:(IBDrumType)type
                           atIndex:(NSUInteger)index isOn:(BOOL)isOn;
+ (void)sendDrumUpdateWithTempo:(NSUInteger)tempo;
+ (void)sendDrumUpdateWithStartOrStop:(BOOL)start;
+ (void)sendPianoUpdateWithKeyIndex:(NSUInteger)index isOn:(BOOL)isOn;
```

Drum

I used a similar open-source project called **BBGroover** (https://github.com/pwightman/BBGroover), which is an easy-to-use scheduling/sequencing library for drum beats, and I made my own modification.

The public APIs provide functions as follows:

```
- (void)turnOnDrum:(IBDrumType)type atIndex:(NSUInteger)index;
- (void)turnOffDrum:(IBDrumType)type atIndex:(NSUInteger)index;
- (void)updateTempo:(NSUInteger)tempo;

- (void)start;
- (void)stop;
- (void)resume;
- (void)pause;
```

The functions are just as the same as the names of those methods.

Meanwhile, the model of the drum provides some delegate methods for other classes to use, such as:

```
- (void)didTick:(NSUInteger)tick;
- (void)didUpdateTempoTo:(NSUInteger)newTempo;
- (void)didUpdateModel;
```

Those method are optionally implemented in the delegate.

Piano

The implementation of model of piano has only one public API, which is:

```
- (void)playKeyAtIndex:(NSUInteger)index;
```

It will generate sound of the required piano key.

4.2 View Implementation

Drum

Following the previous design, two classes is needed for beats update and drum machine.

The beats view is called `BBTickView`, and the view of drum machine is called `BBGridView`, They are extracted from **BBGroover**. In reality, the tick view is just a floating

little dark grey rectangle above the grid view of the drum machine. The tick will change its place with the update of current beats. The grid view is just a table, with turned-on slot marked dark grey.

Those two classes both have `delegate` and `datasource` to provide data that the view is needed because in MVC design pattern, View does not contain any data. Instead, its data is provided by Controller. The `datasource` of the `BBGridView` provides the information of how many columns and rows, and the delegate is like the callback of user interaction. They are as follows:

Delegate:

```
- (void)gridView:(BBGridView *)gridView
wasSelectedAtRow:(NSUInteger)row
          column:(NSUInteger)column;
- (BOOL)gridView:(BBGridView *)gridView
 isSelectedAtRow:(NSUInteger)row
          column:(NSUInteger)column;
```

Data Source:

```
- (NSUInteger)rowsForGridView:(BBGridView *)gridView;
- (NSUInteger)gridView:(BBGridView *)gridView
         columnsForRow:(NSUInteger)row;
```

Finally, the whole look of the drum is as follows:

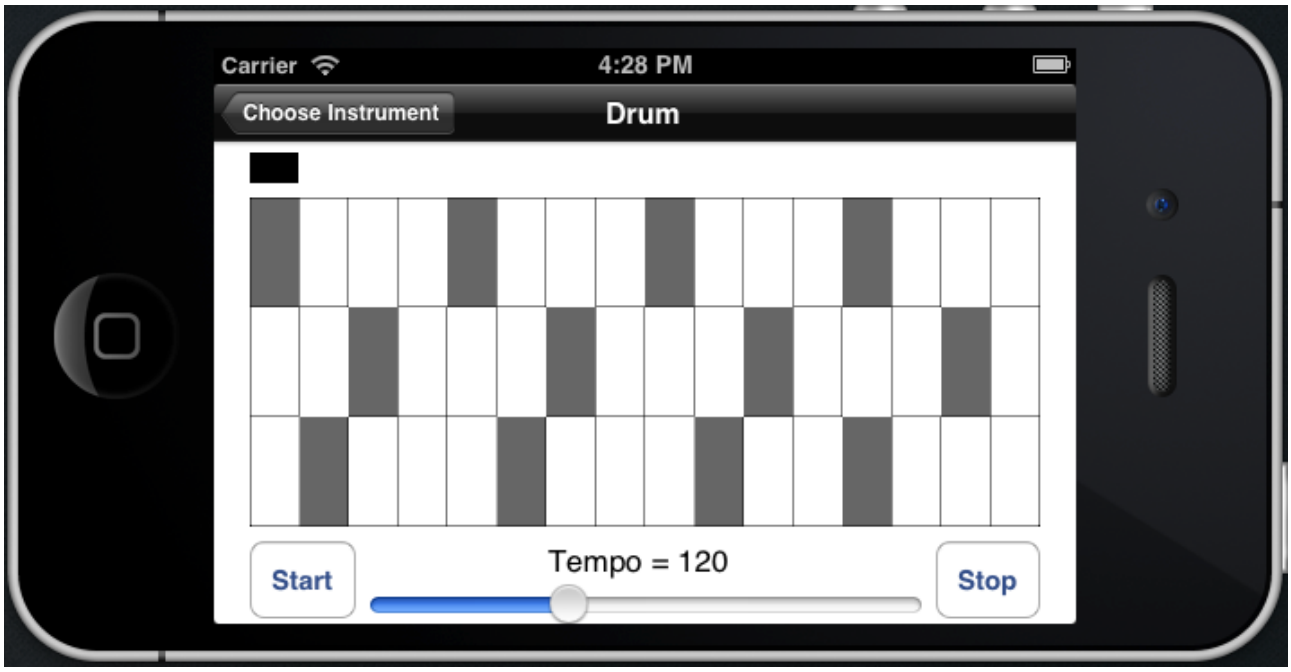

Figure 2. The view of drum machine.

Piano

The view of the piano is complicated. Basically, there is a view to select range of octave and a view to show the keyboard. They follows the requirement in the design stage. Details can be found in the source code.

The final view of the piano is as follows:

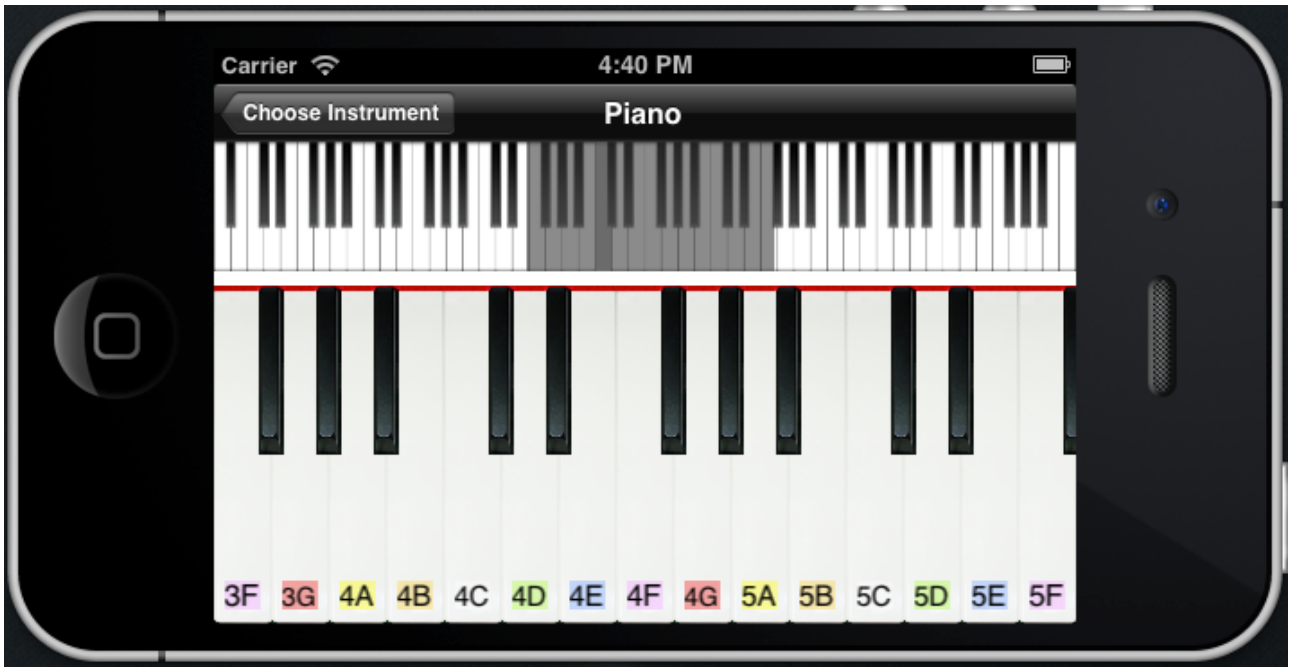

Figure 3. The view of piano

4.3 Controller Implementation

The Controllers of Instruments

Different controller of different type of instrument behave almost the same, except for calling specific functions in different instruments.

Firstly, the controllers implement methods to deal with the user interaction on the device. This is achieved by target/action mechanism and implement delegate method provided by view classes. Basically, target/action mechanism is an easy why to deal with user interaction such as pressing a button, and delegate method is the way to let controllers know that a user gesture such as tap, pin, pan, etc., happens and the implementation is the handler of those gestures. In the implementation, generally the controller will call the method provided by models based on the parameters passed by view classes. Meanwhile, each user interaction will trigger a network communication, if needed, in order to let all the devices in the same WLAN have the same response.

Secondly, the controllers are responsible for changing the views if the models have updates. Models will be updated through network communications or user interaction. The changes will be passed to controllers by delegates too.

The Other Controllers

Some other controllers are needed to fulfill the App. One is the controller to choose which instrument to use, and the other one is to show the device list that are connected in the same WLAN. The controller of choosing instrument is just a table to show all the instruments that are available. After clicking one instrument, it will segue to the view/controller of that instrument. The controller of showing band members are just list all the device names.

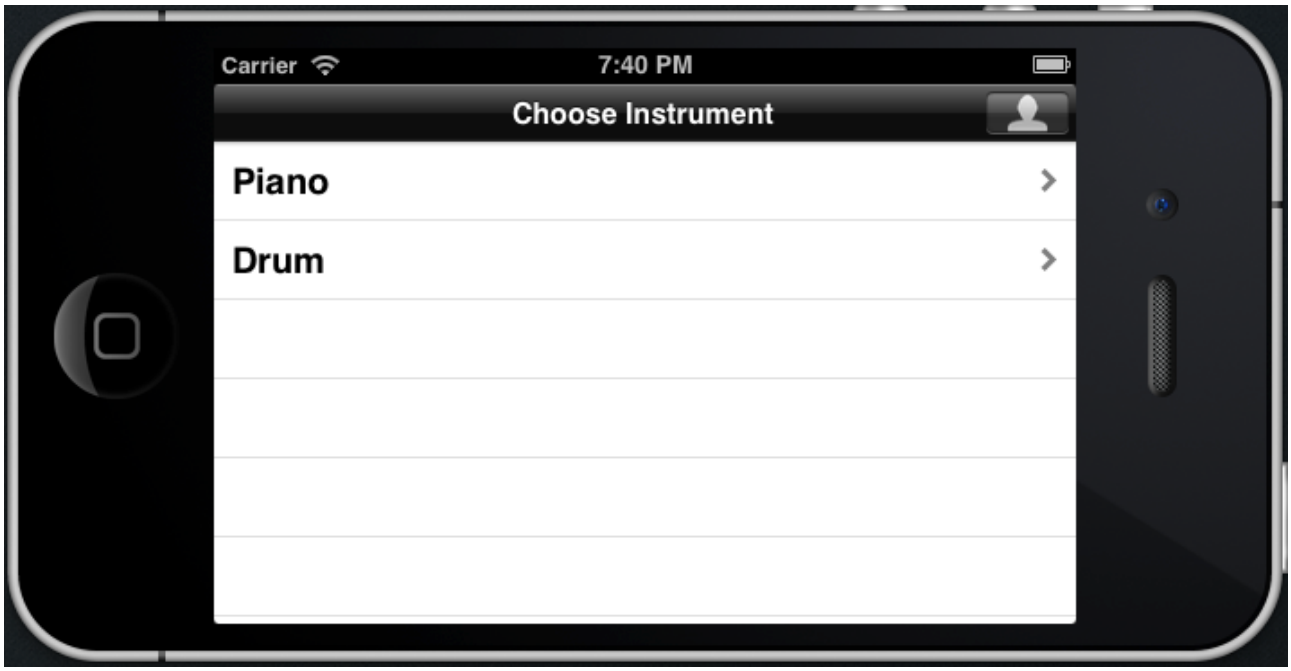

Figure 4. The Controller of choosing instrument

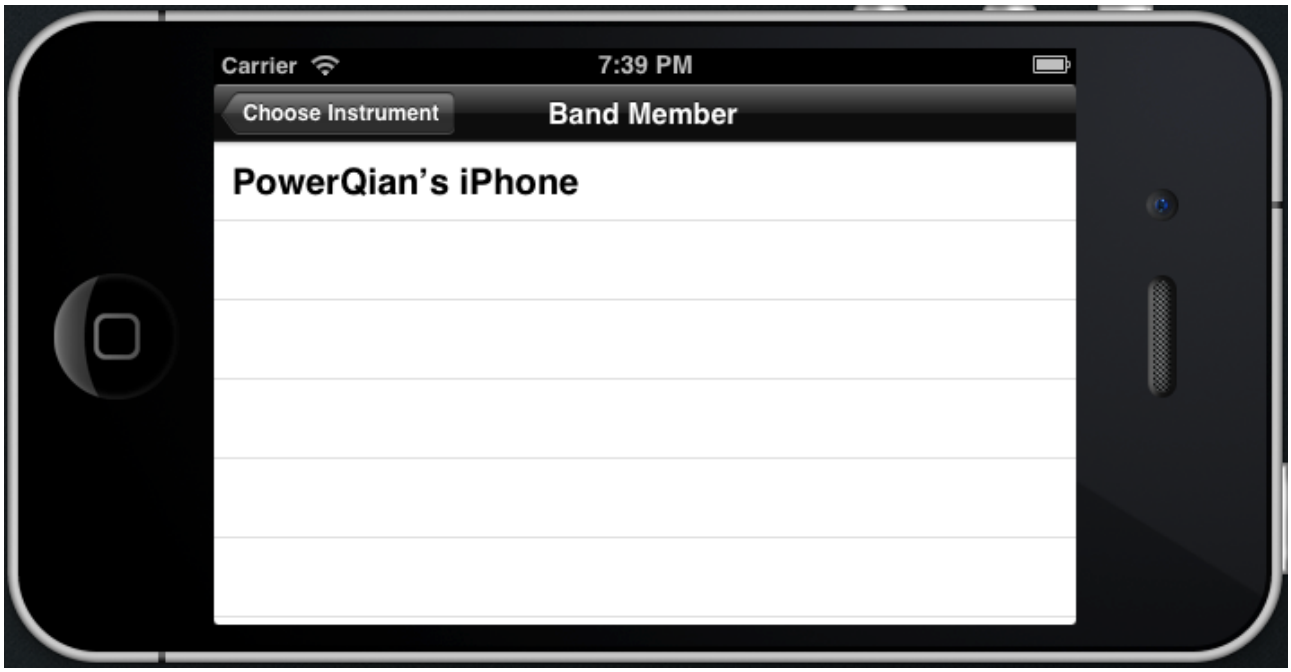

Figure 5. The Controller of showing band members

4.4 Summary

After all the implementations, the final relationship diagram is as follows:

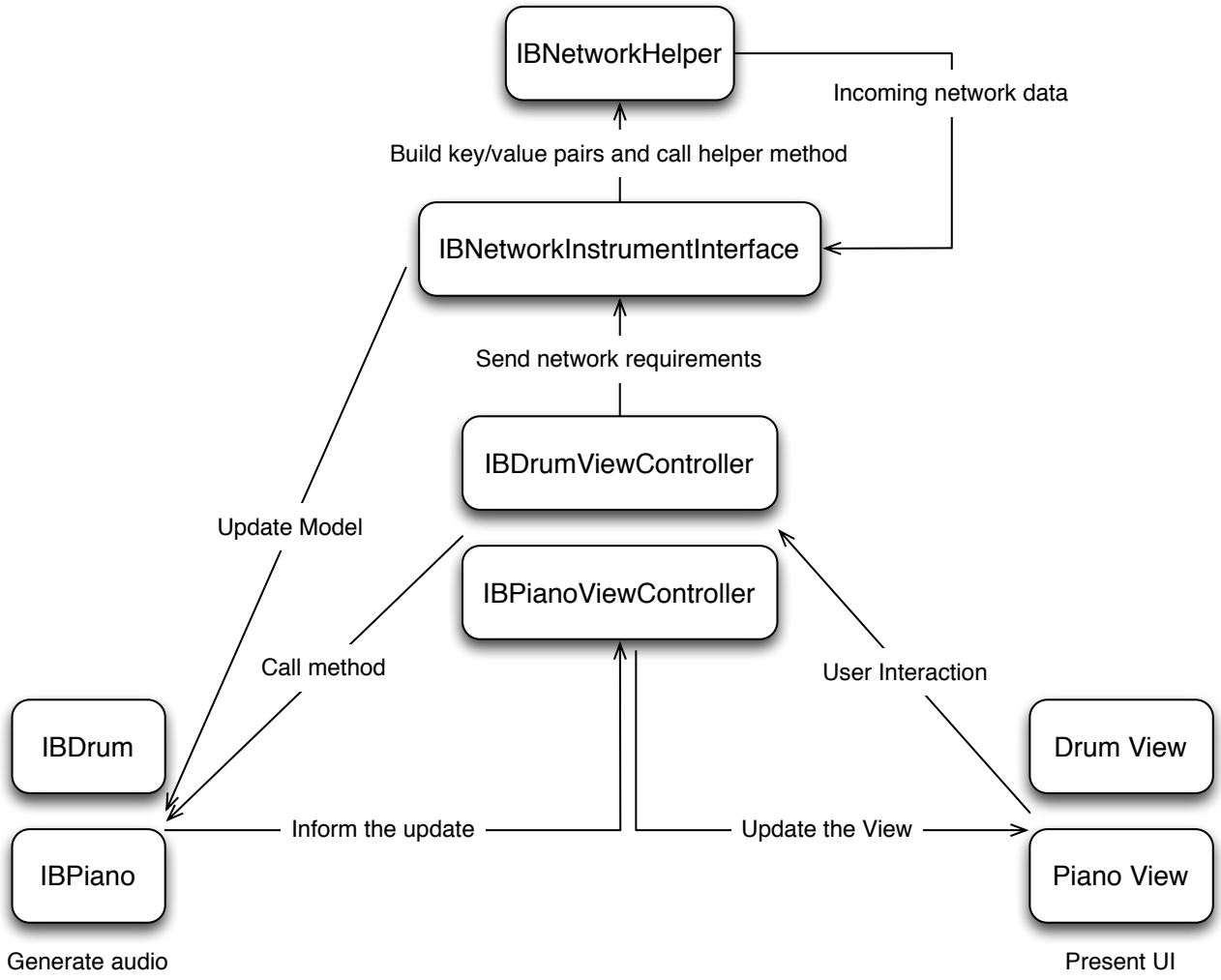

Figure 6. The relationship between classes

V. Conclusion

This **Musical Instrument App on Mobile Devices** project, which I call **iBand**, is an entertaining iOS App that let users play instruments together through Wireless LAN. Currently, it consists of two instruments, piano and drum. More instruments can be easily added using the structure of the existing code. The sound of the instruments uses WAV audio files. The audio files is played efficiently using **ObjectAL** library. All joined devices exchange their sound information through WLAN so that they all have the same sound effect. **DTBonjour** is introduced to do the WLAN network communication. Each time user presses a piano key or turn on one slot in drum machine, the audio will be played and a message will be sent through WLAN to other devices to tell them to play that audio.

This project requires different area of knowledge and the strong programming ability. The whole structure of the project needs to be carefully designed since there are two major part, network and audio, to coordinate well with each other. Meanwhile, the implementation of instruments makes use of the advantage of Object-Oriented Programming so that it provides good extensibility.

The final result of this project achieves the goals and meets the requirements in the design stage with satisfaction. The App is used in all my devices perfectly.

Source Code

The source code of this project can be found at https://github.com/powerqian/iBand.

Demo Video

YouTube: http://youtu.be/rmv9mLy9mq4

Youku: http://v.youku.com/v_show/id_XNTU3NjgyMjYw.html

Tudou: http://www.tudou.com/programs/view/slTKw3t4fPY/

VI. Appendix

Minutes of the 1st Project Meeting

Date: Thursday, 7 March 2013

Time: 11:30 AM

Place: Room 3512

Attending: Prof. Rossiter, QIAN Li

Absent: None

Recorder: QIAN Li

Approval of minutes

This is first formal group meeting, so there were no minutes to approve.

Report on Progress

QIAN Li demonstrated his idea on this project and showed a piano App.

Discussion Items and Things To Do

- Scope of the project
- Predesign for the project
- Support Wireless LAN network communication.

Meeting adjournment

The meeting was adjourned at 12:00 PM.

Minutes of the 2nd Project Meeting

Date: Wednesday, 27 March 2013

Time: 11:45 AM

Place: Room 3512

Attending: Prof. Rossiter, QIAN Li

Absent: None

Recorder: QIAN Li

Approval of minutes

The minutes of the last meeting were approved without amendment.

Report on Progress

QIAN Li made the demo of playing piano on one device and the piano is also played on the other device, which means the Wireless LAN network communication works.

Discussion Items and Things To Do

- Improvements on piano and adding other sound of instrument.

Meeting adjournment

The meeting was adjourned at 12:15 PM.

Minutes of the 3rd Project Meeting

Date: Monday, 15 April 2013

Time: 12:00 PM

Place: Room 3512

Attending: Prof. Rossiter, QIAN Li

Absent: None

Recorder: QIAN Li

Approval of minutes

The minutes of the last meeting were approved without amendment.

Report on Progress

QIAN Li demonstrated the drum machine.

Discussion Items and Things To Do

- Making the drum machine work with Wireless LAN network.

Meeting adjournment

The meeting was adjourned at 12:30 PM.

Minutes of the 4th Project Meeting

Date: Thursday, 16 May 2013

Time: 12:00 PM

Place: Room 3512

Attending: Prof. Rossiter, QIAN Li

Absent: None

Recorder: QIAN Li

Approval of minutes

The minutes of the last meeting were approved without amendment.

Report on Progress

QIAN Li demonstrated the final result of the project, which meet all the requirements.

Discussion Items and Things To Do

- Finish the report
- Record a demonstration video

Meeting adjournment

The meeting was adjourned at 12:30 PM.