GeoChat on iPhone
A Map-based Chat Application on iPhone

Li Lian
Abstract

**GeoChat on iPhone: A Map-based Chat Application on iPhone**

Li Lian

Mobile applications, which have been widely used, are influencing our daily lives more and more deeply. A considerable number of advanced technologies and innovative experiments are being undertaken by developers in order to bring more convenience and enjoyment to people. One of the hot investigations in mobile applications is map-based services for smart phones.

This thesis will address the implementation of a map-based chat application for iPhone subscribers named GeoChat, which has been previously developed using Java ME. As implied by the name, a user can search and join chat rooms on geographical locations, and talk with the others who are interested in this location too. A user is also allowed to create chat rooms and propose topics for the places according to his or her preferences.

Since the development of iPhone application is not as fledged as that of Java ME, a comparison between the development processes of iPhone SDK and Java ME will be presented after introducing the implementation procedure of GeoChat on iPhone, through which we can learn advantages from these two development platforms and achieve more with less effort in the future.

Handledare: Lars-Erik Holmquist
Åmnesgranskare: Olle Eriksson
Examinator: Anders Jansson
IT 10 037
Tryckt av: Reprocentralen ITC
# Contents

I Introduction ................................................................................................................................................. 3  
1.1 Background ............................................................................................................................................... 3  
1.2 Motivation .................................................................................................................................................. 4  
1.3 Goals ....................................................................................................................................................... 5  
1.4 Delimitation .............................................................................................................................................. 6  

II Method .......................................................................................................................................................... 7  
2.1 Knowledge study ....................................................................................................................................... 7  
  2.1.1 IDE (Integrated Development Environment) ....................................................................................... 7  
  2.1.2 Cocoa .................................................................................................................................................. 7  
  2.1.3 Programming language ...................................................................................................................... 7  
  2.1.4 Review of previous Java version ........................................................................................................ 8  
  2.1.5 Data storage on Server ...................................................................................................................... 9  
2.2 Problem Analysis ..................................................................................................................................... 10  
  2.2.1 Map ................................................................................................................................................... 10  
  2.2.2 Chat room marker ............................................................................................................................ 12  
2.3 Use case .................................................................................................................................................. 14  

III Design .......................................................................................................................................................... 17  
3.1 Software architecture ............................................................................................................................... 17  
3.2 MVC Design Pattern .............................................................................................................................. 17  
3.3 Delegation Design Pattern .................................................................................................................... 19  
3.4 User Interface design ............................................................................................................................. 21  

IV Implementation .......................................................................................................................................... 23  
4.1 Flow chart ................................................................................................................................................. 23  
4.2 Navigation controller ............................................................................................................................... 24  
4.3 View controllers ...................................................................................................................................... 25  
  4.3.1 Initial view controller ......................................................................................................................... 25  
  4.3.2 Map view controller ......................................................................................................................... 26  
  4.3.3 Chat room view controller .............................................................................................................. 26  
4.4 Network communication ......................................................................................................................... 27
I Introduction

Since the launch of the first commercial citywide cellular network in Japan by NTT in 1979, mobile phones have been invented and used for 30 years, and nowadays are ubiquitous in people's daily life. They were not only improved in appearance, but also embedded with more functionality. From the first generation of mobile phone to the up-to-date smart phone, software development technology for mobiles has advanced fleetly during the decades. With a huge increment of mobile phone subscribers, more and more applications and software come with the tide of fashion. IPhone, one typical smart phone which was released in June 2007, has made Apple briefly the third largest mobile phone manufacturer by revenue, after Nokia and Samsung within two years [7]. To date (Oct. 2009), the number of available iPhone applications almost reaches 77,000, which is a great amount compared to other devices [5]. The free SDK and convenient iPhone simulator provide a very straightforward path for engineers to realize whatever ideas they imagine. As the market share of iPhone all over the world is growing tremendously, there is no surprise that uncountable iPhone applications will be published frequently in the future. This is exactly what we would like to see -- these inventive applications assist our business and decorate our life better and better.

This thesis will present an iPhone application named “GeoChat”, which is a map-based chat application for users to communicate with people related with geographical locations. Using GeoChat, users can navigate on the map to find chat rooms in different places, zoom levels and various topics. If an idea pops up in mind, one can also create a new chat room on the map and entitle it with the topic that he or she prefers.

1.1 Background

Applications developed for mobile devices have been widely used since the beginning of this century. It becomes a prevalent trend that more and more mobile programs are being developed and put into use among users, since people can get information, gain knowledge and relax by using them. Thanks to the rapid development of electronic products, we can manage numerous unbelievable businesses by mobile devices now. Besides the functions that are indispensable for ordinary mobile phones, such as making calls and sending messages, thousands of applications for different purposes are booming every year. Mobile phone subscribers can make use of them to read web feeds, send emails, listen to music,
and entertain. With the avalanche of smart phones, mobile devices can be considered as computers in hands, which can deal with various affairs while saving space and effort, rather than just be “phones”.

Mobile Life Centre, in which this thesis is held, is a joint venture that does research in mobile services and ubiquitous computing in Kista, Sweden. There are four groups, namely Involve, Future Applications Lab, Game and Mobility. GeoChat was previously developed using Java by the Future Applications Lab (FAL). Since map-based chat application is novel although communication tools are ubiquitous in our routine life, FAL group would like to implement a version of GeoChat on iPhone so that it can be applied in another widely used platform. Meanwhile, I am also responsible to work on the comparison between these two versions from different perspectives.

1.2 Motivation

Map services are getting more and more popular currently as the online map and GPS (Global Positioning System) have been pervasively used. There are several great corporations that have integrated global map services, like Google and Yahoo!, which allow users to view the map in different modes as standard, hybrid and satellite. However, the core applications for these services are finding locations and ways on the map but not communication. Motivated by creating an application that is based on map but also able to let people communicate with each other, the mobile software GeoChat is developed by Mattias Rost from Mobile Life using Java.

Chat rooms are very prevalent nowadays especially on personal computers. However, to embed an application which performs the utility of a chat room on a mobile device is rare till now. Additionally, the chat rooms previously used are mostly subject-oriented, namely the chat rooms are distinguished by topics. Nevertheless, communication based on geographical areas also make great sense as people may like to know something in a particular place or have strong interest in a certain area. For instance, a student who will be exchanged to another university can chat with people currently studying there or who are familiar with the city where that university locates; a person who wants to travel to a resort will likely get some useful information from people who know there well. By doing this thesis, the iPhone GeoChat application can be used by the iPhone users, and should be able to communicate with the others who have GeoChat installed on other Java-supported devices, e.g. Nokia and Sony Ericsson, as well. For the reason that iPhone development is not as universal as other mobile systems, we also would like to study the disparity of developing process between Java ME and iPhone SDK (implemented
by Objective-C) after finishing the implementation. On all accounts, we hope this GeoChat could combine topics with geographical locations, in other words, people can talk about something which is correlative with a place.

### 1.3 Goals

This application should run on the device which supports Apple iPhone OS 3.0 or later version. The aim of this thesis is to implement GeoChat on iPhone and analyze the difference of developing processes between Java ME and iPhone SDK. The goals are as follows.

- Implement a map as the bottom layer of GeoChat which can be used to zoom and navigate on.
- Get chat room information and conversations from the server and update in time.
- Mark chat rooms of different zoom level with different tags.
- Create chat rooms on any position of the map.
- Implement the features of a chat room with “message box”, “input area”, “view chats”, “check participants” etc.
- Make sure GeoChat of iPhone version and Java version can interact correctly.
- Install GeoChat on real devices, i.e. iPhone or iPod Touch, and test it.
- Write the analysis of the difference of developing process between Java ME and iPhone SDK.

Eventually, the GeoChat on iPhone should be able to communicate with that of Java version.
1.4 Delimitation

The chat rooms of GeoChat exist there once created. Chat rooms which are not visited for a while, e.g. one month, should be automatically removed from the server, otherwise there will be more and more chat rooms that are not being used at all and the server cannot afford this finally on some day. The data on the server is not managed automatically yet. For GeoChat on iPhone, there is only one chat room for each geographical location, which means only one chat room could be set at one absolute place. However, since this is a chat application and the chat rooms are not necessary to be set at precious positions, it is almost not possible to have more than one chat room at the same location with the accuracy to 6 decimals, especially for there are not many chat rooms on the map currently.
II Method

2.1 Knowledge study

2.1.1 IDE (Integrated Development Environment)

Xcode is the development environment of Apple’s Mac OS X. At the heart of the Xcode tools package is the Xcode IDE, a graphical workbench that tightly integrates a professional text editor, a robust build system, a debugger, and the powerful GCC compiler capable of targeting Intel and PowerPC regardless of host platform. Xcode is both easy to use, and yet powerful enough to build the largest Mac OS X applications. For my thesis, I used Xcode 3.1 to develop GeoChat under Mac OS 10.5.

2.1.2 Cocoa

Cocoa is one of the five APIs of Mac OS X, and typically Cocoa applications are developed with Xcode. Using Cocoa with the Xcode IDE is simply the best way to create native Mac applications. What is more, Cocoa uses the Model-View-Controller (MVC) design pattern throughout. That is also the way that I follow to design and implement my application, but I apply some other design patterns together with it.

2.1.3 Programming language

Apple has its particular programming language for developing applications on Mac OS X and iPhone OS, named Objective-C. It is an object-oriented programming language integrates ANSI C language and extends it to full object-oriented capabilities in a simple and straightforward way. The syntax of Objective-C is somewhat different from other object-oriented programming language, and I would like to introduce it a bit by giving several typical examples.

1) Data type
Objective-C supports the basic data types of C. Besides that, it has a special data type: id, which is the general type for any kind of object regardless of class and defined as pointer to an object data structure. Sometimes we use this weak type to declare a variable which can accept arbitrary object.

2) Method declaration
In Objective-C, there are two types of methods: class method and instance method. An instance method applies to an instance of the class whereas a class method applies to the class itself.
Take an instance of an instance method, the syntax of its declaration is:

-(void) insertObject: (id) anObject atIndex:(NSInteger) index;

where “-” is the method type identifier, “void” is the return type, “id” and “NSInteger” are parameter types, “anObject” and “index” are parameter name, while “atIndex” is a prompt of the second parameter. The name of the method actually is “insertObject: atIndex:” including the colons in the double quotation marks.

3) Message

When we want to call a method, we do so by “messaging” the corresponding object.

[myArray insertObject: anObj atIndex: 0];

This will insert anObj to the first index of myArray. Messages are embraced by [].

4) Protocol

A protocol is similar to the “interface” in Java and C#. It is a set of methods which any class can declare to implement, but it only defines some method signatures without any implementation.

Current version of Objective-C supports a protocol with optional methods so that a class does not have to implement all the methods in a protocol but only the mandatories. The syntax of define a protocol is:

@protocol FlipsideViewControllerDelegate
- (void)flipsideViewControllerDidFinish:(FlipsideViewController *)controller;
@end

This is a piece of code from my project. In this protocol, only one method is declared to flip the window to the other side, and it is implemented by Class NickViewController, thus the body of this method will be defined in that class.

2.1.4 Review of previous Java version
Before getting down to program iPhone version's GeoChat, I reviewed the completed version of GeoChat written in Java, experienced it on a Sony-Ericsson smart phone, and got an outline about this software. The starting point of it was displaying a map on the screen. Stretching from this point, any other operations and functionalities related with the map were implemented one by one. I was planning to imitate this and tried to use the same way to implement GeoChat on iPhone. However, with deeper understanding of iPhone application development, I learnt the design patterns in Objective-C programming and found they can formulate the development process more neatly and simply. Hence, I used the MVC and delegation design pattern in my application, and implemented it in a different manner.

2.1.5 Data storage on Server

As my application will share the database with the Java version, I studied the data storage on the server side in order to store the data of GeoChat on iPhone in the same database directly and uniformly. When accessing the server, the database administrator can operate different commands by giving relevant arguments. The following pictures show all the communication actions between client and server. I will introduce the implementation in chapter 5.
When launching GeoChat, the application begins with an input dialog which requires a nickname. After input a valid nickname (without special characters), a session ID is generated which is named chatid in the server. It is considered as an identification of a nickname and passed as a parameter to access the database, and will be invalid until user end up the application. Thereby, it determines the success or failure of turning into next view or command.

2.2 Problem Analysis

2.2.1 Map

An essential element of this application is the map, which fits with the title properly. In the Java version of GeoChat, it was implemented by downloading Yahoo! map from their server tile by tile and displayed on the window. However, when developing it on iPhone, it is very exciting to obtain the newest framework -- Map Kit, which could be employed perfectly to satisfy all the requirements that GeoChat's map needs. The definition of the Map Kit framework in iPhone OS reference library is as follows.

The Map Kit framework provides an embeddable map interface for your application. Use it to display map or satellite imagery from the windows and views of your custom applications. You can also use the framework to annotate your maps with points of interest and other custom information [6].

Once the map is added as a layer of the application, the typical iPhone gestures are integrated at that time. Users can pan and zoom the map freely. Pinching, which is considered as a distinct feature of iPhone can be used as well. However, single touch cannot be captured on the map view since it is disabled in order to fulfill the other functions like zooming, panning, pinching. If a touch event could be responded by the screen, it will take a lot of resource and memory to get the touch information every time when user is navigating on the map. However, the continuously generated location information is unnecessary most of the time. Thereby, I used the class “MKAnnotation” of map kit instead, to achieve selecting chat room. This is
quite different from the java version since that was based on keyboard control. The annotation problem will be introduced in next subsection.

Since the chat rooms varied according to zoom levels, it is required to get the current zoom level of the map whenever it is moved. There is not any available method in the library which can be used directly to get the current zoom level of the map, so I have to define a method by myself to achieve this. In order to find a proper solution to get the zoom level, I counted that there are 17 levels of the embedded Google map of MKMapView class. The number of zoom levels is the same as that of Yahoo! map used in Java version. It is significant to get the consistency because they share the same server side.

I recorded the movement of longitude and latitude when zooming in and out the map and found the longitude changed in geometric proportion but the latitude did not. Therefore, I use the change of longitude to define zoom levels and the method is as follows. Owing to the pinch gesture on iPhone, we may not stay at a integral zoom level, saying zoom level 7 or 8, but in between like zoom level 7.6. This may be the reason that why MKMapView does not have a method of getting current zoom level, because we can zoom to whatever extent as we want to. Therefore, we have to consider the zoom level of a chat room is integer although it may be not. Otherwise, it will not match that of the Java version. Hereby, I assign the current zoom level (czl) belonging to [n, n+1) to be n, in other words if n <= czl < (n+1), then czl = n, where n is the integral zoom level. For instance, if the real czl = 6.5, we consider it is 6.

```c
-(int)getZoomLevel:(MKMapView *) m_map
{
    int n = 0;
    int zoomlevel;
    CLLocationCoordinateDegrees lon = 225;
    CLLocationCoordinateDegrees currentLonSpan;
    currentLonSpan = m_map.region.span.longitudeDelta;
    if((currentLonSpan > (lon/pow(2, (n+1))) && (currentLonSpan <= lon))
    {
        zoomlevel = 0;
        return zoomlevel;
    }
    for (n = 1; n < 18; n++)
    {
        if((currentLonSpan > (lon/pow(2, (n+1))) &&
            (currentLonSpan <= (lon / pow(2, n))))
        {
```
2.2.2 Chat room marker

In Java version's GeoChat, the chat rooms are marked as squares. When navigating on the map, there might be some overlaps of the chat rooms so that people can switch between them and choose which one they want to join.

![Figure 3: chat room markers in Java version](image)

However, the iPhone Map Kit disabled the single touch capability on the map, which means, users can not touch on these “squares” to choose chat rooms. There is a solution to get the touch point on the map, which could be described in a few steps:

1. Add a transparent sub view to the map view.
2. Call methods to get touch event on that view.
3. Pass the point value to map view and convert it to longitude and latitude.
4. Remove the sub view.
Nevertheless, the map is “dead” -- cannot move at all when using this method to solve the problem. Thus, I have to give up using squares to represent chat rooms but find another approach to implement it. I tried to find a method to get a chat room's information when touching on the markers and noticed a class entitled “MKAnnotationView” in Map Kit, which inspired me to denote the chat rooms with annotations instead of squares.

[Figure 4: chat room markers in iPhone version]

In the previous Java version, the information of a chat room, such as longitude and latitude, is saved as Mercator projection value on the server. Because the map is displayed on the screen tile by tile after the real map is Mercator projected to a plane.

The Mercator projection is a cylindrical map projection presented by the Flemish geographer and cartographer Gerardus Mercator, in 1569. It became the standard map projection for nautical purposes because of its ability to represent lines of constant course, known as rhomb lines or loxodromic, as straight segments. While the linear scale is constant in all directions around any point, thus preserving the angles and the shapes of small objects (which makes the projection conformal), the Mercator projection distorts the size and shape of large objects, as the scale increases from the Equator to the poles, where it becomes infinite [8].

Thus, for Java version, the coordinate of each point on the map is a MP value, and is stored on the server in this type. However, we can directly call methods to get the geographical latitude and longitude value of each point on the map of MKMapView in iPhone version. But in order to keep the identity of two versions values storing on
the server, I need to converse the real latitude and longitude to MP value while communicating with server. Or else, different type of mobile subscribers cannot communicate with each other using GeoChat since the same chat rooms may not be found at the same location on the map. By converting the longitude and latitude values, I use mathematic formulas of Mercator projection. The following equations determine the x and y coordinates of a point on a Mercator map from its latitude φ and longitude λ (with λ₀ being the longitude in the center of map).

\[
\begin{align*}
x &= \lambda - \lambda_0 \\
y &= \ln \left( \tan \left( \frac{\pi}{4} + \frac{\varphi}{2} \right) \right) \\
&= \frac{1}{2} \ln \left( \frac{1 + \sin(\varphi)}{1 - \sin(\varphi)} \right) \\
&= \sinh^{-1}(\tan(\varphi)) \\
&= \tanh^{-1}(\sin(\varphi)) \\
&= \ln(\tan(\varphi) + \sec(\varphi)).
\end{align*}
\]

This is the inverse of the Gudermannian function:

\[
\begin{align*}
\varphi &= 2 \tan^{-1}(e^y) - \frac{\pi}{2} \\
&= \tanh^{-1}(\sinh(y)) \\
\lambda &= x + \lambda_0.
\end{align*}
\]

[Figure 5: mathematic formula of Mercator projection]

The scale is proportional to the secant of the latitude φ, getting arbitrarily large near the poles, where φ = ±90°. Moreover, as seen from the formulas, the pole's y is plus or minus infinity. Making use of these formulas, I convert the chat rooms' geographical longitude and latitude into MP values and send them to the server when users create new chat rooms; while displaying the existed chat rooms, I get MP values from the server first and convert them in order to show particular chat rooms on the map view layer on iPhone version.

2.3 Use case

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. It is made up of a set of possible sequences of interactions between systems and users in a particular environment. It consists of a group of elements (for example, classes and interfaces) that can be used together in a way that will have an effect larger than the sum of the separate elements combined. A use case should contain all system activities that have significance to the users, thus
it can be thought as a collection of possible scenarios related to a specific goal, indeed, the use case and goal are sometimes considered to be synonymous [1]. Here is the use case diagram for client and server sides of GeoChat.

- **Navigate on the map**: Basically, this is a map-based application, so it could be used as an ordinary map at least and has all the functions that a map service includes.
- **Find chat rooms**: For those who are interested in a particular location, they can try to find chat rooms there.
- **Create chat room**: Sometimes, there may be no chat room exists in the place that the user would like to chat. Thereby one can create a chat room in an interesting place.
- **Chat**: This is what we do in a chat room.
- **View participants**: Users can know how many people are there in the chat room and who are they.
- **Change nickname**: If one would like to change another nickname, it is viable in GeoChat.
- **Locate me**: It is a new function added to GeoChat, which makes much sense for new comers or visitors to a strange place. By clicking “Locate Me”, users will see their current place on the map, which helps them find the place they are locating in and probably some chat rooms around her/him in different zoom level.

![Use Case Diagram for Client](image)

As for the server side, the use cases are as follows.
- Login: It is much secure to access the system.
- Logout: Quit the system in secure way in case data lost.
- Update chat rooms: The administrator can decide which chat rooms should be removed from server for a proper reason.
- Manage users: The administrator has the permission to manage users, i.e. decline their access.
- Settings: All the information in the database could be modified.

[Figure 7: use cases for administrator]
III Design

3.1 Software architecture

This is a client/server mode software overall. Owing to the shared database of GeoChat in two versions, I implemented the client side of this software and fulfill the data transportation protocol according to the implementation on the server side of Java version.

3.2 MVC Design Pattern

MVC model is the most used design pattern which assigns objects as one of the roles -- model, view and controller in Cocoa applications. It does not only define the role of an object but also specifies the relationships between them as well as the communicating ways. Each role is comparatively independent from the other and sometimes referred as a layer, such as control layer [2].
The advantages of MVC model are numerous. First of all, applications having an MVC design are more easily extensible than other applications; moreover, the objects are highly reusable since they are separated from each other; additionally, it is much convenient to modify interfaces; last but not least, this kind of design pattern could give a clear structure of the application and enhance the efficiency of the implementation. For the GeoChat application, I use this model to design and assign different tasks.

Model --- This role is considered as a data encapsulation for the attributes of an object, as well as the logic of processing the data. In GeoChat, the Class ChatRoom, ChatMessage and MapAnnotation are of this type, where correlative arguments of the model are defined and the setup methods are fulfilled.

View --- This is the interface of an application and also the channel that users interact with the application. It should respond to user's actions, such as input and touch event. In iPhone SDK, there is an integrated tool named Interface Builder, which helps programmers design their view directly with various components, e.g., navigation bar, button, label, text field and so on.

Interface Builder is an application for designing and testing user interfaces. Developers can use Interface Builder to create user interfaces that follow the Mac OS X human-interface guidelines by dragging user-interface elements from a palette of predefined controls and dropping them into the window or view they are configuring. Interface Builder works closely with Xcode to provide a development experience that facilitates the concurrent but specialized development of an application's user interface and business logic [3].

Normally, the file ended with “.nib” is created from Interface Builder and represents views of different layers. However, the view could also be defined in the code
programmatically. GeoChat has several nib files representing the interfaces of different views, some of them were implemented by drag-in components in Interface Builder, while others were defined and added to views in the code.

Controller --- It is the intermediary of the View and Model roles of the application. Controller object performs as a medium responding user actions and updating the Models, and it also manages of updating views when Model objects are modified. In certain points, I figure it like a collection of methods that implement the functionalities of an application, and it is the heart of the application for its dispatching ability. In my application, the controller takes the most significant place indubitably for almost every required function of GeoChat are implemented in each view's corresponding view controller.

3.3 Delegation Design Pattern

Delegation design pattern is a pattern where one object periodically sends messages to another object specified as its delegate to ask for input or to notify the delegate that an event is occurring. We use it as an alternative to class inheritance for extending the functionality of reusable objects. It is similar as the interface in Java, which is a group of related methods with empty bodies. When a class declared to implement an interface, all methods defined in the interface must be implemented in that class. Here is the different way of implementing interface and delegate method in Java ME and iPhone SDK. First of all, the keyword interface in Java is entirely different from that in Objective-C. It is better to distinguish the difference in order to understand the example code clearly. In Java, the keyword interface is used to declare an interface which consists of a set of methods and can be implemented by classes. In Objective-C, the keyword interface is only used to declare a class in .h files.

```java
public interface DownloadListener
{
    public void downloadComplete();
}

public class MapLayer implements DownloadListener
{
    ...

    public void downloadComplete()
```
From the java code above, we can see that the interface DownloadListener involves only one method downloadComplete(), and the class MapLayer which implements this interface defined the function of this method. While, the delegation pattern in Cocoa is done in another way.

```
@protocol MyUIViewDelegate
-(id) fun1;
-(id) fun2;
@end

@interface MyUIViewController: UIViewController <MyUIViewDelegate>
...
@end

@implementation MyUIViewController
...
-(id) fun1
{
    ...
}
-(id) fun2
{
    ...
}
@end
```

```
@interface MyUIView
{
    id<MyUIViewDelegate> delegate;
}
```
In Objective-C, each definition, declaration or implementation is enclosed in two “@” denotation as shown in the code. A protocol only defines some method signatures without any implementation. As we can find the class MyUIViewController extends from UIViewController is declared after @interface, and conforms MyUIViewDelegate protocol methods (which is represented in the way of putting the protocol name in < >). So the methods in MyUIViewDelegate protocol are implemented in class MyUIViewController. And another class MyUIView declares a delegate of MyUIViewDelegate, which means the methods that are implemented in class MyUIViewController can be called by the delegate in class MyUIView.

### 3.4 User Interface design

User interface is the so-called “View” in MVC model. To design a pleasing interface, the need of users is the first factor to be considered. Meanwhile, it is very important to make the appearance of an application neat and without unnecessary attention. For iPhone application development, Interface Builder which has been introduced earlier is a convenient tool provided by iPhone SDK to design user interfaces.

Basically, there are three main views I designed for GeoChat on iPhone. For the initialized window, I set a prompt label and a text field to receive user's nickname. The interface of this initial window is mostly the same as the java version except a flip-side button on the lower right side of the screen, which displays a specification of GeoChat by flipping the window to the other side. I add this button because it is quite necessary for the users to read the general instruction of the functionality of this application. Additionally, this flip-side view animates attractively and saves more space.
After typing in a nickname, users will reach the most significant view of GeoChat, that is, the map view with chat room markers on it. In this view, I hide all the other interface components except two translucent buttons -- the back button and the option button, for displaying the map on the screen as much as possible. Or else, the upper screen will be taken up by a navigation bar which I use to switch between different views. All the functions that related with map, e.g., create chat room, toggle satellite, can be manipulated in this “map” view. Moreover, I use pin annotations in different colors instead of squares to represent chat rooms. The essential reason of this change has been interpreted in previous chapter. What is more, iPhone SDK has already provided this class for marking on the map, and the interaction is more direct with this vivid pin.

As for the view of the chat room, I prefer to have a natty and easy-interacting interface which gives users the clearest text field for chatting and reading messages. One bear watching point is that, since iPhone's virtual keyboard pops out which will lay over part of the input field when users touch editable fields on the screen, the position of the text view must be justified in order to make it visible for the users once the keyboard shows up. I achieve this by shrinking the displaying conversation field and stretch the input field when touch begins, and resume them when touch ends. If users would like to view the participants in a chat room, it is also feasible by touching the button “participants”. Under this situation, a sub view will be added to the chat message view and removed until users press the button “chat” to look through the messages.
IV Implementation

4.1 Flow chart

Before starting to program, I drew a flow chart of GeoChat in order to get a clearer path of implementing it. In this diagram, we can see the steps of manipulating this application as well as the general functions that GeoChat should include. The flow chart is important for that we will get a simple start point of the application and keep on expanding it to fulfill all the functions with more and more complex methods. The entry point of this program is the nickname view which requires an input nickname from the user and returns a session ID for the access for one time. If a session ID is successfully generated, it is passed as a parameter when users do every option until they log out this application. Thereby, it determines the success or failure of turning into next view. With a valid session ID, users will be taken to the central view -- the map view of this application, where the map could be set to “standard” or “satellite” mode. It is quite distinct that most of the functions of GeoChat are accomplished in this map view except chat. Hence, I can create a separate view for chat and implement all targets by following this flow chart.

[Figure 10: Flow chart]
4.2 Navigation controller

Initially, taking Java version as reference, I also attempted to use a basic window for the application and add or remove views onto or from it to fulfill different functions. For example, when asking the user to input a nickname, I added view with a label and a text field as prompt and receiver to the window. After that, I removed them and added another view with a map and two buttons to do the other operations. I changed views on the same window over and over. This is certainly a way of implementing it although it is far from convenient. However, I encountered a little problem with this technique when removing a view from its super view and add another sub view on the base view. The new interface looks like being moved upwards for some pixels so that all the components (button, label) do not stay in the original place. Apparently, this is not an appropriate method to solve the problem.

Afterwards, I found the explanation from “iPhone SDK Release Notes for iPhone OS 2.2” where this is stated as a known issue: If a view is detached from a given UIViewController as a top-level nib object, but connected as an outlet, the view’s origin is incorrectly moved upwards by approximately 24 pixels. Therefore, I was compelled to apply other methods to switch between views and found navigation controller which is quite convenient and easy to switch between views.

The UINavigationController class implements a specialized view controller that manages the navigation of hierarchical content. It can be used to control between different views of the application. This navigation interface makes it possible to present data efficiently and also makes it easier for the user to navigate that content since it gets rid of adding and removing views all the time. The navigation controller dominates several view controllers in my application, and the relationship between them is shown in the follow picture, and the switches between these views are pointed out by red arrows.
4.3 View controllers

In iPhone SDK, the UIViewController class is set to be responsible for controlling associated view, managing the changes of the view and achieving the operations on the view. In MVC model, it is obviously that the controller corresponds with data, view as well as the server side. Therefore, all the significance of this application is tied with the following view controllers.

4.3.1 Initial view controller

This is a window-based application which is controlled by a navigation interface to switch between several views, so the initial view is implemented on the window with a navigation bar, and loaded from the corresponding view of "NicknameViewController". This view controller inherited from UIViewController is responsible for receiving a user's nickname and bring the user to the main view as long as it is valid. The format of the nickname should begin with a letter and only include number and underline but not blank or other special characters as #, €, § etc. Another action in this controller is the short introduction of GeoChat shown on the other side of the window when clicking the information button in the lower right corner of the view. Two delegate protocols, UITextFieldDelegate and FlipsideViewControllerDelegate, are implemented in order to interact with keyboard inputting and flip to the introduction view.
4.3.2 Map view controller

This controller could be considered as the heart of the application because the kernel idea that GeoChat holds is implemented by it. The first element I add to the view is a map, which is defined as an object of MKMapView class. Then I set the center of the map to Stockholm where GeoChat is created and the zoom level to a middle level which looks appropriate for users. The map has the same features as an online map.

The coming task is so significant that we cannot use GeoChat without achieving it precisely, which is how to get immediate chat rooms shown on the map. One viable way is to get the information of chat rooms from the server every time the map is changed, no matter panned or zoomed. By implementing this, five parameters should be sent to the server including the biggest and smallest longitude and latitude of the map currently displayed on the screen as well as the zoom level of it. Once the map is moved, we capture these values and the chat rooms' information will be returned from the server. This is done in the following delegate method of MKMapViewDelegate protocol.

- (void)mapView:(MKMapView *)mapView regionDidChangeAnimated:(BOOL)animated

The name of this method implied that all the manipulations are acted after the region of the map has been changed, namely when any piece of code written in this method will be performed once the map is moved. Thus, we send a message like this to update the existed chat rooms in a certain region of the map shown on the screen, [helper ChatRooms:mapView sessionid: ssid]; where helper is an instance of a class responsible for getting chat room information from the server. We call the method ChatRooms: sessionid: (in Objective-C, the name of a method is rendered like this, including the colons) by passing two arguments the map view and the session id which is generated from the beginning.

To denote the chat room markers on the map, I applied pin annotations by using the class MKAnnotationView of the map kit. It is a key point of implementing GeoChat resulting in that these chat room markers are main pipes for users to interact with the application, i.e. without seeing specific and exact chat rooms on the map, they cannot choose which one to join in.

4.3.3 Chat room view controller

When a user select and join a chat room, the window will switch to it. Thus, users can chat with the others in the chat room. The chat room view controller takes the responsibility of chatting and handles all data updates of chat messages and
participants in a chat room. Participators in a chat room can type with the virtual keyboard in the text area and send the message so that everyone in the same chat room will see what the others said. This is done almost the same as our Java version. There is a little tricky problem is the virtual keyboard will shut out the users to see other dialogs when they are typing. I solve it by moving the displaying message field upwards for some pixels. Since it is scrollable and new message is always appended to the last row, users can view messages while typing. If the keyboard is hidden, the pattern of this view will be back to the original proportion. As for updating data, there are two separate threads monitoring the server and sending updates every single second. Once somebody leaves the chat room, this view controller has to tell the server to remove this user from the participants of this chat room.

4.4 Network communication

GeoChat is an online application that requires network coverage when subscribers tend to use it. Thereby, the communication between client side and server side is indispensable. There are two kinds of situation in this application depending on whether we have to get the returned data for later use. If not, we only need to use NSURLConnection to post data to server. Take an instance, when a user wants to join a chat room, we will just send a request to server with chat room number and the user's session ID, so that the server will deal with the participants in that chat room. The client side does not have to make any response or relevant action after post this message to the server. In this case, I created a class named “NetworkConnection” and called class method

```objc
+(NSURLConnection *)connectionWithRequest:
(NSURLRequest *)request delegate:(id)delegate
```

to create and return an initialized URL connection and begin to load the data for the URL request. The delegate for the connection is the class itself. So, four delegate methods of NSURLConnection were implemented in this class in order to fulfill the basic requirements of posting data to the server.

```objc
-(void)connection:(NSURLConnection *)connection
didReceiveResponse:(NSURLResponse *)response

-(void)connection:(NSURLConnection *)connection
didReceiveData:(NSData *)data

-(void)connection:(NSURLConnection *)connection
didFailWithError:(NSError *)error
```
In fact, the URL request that we send is “http://www.sics.se/~rost/geochat/chat.php?v=5&cmd=join&chatid=CHATROOMNUMBER@&sessionid=SESSIONID”. However, most of the time we do not only post data but also make use of the returned values. For example, when a user input a nickname, a session ID guarantees his/her status is generated. Since we will use it as the nickname no matter joining a chat room or chatting with others or creating new chat rooms, it is necessary to get this value from the returned xml file. Similarly, when we send a request to get the chat rooms in a certain range, the return information is also enclosed in an .xml file which looks like this.

Request

http://www.sics.se/~rost/geochat/chat.php?v=5&cmd=start&nick=Lydia

Return value
<?xml version="1.0" encoding="UTF-8" ?>
<sessionid>n7fheb2lk5pv8d873a32tg92k3</sessionid>

When a user logs in with the nickname “Lydia”, the session id which is enclosed in an .xml file is actually “n7fheb2lk5pv8d873a32tg92k3”.

Request

http://www.sics.se/~rost/geochat/chat.php?v=5&cmd=chats&lon1=0&lat1=0.4&lon2=0.8&lat2=1&minz=4&maxz=10

Return value
<?xml version="1.0" encoding="UTF-8" ?>
<chats>
  <chat>
    <id>5</id>
    <longitude>0.8</longitude>
    <latitude>0.42</latitude>
    <z>8</z>
    <topic>whatwhat</topic>
  </chat>
</chats>
When we want to display chat rooms on the map, we have to send a request to the server with the minimum and maximum value of longitude, latitude (Mercator projection value) and zoom level. Thus, we can get all the chat rooms’ information of chat room id, longitude, latitude, zoom level and topic within this range, which is marked as bold characters in the above xml file. To add annotations on the exact position of the map, we need to parse these values. NSXMLParser, a class provided by Cocoa, can solve this problem conveniently by implementing the following delegate methods.

- (void)parser:(NSXMLParser*)parser didStartElement:(NSString*)elementName namespaceURI:(NSString*)namespaceURI qualifiedName:(NSString*)qName attributes:(NSDictionary *)attributeDict

- (void)parser:(NSXMLParser*)parser didEndElement:(NSString*)elementName namespaceURI:(NSString*)namespaceURI qualifiedName:(NSString*)qName

- (void)parser:(NSXMLParser*)parser foundCharacters:(NSString*)string

Implying from the name of the methods, it is not difficult to understand that the useful value is found by matching the start and end tag enclosed it. If this kind of conformation exists, we save the current element in a string for later use.

4.5 Data Update
We can move the map to refresh the information of chat rooms, however, we have to create a single thread to monitor the server every second in order to get chat logs and participants updated. Therefore, a new thread is created when a user joins in a chat room and terminated when he/she leaves. The RUN method is implemented as follow, where NSThread is a class controls the execution of a thread.

-(void) run:(id)sender
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    @try{
        while(isExit==NO){
            double interval = 1;
            [NSThread sleepForTimeInterval:interval];
            [self performSelectorOnMainThread:@selector(chatsUpdate) withObject:nil waitUntilDone:NO];
        }
        isExit = NO;
    }@catch(NSException *e){
    }@finally{
        [pool release];
        [NSThread exit];
    }
}


V Testing

5.1 Memory management

In Cocoa programs, memory allocation is strictly controlled. The fundamental rule of memory management is:

You take ownership of an object if you create it using a method whose name begins with “alloc” or “new” or contains “copy” (for example, alloc, newObject, or mutableCopy), or if you send it a retain message. You are responsible for relinquishing ownership of objects you own using release or autorelease. Any other time you receive an object, you must not release it [4].

The reason that we need to follow the rules rigorously is because objects are constantly creating and disposing of other objects in an Objective-C program. Usually, an object creates things for private use and can dispose of them as needed so as to free the memory for other use. However, the ownership becomes not quite clear when an object sends a message to another object and passes a value. Under this situation, memory leak takes place unless we manage every object demonstrably. Because there is not any garbage collection in the iPhone OS, memory management should be done manually, and you cannot simply call “dealloc” to free the resources and references of a certain object because this object may contain references to other objects that will not be real located. This kind of situation will cause memory leaks, so we use retain which will increment the value of a certain variable for the instantiated object, and release which decrements the value, and the referenced object is deallocated when it reaches zero. In other words, an object should be released after usage once it is set to retain. Otherwise, a potential memory problem may happen sometimes. Take an instance like this:

File: example.h

@interface example: NSObject
{
    NSString anObject;
}

@property (nonatomic, retain) NSString anObject;
@end

File: example.m

@implementation example
@synthesize anObject;
-(void) initialize
{
    anObject = [[NSString alloc] initWithString: @"Hello"];
    [self test];
}

-(void) test
{
    anObject = [[NSString alloc] initWithString:
         @"Not released the first object yet!"];
}

-(void) dealloc
{
    [anObject release];
    [super dealloc];
}
@end

In this example, object “anObject” is allocated at first and released in the end. It seems there should no memory leak exist. Whereas, when searching the code for all references to anObject, we have allocated a new string and assigned the pointer to anObject in method “test”. The problem is that we never released anObject before it pointed to something else. Therefore, the first allocation is still there and we cannot get the memory back. The release call in dealloc method is the release of the second string actually since that is where the pointer is pointing to. Although it may not affect the program to work correctly, the potential problem is still there. By solving this memory leak, we should release the first string before it is allocated again in the test method.

-(void) test
{
    [anObject release];
    anObject = [[NSString alloc] initWithString:
         @"Not released the first object yet!"];
}

In this application, I will use a tool which is involved in iPhone SDK named “Instrument” to detect memory leak. However, it is impossible to avoid all potential memory leaks in a complicated program since some of them may not influence the
execution but still exists. Therefore, we can only try to reduce this problem as much as possible. The testing result will be addressed in section 5.3.

5.2 Deployment of GeoChat on iPhone

For this GeoChat version, we have not decided to distribute it on the app store, but just install it on an iPod Touch to test the functionalities currently. The process of deploying an iPhone application on a real device is a bit complicated. The steps are listed here.

1. Apply for an iPhone application developer ID.
2. Generate a certificate signing request, and wait for the approval from team agent or admin (if you apply an individual ID, you can manage it by yourself). After acceptance, download the certificate and install it on the local machine.
3. Submit the device’s UDID to the team admin and he/she will register your device in the team with a name and the unique UDID.
4. Team admin should generate an app ID for your application and create a development provisioning profile.
5. Download the development provisioning profile. As long as your Apple device ID and iPhone development certificate are included in this profile, you are able to test the application on a real device.

Since Apple protects its copyright strongly, I will only describe the basic procedure but not put any screenshots here.

5.3 Result and rectification

I used black box testing to test the functionalities of GeoChat on iPhone. The devices that I used to test the communication between iPhone version and Java version were an iPod Touch and a Sony-Ericsson smart phone. For possible memory leaks, a testing tool called Instruments integrated in Xcode environment was employed, which can track the data transfer and easily pinpoint the problem area. The major functions of GeoChat were fulfilled and worked correctly. However, there were still a few problems found during my test, and I rectified them with proper approaches.

Annotation problem:
When users touch a pin annotation locates on the edge of the screen, the application will exit unexpectedly. The pin will be relocated to an inner position on the screen so that the popup bubble is able to be displayed. Thus this will cause the move of the map. However, the application will update the chat room annotations once the map is moved by removing all the chat rooms and repainting the ones should be appear in
the new map range. Consequently, the pin on the edge is removed, and the program exits unexpectedly. I solve this problem by adding a message in the delegate method:

-(void)mapView:(MKMapView*)mapView regionWillChangeAnimated:(BOOL)animated

Thence, the deleted pin annotation will show up again when the map is moved.

Memory leak detected by Instruments:
Here is an example of the object allocation and release process of the project. Some of the memory is allocated by system, while some is by me when defining objects. The size of the memory should be 0 after we run the application. However, we can find from the picture that there is memory of “NSXMLParser” which is only allocated by not released. This is caused because of my carelessness although it may not lead to an error. Nevertheless, the potential hazard of this application still exists. I rectified this kind of mistake by checking all the objects’ memory allocation and releasing them after usage.

[figure 12: memory leak detected by Instruments]

After the rectification, GeoChat on iPhone can be successfully used and communicate with that installed on Java-supported mobile phones.
VI Comparison

After finishing the development of GeoChat on iPhone, I compared the difference of Java ME and iPhone SDK development process from various aspects. The significance of this comparison is to comprehend Java ME and iPhone SDK development more thoroughly and find out the advantages and disadvantages of using the two ways to develop software and even think and solve problems from different angles. Primarily, it is admitted that there are similarities of the two versions as they are the same application and realize the same functions basically. Whereas, the disparities are obviously as well because of the different software development platforms we use. I will compare them particularly on GeoChat, and generalize some similarities and differences of the development process thereafter.

6.1 Comparison on GeoChat

6.1.1 Similarity

i. Requirements and goals

Because GeoChat is an application with fixed functionalities that should be realized, the requirement will not vary no matter in which way the application is implemented. Although we can achieve every target with different approaches, the goals are never changed.

ii. System architecture

The system architecture of the two versions are both in client-server mode, where client represents a user of GeoChat who sends requests to the server via network, and server is a high performance host which runs some programs of managing data and sending response to clients.

3. Shared database

As mentioned previously, GeoChat implemented by Java has already been developed before I get down porting it to iPhone, the database built for Java version can also be employed by iPhone. Thus, there is no modification in this shared database. However, this situation may not adaptable for all software owning to different aims of an application.
6.1.2 Disparity

i. Design pattern

A design pattern is a template for solving general and recurring problems in a particular context. By using certain design patterns in your application, it can help you increase the efficiency of figuring out a problem and make things easier to be rendered. In an application, maybe there is not only one design pattern is used since some patterns are naturally fit together. We can apply different design patterns in the application development to solve different kind of problem. In GeoChat on iPhone, I used MVC design pattern and delegation design pattern for iPhone application development particularly. Both Java and Cocoa have dozens of design patterns varied according to different effect. As long as they are utilized correctly in our development, they will play important roles in solving problems and building architectures.

ii. Interaction

The screen touch and virtual keyboard make the interaction of input and navigation varied from iPhone to Java-supported devices. For GeoChat on iPhone, all interactions between users and the application are done by touching directly on the screen. Interaction components such as button, label and text field are provided by the Cocoa API, when inputting some text in a text field, a virtual keyboard pops up immediately after we touch the text field. To achieve this, we only need to conform to some protocols in corresponding classes, like a class with a text field defined in it should conform to the UITextFieldDelegate protocol. However, when developing with Java ME, the text field and input method all need to be implemented programmatically, which takes more effort in coding, and the non-full keyboard makes the selection of word even harder to be fulfilled.

![Virtual Keyboard](image)
In the picture above, when users touch the text field where a blue cursor stays, a virtual keyboard pops up from the bottom of the screen. Furthermore, the style of the keyboard can be selected due to various functions. For instance, a keyboard of “Return” type does not do anything after the input, but a “Done” keyboard will return the current view that the text field in; There are also other types of keyboard as “Go” and “Search”, which are particularly applied when typing a website or searching some materials. Whereas, the interaction interface of Java version is defined by coding.

iii. Map layer

The map is implemented in different way since iPhone SDK has a map kit which is able to be deployed for creating a Google map in the application directly, and the zooming and panning functionalities are integrated already. Nevertheless, the map created by Java, composed by dozens of small tiles, has to be downloaded and repainted on the screen every time it is moved or zoomed. The speed of navigating on a map refreshed tile by tile is rather slow than on an intact map.

iv. Control of different views

The class UINavigationController in iPhone SDK is perfectly responsible for switching between views, thus we can implement each view respectively and use navigation controller to control them. In this way, a single view is neater and contains less information in an iPhone application. While, to implement the application without view switches also has advantages since only one view need to be designed, and we only need to add and remove components from that view. This is the way that Java version applied.

v. Interface design

Interface Builder is an intuitional tool for developers to build interfaces with iPhone SDK, but in java version, the interface are implemented manually by coding and took much more work to accomplish. Compared with GeoChat on iPhone, the complicated and nice-looking interface is onerous to be implemented by Java. As shown in the following pictures, the send and back button in GeoChat on iPhone is more intuitive than the prompt in Java version.
vi. Chat room marker

The chat room markers, which users can choose by touching the pin annotations on the map in iPhone version's GeoChat, is totally different from that of Java version. The latter was denoted by a square varied in different map zoom level which is not as distinct as the former. But the zoom level of a chat room is more obvious because the size of the square changes according to the map's zoom level, while a pin cannot resize when zoom level differs.
The two pictures reveal two methods of denoting chat room markers. In Java GeoChat, squares represent chat rooms which can be displayed in the range of \([z-3, z+3]\) (where \(z\) is the current zoom level of the map). Therefore, a square should have seven sizes that magnifies from a dot to almost a full screen size. We navigate on the map by controlling a cross cursor. When the cross is in the overlap of some squares, we are able to use key (4) and (6) on the keyboard to choose the chat room that we want to join in. Nevertheless, the application implemented by Objective-C on iPhone does not use squares to represent chat room markers but pin annotations. Since the pins are not resizable, I use different color to distinguish chat rooms created in different zoom levels. But there are only three colors available of a pin's property, and this is a weakness to specify chat rooms located in different zoom levels on the map compared with Java version.

vii. Implementation sequence

The sequence of implementing GeoChat by Java and Objective-C can be different owning to disparate orientations. My application is implemented according to views, but the Java version is implemented due to functionalities.

viii. Network communication

After sending a request URL to the server, we get the return value in xml format. We will parse xml file in order to receive useful information in it. This is easier to be completed by Objective-C as long as we implement three methods that are mentioned in section 4.4, we can get the elements from an xml file. Although it is a bit complicated to achieve this by Java, the steps are also clear and not difficult to grasp.

• Load the xml file into an input stream.
• Create an instance of XMLParser
• Parse the xml into “Document” type
• Get the root element then put the elements of this root one by one into a list

ix. Memory management

Since Java has a garbage collection mechanism but iPhone OS does not, it is more significant for iPhone developers to care about the memory management and manually release objects in case leak problems happen.

x. Installation of the application on real device
To install an application on iPhone, we have to register for an iPhone development program, which means it is charged to add an application to a device. Before getting the permission to install an application, we need to apply for a certificate, add device, generate App ID and create provisioning file. And all of these only can be done by the administrator. So if you are a team member as me, it is even harder to go through everything quickly. Nevertheless, an application developed by Java ME is free for the developer to install it on his/her devices without so many troublesome procedures.

6.2 General comparison between Java ME and iPhone SDK

We can generalize some notions for Java and iPhone application development after the concrete comparison is conducted. For the development process of iPhone application, the software structure is more fixed than developed by Java, since iPhone SDK contains many frameworks which are particularly provided for iPhone application development. But Java ME can be utilized to develop applications for any java-supported device in more flexible ways.

iPhone SDK is more convenient to use compared with Java, for the reason that a lot of APIs are packaged and can be employed directly. While developing by Java, it may take more effort to realize a goal although the approaches are often optional.

Visual interfaces are simpler to be developed by iPhone SDK than Java ME because of the professional tool Interface Builder it provides.
VII Conclusion and future work

GeoChat on iPhone worked well on real devices. The process of creating it did not go smoothly as expected, but I gained treasurable experience from solving puzzles. I obtained most of the relevant knowledge from the Internet especially the official webpage of Apple’s iPhone app development. While implementing, I became more and more familiar with Xcode IDE and Objective-C programming. Although Java version existed, I still tried to design the application from scratch to make it in Cocoa style. Besides all the features in the previous Java version, GeoChat on iPhone has an extra function, “Locate Me”, which is to locate the user’s current position.

From my perspective, despite the successful real device installation of GeoChat on iPhone, it is still an early prototype which needs to be optimized for being a real product. Therefore, I listed some points that can be worked on in the future.
1. If Mobile Life would like to distribute GeoChat on Apple’s App Store, a performance test needs to be accomplished, e.g. using automatic testing tools. And also, a load test of the server is also necessary to assure the quality of service.
2. The chatting function could be improved. In another word, users can chat with images, such as a cartoon expression, as well as textual characters.
3. The interface can be redesigned by interface designers.
4. Currently, users can only create one chat room at a precise geographical location. Take an instance, if there is a chat room at (58.430967, 18.229018), the first represents latitude and second for longitude, users are not allowed to create another chat room at this location. Although this may not happen in practice, we still would like to allow users to create more than one chat room on a precise geographical location in future versions.
5. If many people show interests in this application, we will consider creating a registration system to provide extra functions for members.
References


