

Shanghai Jiao Tong University
Wireless Communications Principles and Applications

Find your friend - An Android application

Presented to
Dr.Xin Bin Wang

Andreas Ziegler 7090309048
Yousef Boarik 5090309553

SHANGHAI JIAO TONG UNIVERSITY
The Faculty of Engineering
Department of Electronic Engineering

2012.05.24

Contents

1 Abstract	2
2 Directories	3
2.1 Figures	3
2.2 Tables	3
2.3 Definitions	3
3 Introduction	4
4 Basics	4
4.1 Used software	4
4.2 Used hardware	4
4.3 Theoretical basics	5
4.3.1 Google Maps	5
4.3.2 Network sockets	7
5 Planing	10
5.1 Summarization	10
5.2 Division of the work	10
6 Implementation	11
6.1 User interface	11
6.1.1 Client	11
6.1.2 Server	12
6.2 Main application (onCreate())	13
6.2.1 Server site	13
6.2.2 Client site	14
6.3 getData()	15
6.3.1 Server site	15
6.3.2 Client site	16
6.4 connect() (only on the client site)	17
6.5 Google Maps	17
6.5.1 Permission to access to the service	17
6.5.2 FindYourFriend Activity class and displaying the zoom control	18
6.5.3 GPS Location Mapping	21
7 Tests	22
8 Conclusion	22

Abstract

- Job definition** In the course “Wireless Communication Principles and Application” one part of the course was a project. There were different options for the projects: Writing a theoretical paper, develop something with the ns-2 network simulator or writing an application for a smart phone platform. We choose the last one because we want to collect some experiences about developing software for smart phones. Because we heard and read a lot about Android, we decided to develop an application for Android smart phones and see if it is really so easy.
- Project definition** After discussing a few ideas, we decided, that we will develop an application, which helps us to find each other on an area like a university campus. Our application has to help us to find our friend with the help of Google Maps and a network connection.
- Procedure** During our project we went through different project states.
- First, we had to choose an application to develop: After discussing a few ideas, we decided, that we develop an application which help us to find a friend on the university campus.
 - Because programming in Java and developing for Android was new for both of us, we first learned the basics about programming with Android.
 - After we had the basic knowledge about programming with Android, we divided our application in two parts, that both of us can work as independent as possible.
 - We combined both parts of the application after both parts were finished.
 - After the programming part was finished, we created the presentation to present our project in the lesson.
- During the whole project work, we always updated our documentation.

Directories

2.1 Figures

List of Figures

Fig. 1: Google maps.	5
Fig. 2: Google API key.	5
Fig. 3: Server and client Socket.	8
Fig. 4: User interface	11

2.2 Tables

List of Tables

Tab. 1: Abbreviation	3
Tab. 2: Division of the work	10

2.3 Definitions

Android	Google smart phone platform and operation system
IDE	Integrated Development Environment
Eclipse	An IDE for developing Android applications
Android SDK	Development platform and tools for programming with Android
Emulator	An Android smart phone emulator provided by the Android SDK
TCP	Transport layer protocol
API	Application Programming Interface

Table 1: Abbreviation

Introduction

Our Android application should help people to find each other in an area like an university campus. The application uses Google Maps to show the location of the user and also the location of the friend. For sending and receiving the locations, server and client sockets are used, so that the application can use a reliable TCP connection to communicate.

Basics

4.1 Used software

Eclipse	Eclipse is an IDE which can be used for developing software in a lot of different programming languages, for example Java for Android. We also used Eclipse because Eclipse is a platform independent IDE so we could use it on Windows 7, Mac OS X and also on Arch Linux.
Android SDK	We used Eclipse in combination with the Android SDK, which makes it easy to develop, simulate and debug software for Android smart phones. The Android SDK also includes an Android Emulator, which makes it possible to develop Android application without having a real Android device.
L^AT_EX	We used different editors to write the documentation of our project in L ^A T _E X.

4.2 Used hardware

The only Android smart phone we had, was a HTC Desire which belongs to Andreas Ziegler.

Development environment: We connected Eclipse with the HTC Desire smart phone and also with the Android emulator. In this way, we were able to develop, test and debug the application. For the connection of the both “devices”, we used a wireless lan network.

4.3 Theoretical basics

4.3.1 Google Maps

Basics: Google Maps is one of the many applications bundled with the Android platform. In addition to simply using the Maps application, we could also embed it into our own applications and make it do some very cool things, such as

- Change the views of Google Maps.
- Obtain the latitude and longitude of locations in Google Maps.
- Add markers to Google Maps.
- Add zoom control.



Figure 1: Google maps.

How to use: Before we could integrate Google Maps into our android application. We needed to apply for a free Google Maps API key. We first created a certificate and get its MD5 fingerprint, then we registered with the MD5 on the website code.google.com to get the Google key.

Here we faced one problem that we couldn't find the MD5 for JDK 1.7, all we got is SHA1 fingerprint which doesn't work with Google key. So we solved this problem by using JDK 1.6.

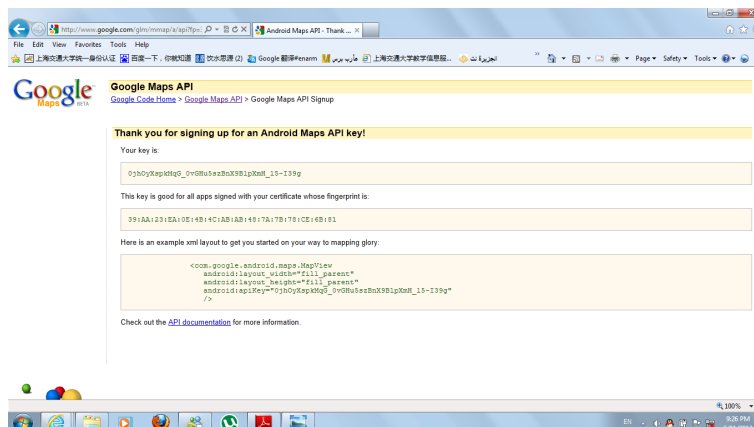


Figure 2: Google API key.

In order to display Google Maps in our application, we first needed to have the INTERNET permission and other certain permissions in our manifest file to use location information. We then added the "com.google.android.maps" element to embed the map within the activity.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
//-----
<uses-library android:name="com.google.android.maps"/>
//-----
```

In our program for Google Maps we used two big package which contain several important classes for our application:

- Android Location package
- Com.google.android.maps

Android Location package classes :

Following are some important classes present under the android location package.

LocationManager: The class provides access to the location service. It also provides facility to get the best Location Provider as per the criteria.

LocationProvider: It's an abstract superclass for location providers. A location provider provides periodic reports on the geographical location of the device.

LocationListener: Provides callback methods which are called when location gets changed. The listener object has to be registered with the location manager.

Criteria: The class provides the application to choose suitable Location Provider by providing access to set of required properties of the LocationProvider.

Com.google.android.maps classes:

This package contains classes related to rendering, controlling and overlaying information on the Google Maps on the android devices.

MapActivity: It is the spacing activity defined to show the Google Maps. The MapActivity takes care of the low-level networking.

MapView: MapView is the view that supports and displays the map. This must be contained by a MapActivity.

MapController: MapController is the object used to move the map around the screen.

In order to display Google Map view, we need to update our Activity class (FindYourFriendClientActivity). This class should extend MapActivity class in place of Activity class. We also need to import com.google.android.maps.MapActivity package to support MapActivity class. We also need to override isRouteDisplayed method of MapActivity class. This method is used for Google's accounting purposes, and it should return true if the application is displaying routing information on the map. For most simple cases, return false.

```
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;

public class FindYourFriendClientActivity extends MapActivity {
    .....
    .....
    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
}
```

Add Zoom (in/out) Controls

MapView class has an in-built method `setBuiltInZoomControls`. Invoking this method with `true/false` enable/disable Zoom in/out controls. In order to invoke this method, we need to find the instance of MapView class by calling `findViewById` with id of MapView control. `id="@+id/mapView"` from the `main.xml`. One important thing to note here is that the zoom controls will become enable once we will touch/click the map view.

We can select, if we want to show Satellite, Traffic or Street view in maps. This is simply achieved by calling `setSatellite`, `setStreetView` and `setTraffic` methods.

```
// Displaying Zooming controls
mapView = (MapView) findViewById(R.id.mapView);
mapView.setBuiltInZoomControls(true);
mapView.setTraffic(true);
```

Add GPS Location Mapping

Android provides location based services through `LocationManager` (package `android.location`) class. This class provides periodical updates about the location of the device. In order to use `LocationManager`, we need to get a reference of `LocationManager` class by calling `getSystemService` method. Later on, we need to register for location updates by calling `requestLocationUpdates` method. We need to create a class implementing abstract `LocationListener` class. This class will be registered with Location Manager to receive updates of location. We need to override all four methods of this class, namely `onLocationChanged`, `onProviderDisabled/Enabled`, and `onStatusChanged`. As we are just interested in getting location updates, we will modify the code of `onLocationChanged` to navigate to the new location received in the map view. This is achieved by calling `animateTo` method of `MapController`.

```
private LocationManager lm;

lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
LocationListener locationListener = new MyLocationListener();
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);
}
private class MyLocationListener implements LocationListener
{
    @Override
    public void onLocationChanged(Location loc) { }

    @Override
    public void onProviderDisabled(String provider) { }
    @Override
    public void onProviderEnabled(String provider) { }
    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) { }
```

Add a Location Marker

A map overlay can be added showing the user's current location. To add an overlay, define a class that will extend `Overlay` class. The overlay display a text "Here I am.." and "my friend is here..." on the map.

```
canvas.drawText("I am here...", myScreenCoords.x, myScreenCoords.y, paint);
canvas.drawText("My friend is there...", friendScreenCoords.x, friendScreenCoords.y, paint);
```

4.3.2 Network sockets

Basics: To send and receive data from application to application over a network connection, sockets must be used. The setup of a server and a client socket is visible in figure 3. On the server site, after a socket is created, it must be binded to a port (Portnumber). When the server socket has its portnumber, it listen to incoming connections, which it will accept. When the server socket is connected, it is ready to send and receive data. On

the client site, it is a bit easier. When the client socket is created, it is ready to connect to a server. When the client is connected, it is also ready to send and receive data.

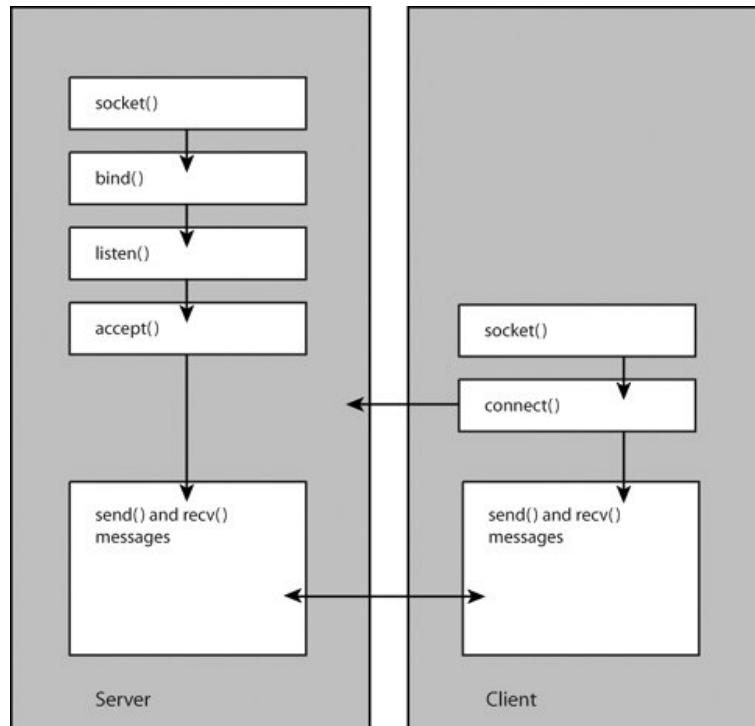


Figure 3: Server and client Socket.

How to use: First we will look at the implementation of the client. After we will also cover the implementation of the server site.

First, we have to create a socket object.

```
Socket client;
```

After the socket object is created, we can try to connect the client to a server.

```
client = new Socket("www.example.com", 4321);
```

With the now created and connected socket object, we can create an output “channel”. With the following command, we connect the object *out* to the output stream of the socket object (*client*).

```
out = new PrintWriter(client.getOutputStream(), true);
```

For the input we can do the same. With the following command, we connect the object *in* to the input stream of the socket object (*client*).

```
in = new BufferedReader(new InputStreamReader(client.getInputStream()));
```

Together with the exception handling, the code for the client looks like this:

```
Socket client;
```

```
try{
    client = new Socket("www.example.com", 4321);
    out = new PrintWriter(client.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(client.getInputStream()));
} catch(UnknownHostException e) {
    System.out.println("Unknown host: www.example.com");
}
```

```
} catch (IOException e) {  
    System.out.println("No I/O");  
}
```

On the server site, we first create a server socket.

```
ServerSocket server;
```

After the server socket is created, we bind the socket to a portnumber.

```
server = new ServerSocket(4321);
```

Now we also creat on the server site a client socket object.

```
Socket client;
```

With the *accept()*-command, we let the client connect to the server.

```
client = server.accept();
```

After the server accepts the connection of the client, we can also creat an input and output “channel”.

```
in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
out = new PrintWriter(client.getOutputStream(), true);
```

Together with the exception handling, the code for the server looks like this:

```
ServerSocket server;
```

```
try{  
    server = new ServerSocket(4321);  
} catch (IOException e) {  
    System.out.println("Could not listen on port 4321");  
}
```

```
Socket client;
```

```
try{  
    client = server.accept();  
} catch (IOException e) {  
    System.out.println("Accept failed: 4321");  
}
```

```
try{  
    in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
    out = new PrintWriter(client.getOutputStream(), true);  
} catch (IOException e) {  
    System.out.println("Read failed");  
}
```

Planing

5.1 Summarization

At the beginning of the project, both of us, Yousef Boraik and Andreas Ziegler, studied the basics about Android programming. With the basic knowledge, every one of us, focused on his part. At the end, we combined both parts and optimized some details for the final application.

5.2 Division of the work

That we were as independent as possible from each other, we divided the project work.

Andreas Ziegler	Network communication
Yousef Boraik	Google Maps

Table 2: Division of the work

Implementation

6.1 User interface

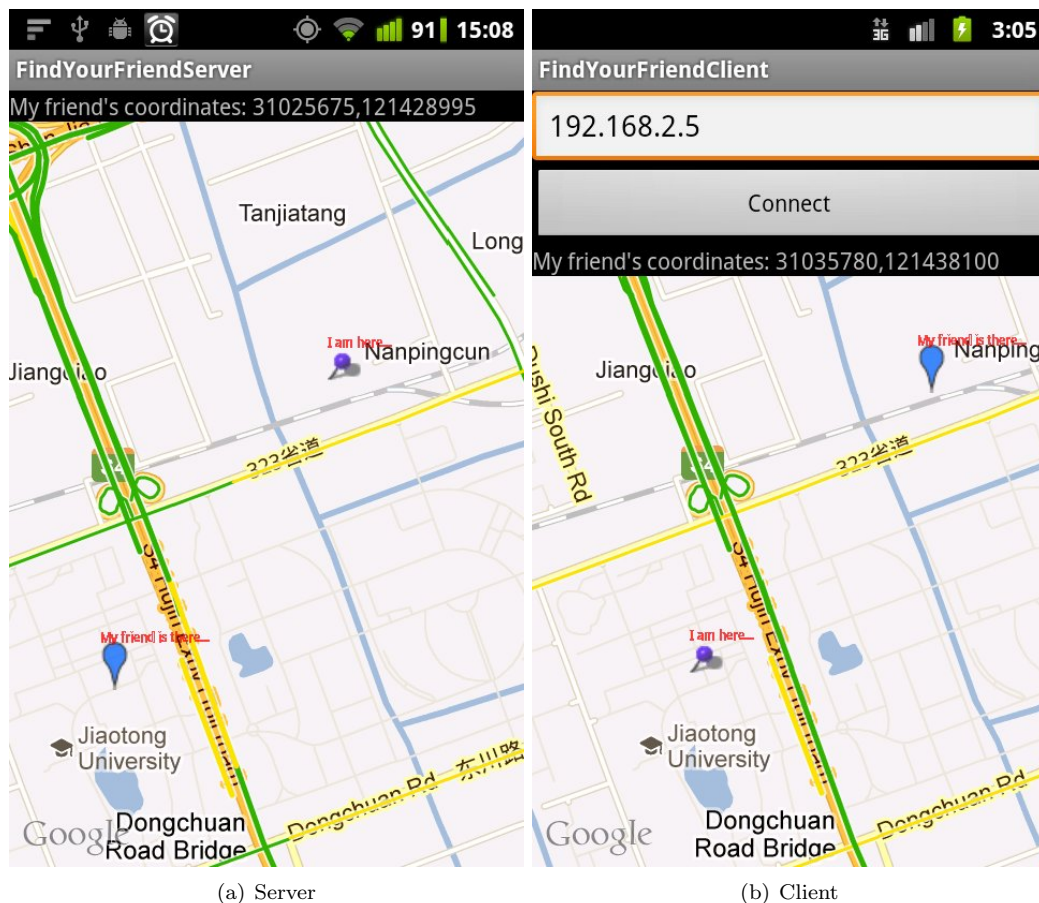


Figure 2: User interface

The user interface of our application is quite basic and includes a text field and Google Maps. On the client site, there are also one input field and a button, which is used to define the address of the server and to connect to the server. The user interface is defined in `main.xml`.

6.1.1 Client

One important point is the `android:onClick="connect"` tag, which defines the name of the method, which will be executed, when the button is pressed.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <EditText
```

```
        android:id="@+id/serverIP"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/IP" />

<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/connect"
    android:onClick="connect" />

<TextView
    android:id="@+id/text"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<com.google.android.maps.MapView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mapView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="0NhhMUyOb6b7IqTAwdQgzsgNXGv8hOIKeL_NUkQ" />

</LinearLayout>
```

6.1.2 Server

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <com.google.android.maps.MapView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/mapView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey="0NhhMUyOb6b7IqTAwdQgzsgNXGv8hOIKeL_NUkQ" />

</LinearLayout>
```

6.2 Main application (onCreate())

6.2.1 Server site

Next to the default procedure, which creates the layout, the text input and output object were defined and the network communication get started from here. The interesting and important thing is, that we use a separate thread for the part of the network connection. The connection process could take some time and when it would be in the main thread, it would block the user interface. To prevent this, the network communication part runs in its own thread and so it don't bother any other parts of the application. In Android it is just possible to show or update the user interface from the main thread. Because our network communication runs in an other thread, we need the *post()* method of the user interface object, to update it.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Displaying Zooming controls
    mapView = (MapView) findViewById(R.id.mapView);
    mapView.setBuiltInZoomControls(true);
    mapView.setTraffic(true);

    mText = (TextView) this.findViewById(R.id.text);

    mc = mapView.getController();
    String coordinates[] = {"31.035780", "121.438100"};
    double lat = Double.parseDouble(coordinates[0]);
    double lng = Double.parseDouble(coordinates[1]);
    p = new GeoPoint((int) (lat * 1E6), (int) (lng * 1E6));

    String friendCoordinates[] = {"31.025675", "121.428995"};
    double friendLat = Double.parseDouble(friendCoordinates[0]);
    double friendLng = Double.parseDouble(friendCoordinates[1]);
    pFriend = new GeoPoint((int) (friendLat * 1E6), (int) (friendLng * 1E6));

    mc.animateTo(p);
    mc.setZoom(16);
    //---Add a location marker---
    MapOverlay mapOverlay = new MapOverlay();
    List<Overlay> listOfOverlays = mapView.getOverlays();
    listOfOverlays.clear();
    listOfOverlays.add(mapOverlay);
    mapView.invalidate();

    lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    LocationListener locationListener = new MyLocationListener();
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);

    new Thread(new Runnable() {
        public void run() {

            while(true) {
                String str = getData();
```

```
String[] separated = str.split(",");
int latitudeE6 = Integer.parseInt(separated[0]);
int longitudeE6 = Integer.parseInt(separated[1]);
pFriend = new GeoPoint(latitudeE6, longitudeE6);

mText.post(new Runnable() {
    public void run() {
        mText.setText("My friend's coordinates: " + pFriend.toString());
    }
});

mapView.post(new Runnable() {
    public void run() {
        mapView.postInvalidate();
    }
});
}
}).start();
}
```

6.2.2 Client site

On the client site, we just have the default procedure and create the input and output text fields. Compare to the server site, we have one text input more (*inputServerIP*), which we need, that the user can input the address of the server. On the client site, the network connection get started by the method *connect*, which will be executed by pressing the button and so this code is not in the *onCreate()*-method.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Displaying Zooming controls
    mapView = (MapView) findViewById(R.id.mapView);
    mapView.setBuiltInZoomControls(true);
    mapView.setTraffic(true);

    inputServerIP = (EditText) findViewById(R.id.serverIP);
    mText = (TextView) findViewById(R.id.text);

    mc = mapView.getController();
    String coordinates[] = {"31.025675", "121.428995"};
    double lat = Double.parseDouble(coordinates[0]);
    double lng = Double.parseDouble(coordinates[1]);
    p = new GeoPoint((int) (lat * 1E6), (int) (lng * 1E6));

    String friendCoordinates[] = {"31.035780", "121.438100"};
    double friendLat = Double.parseDouble(friendCoordinates[0]);
    double friendLng = Double.parseDouble(friendCoordinates[1]);
    pFriend = new GeoPoint((int) (friendLat * 1E6), (int) (friendLng * 1E6));

    mc.animateTo(p);
    mc.setZoom(16);
    //---Add a location marker---
    MapOverlay mapOverlay = new MapOverlay();
    List<Overlay> listOfOverlays = mapView.getOverlays();
```

```
listOfOverlays.clear();
listOfOverlays.add(mapOverlay);
mapView.invalidate();

lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
LocationListener locationListener = new MyLocationListener();
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);
}
```

6.3 getData()

6.3.1 Server site

The *getData()*-method is responsible for the whole network connection, which includes setting up the connection and sending and receiving data. The code is based on the code, which is explained in the theoretical part of the documentation. The difference here are, that we use a flag (*flagConnectServer*), that we just establish a connection if there is not already an active connection. The application also just transmit and receive data if there is a successfully established connection (*flagConnectClient*).

```
public String getData() {

    String result = "";

    if (!flagConnectServer) {
        try {
            server = new ServerSocket(4321);
            flagConnectServer = true;
        } catch (IOException e) {
            System.out.println("Could not listen on port 4321");
            flagConnectServer = false;
        }
    }

    if (!flagConnectClient) {
        try {
            client = server.accept();
            //read from client through inputstream
            dataInputStream = new DataInputStream(client.getInputStream());
            //write to stream through outputstream
            dataOutputStream = new DataOutputStream(client.getOutputStream());
            flagConnectClient = true;
        } catch (IOException e) {
            System.out.println("Accept failed: 4321");
            flagConnectClient = false;
        }
    }

    if (flagConnectClient) {
        try {
            dataOutputStream.writeUTF(p.toString());
            result = dataInputStream.readUTF();
        } catch (IOException e) {
```



```
        System.out.println("No I/O");
    }
    try {
        Thread.sleep(500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
return(result);
}
```

6.3.2 Client site

Also the client site is similar to the code in the theory part of the documentation. The difference is also here, that we use a flag (*flagConnect*), to prevent connecting a second time.

```
public String getData(String serverIP) {

    String result = "";

    if (!flagConnect) {
        try {
            serverIP = inputServerIP.getText().toString();
            client = new Socket(serverIP, 4321);

            dataOutputStream = new DataOutputStream(
                client.getOutputStream());
            dataInputStream = new DataInputStream(client.getInputStream());
            flagConnect = true;

        } catch (UnknownHostException e) {
            System.out.println("Unknown host: www.example.com");
            result = "Unknown host: www.example.com";
            flagConnect = false;

        } catch (IOException e) {
            System.out.println("No I/O");
            result = "No I/O";
            flagConnect = false;
        }
    }

    if (flagConnect) {
        try {
            dataOutputStream.writeUTF(p.toString());
            dataOutputStream.flush();
            if (dataInputStream.available() >= 0) {
                result = dataInputStream.readUTF();
            }

        } catch (IOException e) {
            System.out.println("No I/O Send/Receive");
            result = "No I/O";
        }
        try {
            Thread.sleep(500);
        }
    }
}
```

```
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}  
return (result);  
}
```

6.4 connect() (only on the client site)

The method *connect()* get executed, when the connect button on the user interface is pressed, because we defined it in the main.xml. The code is the same, like in the *onCreate()*-methode on the server site.

```
public void connect(View view) {  
  
    new Thread(new Runnable() {  
        public void run() {  
            String serverIP = inputServerIP.getText().toString();  
  
            while (true) {  
                String str = getData(serverIP);  
                String[] separated = str.split(",");  
                int latitudeE6 = Integer.parseInt(separated[0]);  
                int longitudeE6 = Integer.parseInt(separated[1]);  
                pFriend = new GeoPoint(latitudeE6, longitudeE6);  
  
                mText.post(new Runnable() {  
                    public void run() {  
                        mText.setText("My friend's coordinates: " + pFriend.toString());  
                    }  
                });  
  
                mapView.post(new Runnable() {  
                    public void run() {  
                        mapView.postInvalidate();  
                    }  
                });  
            }  
        }  
    }).start();  
}
```

6.5 Google Maps

6.5.1 Permission to access to the service

As explained before we need to add some permissions to our manifest file to use location information.

```
<?xml version="1.0" encoding="UTF-8"?>  
-<manifest android:versionName="1.0" android:versionCode=  
"1" package="sjtu.wcap.findyourfriendclient" xmlns:android=  
"http://schemas.android.com/apk/res/android"> ]
```

```
<uses-sdk android:minSdkVersion="10"/>
<!-- Add permissions -->
<uses-permission android:name=
"android.permission.INTERNET"/> <uses-permission android:name=
"android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
-<application android:label="@string/app_name" android:icon=
"@drawable/ic_launcher">
<!-- Add Google Map Library -->
<uses-library android:name="com.google.android.maps"/>
-<activity android:name=".FindYourFriendClientActivity" android:label=
"@string/app_name"> -<intent-filter> <action android:name=
"android.intent.action.MAIN"/> <category android:name=
"android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
</manifest>
```

6.5.2 FindYourFriend Activity class and displaying the zoom control

The code below implement FindYourFriendActivity class which is the main class for displaying GoogleMaps also implement the MapOverlay class which included the zoom control and the location marker,also it included the GeoPoint object which represent a geographical location and then the code translate the Geopoint to screen pixels.

```
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Point;
public class FindYourFriendClientActivity extends MapActivity {

private static MapView mapView;
private static MapController mc;
private static GeoPoint p;
private static GeoPoint pFriend;

private LocationManager lm;
private TextView mText;
private EditText inputServerIP;
private static Socket client = null;
private static DataOutputStream dataOutputStream = null;
private static DataInputStream dataInputStream = null;
private static boolean flagConnect = false;

class MapOverlay extends com.google.android.maps.Overlay
{
/*@Override
```

```

public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when) {
    super.draw(canvas, mapView, shadow);
    //---translate the GeoPoint to screen pixels---
    Point screenPts = new Point();
    mapView.getProjection().toPixels(p, screenPts);
    //---add the marker---
    Bitmap bmp = BitmapFactory.decodeResource(getResources(), R.drawable.pushpin);
    canvas.drawBitmap(bmp, screenPts.x, screenPts.y-50, null);
    return true;
}*/

public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when) {
    Paint paint = new Paint();

    super.draw(canvas, mapView, shadow);
    // Converts lat/lng-Point to OUR coordinates on the screen.
    Point myScreenCoords = new Point();
    mapView.getProjection().toPixels(p, myScreenCoords);

    paint.setStrokeWidth(1);
    paint.setARGB(255, 255, 255, 255);
    paint.setStyle(Paint.Style.STROKE);

    Bitmap bmp = BitmapFactory.decodeResource(getResources(), R.drawable.pushpin);

    canvas.drawBitmap(bmp, myScreenCoords.x, myScreenCoords.y, paint);
    canvas.drawText("I am here...", myScreenCoords.x, myScreenCoords.y, paint);
    Point friendScreenCoords = new Point();
    mapView.getProjection().toPixels(pFriend, friendScreenCoords);

    paint.setStrokeWidth(1);
    paint.setARGB(255, 255, 255, 255);
    paint.setStyle(Paint.Style.STROKE);

    Bitmap bmpFriend = BitmapFactory.decodeResource(getResources(), R.drawable.marker);

    canvas.drawBitmap(bmpFriend, friendScreenCoords.x,
        friendScreenCoords.y, paint);
    canvas.drawText("My friend is there...",
        friendScreenCoords.x, friendScreenCoords.y, paint);
        return true;
    }

    @Override
    public boolean onTouchEvent(MotionEvent event, MapView mapView) {
        //---when user lifts his finger---
        if (event.getAction() == 1) {
            GeoPoint p = mapView.getProjection().fromPixels((int) event.getX(), (int) event.getY());
            Toast.makeText(getApplicationContext(),
                "Location: " +
                p.getLatitudeE6() / 1E6 + ",|" +
                p.getLongitudeE6() / 1E6,
                Toast.LENGTH_SHORT).show();
        }
        return false;
    }
}

```

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Displaying Zooming controls
    mapView = (MapView) findViewById(R.id.mapView);
    mapView.setBuiltInZoomControls(true);
    mapView.setTraffic(true);

    inputServerIP = (EditText) findViewById(R.id.serverIP);
    //inputText = (EditText) findViewById(R.id.sendText);
    mText = (TextView) findViewById(R.id.text);

    mc = mapView.getController();
    String coordinates[] = {"31.025675", "121.428995"};
    //String coordinates[] = {"1.000001", "1.000002"};
    double lat = Double.parseDouble(coordinates[0]);
    double lng = Double.parseDouble(coordinates[1]);
    p = new GeoPoint((int) (lat * 1E6), (int) (lng * 1E6));

    String friendCoordinates[] = {"31.035780", "121.438100"};
    //String friendCoordinates[] = {"1.000002", "1.000004"};
    double friendLat = Double.parseDouble(friendCoordinates[0]);
    double friendLng = Double.parseDouble(friendCoordinates[1]);
    pFriend = new GeoPoint((int) (friendLat * 1E6), (int) (friendLng * 1E6));
    mc.animateTo(p);
    mc.setZoom(13);
    //---Add a location marker---
    mapOverlay mapOverlay = new MapOverlay();
    List<Overlay> listOfOverlays = mapView.getOverlays();
    listOfOverlays.clear();
    listOfOverlays.add(mapOverlay);
    mapView.invalidate();
    lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    LocationListener locationListener = new MyLocationListener();
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);
}

-----Net Work communication related code

* User can zoom in/out using keys provided on keypad */
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_I) {
        mapView.getController().setZoom(mapView.getZoomLevel() + 1);
        return true;
    } else if (keyCode == KeyEvent.KEYCODE_O) {
        mapView.getController().setZoom(mapView.getZoomLevel() - 1);
        return true;
    } else if (keyCode == KeyEvent.KEYCODE_S) {
        mapView.setSatellite(true);
        return true;
    } else if (keyCode == KeyEvent.KEYCODE_T) {
        mapView.setTraffic(true);
    }
}
```

```
        return true;
    }
    return false;    }}
```

6.5.3 GPS Location Mapping

The code below implement the LocationManager class which provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to fire an application-specified Intent when the device enters the proximity of a given geographical location.

```
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
.....
.....
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        .....
        .....
        lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        LocationListener locationListener = new MyLocationListener();
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);
        private class MyLocationListener implements LocationListener
        {
            @Override
            public void onLocationChanged(Location loc) {
                if (loc != null) {
                    Toast.makeText(getApplicationContext(), "Location changed : Lat: " + loc.getLatitude() +
                        " Lng: " + loc.getLongitude(),
                        Toast.LENGTH_SHORT).show();
                }
            }
            p = new GeoPoint((int) (loc.getLatitude() * 1E6), (int) (loc.getLongitude() * 1E6));
            mc.animateTo(p);
            mc.setZoom(18);
        }
        @Override
            public void onProviderDisabled(String provider) {
        }
        @Override
            public void onProviderEnabled(String provider) {
        }
        @Override
            public void onStatusChanged(String provider, int status, Bundle extras) {
        }
    }}
```

Tests

To test our application, we used one real Android smart phone and the Android emulator from the Android SDK. Because the Android emulator was not able to provide server sockets, we run the server application on the real Android smart phone and the client on the emulator.

Conclusion

Result

After the implementation, the test and debug process, our application finally worked in our test environment. We are happy, that we can say, that our first application, that we wrote for Android finally works.

We spent a lot of time to learn how to use Java and Eclipse to develop applications for Android. It was a great experience to get an idea, what it means to develop an application for Android.

Improvements

To be able to search more than just one friend, we could change our server-client model to a server-to-multiple-clients model or work with a central server, which runs on a server pc and the smart phones connect to this server as clients.

Also in Google Maps, we can improve our application by drawing a line between my location and my friend location, and showing the distance between the two locations.