Practical 1: Geomessaging

Application Description

There are three main parts of the Geomessaging application. The main activity, GeoActivity, upon creation appears as figure 1.(i). The user is allowed to take a photo and enter a descriptive text; the button after this action is replaced by the photo, 1.(ii). The second central element is a public server that allows various users to upload their footage. Figure 2 gives an overview of the data contained on the server (that may be returned as an XML file by an API call). In addition to the photo and text, the location is important, which is passed along with the photo and text on submission. The last main element of the application is a background service, that is initiated at the time the application starts. It is triggered by a location-service to download and identify messages in the vicinity of the user; the ones that are close by are indicated by notifications given in the Android OS status bar. In figure 1.(i) and (ii) we can see that there are already three such notices present, figure 1.(iii) shows the expanded view. Pressing one of the notifications will open up something akin to figure 1.(iv); a greeting left by another user.

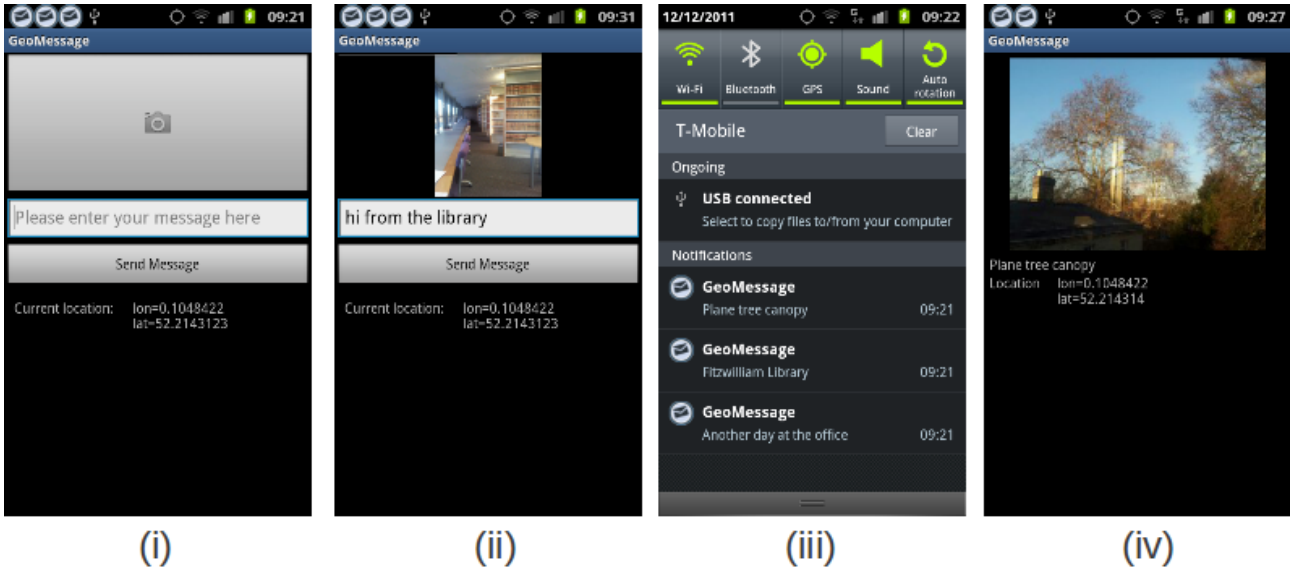

Figure 1: Series of screenshots

```
▼<messages>
   <message message="Plane tree canopy" latitude="52.214314" longitude="0.1048422" picture="image8774922402735242482.jpg"/>
   <message message="Fitzwilliam Library" latitude="52.214314" longitude="0.1048422" picture="image8095138555970404711.jpg"/>
   <message message="Another day at the office" latitude="52.214825" longitude="0.10657347" picture="image7851257954987328412.jpg"/>
 </messages>
```

Figure 2: The server API provides information about the uploaded images

Implementation

The most important task of GeoActivity is to provide the message posting functionality and to initiate the class BackgroundService. The background service is enabled by creating an Intent, which indicates that this part of the application will be a longer-running operation that will keep running in the background. In our implementation both GeoActivity and BackgroundService have their own LocationListener for receiving callbacks on GPS updates. GeoActivity will disable its listener when it loses focus, but the listener of the background service will keep running; while the user potentially will be interacting with a variety of unrelated applications in the meantime. It is worth noting that actively listening for GPS changes in this way is expensive on power (implementing a PASSIVE_PROVIDER for the background would probably have been a good alternative).

Considering the background service in more detail, the key functionality is the method

```
processLocationUpdate(Location location)
```

This callback is triggered by LocationListener (to conserve power and bandwidth, I set it to indicate on 100s intervals, and not less than 50m distances). The method downloads messages from the server, and will for each message in a given proximity (I've set this to 250m), produce a notification to the user. The method was intended to use the server API to retrieve messages within a given proximity, but on closer inspection the method ServerHelper.getMessages(...) always seemed to download the entire set of messages regardless of the values of the attributes.

To overcome this problem, I added to the Message class lat, lon attributes, and the following method

```
checkProximity(double lat, double lon, double distance)
```

which will return a boolean value to indicate if the given message is within range (distance given in metres). The method takes the cosine of the latitude (the angle of curvature of the Earth) to arrive at an approximate conversion from degrees longitude to metres (latitude is the same at any point on the Earth, about 111km per degree).

Adding the functionality of checking proximity in the Message class also necessitated adding to the XML parser in the ServerHelper to extract the lat,lon values from the XML returned by the server, these are now added to each new Message object as they are instantiated in the parser.

I also rewrote the equals(Object obj) method in the Messages class to check equality against lat,lon and message text, instead of pictureURI. The reason this was done was in consideration to the posting of messages in the GeoActivity. When a message is posted initially, GeoActivity doesn't have knowledge of the pictureURI created by the server, but it was still necessary to add the posted message in the Set<Message> notifiedMessages maintained by the background service. Otherwise, the user would receive notifications on messages posted by self. The Set of notified messages was therefore changed to a public static variable to allow access by GeoActivity.