Harsh Mer

Profiler - Ringer mode change Service

Introduction

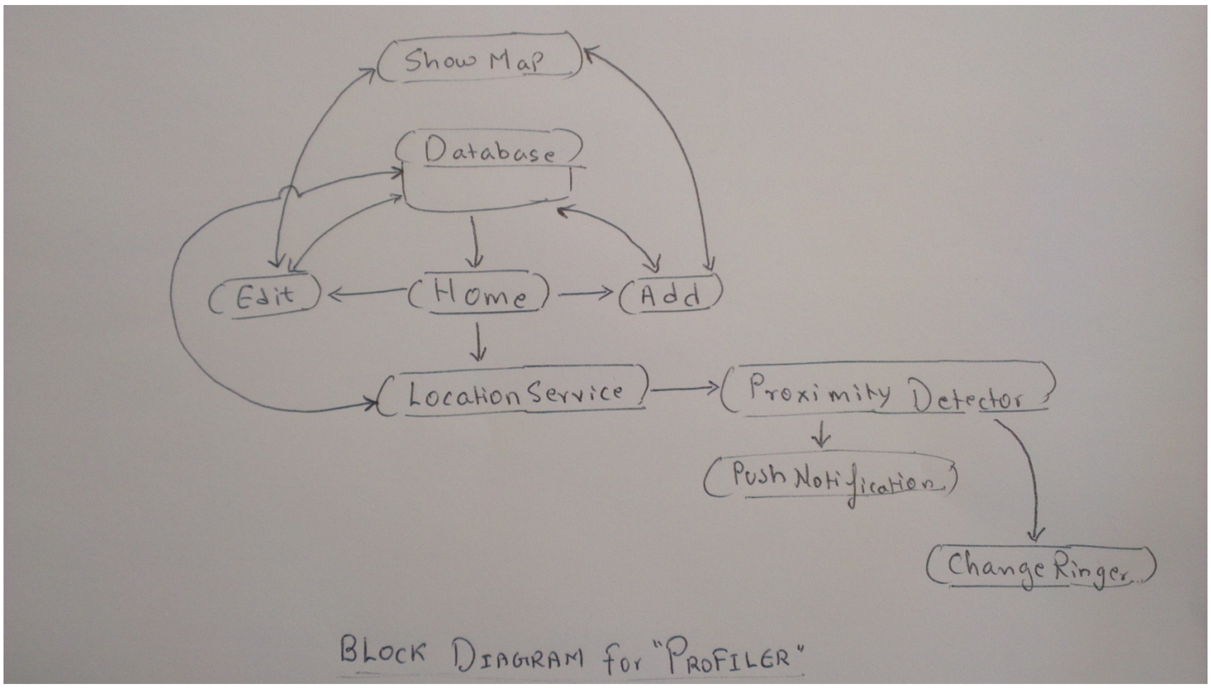
Profiler is an Android application which is based on Location aware profile changing of the mobile phone. This application provides the user with an ability to change the profile mode of their Mobile phone (Silent, Loud or Vibrate) automatically by storing the location co-ordinates of a particular location along with the profile mode which needs to be activated when it reaches that desired location. A location monitoring service is used to achieve this functionality.

Basic Usage

The application starts with an activity displaying a list view of all the locations (in form of name which the users have stored it in the application like for ex. “Home”, “Office”, “College” etc...). User is provided with a button on that same activity to add a new location which would take the user to the activity which reads the current location and allows the user to store the name against this location co-ordinates along with the profile mode which user needs to set against that location. Once the location is set, whenever user comes near to that location i.e. within the radius of that location, a notification is fired and the profile mode of the mobile phone is changed according to the profile set for that location.

This allows the user an ability to free themselves from the overhead of changing profile modes manually as there is always a room for an error where user would sometime forget to change the profile mode of the phone which in turn would interrupt certain discussions or the work. This Application AKA *Profiler* removes this overhead and does this task automatically.

Below is the Block Diagram of the Profiler application that explains the control flow of the application.



Structure of the Application

The structure of the application is a three tier architecture. It deals with the front end which is the GUI, the back end which is the database that handles all the location details and the names and the profile modes corresponding to it and a background service which handles the proximity sensing and changing the ringer mode of the mobile phone. The database is installed already in the application. So there is no overhead of creating the database every time the application is installed. The fetching of the data from the database and comparing it with the current location is done by the background location service. Basically this application is dealing with the location context.

Design & Implementation

Basically the design deals with three activity screens:

- Home screen
- Add or Edit screen
- Show map screen

Home Screen

The home screen which is the first activity when the application starts deals with the display of all the locations stored in the database along with the information about which profile is currently in a List View. Home screen also consist of an "Add" button which provides the facility to add new locations and profiles to be associated with.

During the first run of the application, the database would be empty and the location service will be initiated but will have no data to add proximity alert on. Once the user clicks on the add button the second activity will be called which deals with adding or editing profiles. The detailed description of the functionality of the second activity is described in the next section. When the user is in the second activity, the current location is acquired and the profile mode for this location is chosen by the user.

However, if the current location is not obtained by the location manager, the user also has the option of going to the map view and getting the desired location and importing the location co-ordinates from the map view into the activities location fields and set the location in the database by clicking on save button.

Once the save button is clicked, the control switches to the first activity which retrieves the data from the database and populating it in the list view & the location service is triggered and also a proximity alert is added to that location with a predefined radius of 30 meters.

The list view is a custom made list view with a custom adapter (ArrayAdapter) where every item in the list view is the name of the location co-ordinates stored in the database, along with a small text which displays what is the profile mode for each of the item. The theme used for the whole application is a black theme as it consumes less power and hence saving quite a bit of energy in display purpose.

Edit Functionality

To edit a particular location (list item), each location is long pressed and a popup menu appears with a command to edit. On pressing the edit command from the popup menu, the control is redirected to the second activity but with all details associated with the item which is clicked from the database.

For ex, if the list item namely “home” is clicked, all the details regarding home would be retrieved by the database class and the control would be transferred to the second activity filling up the text boxes in the activity with all the details associated with the location “home”.

Once again user can modify the details and hit the save button which will add a proximity alert to that location.

Add or Edit screen

This is the second activity in the application. This activity retrieves the current location by using the location service when clicked on the “show location” button. This button gets the current location. The onClick event of the button creates an instance of GPSTracker activity which extends the Service class in the android Library and implements the LocationListener. This activity uses the location Manager to retrieve the current location either from the GPS or the Network (depending upon which is active) and populates the Longitude and latitude in the respective fields.

The user is also provided with a spinner which holds the profile mode values like Loud, Silent, Vibrate. The selected value from the spinner gets attached with the location co-ordinates and once pressed on the Save button, the details from every field is saved in the database. The onClick event of the save button gets all the values from the fields and creates a new database class object which opens the database. A new instance of Location class is created which holds all the values retrieved from the fields in the properties (like for an instance longitude, latitude, name & profile Name). Using the database object, the store location function from the database class is called and it is passed the database object and the location object, which in turn is used to store the location in the database. Once the store function is executed, it returns a rowCheck value which can be used to check if the query executed correctly.

The edit screen is the same with a difference in populating the values in the field. The values gets populated from the database depending upon the list Item which has been pressed upon. Rest everything remains the same.

If in any case the show location button cannot get the current location, the “Show Map” button opens up a Map View in an another activity which allows the user with a facility to manually pick the location, put the marker and get the location co-ordinates populate into the previous activity (Add or Edit activity) automatically.

DatabaseAccess Class

This class controls all the database access, modification and deletion related queries. The database is created before using a plugin called SQLite manager in the Browser Mozilla Firefox. This database file is imported and copied to the root folder of the application programmatically. This removes the overhead of creating the database in the application programmatically.

Location Service

This service is started as the first activity becomes active. This service extends the service and implements the LocationListener. On start of the service the LocationManager object gets the context location service and requests for the location updates every 400ms. Using the database object the database is opened and the data is retrieved from the database and passed it to the addProximityAlert method to add proximity alert on each record. This method creates the pendingIntent which will perform a broadcast when the user is approaching any location which is stored in the database. The location co-ordinates obtained from the database, the pendingIntent and the radius is passed to the addProximityAlert method using the LocationManager object.

Furthermore, this broadcast will be received by a broadcast receiver which will handle the process of changing the ringer profile of the phone depending upon the location and its profile saved in the database.

The broadcast receiver on receiving the broadcast would check for the proximity entering for the mobile phone and set the Boolean flag accordingly. If the Boolean flag is set to true then a notification will pushed and the ringer profile of the phone would be changed accordingly by using the AudioManager.

Challenges

The biggest challenge in making this application work was detecting the proximity entering and switching the ringer profile of mobile phone. The location service retrieves the current location using either the GPS or the network. The problem with the function getLatitude() and getLongitude() of the LocationManager is they acquire the inverse values. The whole application of setting the proximity alert to the location co-ordinates was correct but the values which were being passed were inverse and hence the proximity alert couldn't work correctly. We tried to look for the solutions, but couldn't know until we found out using the help of debugger that the values we are getting are inverse and hence we had to change the order of the values which are being passed to the addProximityAlert() method to set proximity alert on the location.

Testing

Basically the testing of the application was done by changing the radius around the desired location. According to the documentation of the proximity alert, *"due to the approximate nature of position estimation, if the device passes through the given area briefly, it is possible that no Intent will be fired. Similarly, an Intent could be fired if the device passes very close to the given area but does not actually enter it."*

Hence we had to briefly check for the application to be working. For that matter, we had to completely walk out of the locations range, and then walk back in to see if the intent is fired and broadcast receiver receives the broadcast and changes the ringer profile.

Also testing has been done for not allowing the user to store a null values (empty values) while adding or editing the records in the second (Add/Edit) activity. A simple check is run before the data is being entered to the database whether it is empty. If it is empty, user will be notified with a toast message that the values can't be empty or it can't be the same as the name already exist. So a check for redundancy is also being tested.

Harsh Mer

My Contribution

This application is developed as a team. But in a team every team member has a role to play. My role was to design the front end which comprises of the custom ListView and the design & the back end which is nothing but the database.

Front End

List View:

Every list item has two values

- Name
- Sub name

This list view is set with a custom adapter which extends the ArrayAdapter<T>. This adapter gets the Name (name of the location) and the Sub name (name of the ringer profile for that location) from the ArrayList (constructed by retrieving data from the database) for both which is passed to the custom adapters constructor while instantiating it. The values from the ArrayList is then set to the name and sub name respectively.

The GUI also contains an Add button which transfers the control to the second activity which is Add/Edit activity.

The edit functionality works by long press on every list item, and it will eventually pop up with an alert box with a button for edit. Once the button is clicked, the control is transferred to Add/Edit activity.

Back End

Back end comprises of the database activity. For creating the database, I have used the SQLite Manager plugin for the Mozilla Firefox browser. This plugin gives you an ability to create a .sqlite database with a GUI. After creating the database, the .sqlite file is then imported to the Assets folder.

In the database class, the database is first checked for its availability in the applications root directory “/data/data/com.nightowl.profiler/databases”. If the database is not available or it doesn’t exist then the directory “database” is created and the finally the database is copied to that location. If the database does exist, then it is opened for the operations. This class deals with all database related operations like Reading, updating and deletion of records.

A simple check operation is applied to see if there are no duplicate names (location Name) in the database as it will cause a problem for redundant data stored in the database. Also none of the fields in the database allows null values and hence a check for the null values is being taken care off programmatically.

Harsh Mer

The structure of the database is:

```
CREATE TABLE "location" ("id" INTEGER PRIMARY KEY  NOT NULL , "name" VARCHAR NOT NULL , "longitude" DOUBLE NOT NULL , "latitude" DOUBLE NOT NULL , "profile" CHAR NOT NULL )
```