School of Electrical and Information Engineering

University of the Witwatersrand, Johannesburg

ELEN 4010 Software Development 3

Project 2010

Dribble: The Design, Implementation and Analysis of a Geosocial Networking System - Mobile Client

May 2010

Chad Epstein 0706965X

Partner:

Ashleigh Campbell 0605245J

http://github.com/ajpaverd/Dribble

DESIGN AND IMPLEMENTATION OF DRIBBLE MOBILE APPLICATION

Chad H. Epstein

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: Dribble is a location-based social networking client-server application. A user is able to receive messages called Dribs within three kilometers of the user's current location. The mobile application was developed using Android 1.5 for the HTC Dream mobile phone. The mobile development team used a form of version control and divided the work over seven weeks and amongst team members. The application was tested using logging and integration test with the server and was found to function effectively however improvements can be made by refreshing content automatically in the background and porting it to other platforms. This would allow it to appeal to a wider market if it is to be deployed in the future.

1. INTRODUCTION

Social networking applications are becoming very popular at the moment and have continued to grow over the past 10 years [1]. Determining the location of a user has become easier as technology develops and since this has been implemented in common devices such as cellphones. Many of the latest generation cellphones contain a form of GPS positioning built in and this trend will become more apparent in the future. Currently there are a number of so-called "geosocial-networking" applications which include location-enhanced technology. One of the most popular of these is called Foursquare which adds a game element to social applications such as Twitter and Facebook [2].

The aim of this project was to build and release an open-source software application. A cellphone application called Dribble was developed and would be classified as a geosocial networking system because of its core focus on location-based real-time social information sharing. The application allows users to send and receive messages to and from their mobile phone to topics that are within the range of a user's current location. The application has a cellphone client and a remote application server however this report will focus mainly on the cellphone application.

Assumptions and constraints for the mobile application are stated in Section 3, followed by a general overview in Section 2. The application contains many design components and these are detailed in Section 4. The project was managed using a version control system and a logging system described in Section 5. To ensure proper functionality, the system was tested and the tests and test results, along with possible improvements to the project are provided in Section 0. Finally, there is a conclusion to the report in Section 7.

2. ASUUMPTIONS AND CONSTRAINTS

2.1. Assumption

There were several assumptions that need to be made when designing Dribble. It is assumed that the Android cell phone has sufficient access to the Internet either over one of the network communication technologies mentioned previously as well as sufficient bandwidth and speed to run effectively. The application will still retain most functionality without displaying the maps, however in order for them to function it requires that the Google Map server is available and can deliver results fast enough. It is also assumed that the remote server is always accessible and reliable. This means that the server functions correctly, that there is very little downtime, and that the server can handle and process requests. It is also assumed that even when the application is used by a number of users, the server will be sufficiently load balanced such that there is no additional lag when a large number of users use the service simultaneously.

2.2. Constraints

There are constraints that need to be considered due to the project requirements, the mobile development platform the application's client-server architecture. The application was developed using the Android development platform, therefore it is currently limited to Android devices only. Using a mobile device means that there are inherent constraints because of the limited processing power, memory, and storage of these types of devices. Once the application is running on the device, the network bandwidth and network speed limit how much data can be sent to the server and how fast. The protocol used to send this information to the server cannot be modified and must be used by all mobile devices for the service to work correctly. The application also depends on the server availability at any given time and how quickly it can receive and process inputs and return results. The DribSubjects and Dribs sent to the server are limited to maximum lengths of 20

and 144 characters respectively to encourage users to keep these concise and prevent them from being overly long. Once completed, the application must be released as open source and this means that all third party libraries used during development must be open source too.

3. PROJECT OVERVIEW

A generalised system model of the overall Dribble system is depicted in Appendix A, Figure 1. The system is primarily divided into two components – the mobile application (Android) and the remote application server (Glassfish). The mobile application is executed on the HTC Dream mobile phone and provides the GUI with which the users will interact. The mobile phone initiates a connection to the server via web services, set up on the remote application server. There is a common package used with both systems containing the`Drib' and `DribSubject' classes which are used to convey messages and message subjects, respectively, to and from the server. The objects are serialised in order to be sent over a HTTP connection, using: 3G, EDGE, GPRS, or HSDPA depending on the capabilities of the phone and network. The sent object is deerialised on the server side, and processed using Enterprise JavaBeans. The database is queried with the relevant request and returns the desired result. This result is returned using Enterprise JavaBeans and serialised to be sent back to mobile application using RESTful web services. The mobile application uses Java and XML to graphically display the Dribs and DribSubjects.

3.1. Use Case Model

The UML use case model of the Dribble application is shown in Appendix A, Figure 2. This model depicts the major functional use cases as seen from the user's perspective. This functionality is provided by the mobile application which is in turn supported by the server application. As shown in the figure, the purpose of the application is for users to upload messages about events relevant to their current locations and then view the messages of other users. Since all messages are anonymous, all messages are available to all users in the area. By providing this functionality, Dribble can be classified as a geosocial networking system.

4. APPLICATION DESIGN

The mobile application is the client of the overall Dribble system. A detailed structure is shown in Appendix A, Figure 3. To come up with a design, first, a description was established highlighting the problem statement from the mobile application perspective and generating key requirements necessary. Next, a prototype GUI was constructed to evaluate the basic GUI elements, interaction flow, ease of use, and functionality that would need to be developed. This was re-evaluated after constraints became apparent whilst developing the design and research into the implementation. Next a use case model was developed to analyse the scenarios the user would navigate through as described in Section 3.1. A key concept and trend in software development is the implementation of modular code. This is a technique that is used throughout the design and source code, and requires that the system be decomposed into separate components, which can be added or removed without affecting one another. In this application modularity is achieved between the presentation and logic layers of the software system. The presentation layer is scripted in XML and links to the Java source code (logic layer), through the use of the `R.java' class which is automatically generated. Further, modularity was implemented through the use of packages and web services.

4.1. Platform Description

The project made use of the Android SDK which provides the tools and APIs necessary for development. The programming language used to develop the mobile application is a form of Java that is specific to the Android platform. This was integrated into the Eclipse IDE using the Android Development Tools (ADT) plug-in which provided some benefits. It generates the correct project structure for the application, and keeps resources, such as XML layout or images, compiled automatically. It also provides tools integration, custom XML editors, and a debug output pane called LogCat all of which will be discussed later [3]. It is possible, however to use other IDE's, but it would take more effort and time to use and therefore Eclipse is the recommended IDE by Android [3] There have been recent releases of the Android operating system and there may be platform incompatibilities when developing for different firmware versions. The phone used for development (HTC Dream) contained Android version 1.5 and the same application may therefore not work on higher firmware versions, the latest being Android 2.1[4]. The firmware is not obsolete however, but rather the opposite since at the time of writing the report, Android version 1.5 is still being used on a large majority of Android devices [4].

The reason Android specifically was chosen for development is that it is a modern development platform which is fast gaining popularity and an Android phone was provided to the group try out the new technology.

4.2. Android Application Structure [5]

An Android application is structured differently to a typical Java or JavaME application. The compiled Java code — along with any data and resource files required by the application — is bundled by into an Android package, an archive file marked by an '.apk' suffix. This file is used for distributing the application and installing it on mobile devices. All the code in a single .apk file is considered to be one application.

A central feature of Android is that one application can make use of elements of other applications. These do not have to be incorporated but can be called and started when necessary therefore there is no single entry point like in a typical Java application. An Android application has a few components. Activities - an activity presents a visual user interface for one focused task the user can perform. All must inherit the Activity base class and there can be more than one. One Activity is marked as the main one for startup. The visual content of a window is made up of a hierarchy of views derived from the base View class where the user interaction takes place. View control a portion of the screen and their layout is usually organized by their parents called ViewGroups. Android has built in views to use. Other components are Services, Broadcast Receivers and Content Providers however these were not used in the application. An Android application also contains a manifest file which is used to inform Android about the applications components. Finally, Android stores the resources used for the program separate to the code. This includes the 'layout resource' for XML layout files which describe the layouts and views, a 'drawable' resource for images and image states, and a 'values' resource for values that can be accessed throughout the code.

4.3. Graphical User Interface

The application begins by displaying a custom splash screen for a predefined time containing the group logo to help identify the application and act as a starting point. This is shown in Appendix A, Figure 4. This is the main Activity when starting the program. The remainder of the user interface was designed with a tab layout as the main navigation tool. This allows for fast and intuitive interaction with the program, especially for touch-screen enabled devices, and creates a distinct separation of application activities. The tabs are (from left to right): 'Map', 'Topics', 'Dribs', and 'New' shown in Appendix A, Figures 5-9. 'Map' contains location-related functionality to be discussed later. The 'Topics' tab allows the user to obtain a list of the most highly ranked DribSubjects in the area surrounding him or her. The 'Dribs' tab contains a list of Dribs for the selected DribSubject and the 'New' tab allows for the creation of new Dribs and DribSubjects. Each tab is its own Activity and has its own XML layout file attached to it to display views. The code for each tab is separate from its view layout and the same XML layout can be used for multiple activities if necessary. Tabs are defined as TabWidgets, are grouped together by a TabHost, and created and displayed in a TabActivity. The content of a tab is defined by its TabSpec. Each tab has its own icon contained in the 'drawable' resource. The icon has separate images for when the tab is active or inactive. Defining which icon is used for which state is an xml file contained in the 'drawable' resource as well. The content of a tab is a FrameLayout used to display the tab views. By default, the Android application does not handle rotation of the screen from portrait to landscape and vice versa automatically. Therefore functions were added in the manifest file and in the code to respond to rotations and to reload the correct layout as necessary.

4.4. Location Retrieval and Implementation

For the Map tab, the Google Maps library and APIs built into the Android SDK where used to perform map activities. This includes: obtaining a GPS location, and the ability to show, pan, and zoom a map. It also allows for overlay items to be displayed on the map, which, for the Dribble application, was used to show topic locations and details within the user's sphere of reference. On startup, the Dribble application uses a thread to attempt to obtain the users location from: GPS if available, otherwise from an approximate network location using cellphone towers. While the application is requesting a location, the map is displayed and tiled using data obtained from Google via the Internet. To use this function in an emulator environment, it was required that a developer's key be retrieved from Google that is locked to the developer's machine. This was done by executing commands to generate a ssh hash which was entered in a Google website to obtain the key. The key is not necessary when deployed to the cellphone. It was also necessary to set permissions in the manifest file to allow the cellphone to use

GPS and Internet capabilities. Once a lock on a location has been obtained the map zooms and animates to this position. Locations retrieved from the cellphone are stored as integers representing microdegrees as this is the data type Android location-orientated functions use.
The map contains a default overlay item until topics are fetched from the server. After topics are fetched from the server, an overlay item is generated for each topic and added to the map. The overlay items are cleared and re-generated each time topics are fetched to ensure that the map only shows topics that are applicable to the area surrounding the user. The overlay items have a listener responding to user taps which displays the subject's name once performed. This can be extended to display other details. Once the first location fix is obtained, a thread is started that runs throughout the duration of the program that constantly refreshes the user's location. It does so by implanting a LocationListener interface that runs in the background and waits for location changes which are stored by the thread at set intervals. For testing purposes, this was set at every 10 seconds. This delay could be increased in a production version as a short interval may consume unnecessary resources. Other activities access the thread throughout the program to use the location details when requesting DribSubjects, sending Dribs, and sending likes or dislikes.

4.5. Activities

The application contains a number of activities used to represent the logical functionality of the program. The 'Topics' tab extends ListActivity to display a list of subjects available to the user fetched from the server (discussed later). It also stores the last accessed DribSubject which available to other activities. Once topics are fetched, overlay items for each are created and added to the map. Users may filter through subjects by typing the corresponding letters on the keyboard. When a subject is clicked, the tab is changed to the 'Drib' tab which also extends ListActivity. Here lists of messages or Dribs are retrieved from the user and displayed. The dribble icon is displayed next to each Drib along with: a timestamp of the message showing the date and time, the message rank which is calculated at the server, and a like and dislike button for rating messages. Below the messages is a reply button for replying to Dribs. These are all views defined in the XML layout file for the tab. When a like or dislike button is pressed, the message is sent to the server to increase or decrease the rating. The reply button takes the user to the 'New' tab where a message can be created. There are two types of views for this tab, one for replying to messages and one for creating new subjects. When replying, the subject creation field is disabled. The message text area limits messages to 144 characters and the subject area to 20 characters. When the Submit button is pressed, the message, along with the subject ID is sent to the server for processing. Each time the 'Topics' or 'Dribs' tabs are clicked, the content is refreshed to the most up-to-date information from the server. This ensures that even if the user changes location, they always receive the latest content. This, however, still requires user interaction. A better implementation would be to use a thread that refreshes content automatically in the background at certain intervals which would not require user input. However, a possible disadvantage to implementing this functionality is the added processing for the application which could slow it down and run the battery down.

4.6. Communication

The last component of the program is the communications with the server. Three packages were implemented in the application: `common.dribble' (contains the classes common to the mobile application and the server, and wrapper classes); `dribble.dribbleapp' (contains the classes implementing mobile functionality) and finally `XStream' (discussed below). The `common.dribble' package provided a common interface between the server and the client. It contained the main classes `Drib' and `DribSubject', which were instantiated every time a user created a message or message subject. These objects contained important data used for processing such as names, latitudes and longitudes, number of posts, views, a rating, and a popularity score. Before an object can be sent, it needs to be serialized into XML as RESTful web services on the server-side application. Objects are automatically serialized with a third party library called XStream, and sent via a HTTP Put request (no object is sent when using the HTTP Get request). Due to the incompatibility of the different serialization packages utilised on the mobile application and the server, two wrapper classes were introduced (one for each common class). The purpose was to ensure the XML root nodes conformed to a standard structure, defined on the server side. Each topic or message is sent to a domain defined in the DribCom class. The URL changes, however depending on what is being sent. Subjects and messages are received with a GET request containing the latitude, longitude and number of results wanted, and messages and message likes or dislikes are sent with a PUT

request. The GET request returns an InputStream, which is first converted to a string and then deserialised with the XStream API's before being used.

5. PROJECT MANAGEMENT

Project management is vital to the project's development. Resources such as time and skills need to be respectively distributed and assigned to tasks. Members need to be allocated tasks or sub-tasks. Aspects include version control and team management. Project management with respect to the mobile application will be discussed below.

5.1. Version Control

In order to manage the source code and configuration of the project, version control systems were used. This facilitated individual development but also ensured that changes were integrated back into the central project.
For the mobile application, a pair programming methodology was mainly used. This meant that there was no need for extensive use of a version control system. However, in order to centralise and backup development work, the DropBox collaboration system was used [6]. The Dribble mobile application is not the standard type of project for which version control systems are designed, therefore the advantage of DropBox is that its simplicity avoided any major conflicts, and maintained an adequate backup of the system.

5.2. Team Management

A large portion of the project was coded using the technique of pair programming. This technique involves two programmers sharing one workstation, both contributing equally to the development of the same source code. Time management activities are provided with the timesheet in Appendix B, Table 1. Task times were calculated by measuring the start and end of each programming session and adding up the times. There is also a basic week-by-week Gannt chart description in Appendix B, Figure 1.

6. DESIGN EVALUATION

6.1. Trade-offs

Some aspects of the project needed to be reconsidered to make it easier to implement and faster and easier to use. Initially, a tag cloud was initially going to be used to represent DribSubjects on the mobile device and this was changed to use ordered lists which essentially perform the same task. To provide faster transmission, whole DribSubjects are not sent to the server but rather only the unique ID representing the DribSubject which enables less overhead over the network. Other trade-offs occurred when= dealing with positioning. The radius around DribSubjects and Dribs is represented in meters, however latitude and longitude obtained from the mobile device are returned as microdegrees, thus a conversion is necessary. An estimation of this conversion factor for South Africa was found to be approximately 8000 degrees per kilometre by trial and error. Also for simplicity, the area around

6.2. Testing

Due to the variety of components which comprise this system, a number of different approaches were used to test and ensure system functionality. These included the extensive use and analysis of log files, multi-platform testing and integration testing. log files were used to monitor the status of the application and to detect and identify any errors which occurred. In order to meet the requirement of using standardised components and subsystems, the logging was implemented using the standard capabilities of the Android emulator. By following this recommended convention, the logging output could be processed and displayed in the standard format for each platform. The logger used was called LogCat. LogCat was utilised for session logging, and custom logs were embedded in the code to highlight function executions, thread and error checking and the ability to monitor warnings and override methods.
Multi-platform testing was used on both the mobile and server sides of the system. On the mobile side, the application was tested using both the Android emulator and an HTC Dream device. This was done to confirm that the application functions on both the emulator and a real target device. The integration testing was the final high level testing of all the functionality in the system. This was performed from both the emulated and physical mobile devices. This type of testing included all the factors which would be present in the production system including the possible performance limitations and reliability issues associated with using a mobile data network. Given more time, it could be possible to test the application using profiler tools to see where exactly slowdosn occur in the code. Android has such a tool available but it is currently not integrated in the Eclipse environment. Although the required tools were investigated and acquired, this testing did not take place due to time constraints.

6.3. Possible Improvements

The project has potential for future work and there are many improvements that can be made. Currently, the application does not support having punctuation marks in Dribs as a result of a bug on the server side. This is an issue that would need to be resolved for future work as it limits the type of messages a user can send. Drib content can be refreshed manually by the user but if not performed for some time it could mean that the user may be viewing out-of-date information. It would therefore be better to use a thread that refreshes content automatically in the background at certain intervals. The chosen platform limits the number of users that can benefit from the application. The current application runs on one platform and one version only (Android v1.5). To grow the user base, it would be beneficial to port it to other platforms such as Nokia's Symbian or the Blackberry architecture, and to higher firmware versions. The project was designed to be sustainable and has real-world applications and could be marketed and developed further to gain popularity amongst users with a main goal of eventually generating a profit. The application source is currently hosted on a central version control server as an open source project, however if the application were to make a profit, the project would need to forked to another development branch which would have be closed source to protect intellectual property.

7. CONCLUSION

The mobile application was designed on the Android platform which was a newly investigated modern programming environment. The application was run on the HTC Dream Android device and consists of a tabbed interface. Users are able to send messages to subjects located within a certain distance around them and view subjects near their location on a Google Map. To do so, a thread continually updates their location. The application used a form of version control and team management was implemented to divide the work. Testing was performed using both the emulator and actual device along with logging. There were some trade-offs that had to be made in the design due to time and platform constraints. Although the application functions effectively, there are still possible improvements to be made with the end goal of having the application marketable.

REFERENCES

[1] T. Economist. “A Special Report on Social Networking." The Economist, January 2010. URL http: //www.economist.com/specialreports/ displaystory.cfm?story_id=15351002. Last Accessed: 29 April 2010.

[2] Foursquare. “foursquare." URL http:// foursquare.com. Last Accessed: 29 April 2010.

[3] Android. “ADT Plugin for Eclipse.” URL http://developer.android.com/ sdk/eclipse-adt.html. Last accessed 2 May 2010

[4] Android. “Platform Versions.” URL http://developer.android.com/ resources/dashboard/platform-versions.html. Last accessed 2 May 2010

[5] Android. “Application Fundamentals”. URL http://developer.android.com/ guide/topics/fundamentals.html. Last accessed 2 May 2010

[6] Dropbox Home. “DropBox." URL http:// www.dropbox.com/. Last Accessed: 29 April 2010

APPENDIX

A. Diagrams

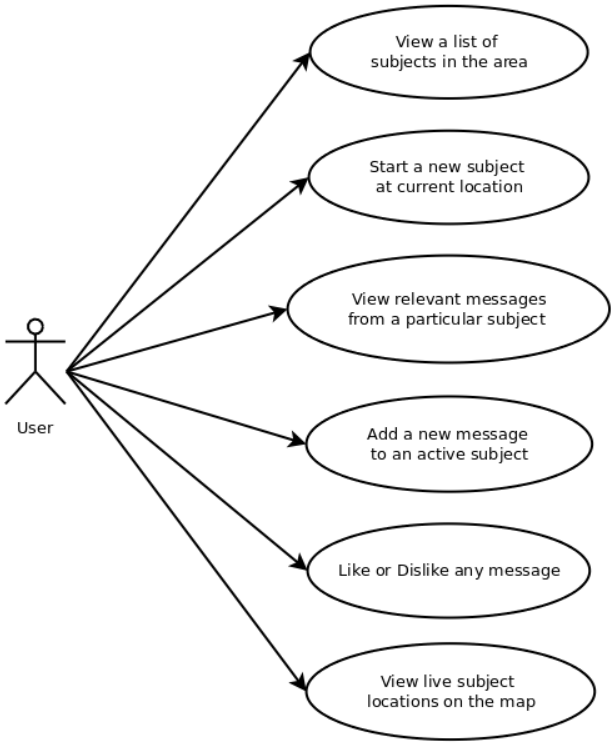

Figure 2 Mobile use cases

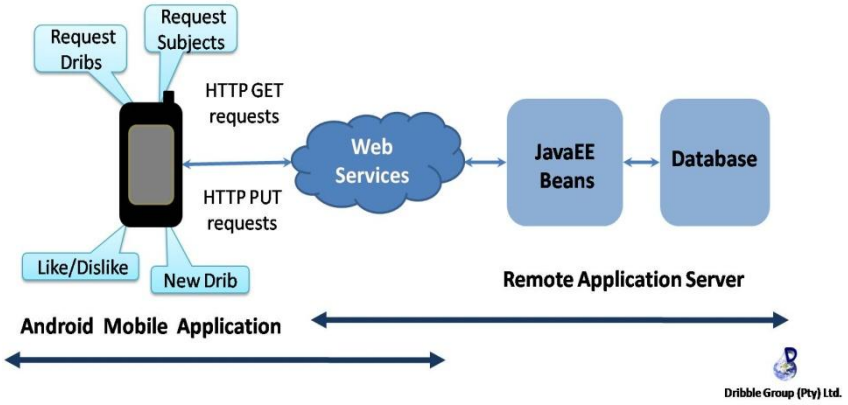

Figure 1 Overall system functionality

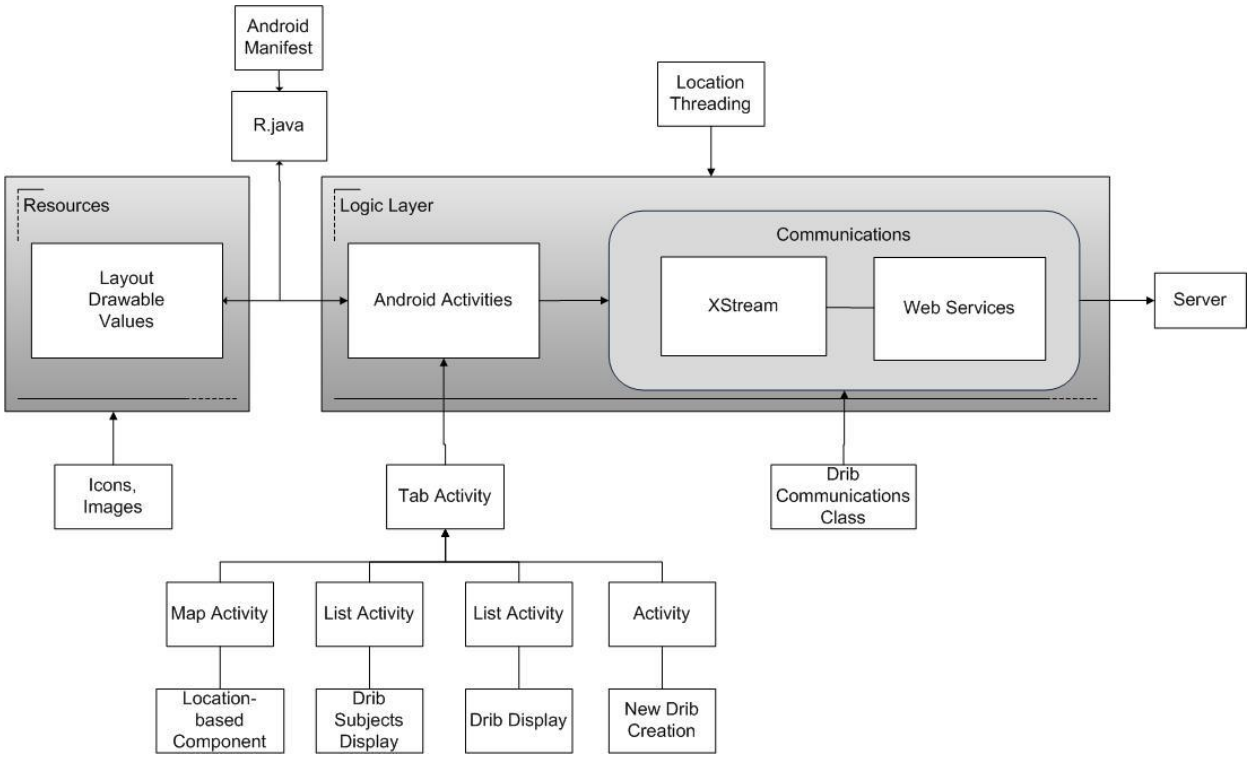

Figure 3 Detailed system diagram

Figure 4 Splash screen

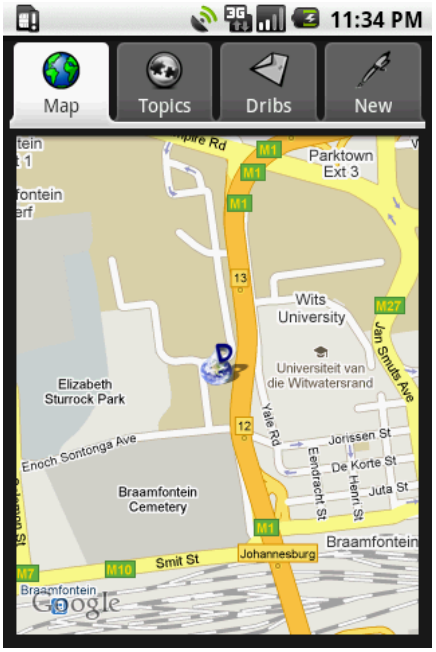

Figure 5 Map tab

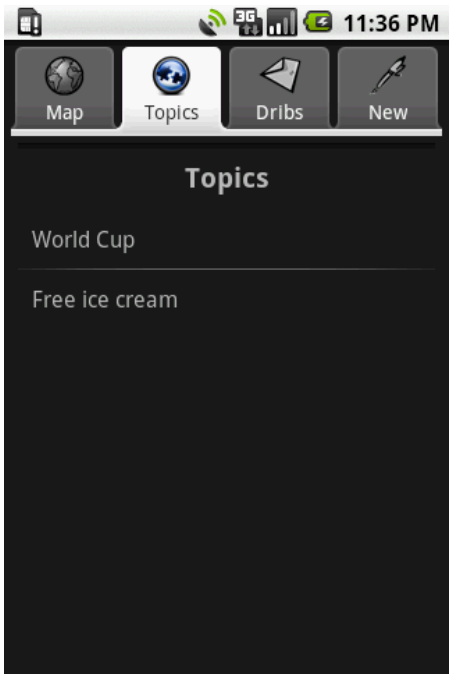

Figure 6 Topics tab

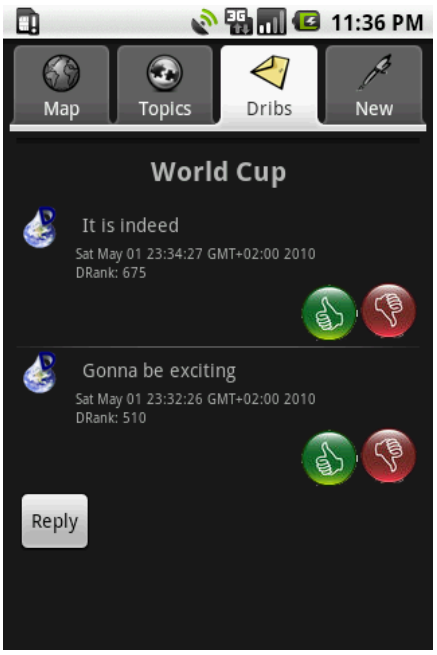

Figure 7 Dribs tab

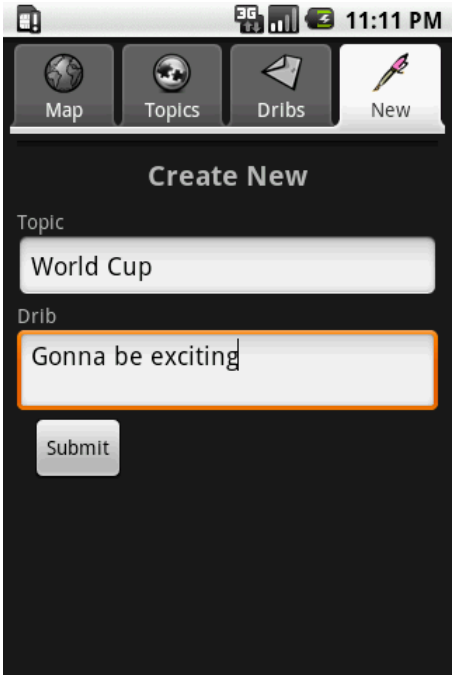

Figure 8 New tab showing reply message

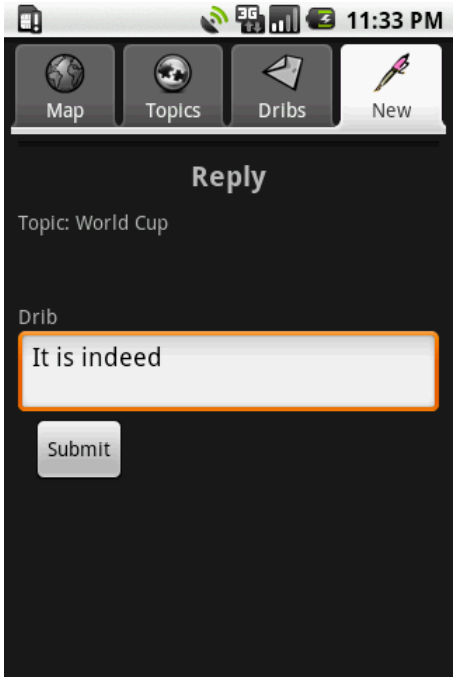

Figure 9 New tab showing new message

B. Timesheet

Table 1 Individual task activity timesheet

	Chad	
Task	Estimated Time (hours)	Actual Time (hours)
Location-based functionality	9.45	15
User Interface	9.45	20
Activity Implementation	4.725	10
Communications	3.15	3
Additional Functionality	1.575	1
Total	31.5	49

	Week 1	**Week 2**	**Week 3**
Location-based functionality			
User Interface			
Activity Implementation			
Communications			
Additional Functionality			

	Week 4	**Week 5**	**Week 6**
Location-based functionality			
User Interface			
Activity Implementation			
Communications			
Additional Functionality			

	Week 7
Location-based functionality	
User Interface	
Activity Implementation	
Communications	
Additional Functionality	

Figure 1 Week-by-week Gannt chart