SWE599 Project, Spring 2014
Instructor: Fatih ALAGÖZ

SWE599 Project
LRB: LittleRedButton
Design Specifications Document

Revision 1.0
23.03.2014

By: Mustafa Göksu GÜRKAŞ
Student Id: 2011719225

Revision History

Revision	Date	Explanation
1.0	23.03.2014	Initial design specifications

Table of Contents

Table of Figures

1. Executive Summary

The LittleRedButton project is a mobile application design and development project for users who need urgent help in some cases. The goal is firstly shoot a picture of the scene where the user is, then to locate the user who declares an emergency occasion and finally to share his/her GPS location coordinates (with the photo taken as an attachment) through various selected notification channels to the people whom the user selected as recipients of the application previously. The recipient list is created and the notification channels are selected formerly by the user. When the user needs to notify his friends that he/she is in an urgent situation, he/she simply just presses the red button. The LittleRedButton sends a Facebook post, a tweet, an e-mail or a SMS to its recipients immediately.

In latter sections, the development view and reuse capabilities of the considered design is described.

2. Introduction

2.1 Purpose

The purpose of this software project is to design and develop a new mobile application on Android devices. The mobile application is to be implemented with JAVA for Android and by using a database system like MySQL. One aims to accomplish the following tasks by using this system.

1. Users shall be able to locate himself/herself by using his smart phone's GPS/Network location services.
2. Users shall be able to add or delete his/her contacts in his/her smart phone as recipients.
3. Users shall be able to display his/her contact list that takes place in his/her smart phone memory.
4. Users shall be able to login to/logout from his/her Facebook account by using this application.
5. Users shall be able to notify his/her friends with his/her current GPS location coordinates as an emergency message via his/her Facebook profile.
6. Users shall be able to login to/logout from his/her Twitter account by using this application.
7. Users shall be able to notify his/her friends with his/her current GPS location coordinates as an emergency message via his/her Twitter profile.
8. Users shall be able to notify his/her selected recipients with his/her current GPS location coordinates as an emergency message via SMS.
9. Users shall be able to notify his/her selected recipients with his/her current GPS location coordinates as an emergency message via e-mail.
10. Users shall be able to let the application track himself/herself by sending emergency messages with his/her current (updated) GPS location coordinates to above channels every 60 seconds.
11. Users shall be able to display his/her current GPS location coordinates on Google Maps.
12. Users shall be able to display nearest police stations and health institutions to his/her current GPS location coordinates on Google Maps.
13. Users shall be able to configure the application's behavior by arranging the settings of the application.

There will be only one actor in this mobile application which is the user himself/herself.

When the user declares an emergency occasion, the GPS location coordinates are extracted by using the mobile platform's location services. For sharing through different channels (such as SMS, e-mail, Facebook and Twitter), the mobile platform APIs and frameworks will be used.

The design details of all application functions and the user interface are given in the following sections of this document.

The design is based on LRB Requirements Specification Document, Revision 1.0, in file Gurkas-MustafaGoksu-LittleRedButton-RSD-2014-03-09-Rev-1.0.pdf [1].

The design approach of this project consists of clearing the meaning of the basic native mobile application architecture. For design representation notation, UML class diagram is used as the best way to show the structures of a system, the information flow and return values in the same notation. Each class is examined in detail and brief summary about how and why they are used are explained in latter sections.

2.2 Scope

The scope of the project is LittleRedButton – emergency declaration mobile application. The list of features is composed of:

- Configuring application settings and notification channels
- Adding Recipients from phone contacts to the Recipient List
- Removing Recipients from the Recipient List.
- Locating the user's GPS location coordinates
- Shooting the picture of the scene that the user is currently at
- Sending a post including the location coordinates and current scene at that coordinates to user's Facebook wall
- Sending a tweet including the location coordinates and current scene at that coordinates to user's Twitter status
- Sending an emergency SMS containing the location of the user to the people that take place in recipients list
- Sending an emergency e-mail containing the location coordinates and current scene at that coordinates to the people that take place in recipients list
- Locating the nearest police stations and health institutions to the current coordinates of the user and mark/display them to the user on Google Maps.

2.3 Glossary

GPS: Global Positioning Service
MVC: Model View Controller
OAuth: Open Standard for Authorization
SLA: Service Level Agreement
SMS: Short Messaging Service
UML: Unified Modeling Language

2.4 References

1. LRB Requirements Specification Document, Revision 1.0 (Gurkas-MustafaGoksu-LittleRedButton-RSD-2014-03-09-Rev-1.0)
2. ANSI/IEEE 1471-2000, Recommended Practice for Architecture Description of Software-Intensive Systems (https://standards.ieee.org/findstds/standard/1471-2000.html)
3. Java SE Application Design with MVC (http://www.oracle.com/technetwork/articles/javase/index-142890.html)
4. Model-View-Controller (http://en.wikipedia.org/wiki/Model-view-controller)
5. OAuth – Open Standard for Authorization (http://en.wikipedia.org/wiki/OAuth)
6. Facebook for Android (https://developers.facebook.com/docs/android/login-with-facebook/v2.0)

3. Stakeholder and Scenario Analysis

Here we present the stakeholders and an analysis for a simple scenario. For the definition of requirements traditionally the user was taken as a reference. One could argue what exactly the term user includes but it is obvious that not only the requirements of the user have to be analyzed. The software has to be developed, run, administered, maintained and monitored according to the certain standards or regulations. Each of these aspects is of interest to different people which not necessarily are using the system at all. These groups of people are referred as stakeholders. A stakeholder is defined as an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system [2].

The stakeholders for the LittleRedButton application may be anybody who need urgent notification on an emergency situation. Each stakeholder has interests and concerns, which influence the requirements and also architecture design. During the design, whether the architectural design choices reflect also the interests and concerns or not should be continuously checked.

3.1 System Integrator and Solutions Provider

The system integrators responsibilities are to develop a technically sound application for integration and interoperability of the Mobile applications back-end and available devices and services as well as the availability of system. The developers of the system integrator will use the Android SDK intensively. He needs to perform continuous upgrade and extension services and ensure compliance with traditional standards and regulations in mobile application development. His goal is to deliver the implementation of the mobile applications to contract cost and time effectively. The provider wants to ensure business continuity by way of risk litigation.

3.2 End Users

The responsibilities of the tenants are to use the system effectively and read the user manual before using the system. The user's goal is to declare emergency help message with GPS location coordinates as soon as possible and they want the most user-friendly interface that can be developed.

3.3 Maintenance Crew

The maintenance crew has to monitor the subsystem states and react responsibly to provide maintenance according to the SLA. They are also responsible to deliver diagnostic and logging information to the user data repository. Their goal is to fulfil the SLA´s contractual obligations and ensure cost effectiveness in monitoring and maintenance.

4. Methodology

4.1 Software Architecture Design Fundamentals

The LittleRedButton application design process is based on the standard IEEE 1471 "Recommended Practice for Architectural Description of Software-Intensive Systems" which defines core elements like viewpoint and view. It also describes that stakeholders need to be involved and how to apply stakeholders needs to the architecture. This will be supported by the introduction of "architectural perspectives" which was introduced by Rozanski and Woods [2].

4.1.1 Requirements & Architecture

A defined process has been established to gather requirements in a structured way. In order to achieve this task, discussion rounds with focus groups of expert developers, who will possibly be responsible for the development of LittleRedButton application project have been held. The requirements were then prioritized and a fit criterion selected which allows measuring if a requirement is met or not. Not only from the focus group discussions but also from other sources e.g. standards, best practices, each partner´s experience and so on did the requirements are deduced. In this way we made sure that we collect a broad range of requirements to reflect the wide range of stakeholders. Requirements and architecture influence one another. Requirements are an input for the architectural design process in that they frame the architectural problem and explicitly represent the stakeholders' needs and desires. On the other hand, during the architectural design one has to take into considerations what is possible and look at the requirements from a risk/cost perspective.

4.2 Software Architecture Design Process

4.2.1 Architecture Definition Activities

Rozanski and Woods have based the architectural design process on the following definition: "Architecture Definition is a process by which stakeholder needs and concerns are captured, architecture to meet these needs is designed, and the architecture is clearly and unambiguously described via an architectural description." [2] A broad set of principles should be considered if the architectural design should be of good quality. Stakeholders have to be engaged to collect their concerns so the requirements can be balanced if there are conflicting or incompatible ones. The architectural design must allow for effective communication between all stakeholders and it must be structured to ensure continuous progress. Given the complexity of the project the design and also the process has to be flexible so we can react quickly to changing requirements and environments.

5. System Design

5.1 Overview

This chapter describes the Functional viewpoint, which is currently the main focus of LittleRedButton analysis process.

5.2 Architecture

Typical native mobile application logic will be applied to the architecture of this project. The basic architecture of a typical Android application is displayed below:

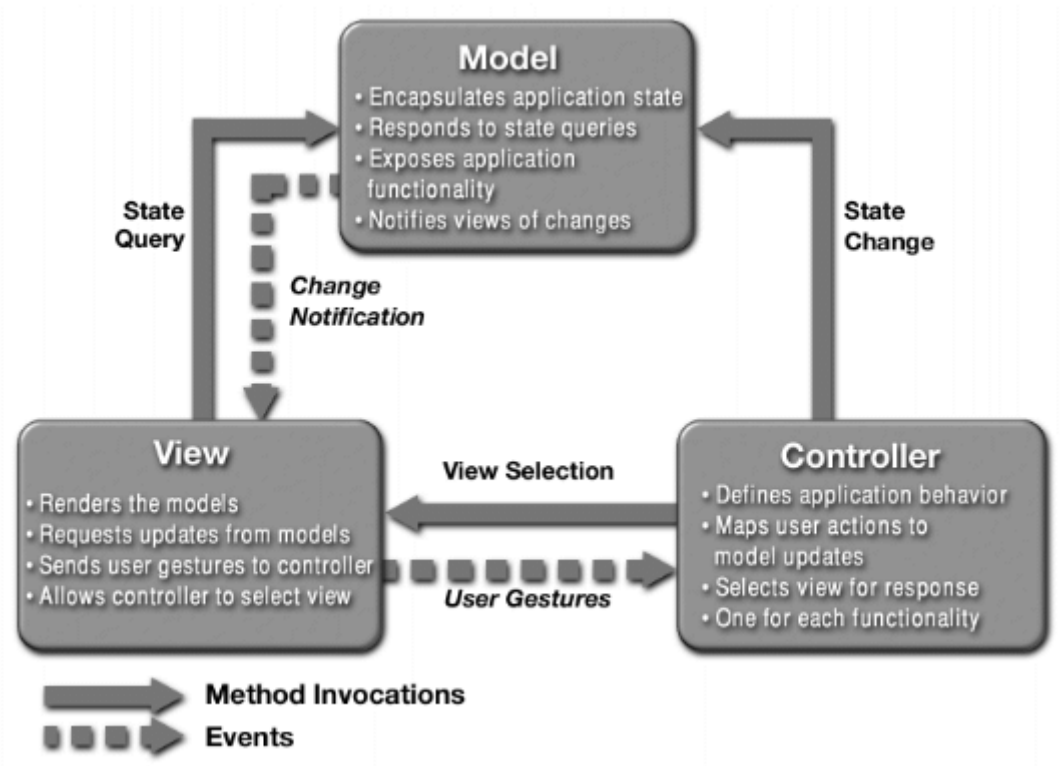


Figure 1: A typical collaboration of the MVC components

In this architecture, there are 3 roles which are model, view and controller. The actions, views and activities in Android are the baked in way of working with the Android UI and are an implementation of a model-view-view model pattern, which is structurally similar (in the same family as) model-view-controller. Android "wants to" follow a MVC pattern, but view & controller are generally really coupled in activities. In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them.

- A **controller** can send commands to the model to update the model's state. It can also send commands to its associated view to change the view's presentation of the model. (**Services** for Android. These are background components that behave like UNIX daemons and Windows services. They run invisibly and perform ongoing unattended processing.)
- A **model** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. In some cases an MVC implementation might instead be "passive," so that other components must poll the model for updates rather than being notified. (**Content Providers** in Android. Data Managers that are the recommended form of inter-application data sharing. For Android, Sqlite3 database is appropriate to keep the model data and retrieve the data from the sqlite3 database if necessary for the Android Application.)
- A **view** requests information from the model that it needs for generating an output representation to the user. (**Activities** for Android. This is the application's primary user interface component. Every individual screen of an Android application is derived from the Activity Java class. They are containers for Views.)

This pattern **decouples** changes to how data are manipulated from how they are displayed or stored, while unifying the code in each component. In other words, it's a way of developing apps keeping the data (model) used in the program, and the visual (view) component of the program separate from one another, each interacting only with a controller containing the logic of our program. The view and the model interact only with the controller never with each other.

In a simple application, the Controller can affect changes to the Model based on user input and also communicate those changes to the View so that the UI can be updated. In real-world applications, however, the View will normally also need to update to reflect additional changes to the underlying Model. This is necessary because changing one aspect of the Model may cause it to update other dependent Model state. This requires Model code to inform the View layer when state changes happen. The Model code, however, cannot statically bind and call the View code.

Once the model, view and controller objects are instantiated, the following occurs:

1. The view registers as a listener on the model. Any changes to the underlying data of the model immediately result in a broadcast change notification, which the view receives.
2. The controller is bound to the view. This typically means that any user actions that are performed on the view will invoke a registered listener method in the controller class.
3. The controller is given a reference to the underlying model.

Once a user interacts with the view, the following actions occur:

1. The view recognizes that a GUI action — for example, pushing a button or dragging a scroll bar — has occurred, using a listener method that is registered to be called when such an action occurs.
2. The view calls the appropriate method on the controller.
3. The controller accesses the model, possibly updating it in a way appropriate to the user's action.
4. If the model has been altered, it notifies interested listeners, such as the view, of the change. In some architectures, the controller may also be responsible for updating the view. This is common in Java technology-based enterprise applications. [3]

5.3 System Components

As mentioned earlier, the LittleRedButton has basic native mobile application MVC architecture. In this section, each component shall be examined in detail.

The native mobile application characteristic describes the relationship of cooperating frameworks and services in mobile operating system. The framework and service components provide necessary context for mobile applications to process data. Location Services, GReadServices for Google Maps and SMSManager services are a few examples of services to be used in LittleRedButton application.

5.3.1 Business Logic

The business logic of the application is the code flow in the project after receiving the request until returning the response. The pattern we stick to in this project is Model–View–Controller (MVC) which will be covered in detail in the latter sections. According to this pattern, we separate the application into layers and handle each operation in its own section [4].

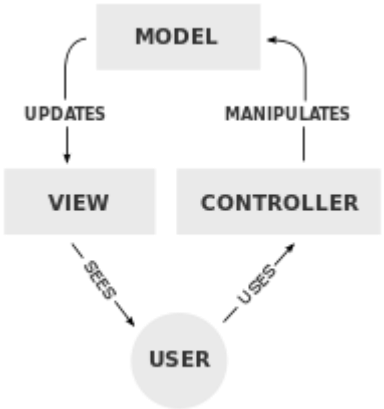


Figure 2: Business Logic in MVC

5.3.1.1 Model

For the model layer of the LittleRedButton application, Recipient object shall be used as a data object. Besides that, in this layer LittleRedButton also have objects like MySQLHelper object to access Sqlite3 database which has the responsibility of access to Sqlite3 database and to operate on it.

5.3.1.2 View

In the view layer, LittleRedButton shall have activity layouts for Android and the necessary code to collaborate this layer to the model and controller layer.

5.3.1.3 Controller

This layer has the business logic in itself. It takes the input parameters from the view layer, uses model layer outputs to construct a result and returns it to the view layer again. So it plays a role of binding the layers and carrying out the controlling and data flowing functions of the application.

5.3.2 Facebook SDK

The Facebook SDK for Android is the easiest way to integrate your Android app with Facebook's platform. The SDK provides support for Login with Facebook authentication, reading and writing to Facebook APIs and support for UI elements such as pickers and dialogs. The Facebook SDK uses Facebook's native app to provide support for authentication when it's present. If you choose not to install this APK to your emulator, you can still develop with the Facebook SDK. In this case your app's Facebook authentication will fall back to taking place via a web view, rather than with the smoother app switching flow. LittleRedButton uses Facebook SDK for Android in order to contact with Facebook servers.

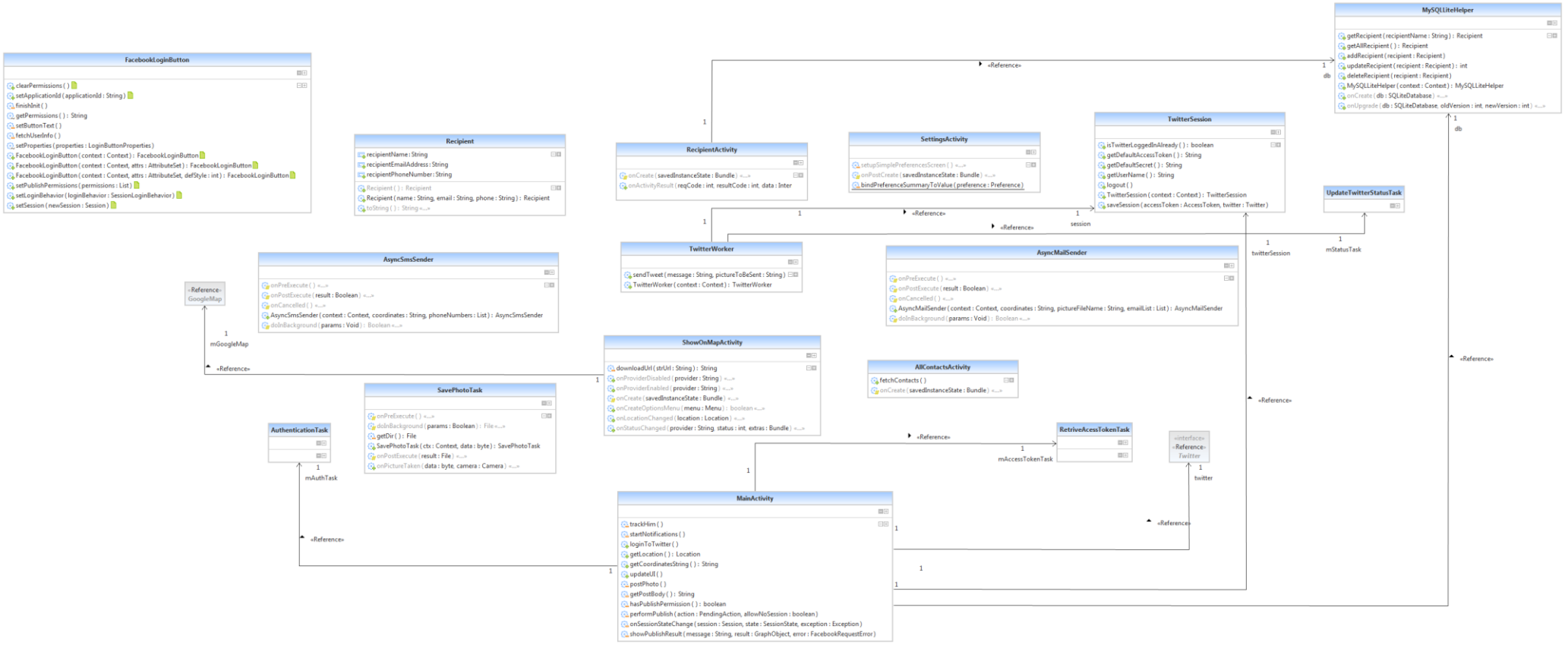
5.3.3 Twitter4j

With Twitter4j, you can easily integrate your Java application with the Twitter services. Twitter4j includes software from JSON.org to parse JSON response from the Twitter API. Twitter4j is featuring:

- 100% Pure Java - works on any Java Platform version 5 or later
- Android platform and Google App Engine ready
- Zero dependency : No additional jars required
- Built-in OAuth support
- Out-of-the-box gzip support
- 100% Twitter API 1.1 compatible

6. Detailed Design

The current structure of the project is displayed below in UML Class diagram representation.



6.1 Data Structures and Methods

LittleRedButton will store the recipients list that the user added by selecting the recipient from his/her phone contacts. **Recipient [name: String, email address: String, phone number: String]** entity will be stored in sqlite3 database. The recipient information will be retrieved from these data structures.

6.1.1 Recipient

Recipient class has attributes of the contact selected among the phone memory. Only the name, phone number and e-mail addresses of contacts which will be used by the application to reach recipients shall be kept.

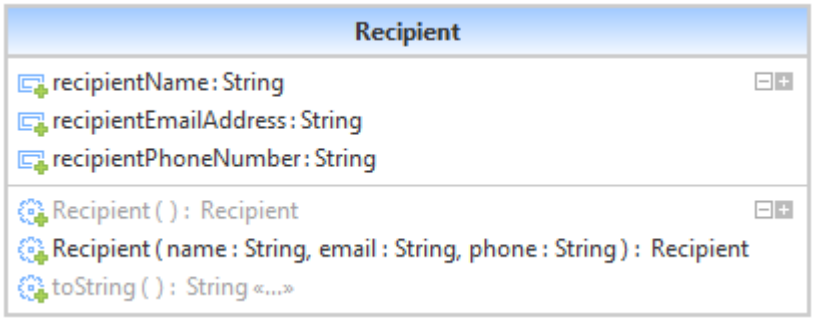


Figure 3: Recipient class

6.1.2 FacebookLoginButton

FacebookLoginButton class is derived from Facebook SDK's Button widget class. Facebook Login allows you to obtain a token to access Facebook's API on behalf of someone using LittleRedButton. One can use this feature in place of building his/her own account system or to add Facebook services to existing accounts.
Facebook Login allows you to obtain a token to access Facebook's API on behalf of someone using your app. You can use this feature in place of building your own account system or to add Facebook services to your existing accounts. The Facebook SDK for Android provides methods to implement Facebook Login for your app. The SDK provides Facebook Login support for the these common scenarios [6]:

- Your app uses only Facebook Login to authorize people using your app.
- Your app provides two login options: Facebook Login and your own login mechanism.
- Your app uses your own login initially and Facebook Login is an option to switch on certain social features.

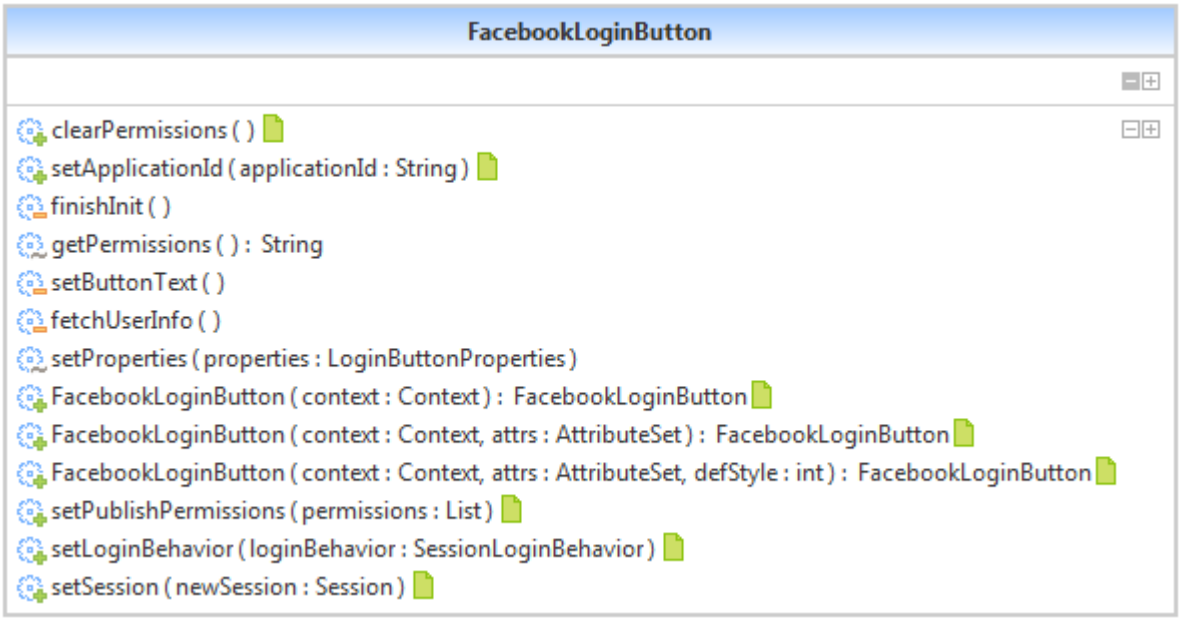


Figure 4: FacebookLoginButton class

6.1.3 RecipientActivity

RecipientActivity class is composed of adding/deleting and displaying the recipients list of the application. It has a "Add Recipient", "Explore All Contacts" and "OK" button. Besides the list of recipients there would be a "Delete" button for each recipient.

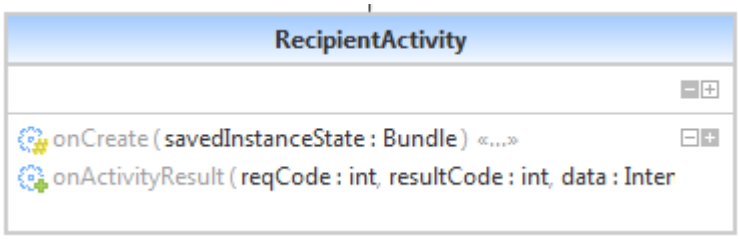


Figure 5: RecipientActivity class

6.1.4 SettingsActivity

SettingsActivity class shall be used in order to configure application settings which are "Message Text" a plain text field to represent message that would be added to emergency message, "Display Name" a plain text field which shall be used to identify the user on outgoing messages, "Gmail Settings" which shall be used if the user wants to use his/her own Gmail account when sending e-mail to recipients and "Track Me" option which is used to enable/disable LittleRedButton's tracking by updating notification channels every 60 seconds.

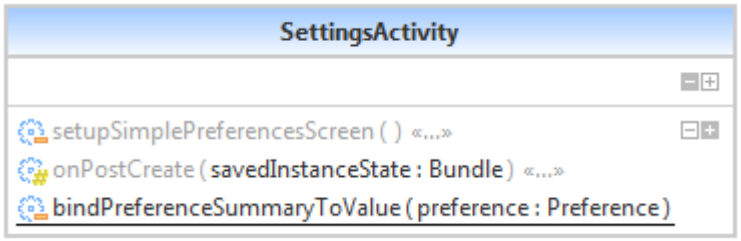


Figure 6: SettingsActivity class

6.1.5 MySQLLiteHelper

MySQLLiteHelper class shall be used to connect sqlite3 database and perform add/update/delete and fetch operations that would be made by the user on recipients list database.

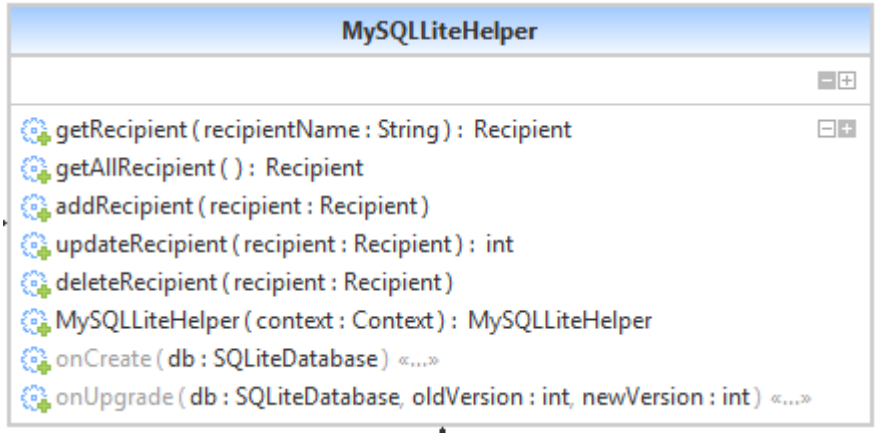


Figure 7: MySQLLiteHelper class

6.1.6 TwitterSession

TwitterSession class is used when the user has logged into his/her Twitter account by using LittleRedButton. All credentials provided by Twitter4j API shall be stored in TwitterSession class. When the user logs out from his/her Twitter account TwitterSession class attributes shall be cleared by the application.

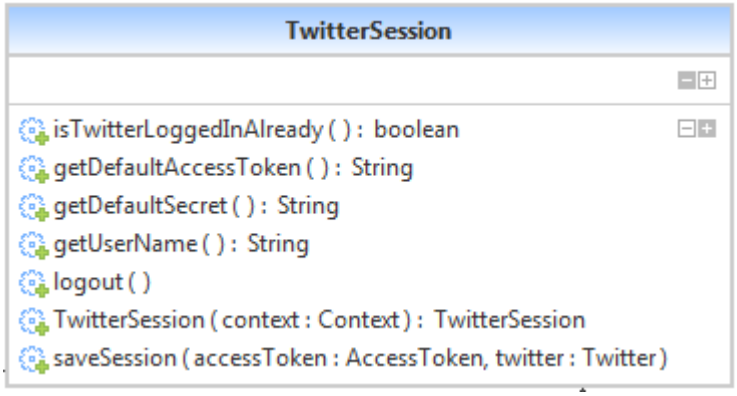


Figure 8: TwitterSession class

6.1.7 AsyncSmsSender

For each recipient that takes place in the recipients list database, AsyncSmsSender class sends a well-formed SMS message containing the emergency message asynchronously. This class is derived from AsyncTask of Android.

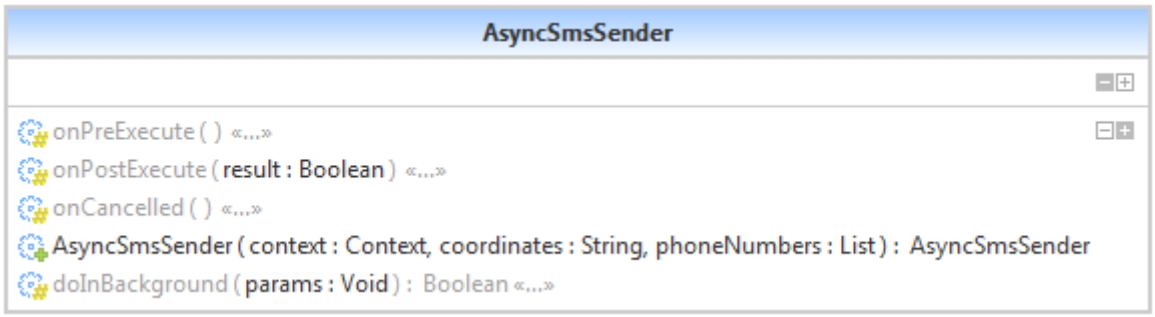


Figure 9: AsyncSmsSender class

6.1.8 TwitterWorker

TwitterWorker class shall be used to update user's Twitter status in collaboration with the TwitterSession and UpdateTwitterStatusTask classes. An emergency message with a picture of the user's current scene shall be posted to Twitter via Twitter4j API by using this class functions.

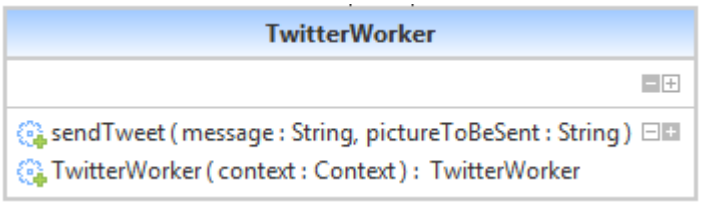
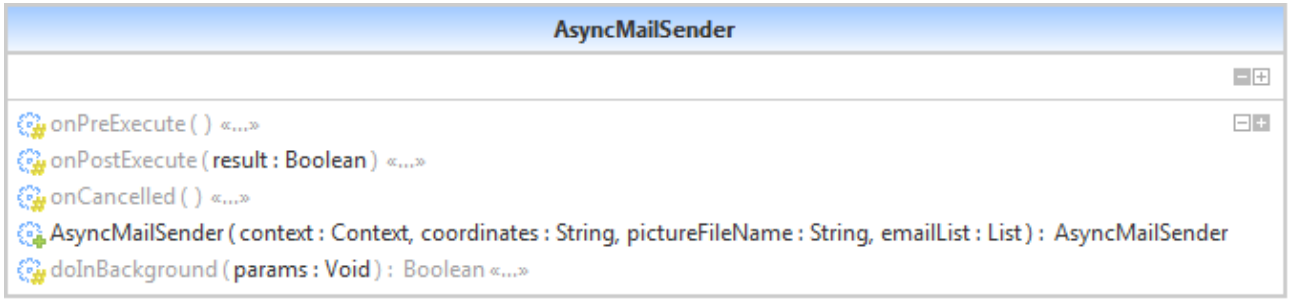


Figure 10: TwitterWorker class

6.1.9 AsyncMailSender

AsyncMailSender class is used to send the current GPS location coordinates with a picture of the user's current scene to the recipients that the user previously selected as an e-mail. If the user sets his/her own Gmail account credentials formerly e-mail shall be sent from his/her account. If user lefts Gmail account settings empty, application shall use its default e-mail configuration to send e-mails. If the user sets the display name through application settings then the e-mails being sent shall be displayed from that name in recipient's inbox. This class is derived from AsyncTask of Android.



,

Figure 11: AsyncMailSender class

6.1.10 UpdateTwitterStatusTask

UpdateTwitterStatusTask shall be used to update user's Twitter status with the given message and the picture asynchronously using twitter4j library functionalities.

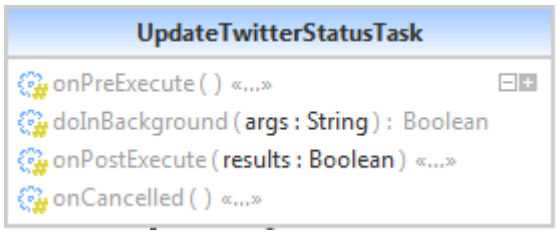


Figure 12: UpdateTwitterStatusTask class

6.1.11 SavePhotoTask

SavePhotoTask is an asynchronous task that is used to save the picture which is shot by the application on user's emergency declaration. Listening to onPictureTaken event, when the picture data is produced, the data shall be written to external SDCard of the phone. The picture file path shall be used by other functions of the application in order to attach that file to notification channels.

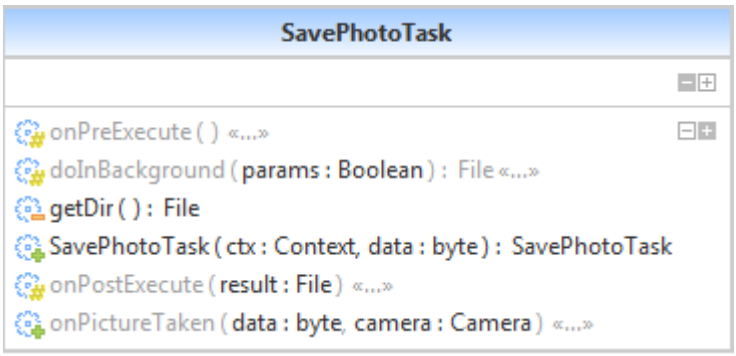


Figure 13: SavePhotoTask class

6.1.12 AuthenticationTask

AuthenticationTask class shall be used to authenticate the user to Twitter using twitter4j libraries. Login with Twitter account in application involves the following process:

- First user shall be asked to authenticate into application using his/her twitter account.
- Once user successfully logged into his twitter account application shall receive his OAuth token and OAuth secret.
- OAuth token and OAuth secret shall be stored in application shared preferences.
- Whenever application needs OAuth tokens, it shall read from shared preferences.
- Once user clicks “Disable Twitter” button application will delete OAuth token & secret from shared preferences.

AuthenticationTask class is an asynchronous class which shall be used to get OAuth request token from Twitter servers by using twitter4j libraries.

OAuth is an open standard for authorization. OAuth provides client applications a 'secure delegated access' to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner, or end-user. The client then uses the access token to access the protected resources hosted by the resource server. OAuth is commonly used as a way for web surfers to log into third party web sites using their Google, Facebook or Twitter passwords, without worrying about their access credentials being compromised. [5] For LittleRedButton case, the application shall use OAuth 2.0. OAuth 2.0 is the next evolution of the OAuth protocol and is not backwards compatible with OAuth 1.0. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. Facebook's new Graph API only supports OAuth 2.0. Google supports OAuth 2.0 as the recommended authentication mechanism for all of its APIs [5].

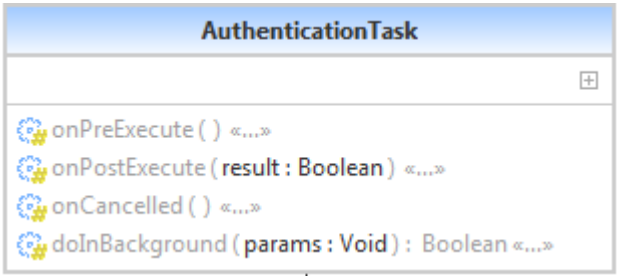


Figure 14: AuthenticationTask class

6.1.13 ShowOnMapActivity

ShowOnMapActivity is an activity that is used to display user’s current GPS location coordinates and the nearest police and health points to that coordinates on Google Maps. LittleRedButton makes use of Google services like Google Map Android API V2 and Google Play Services. Current location of the user shall be marked as green, police stations with blue and hospitals with red markers on Google Maps.

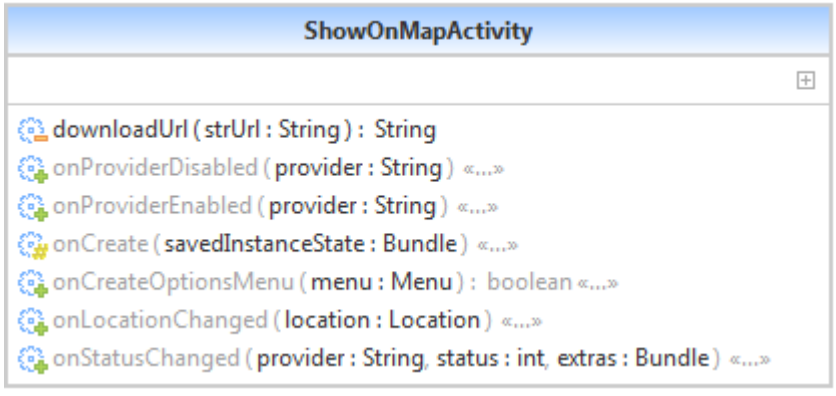


Figure 15: ShowOnMapActivity class

6.1.14 AllContactsActivity

AllContactsActivity class is an Android activity on which all contacts that take place on the phone memory shall be displayed. Some information but not all about a contact shall be displayed on the screen. As LittleRedButton concerns about a contact's name, phone number and e-mail address only those data shall be displayed on AllContactsActivity in order to let the user explore his contacts and their information.

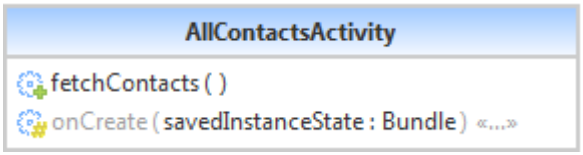


Figure 16: AllContactsActivity class

6.1.15 RetriveAccessTokenTask

RetriveAccessTokenTask class is an asynchronous task of Android which is used to retrieve access token from Twitter servers once the user has performed his login to Twitter via application. After AuthenticationTask is executed, application asks Twitter servers for access tokens to be used during latter calls to Twitter.

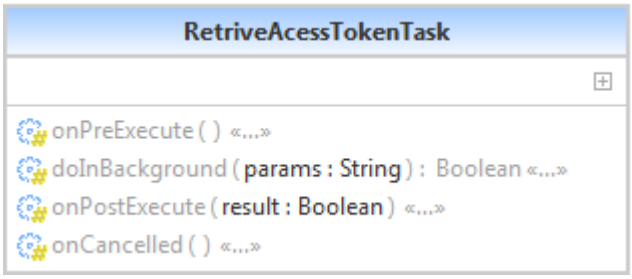


Figure 17: RetriveAccessTokenTask class

6.1.16 MainActivity

As understood from its name, MainActivity class is the main screen of LittleRedButton. All critical operations of the application are performed in this class. Login/Logout operation to/from Facebook and Twitter, grabbing user's location, directing other activities, updating UI components and presenting the emergency declaration are the most vital components of the application that MainActivity class deals with.

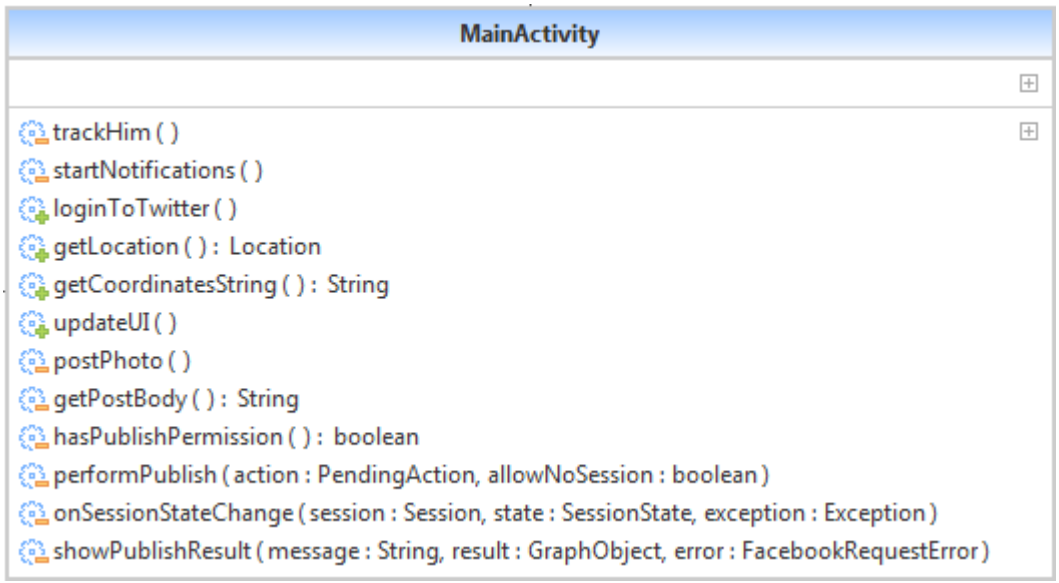


Figure 18: MainActivity class

6.2 UML Activity Diagrams

6.2.1 Facebook Activity Diagram

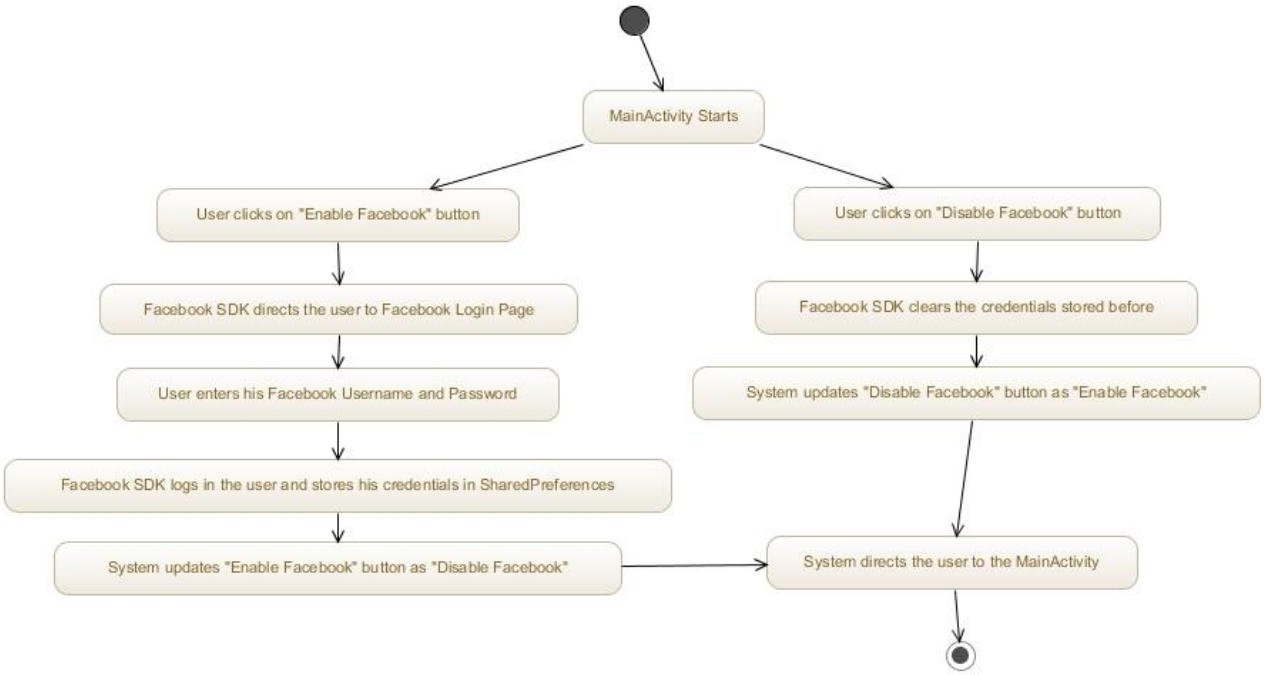


Figure 19: Facebook Activity Diagram

6.2.2 Twitter Activity Diagram

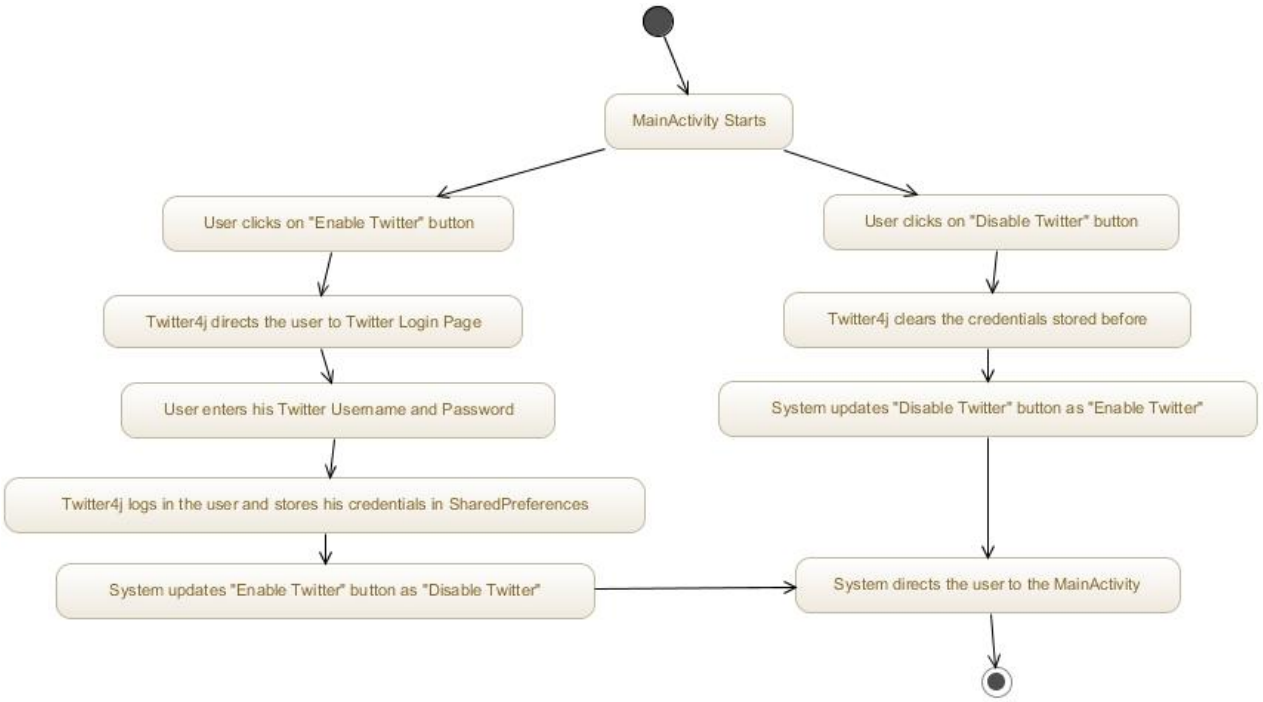


Figure 20: Twitter Activity Diagram

6.2.3 Recipients Activity Diagram

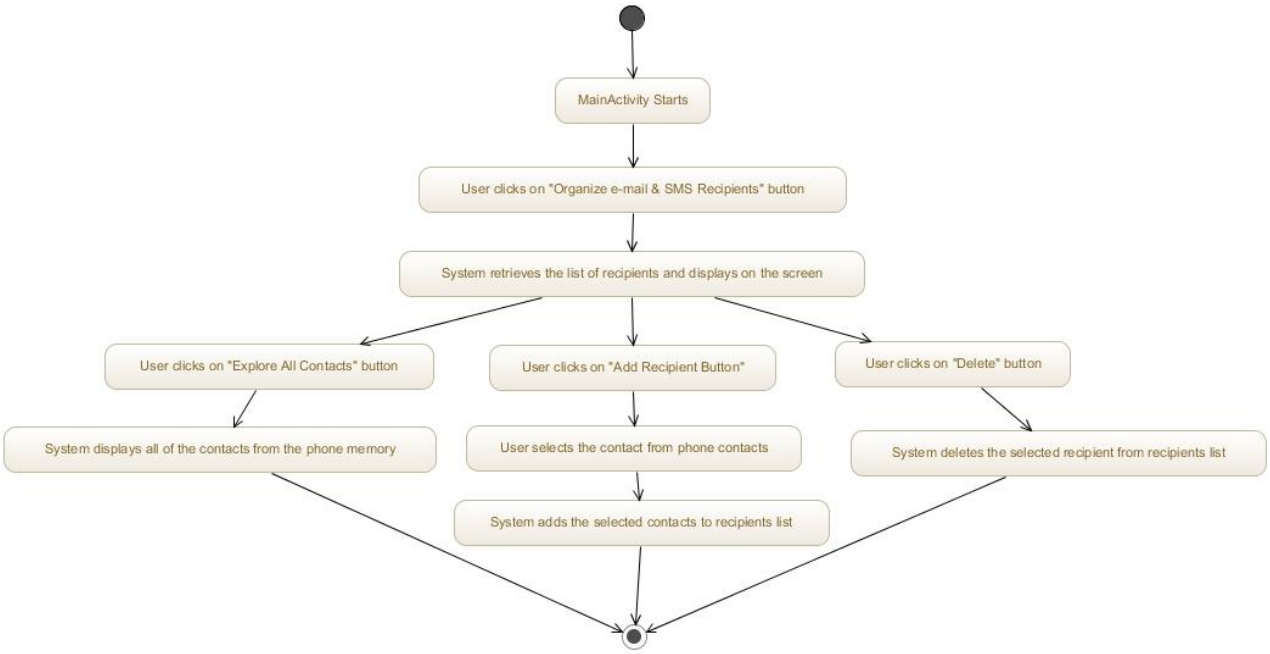


Figure 21: Recipients Activity Diagram

6.2.4 Display Location on Maps Activity Diagram

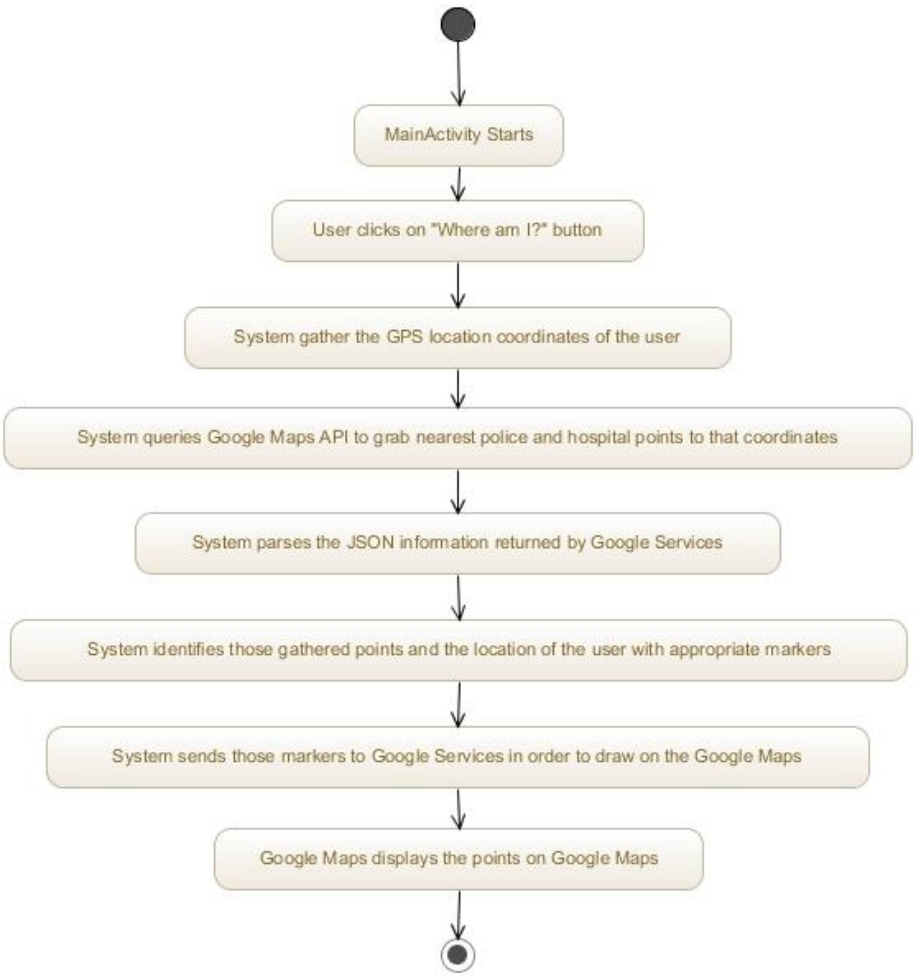


Figure 22: Display Location on Maps Activity Diagram

6.2.5 Get Help Activity Diagram

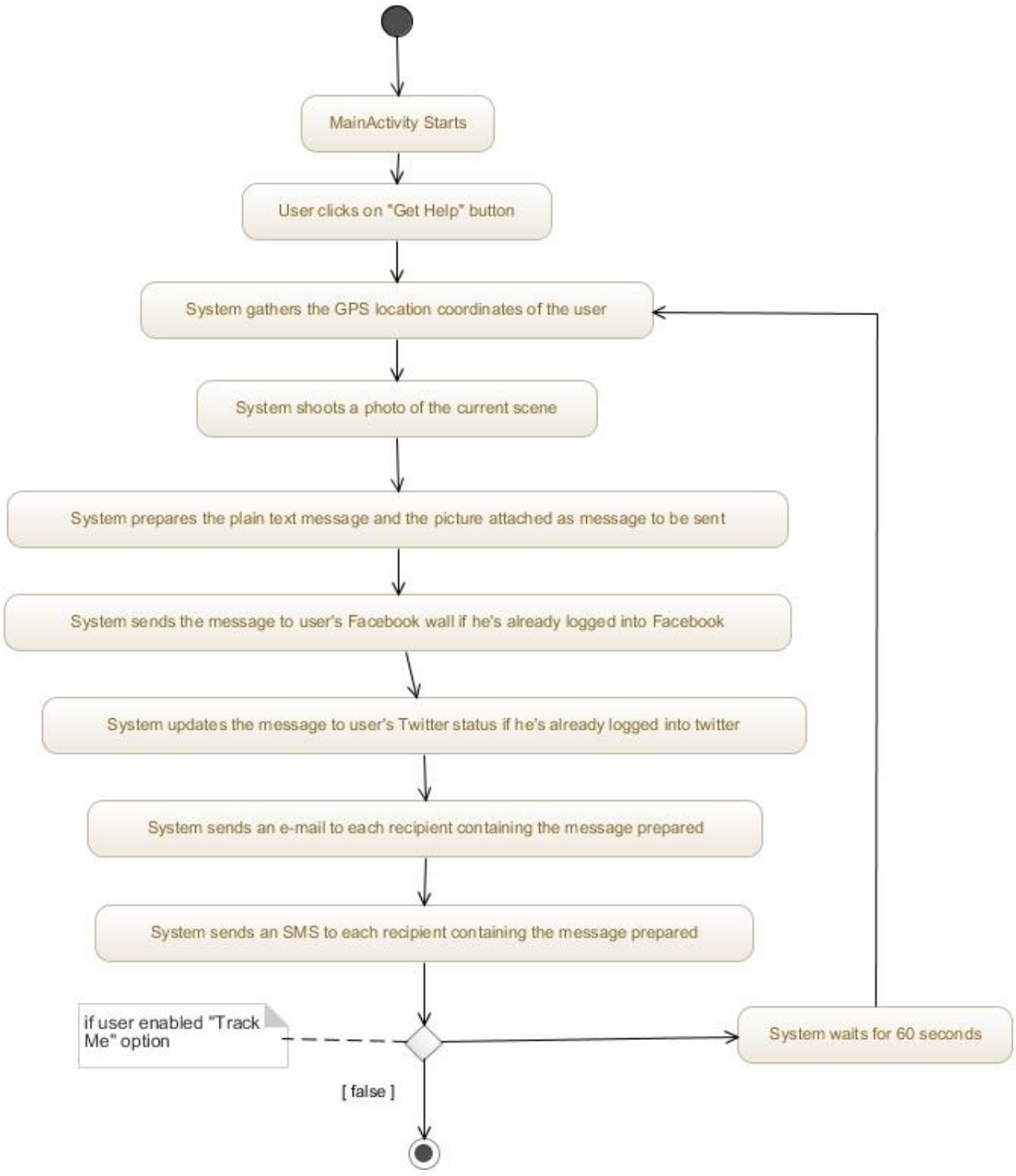


Figure 23: Get Help Activity Diagram

7. Development View

7.1 Overview

For the development viewpoint the module organization is explained in detail which shows the dependencies between modules. In this section, how revision control of source code files in a centralized repository and aspects of the development process to enhance the quality of the software will be highlighted.

7.2 Configuration Management

An important part of modern software development is the Software Configuration Management. Configuration Management is often considered to have originated in the frame of software development in recent years and that it merely consists of a tool for versioning of source files. In software development the core of configuration management is made up of a version control system. Source code (or all source documents of the final product, respectively) is considered a system that spans time and space. Files and directories (which, of course, can contain files) form the space, while their evolution during development forms time. A version control system serves the purpose of moving through this space. Note, that such time and space is not continuous, but rather discrete: Every revision corresponds to a snapshot of a document, which represents a certain state in its history of development. Version control systems are numerous with Bell Labs' SCCS (Source Code Control System) from 1972 being the first one. Since then a multitude of other systems have been designed and used: CSSC, Revision Control System (RCS), Concurrent Versions System (CVS), Revision Control Engine (RCE), Subversion, BitKeeper, ClearCase, Forte Code Management Software, Serena Professional (originally PVCS), Perforce and Visual SourceSafe. Configuration management contains version control, but in addition to this provides further methods of project management. Locating configuration management between formal and complex procedures with rigid mechanisms, and collections of ad-hoc script based on simple version control is a subjective matter. Software configuration consists of software configuration items (SCI) which can be organized in three categories: computer programs, internal and external documentation to these, and data. A baseline is a version of a SCI on which developers and management or customers have agreed upon. A modification of this requires a formal change request. A release is a version which is provided to users, whereas a promotion is to be provided to fellow developers. Several problems for software configuration management can be derived from this:

Version control, i.e. filing of different versions of the same SCI Temporal variation, i.e. allowing modification to occur in parallel, which can be discriminated into two types:

1. Small and bounded activities like fixing of bugs or implementation of a new feature and
2. More complex activities like creating a new program branch, which is developed largely independently of the main development branch (trunk).

Permanent variation, i.e. ability to create different products out of the same sources. These products are also called variants. Build support, i.e. the build process (that is, building of the final software product from its sources) should be supported by configuration management. This is often accomplished through dedicated build tools like Make, Dmake, Imake, Odin, Ant or NAnt, although integrated tools like DSEE, ClearCase or Jason exist, too. Distributed development, i.e. access to the databases of the configuration management should be possible from all over the world. Necessary coordination tasks must also be supported. The Institute of Electrical and Electronics Engineers (IEEE) sees configuration management as a discipline that uses observation and control on a technical as well as on an administrative level. Aims to achieve are:

1. Identification and documentation of functional and physical characteristics of a SCI,
2. Control of changes to these characteristics,
3. Recording and reporting of change processes and implementation states,
4. And auditing of specific requirements.

This results in the following activities that constitute configuration management:

1. Identification of SCIs and their versions through unique identifiers
2. Control of changes through developers, management or a control instance,
3. Accounting of the state of individual components,
4. Auditing of selected versions (which are scheduled for a release) by a quality control team.

Software configuration management deals with the governing of complex software systems.

For source code management we have set up a Subversion repository where all artifacts of the development process will be stored and organized. Subversion, like many other version control systems, manages different versions of documents

by calculating editing operations between these and then only saving these operations. This usually leads to very good compression results. We use subversion because it is a modern revision control system. It has been designed to resolve many of the short-comings of widely-used CVS, while still being free software. Subversion attempts to fix most of the noticeable flaws of the Concurrent Versions System (CVS) and is very powerful, usable and flexible. New to Subversion is the way how it internally treats directories and documents: directories (which are also documents) only contain links to specific revisions of other documents. Thus, when a source file is copied, Subversion will not copy all the data, but rather create a new directory entry referencing the same object. Hence, it is possible to copy the entire development branch at very low cost.

Access to the Subversion code repository is provided through secure SSL connections using our web server. Thus, this repository allows the geographically dispersed group of consortium members to collaborate and share their source code. It tracks changes to source code and allows a versioning of source code files. Additionally, Subversion remembers every change ever made to the files and directories so that earlier versions can be recovered in case that something was accidentally deleted.

7.3 Testing

The testing process covers verification and validation of correct operation and boundary conditions of each requirement given in Gurkas-MustafaGoksu-LittleRedButton-RSD-2014-03-09-Rev-1.0.pdf.

During the development, continuous integration will be handled to avoid any errors and integration failures. This way, any developed code will be tested automatically as soon as they are committed to the code repository. By the help of extra plug-ins, we will also be having the static code analysis checks to keep the code clean in a daily basis.

Quality assurance specifically deals with the management of quality. In this term “assurance” means that under the condition that the processes are followed, the quality management can be assured regarding product quality. Furthermore, quality assurance strives to encourage quality attitudes and discipline among the developers and the management. Quality assurance encompasses software configuration management, quality control and software testing. Software testing can be used to verify the implementation of functional requirements and hence is a popular strategy of risk management. To some extent software testing can also be used to find errors, however, testing cannot prove the absence of errors in software as testing is only as good as its test cases. Furthermore, by the time that testing occurs, it is too late to build quality into the product.

According to the IEEE the test plan is part of the test documentation. It prescribes the scope, approach, resources and schedule of testing activities. Thus it specifies which items and features are to be tested and how, who is responsible for testing and what associated risks are. In this section we want to stress that the actual testing should particularly focus on things that are important to the different stakeholders of the software. Hence the initial scenarios already provide highly useful information on the preferable focus of testing activities.

7.4 Continuous Integration

In order to ensure that changes inside software modules do not affect related modules we will create unit tests which can be executed in order to check that certain functionality is performed correctly. These test cases can be run in an automated fashion so that the developer can easily validate the functional integrity before committing his changes to the central Subversion repository.

Certain agile methodologies like Extreme Programming (XP) propose writing test cases even before implementing the actual code. We leave this up to the developer because in a research project we are sometimes forced to implement functionality that is not well understood. In this case writing unit tests before the code does not make sense. In prototyping situations the developer needs more freedom in trying out several options before the final design decision. In these situations the developer has to add test cases after he made the decision.

Continuous Integration is also a term that has its origin in the concepts of XP. In this community it is common practice to immediately commit every change to the revision control system, no matter how small it is. The rationale behind this is that other developers should always work with the latest version of the code base. Furthermore, Continuous Integration describes a process of executing all test cases in a project in an automated and frequently repeated way. Failures in test cases will be automatically detected and reported to the persons that committed the changed source code files since the last successful test run.

What Continuous Integration is and how it works can be shown by giving a quick example of the process of implementing a small feature, which requires only some few hours of work. Continuous Integration increases the awareness of problems by involved developers by reporting back problems in a very short time frame after these

problems occurred. The error resolving takes less time and efforts compared to failure detection in later stages of the development. The first step is to take a copy of the current integrated source onto the local development machine. This is done by checking out a working copy from the mainline (or trunk) from the revision control system. This local copy can now be modified so that it implements the required feature. In the Continuous Integration paradigm, implementing does not only mean to make changes to the actual code, but it is also required to add unit tests for automated testing of the feature. This is sometimes called self-testing code. Note, that changing the local copy implies the later modification of production code. The developer, who changes the system, makes every effort to ensure that her feature does not introduce a bug: Once she is done (and usually at various points while she is working) she carries out an automated build on her development machine. This takes the source code she is working with, compiles and links it into an executable, and runs automated tests. After that the developer performs manual tests. Only if everything builds and tests without errors is the overall build considered to be good. With a good build, the developer can then think about committing her changes into the repository.

The twist, of course, is that other people may, and usually have made changes to the mainline in the meantime. So first the developer updates her working copy with the changes from the other developers. If the other developers' changes clash with hers, it will (most probably) manifest as a failure either in the compilation or in the tests. In such a case it is the developer's responsibility to fix this and repeat the above steps until her working copy is a good build that is properly synchronized with the mainline.

Once this state is reached, the developer can finally commit her changes into the mainline by committing her changes into the repository. However this commit does not finish her work. At this point, another build is required; but this time on an integration machine based on the mainline code. Only when this build succeeds it is said that the changes are done. The reason for this is that there is always a chance of missing something on the local machine or that the repository was not properly updated. Only when the committed changes build successfully on the integration machine the job is done. This integration build could be executed manually by a person, or be done automatically by a Continuous Integration tool.

If a clash occurs between two developers, it is usually caught when the second developer builds her updated working copy. If not the integration build should fail. Either way the error is detected rapidly. At this point the most important task is to fix it, and get the build working properly again. In a Continuous Integration environment you should never have a failed integration build stay failed for long. A good team should have many correct builds a day. Bad builds do occur from time to time, but should be quickly fixed.

The result of doing this is that there is a stable piece of software that works properly and contains few bugs. Everybody develops off that shared stable base and never gets so far away from that base that it takes very long to integrate back with it. Less time is spent trying to find bugs because they show up quickly. The advantages of Continuous Integration can be summarized as follows:

- When unit tests fail, or a bug is discovered, developers might revert the code base back to a bug-free state, without wasting time debugging.
- Integration problems are detected and fixed continuously - no last minute hiatus before release dates;
- Early warning of broken or incompatible code;
- Early warning of conflicting changes;
- Immediate unit testing of all changes;
- Constant availability of a "current" build for testing, demo, or release purposes;
- The immediate impact of checking in incomplete or broken code acts as an incentive to developers to learn to work more incrementally with shorter feedback cycles.

All kinds of scheduled builds, if done frequently enough, would fall under Continuous Integration. However, nightly builds are too infrequent to qualify. Notable examples of Continuous Integration software include:

- CruiseControl,
- Borland Gauntlet,
- AnthillPro,
- TeamCity,
- Parabuild,
- Pulse.

7.5 Code Inspection

Now there's no guarantee that you get all the integration bugs. The technique relies on testing, and as we all know testing does not prove the absence of errors. The key point is that Continuous Integration catches enough bugs to be worth the cost. The achievements of open source software show that is possible to collaboratively create software of high quality without a highly paid management. In this case the programmers are motivated intrinsically; there is little extrinsic reward for their efforts. Their rewards are commendation, feelings of having done something for the community being proud of "their" software. In commercial software development, there is extrinsic reward (primarily in the form of payments made by the employer), intrinsic motivation is not always given. Therefore creating high quality software in itself might not be the primary goal of developers, because they get paid for reaching milestones. Quality, however, is almost inescapably not a milestone, because it depends (at least partially) on the people getting in contact with it: developers as well as maintainers and users. It is cardinally a soft attribute based on the perception of humans. As a result, it is basically impossible to measure quality as a whole automatically. It would be best to create intrinsic motivation in developers, but how to achieve this – even if management is committed – is an open question. Traditionally this is countered by establishing rigid control mechanisms, for example the ISO-9000 standards or coding rules. In our opinion ISO-9000 is far too rigid for a research project like LittleRedButton. But a set of coding rules, which forbid or promote specific behavior, has been developed and published as a live document in the project Wiki. We are steadily improving these to prevent bad code quality. But even sensible coding standards tend to be overseen or ignored when facing time pressure. Moreover there is no direct feedback from the rules themselves, what would be a precondition for learning. Because of that, our coding standards will be accompanied by software metrics, which automatically verify adherence to coding rules. For this we will rely on the open source CheckStyle library, whose tests range from trivial file size or formatting checks to hidden field detection and magic number tests. However, metrics work on the lowest level of code far from the semantic level. As pointed out before, measuring quality is a hard task, because all-embracing hard criteria for measuring (code) quality automatically cannot be set due to the soft nature of quality.

Appendices

Issues in deployment on mobiles:

While serving all the client browsers from desktop software to an application working on a mobile device, there are a number of issues that crop up which need to be looked into in detail. Some of these issues that are faced are due to different architectures of different devices. A detailed analysis should be made to support all the resolutions for device browsers while coding according to them.