Mobile Voting for Android - GUI design and usability

Radovan Murin

May 6, 2011

1 Purpose

The purpose of this document is to describe the design, implementation and live testing of the Graphics User Interface in the Mobile Voting for Android application created as the mobile part of the Voting system. The document is to be used when designing future versions, adding new functionalities or porting to a new platform. It should give the reader a full understanding of the principles of the GUI. The document does not provide a full understanding of the inner workings of the application nor does it explain the reasoning behind some rules to be followed.

2 Introduction

2.1 Basics

The purpose of the application is to provide a the voting functionality to the system. The application in summary connects to a voting point - server, displays the questions and provides a mean to answer them. A simple principle, but this makes it only harder to design. The basic needs of the GUI is to:

- Provide the user a means to login to the application - not the voting points/servers.
- Provide the user a means to connect to a server, save server details.
- Provide the user a meant to see question text, details and discreetly vote.
- Be extensible and be able to accommodate future changes with minimal effort.
- Be simple, intuitive and minimalistic.
- Be consistent with the operating system's "Look & Feel".

2.2 Design principles

The design principles are to be inspired the ”User Interface Guidelines” for Android applications[1]. They will be a little bit tweaked for our use, because as a voting system, it will have to take into account some safety precautions. Predominantly vote confidentiality and password unreadability.

3 Analysis of the GUI

This section discusses the basic need of the user and how abstract ideas how the GUI should cope with them.
When looking at applications for devices using touchscreens and their review, one observation can be made. The more simplistic and straightforward the application as a whole is, the more popular it is among the user. People do now want over complicated features and often are willing to sacrifice usability for simplicity. Swiftness of the GUI is another trait that is often neglected in the design. One example that comes to mind and has these traits is a Czech made application called ”Pubtran” from Frantisek Hejl [2]. One idea, simple interface and undisruptive helper features made this, in my opinions one of the best applications for Android on the Market.
Our design should take this ideas as a kind of inspiration and build itself on the above principles.

3.1 What the users will need

As a voting application the user will obviously need a way to connect, vote and see the results of his actions. These are basic functionalities that are given by the nature of the application and make the application usable. Next the users will need a communication channel between them and the application. Whenever an error occurs,a vote is sent or virtually anything happens withing the application that affects the user the application must have a way to communicate with the user and give him the information necessary for him to make decisions for how to continue. Last but not least the application has to obey some safety rules and has to sacrifice some user convenience for this. It has to be protected by a password to prevent unauthorized use of the information stored withing the application. GUI properties As stated beforehand, the features are often not the reason for an application's success. Rather it is the feeling of fluidity and ease of use that persuades users to stay with the application and not search for a better designed alternative. For this reason the design of the GUI will have to take in mind that less is often more and the most typical flow of actions and accommodate the layout accordingly. Another property is the jargon of the application.We have to assume that the user has no technical knowledge and try not to use too much technical terms. The question at hand is, what does ”too much” mean? What is the bare minimum of technical knowledge the user has to know to use this application? How to offer assistance to a technically less skilled person? These answers will have to be found out during testing as it is impossible to predict how the users will react. Last, but not least, is the consistency with the operating system. Every operating system has it's philosophy, it's looks and feel. The design should take this into account and place functions where they would be normally found should the application be a base system application - like the setting s application. This will minimize the need of the end user to use manuals as he will find the interface intuitive.

3.2 Hardware demands

Considering that the application should run on a wide range of devices, with various processing power and RAM, it would be best not to make the interface too decorated as this would deteriorate the user experience on slower devices. It is therefore decided that the interface will be plain and simple, with no "eye candy".

4 Design

4.1 Terminology

For a reader that is unfamiliar in the matter of Android user interface programming I give a brief introduction. The GUI is built out of Activities. An activity is in essence one screen layout. More simply a view the canvas, that has components placed on itself. This is not a technically correct explanation, but it is sufficient for the needs of GUI design. An important thing to mention is that every activity is runnable by itself. This feature is not available to the average user, however a power user is able to run the activities in any order. Another part of the GUI is a View. A view is similar to an activity in that it is a canvas with components but in the vertical hierarchy, the view has to be attached to an activity to be displayed. Also it cannot be run. A menu is a container that will pop up from beneath the screen when the menu button is pressed. The menu contains only buttons and they are mostly used as a supplement to finger gestures. A "toast" is a simple message that flashes on a screen for some amount of time. The rest of the terminology is consistent with Java Swing framework terminology and will not be explained in detail. Also, as this is a technical document voting points are referred to as servers.

4.2 Main layout - Overview

The GUI guidelines state that every separate user act should have it's own activity. This divides our application into four activities. The main login, the server listing, server editing and the question listing. The main login activity will prompt the user for his main password. This is the entry activity. When one would try to run the other activities, they simply crash. Not the best behavior, but it works. The server listing activity will be the "lobby" of the application. As it's name suggests, it displays servers and provides options to the user to alter the server details, delete them and finally connect to them. Next there is will be a server adding/editing activity. This activity will consist of text fields that will contain information for successfully connecting to a server. I will enable the user to either connect to the server, or save it to the persistence. Last is the question list activity. From the name it is clear that this is the center-point where the actual voting takes place. This activity will offer the user a means by which they will select answers and sends them to the server.

4.3 Activities - detailed design

4.3.1 Main activity

The main activity is aimed to be very basic. Provide a place to input the password and confirm. It has no other functional features, anything else would create potential holes into the application. On a special occasion - first run, this it will also provide a password setup screen once launched. The main activity should also provide basic copyright information accessible through the menu.

4.3.2 Server listing activity

Previously, I have described this as the "lobby" of the whole application. And this description is, in my opinion, accurate to the intended use of this activity. It will display servers that are known to the device from one source or another. At the present, there are two sources of server information. The user and a beaconing system at the server side which is discussed in the main text of this thesis. These two sources of information are to be separated by a text informing that the servers that follow after the notice are from a different source, the text will inform what is the source. This applies to any future added data source.The separate server entities shall be displayed as buttons. The buttons will have two different triggers that will make them to perform different tasks. To connect to a server the user should simply click the server. The activity also provides the user a means to edit or delete a server entity. When planning where to locate this feature I thought up of two approaches. A menu approach, that would need the user to click on a button that says "Edit server" and then click on a server to edit it, and a touch approach where the user would communicate the wish to edit a server by long pressing it. In the end, the second approach was chosen because of the less steps the person has to make to achieve the desired effect. Also this "long press for a special function" paradigm is well spread in the world of touch screen devices and its users are accustomed to this. Because of these reasons this principle will be used on all such functions. This menu of this activity contains buttons to add a server, edit the main password and exit the application. The exit is not really needed on this platform but some users have a feeling of not being in control, or are not sure what is the state of the application once returned to the main screen of the platform.

4.3.3 Server editing activity

The server editing activity could be named editing/creating activity as the two are the same for the presentation layer. This activity contains a set of labels and text fields. The labels describe the text fields and the user inputs the needed data. As of now the fields that are needed are:

- Friendly name
- IP address
- Port number
- User name
- Password

The the password field characters are to be obscured by bullets (•) to prevent being seen by a third party.
The menu in this activity contains the options to save or only connect to a server with the given data. There is no other features in this activity. In the future this may be extended to accommodate separate user preferences for one server when the need for such features arises.

4.3.4 Question listing activity

This activity is the real aim of the whole application. In short it enables the person to vote. The activity will display a list of questions represented as buttons. This must be consistent with the way the servers are displayed. Again one touch should belong to the principal action - open a list of questions. The list will be a dialog. It contains the answers and next to an answer there are check-boxes. If the user chooses more of less answers than are allowed, it will need to inform the user and prevent him from saving such information. To enforce the right amount of answers the dialog will not allow him to close itself until the criteria for a valid vote are set. An invalid vote is not allowed. To send the answers/votes, two methods should be employed. The first is a one by one method that can be accessed via long press. And the other is mass answers sending and is accessed via the menu. After long pressing the question a dialog will appear that prompts the user for confirmation. The dialog is necessary to prevent "pocket sending" - unintentional sending of votes via incidental inputs. After the user gives the confirmation, the application will send the vote. Mass sending should also be protected against "pocket-sending". The protection can be the same as in individual sending or the fact that the user will have to push a physical button can be taken as such a measure. After choosing this option, selectors will appear next to the question and the user will then choose the questions he would like to send the answers and then he will send them. Again via the menu to prevent unintentional inputs from sending the answers.

Communication with the user The application sometimes needs to send information to the end user. The GUI has to provide a common system that all the components can use to do this. This application should use toasts as a one way (asynchronous) communication channel and dialogs when needing user interaction - synchronous communication. The toast will appear anytime the system makes some alteration or any action. They present a kind of assurance that the users request has been processed successfully. Of course, if requests are unsuccessful an appropriate toast should also appear. The synchronous information flow between the user and the system is created using dialogs. A dialog is a windows, smaller than the screen that dims the activity being displayed and asks for some input. An example of synchronous information flow is the dialog that asks how the system should continue if it has detected and untrusted certificate being used. This dialog should consist of a straightforward explanation of what has happened, what are the options and the consequences to these options. Careful wording is required and the texts have to be very short so that it user really reads them.

4.4 Summary

The main design guidelines for this application are:

- Every activity has its defined scope - purpose. This scope is not to be exceeded for whatever reason. When creating a new feature the GUI should be altered by adding a new activity.
- The definitions of activities are to be kept in this document, and this document should be updated upon adding features.
- When designing system messages for the user the principle of Occam's razor should be followed[1].
- The buttons should have a unified look and similar behavior.
- To achieve seamlessness between the OS and the application the UI should be efficient and quick.
- Predict common user issues and plan for them when placing features.
- One feature can have more than one placement. Redundancy is acceptable when in moderation.

5 Implementation and testing

The goal of this section is to give the potential developer of a new addition a head start in understanding the functions of classes in the GUI and minimize the time needed to learn the code. It describes the state of the GUI as is.

[1]Occam's razor is a principle that simply states that the less information is in the message the less are the chances of misinterpretation

5.1 Tools

GUI design tool For a swift and error free design I used the default Eclipse GUI editor for Android. What the tool does is enable the developer to drag and drop elements that he wishes to be placed and it will automatically generate the appropriate XML file. This tool has its flaws and sometimes the developer has to help the tool by altering the XML by hand. It also features critiques and ideas for code improvement.

Gimp Some icons/pictograms were crated using the GNU Image Manipulation Program. The majority of the pictures did not make it into the final version.

5.2 Implementation

NOTE: This part of the document is not meant to be standalone. It incorporates terminology from the main thesis text. Please read the thesis Implementation section before continuing.

Overview The applications GUI implementation is structured much to the way the design has structured the application. There are the activity classes: main, ChangeServerView, QuestionView, ServerListView. All except the main activity have self explanatory names. Main is the password input activity. Despite having the word View in them, these classes are in fact descendants of the Activity class and should not be confused with View classes. To create a common button type a prototype button class called "DefaultButton" has been created. In it the graphical representations of the buttons are defined.

Implementation details of Activities This part of the describes the parts of the implementation a developer might find hard to understand using only Java doc.

main The main does nothing and waits for the user to input his / her password. Once inputted, it calls a check method and validates the password if all goes well, it initializes the application cryptography and opens the ServerListView. The cryptography initialization is basically associating the open text[2]. Once the password is associated, the other activities can be accessed.

ServerListView Upon launch, the activity starts the beacon listener and gives it its reference. The activity then requests the persistence layer to load the saved servers and displays them. When the beacon listener receives a beacon packet it updates the information and triggers an update of the GUI. An interesting part of this activity is that the list of servers needs constant refreshing - in case a server is deleted, the beacon server stops broadcasting, changes it's name etc. . This is done via a timed event.

[2]Opentext is in cryptology unencrypted text

```
\\ The timed event setup
private Runnable mUpdateTimeTask = new Runnable() {
        public void run() {
             final long start = 5000;
             long millis = SystemClock.uptimeMillis() - start;
             int seconds = (int) (millis / 1000);
             int minutes = seconds / 60;
             seconds     = seconds  15;if (seconds == 0 {
               printServers();
                Log.d("Android mobile voting","seconds < 10");
             }
             handler.postAtTime(this,
                       start + (((minutes * 60) + seconds + 1) * 1000));
        }
     };
```

ChangeServerView This activity can recieve a ServerData entity in the intent[3] with which it is launched. When it does it displays the data of the pushed server. Thanks to this the user can edit or create new servers with the same logic.

Questionview This activity is rather special as it contains the connection class that is used to communicate with the server. It also handles informing the user about connection progress. It does this by using the showNextProgres() method which displays or dismisses the spindle. As the connection is using another thread the questionView has a Handle entity that can be used to run code in the UI thread. This is necessary because the platform prohibits alterations to the GUI from non UI threads.

Other components

5.3 Code criticism

When I started writing the code I had no experience with this platform. Therefore I could not plan in advance the classes and the linkages between them. At this moment there is a log of referencing going on that surely has a better solution. For example the buttons each reference their parents, and the connection has nested references that go to a large number of contained classes. While this works, it is not a good solution. It is the main reason for the application lag on slower devices. Before starting a new project, I would suggest refactoring be made and this document updated accordingly.

[3]An Intent is a collection of information that helps the activity to launch.It gives it information about the context of the launch, and initialization data

5.4 Testing

Computer based As discussed in the main text of the thesis, the application uses Robotium for automated tests.

Real world Upon each major change the application was given to a number (4) testers. These were people that owned an Android powered device and were willing to help out. They were asked to simulate the voting process that was described to them. After each session the testers were asked questions about what they found to be counter-intuitive, how would they prefer the interface to react and if any errors occured. They were encouraged to give no quarter to the application and be as rough to it as they would be in real life. After each testing I have written down notes about their experience and planned my next alteration accordingly.

6 Conclusion and Future

In conclusion, the GUI performs its role. It enables the user to use all the features that are provided by the components beneath the interface. While the coding is not the best it could be due to my lack of skills I can say that it is a success as the final release was tested with two more people who had no prior knowledge of the application and they had no problems with making it work with a quick manual. Despite my best efforts, the application is still experiencing considerable lag when used in some situations. But all in all, the interface does what it needs to to and does it well.

References

[1] User interface guidelines — android developers.
`http://developer.android.com/guide/practices/ui_guidelines/index.html`, Retrieved on 4. 5. 2011.

[2] Pubtran - android market.
`https://market.android.com/details?id=cz.fhejl.pubtran`, Retrieved on 4. 5. 2011.