Developing an Android Application

Brian Belding  
brian.belding@gmail.com  
GSET

Matthew Haake  
mhaak3@comcast.net  
GSET

Ben Parisi  
wildfire655@gmail.com  
GSET

Eric Simeonoglou  
EJSime93@aol.com  
GSET

July 23, 2010

1 Abstract
The menagerie of settings in mobile phones is overly complicated and confusing to the average user. Often this results in the phones doing inappropriate things at inappropriate times. To solve this, we designed an Android application which gives users the ability to define the settings that their phone should have in a certain situation, based on the time of day, day of the week, and gps location. The phone then automatically changes settings such as screen brightness, ringer volume, and wireless connectivity options to the state that the user specified.

2 Introduction
For years, the mobile phone industry has touted the rise of "smartphones" - phones with an advanced featureset and deep computing ability - as the next big revolution in mobile technology. Despite this improved technology, "smartphones" show very little learning and intelligence in their programming. While these phones remain a vast improvement over the basic "feature phones" of the original industry, they still have yet to reach their full potential as far as effectively and efficiently using the systems, sensors, and processing power that they already have on board. Although on paper this hardware has immense potential, much of the time the phones continue to subscribe to the rudimentary structures and software of the feature phones that are their ancestors.

To that end, one of the problems with modern smartphones is that the added features bring added complexity to system settings. Instead of integrating these features in an intuitive and understandable manner, many phone manufacturers simply tack on an additional screen to the existing web of menus and leave it at that. Such an approach fails to take advantage of the capabilities of these phones. They already contain enough sensors to accurately determine the circumstances in which they find themselves, so there is little reason to require manual interaction from the user to do things (like changing settings) which could easily be done autonomously. For example, the phone could automatically change its volume to silent to prevent disruptions during class, or even turn off wifi and bluetooth to save power. This would be a convenience to users by freeing them from the responsibility of maintaining their phone in its proper state, and allow them to simply use all the features the phone has to offer.

We decided to design an application to solve this problem: one that will automatically set the system settings of the phone based on the fulfillment of certain conditions. Using the above example, imagine a student who attends class regularly from 8:00AM to 2:30PM, Monday to Friday. The program can recognize when both of these parameters are true, and will execute the specified actions when that is the case.

We also designed and implemented a feature that simplifies how the user creates situations. When active, this feature collects usage data on each of the phone’s settings, analyzes the data for patterns, and suggests the automation of certain settings based on the data. This optimization process allows for the creation of "smart" scenarios, in which the user may find a way to save battery power or optimize settings in a way that they would not have otherwise known.
3 Background

3.1 Mobile Development

Due to the resource constricted nature of the platform, applications for mobile devices must be designed differently than those for conventional computers. Comparatively, smartphones have lower screen resolutions, less RAM, and slower processors. These constraints force mobile programs to use memory efficiently, make good use of screen real estate, and be mindful of how much they are taxing the device. On the other hand, unlike traditional computers, smartphones often have an array of sensors onboard that are available for easy integration. Accelerometers track the phone’s movement and speed as well as orientation, all of which can be utilized to create intuitive controls for games or provide useful data to applications. Digital compasses and Global Positioning Systems allow the phone to locate itself (and therefore its user) anywhere on Earth, and offer directions and nearby locations via search. Cameras are being implemented in a multitude of ways, such as acting as scanners for phones using Optical Character Recognition. Most importantly, touch screens provide a very natural, innovative interface for the user while eliminating space-consuming devices such as pointers and trackpads.

3.2 The Android Platform

Android is a relatively new mobile platform that was first announced in November of 2007. Created by Google as a part of the Open Handset Alliance, the first phone to use the new Android operating system - the HTC Dream - was released to the public on October 23rd, 2008. Since that point, Android has climbed to hold 13% of the US market share in smartphones. The operating system seamlessly ties in all of the aforementioned hardware features and makes them available for developers to use via its open-source application development system. Developers have complete access to the Android Software Development Kit (SDK) and the Android Application Programmer Interfaces (APIs) in addition to expansive reference material on how to implement them. [1]

The operating system builds upon the Java programming language by layering Android-customized methods and classes on top of it. Android offers a unique programming environment by allowing access to a broad range of controls across the platform. The resulting flexibility lets developers modify more of the default options of the phone than any other popular mobile platform. This characteristic is key to the success of our application because it requires access to the system settings of the phone. In addition, it is imperative that other functions of the phone can run in the foreground while processes of our application run in the background. Other popular mobile platforms place restrictions on or limit the use of multitasking, but Android grants programmers complete access to this functionality.

3.3 Prior Work

There have already been a few attempts at simplifying and automating a user’s settings: Locale, a program created by a team of developers at MIT in the early stages of Android, [2] automates the controls of the phone, but requires the user to think of and program the controls for this automation. Improving upon Locale and other similar applications, our program provides an understandable and information-dense user interface and helps the user create situations by recommending settings via an optimization process in which the user’s activities are monitored over the course of a week and analyzed for patterns at its conclusion.

4 Software Design Decisions

4.1 Conditions and Actions

One of the most crucial design decisions for localization software is choosing which variables define a ‘location’ or ‘situation. Initially, we had planned to include as many conditions as we could control: time, date, location, battery power, caller ID, text message contents, and more. An interface for all of these proved to be disorganized and overly complicated for the user. Because of this, we narrowed the focus of our application to only time of day, day of the week, and GPS location (via Android’s default Google Maps system).

With the conditions established, we needed to specify what actions the user could automate. We chose to allow the user to alter six of the phone’s key system settings: First, the intensity of the ringer volume and screen brightness, and second, whether Wi-Fi, bluetooth, GPS, and airplane mode should be on or off. In addition we added our own unique, non-standard setting: the away message. This allows the user to set a unique string of text (for example, "hey sorry I’m in class I’ll get back to you later!") that can be sent via text message in response to any incoming text or phone call when that set of conditions is true. After, the user is informed of who attempted to contact them so they can give a more appropriate response.
4.2 User Interface

The key to our application is ease of use for the user. Presenting the information in an intuitive manner was therefore an important aspect of our development cycle. Because the application itself is centered around simplifying the user’s everyday life, it was important to keep the interface as simple and understandable as possible. To accomplish this, our entire application is structured around a basic list view that leads to a total of seven different screens. Each screen was designed to maximize information density while still remaining intuitive to the user. To that end, we decided on a user interface paradigm based on forming actual sentences. Conditions and actions were organized in an “if... then” format, with each condition adding more to the sentence. For example, if a user has a class from 8:00 am to 2:30 pm, Monday-Friday and wanted to set their phone to silent for the duration, our edit screen would read “If the time is 8:00 am to 2:30 pm and the day is Monday-Friday, then change volume.” By arranging the different screens in this manner, the user has a very clear idea of how the situations are being defined.

4.3 Multiple Conditions and Priorities

In our initial design, we restricted the user to only one condition per situation. However, this restriction did not allow the user to account for certain plausible circumstances. For example, if a user has a situation describing what to do while he is at school, what happens if he is absent for a day? Or if he set the situation through location, what if he is on school grounds after hours? Because of this complication, we determined it necessary to include the ability to specify multiple conditions for each situation. However, this brought up another problem.

Due to the variety of possible situations that the user can create, it was inevitable that some conflicts could arise between different situations. For example, a person could have distinct situations for while they were at home (which turned their volume up) and for during the night (which turned their volume down). If this person were at home during the night, a conflict would exist over the proper volume setting. To resolve this problem, we assigned each situation a priority based on its position in the list on the main screen. The user can rearrange the list in order to change which situations get priorities over others. Situations closer to the top of the list have a higher priority than those below, and should a conflict occur, the situation higher on the list gets preference.

4.4 Saving and Accessing Files

Our program uses two classes for writing and accessing files. These classes work closely with a list
Figure 3: Screenshot showing how users select the time feature of a condition.

Figure 4: Screenshot showing how users select the days on which the settings will change.

Figure 5: Screenshot of the GPS location selector.

Figure 6: Screenshot of the main screen with its situations arranged by priority.
that contains all of the variables of the situations that were defined by the user. Both classes contain static constructors that create an instance of the class so that when other classes in this program call the method, they do not have to make an object of that class. The write method of the writing class creates a text file that is named after the situation. This text file then stores all of the conditions and actions so that they can be accessed at a later time. The reading class is called once every time the application is started, and once again every time the phone starts up or the conditions or actions in the application are updated. The writing class is called whenever the user decides to change the parameters of the situation.

### 4.5 Services

The basis of our application resides in code that runs in the background of the phone. On Android, such code is called a service. Services can either run while other applications are in the foreground, or while our application is not open at all. In addition, multiple services can run at the same time, and are automatically restarted once enough resources are available. Our service waits for the conditions of any of the user’s situations to be met, then changes the settings to match the situation. When the service is written, it will pull the situation data from the file and keep it in RAM. In order to save processor cycles, the conditions of the situations will be given to the Android system to alert the program should they become true. Proximity Alerts for location and Alarms for time will send Intents to the background service when they become true. Other conditions such as incoming phone-calls already broadcast their state to all programs.

By using the system to alert the program that conditions have been met, the background service is able to stay dormant until conditions change. When the system broadcasts that a setting has changed, it will automatically wake the service to check whether enough conditions have been met to satisfy a situation. If all the conditions for a situation have been met, the service will change the predefined settings to their user-specified values. This method of passive listening for changes minimizes the effects of the service on the phone’s responsiveness and battery life.

### 4.6 Optimization Process

In order to make our application easier to use, we added a process to help the user create situations. We designed a system which, when triggered by the user, studies for one week how the user changes the specific phone settings that are relevant to our application. These settings include whether Bluetooth, GPS, and Wi-Fi are turned on or off. Once the week is over, the data is analyzed to see which settings meet a specific threshold of regular use. Depending on where the data falls in accordance with this threshold, the application can recommend and create situations based on this data. Currently, our application looks for one-hour intervals in which the settings remained constant, which would suggest that the user has a tendency to stay on that setting during that period of time. Each recommendation must be approved by the user before it is finalized. For example, if the optimization process finds that the user never uses Wi-Fi after 8:00 PM, it can recommend a situation that always shuts it off after this time. Then, if the user chooses to accept this, the application will take these conditions and make it into a situation for the user. Should the user decide not to use this situation any longer, they can delete it or edit it in the same way they could with one of the situations that they declared.

### 5 Results and Discussion

Upon the completion and early testing of our design, we found that the majority of users were able to easily navigate through the user-interface and create situations for the phone. To test this, we ran two case studies. The first case study was given to five students. The students were given an outline of an example situation to create for an average school day. They were told to make the phone set the volume to vibrate, turn Bluetooth off, turn Wi-Fi on, and set an away message that says they are in class on Monday through Friday from 8:00AM - 3:00PM at Rutgers University. While creating the situation, each of the users experienced very similar problems.

Most of the problems arose on either the location screen or the actions screen. In the map screen, four of the five users failed to correctly input their desired locations. Although they were successful in navigating to the search bar, the majority of users never actually implemented Google’s mapping engine by pressing the search button. Instead they entered the name or address of the location, and rushed to the done button, never saving a location into their situation’s data. In response, we implemented two pop-up messages to the user to ensure proper data entry. The first is set to activate when the user presses the Done button. If a search is never run or no data is stored to the location variable, then the user is greeted with a dialog that provides them with their options on how to fix the
issue or cancel. To add to this clarity of this process, the second pop-up message informs the user that the location has been entered. Directly below where the coordinate is pin-pointed on the map, a dialog presents the user with the address that it has set.

With these solutions implemented into our application, we began the second case study with twenty people. The results improved drastically from the initial test. Eleven out of the twenty users created the situation perfectly, and eight only made one mistake in the actions screen. Those who failed to correctly create the situation simply misunderstood how the check layout worked. Instead of understanding an unchecked item such as GPS was turned off, they assumed that it was not being implemented (where as it would be completely removed from the screen). To enhance our study, we then added a second situation for the same testers to create. The situation was to be entitled "Beach," and should turn brightness and volume to their maximum levels, as well as turn GPS on all day every Saturday. Six of the original erroneous students completed the second situation perfectly and with greater ease. To prove this quantitatively, we timed the user on each of their runs. On average it took about five minutes and twenty seconds to complete the both situations, approximately three minutes and forty-five seconds on the first and one minute and thirty-five seconds on the second.

6 Conclusions

Through testing, our application has shown to help solve the problems for which it was created. The evaluation of the user interface seems to suggest that it is easy to use and to comprehend, and although there were some early setbacks, they were fixed and retested. These later tests show that the changes help in making the application easier to use. In addition, the optimization feature of our application is written and is in the process of being integrated into the source code. This will be completed shortly after the conclusion of Governor’s School of Engineering and Technology 2010. Following the integration of all features, our application will become available for public download through the Android Market.

7 Acknowledgements

We would like to thank Christine Hung of Ozmosis Learning for being our project advisor and guiding us through the idea, design, and creation of this application. We also thank Angel Irizarry, our RTA mentor, for inspiring us, as well as Blase E. Ur, the Governor’s School Program Coordinator and Ilene Rosen, the Governor’s School Program Director. In addition, we are grateful to Kristin Frank, Head RTA, Jameslevi Schmidt, Research RTA, the Governor’s School Board of Overseers, and the sponsors of the 2010 School of Engineering and Technology: Rutgers University, the Rutgers University of Engineering, Morgan Stanley, the State of New Jersey, Lockheed Martin, PSE&G, the Tomasetta family, The Provident Bank NJ Foundation, Silverline Building Products, and the families of Governor’s School alumni.

References