Mark Ha '13
Adviser: Brian Kernighan
February Paper
02/08/13

"Easy Android" - Developers' Tool for Android Beginners

Introduction

The best way to summarize my thesis is this: my goal is to create a desktop application that enables users with little or no programming / computer science experience to be able to create their own Android applications without necessarily knowing the exact details of the code that goes behind it. So ideally, a user with a clear design and product in mind would be able to convert that idea into an actual, functional application on their phone in minimal time (on the order of one to two hours for first-time users).

The name, "Easy Android," sums up what I hope to accomplish for new Android developers with this project - facilitating Android development. For first-time users, the typical setup process to even begin Android development - to make a "Hello World" application - though well documented and packaged much more nicely than it was originally, requires an assortment of downloads and installations. Specifically, the user needs to have a Java SDK, the Android SDK, Eclipse (or some other comprehensive, high-power editor that can compile and run Android), and finally, any other addition Android SDK tools, extras, platforms, depending on the user's particular needs. Once they have all of these software downloaded and installed, they can finally open up the editor and *attempt* their first program.

After that, there are plenty of tutorials and resources online to guide developers through the "Hello World" step, but for first-time Android developers, especially those who come from non-technical backgrounds, the barrier between "Hello World," and making their own application can be quite significant. While browsing Google search links will eventually answer the question, "How do I make an application that does ___?" it is a process that can be both time-consuming and incredibly frustrating, especially when there is no guarantee of the validity of the "answers" one finds. And so, on top of all the setup that is required to create a "Hello World" application, learning how to program for Android - even for an experienced Java programmer (the language Android code is in) - consumes a considerable amount of additional time. Not only is there the software to consider, but there is also the significant learning curve. In short, this combination of overhead cost presents a substantial impediment to any curious individuals considering starting Android development, and the purpose of Easy Android is to reduce that setup cost.

At the same time, it is important to be realistic and realize that it is impossible to create a tool that would fit the needs of every developer. I decided early that my audience would be first-time Android developers, particularly those with no or relatively little programming experience. Developers who are familiar with Android would probably agree that there is definitely room for improvement in Android development as a whole - for instance, potentially needing to go through every class in the library in order to find the one that gives you the phone's GPS API, or a plethora of complications when dealing with backwards compatibility and making an application that is available to as many Android versions as possible. But on the opposite end of the spectrum, there are the people with no programming experience whatsoever who want to start making Android applications, who struggle with the setup costs and learning curve described above. Ultimately, I decided to target these kinds of users - even without significant programming experience, maybe an introductory computer science course, someone with a clear design or specification in mind would be able to create their own Android application with relatively little effort.

Related Works

I have not yet read any academic papers on this topic, rather, instead I have explored similar applications - applications that seek to, on some level, simplify the programming experience for fresh Android developers. Perhaps one of the more prominent applications that I have heard about is "App Inventor," a project in beta release originally headed by Google, but now under the care of the Massachusetts Institute of Technology. App Inventor takes a very unique approach to programming - perhaps more accurately, the application presents programming and code to the user in a very unique way. Instead of having the traditional, stale lines of code with bothersome syntax, programming in App Inventor is done through the connection of puzzle-piece-like blocks. One example is that instead of the typical Java for loop, there might be a for "block," which takes the necessary arguments that determine its lifetime, but also, the block will be shaped such that another block - what might normally be considered the body of the for loop - fits inside of the for block. By converting all code into puzzle pieces, App Inventor provides a more visually intuitive experience for new Android developers, and also removes the barrier of not knowing Java, or how to code.

A similar product that removes the typing part of coding and converts that into a graphical interface is Scratch, a programming language designed for early programmers (specifically young children) also developed at MIT. Like App Inventor, Scratch uses blocks and drag and drop, rather than actual text to represent many of the logical paradigms that traditional computer scientists know and love. All syntax and code is converted into either blocks or some graphic, which are considerably easier to understand for beginners than the standard syntax and coding rules that one learns in an introductory computer science class.

Both of these applications - App Inventor and Scratch - emphasize the learning process for the user, introducing them to programming through block-based languages. Furthermore, neither is meant to be as powerful a language as say, (actual) Java, or C, or even Python. For instance, Scratch is limited to one-dimensional arrays only, and String manipulation is limited as well. Along the same lines, App Inventor does not easily lend itself to larger scale applications, due to its inability to include external jars (Java Archives) or create true multi-screen applications (instead, you clear the screen and redraw it, essentially). Both also have their own upfront setup costs. That is not to say that one cannot make extremely useful and relevant applications using these products. However, it is apparent that Scratch is not meant to be a full-fledged programming language, nor is App Inventor meant to have the full capabilities of the native Android developing environment. Yet both are arguably much better for beginner programmers, who have yet to learn the ins and outs of syntax and coding.

I hope to model Easy Android after App Inventor and Scratch, specifically in the way they appeal and are much more effective for beginner developers. While neither can really implement an actual, complex product as well as Java or Python, both can handle simple applications just as well as any other language, perhaps even better for inexperience developers, since everything is more or less simplified. In the same way, my plan for Easy Android is to enable any learning developer to efficiently implement simple Android applications.

Progress

Regarding the progress of the project to date, I technically have all of the functionality necessary to create the simplest of applications. For instance, I could make a simple tip calculator, or a contacts browser with the option to click to call, text, or email from start to finish using Easy Android. And every feature I add to the application requires at least a few more lines of code, depending on the complexity and uniqueness of the feature.

In order to develop Easy Android, I spent much of my time building example Android applications that one might expect a user to want. Among those demo applications were a tip calculator, a contacts browser with quick call and text options, an approximate distance tracker using two GPS "checkpoints" activated by the user, a movement detector that uses the phone's accelerometer, and an application that sends out random texts to random contacts. All of these are relatively simple applications from an experienced developer perspective, as well as from the design perspective, yet would probably take several hours for novice Android developers to work out on their own. They also each employ phone-specific functionality that intuitively most people would like to take advantage of. For example, the GPS application takes advantage of the phone's GPS capabilities and could be expanded just slightly to launch Google Maps with the recorded points to help the user remember his or her parking spot. These were some of the more common and useful examples that would have the greatest impact by adding them into the Easy Android application.

Currently, Easy Android operates through the command line, similar to how one would run Python. To elaborate, the application runs as a Java executable, and from there, accepts and parses user input. There are an assortment of commands, each dealing with different parts of creating an Android application, as well as some that offer flexibility in the project. For example, the "create," "build," and "install" commands do what one might reasonably expect: create a new project folder (on disk), compile and build the project, and install the project to an Android phone attached by USB, respectively. Internally, this includes running a script to build the Android project and output the .apk file (Android application executable file format) for the project. Other commands allow the user to change the project name, current class name, package name, while still others help the user debug and decipher the current state of their project by printing out the corresponding Java or xml code, which is actually the Android code that would be compiled if the user built the project at that time. Finally, the most important commands allow the user to add a button, or a text object, or an editable text form, or some custom function to the current class. All of these commands - especially the ones dealing with the actual content of the application - take optional arguments (such as height and width), which when not specified, have generic default values. To illustrate what these commands might look like, here are some examples:

(In this example, pretend that "calculateTip" is a custom function that the user has defined in order to calculate tip accordingly, as well as update the "tiptotal" text item)

```
>button -name calculateButton -text calculate -width fill -height wrap -action
calculateTip
button "calculateButton" added.
>edittext -name editItem -hint "type here" -width 100 -height 50
edittext "editItem" added
>textview -name tiptotal -text 0.00
textview "tiptotal" added
```

With these three simple commands - followed by the "build" and "install" commands to put the application on to the phone, of course - and one simply more complicated custom function definition for the "calculateTip" function, the simplest tip calculator has been created.

And so, currently, the program follows a fairly standard command line structure, though that is all a matter of syntax, and the input method could be altered at any time. As mentioned above, an end-to-end version is complete, which has medium functionality. While it is certainly possible and from a product standpoint probably "necessary" to continue adding features that users can add to their applications (for instance, easy access to the Android "action bar" navigation menu bar, or allowing users to have colors and gradients in their backgrounds), because the framework is already set up, adding each of these features should be relatively trivial, and so, I have decided to move on to other higher priority parts of the application.

Plan

Over the next month and roughly the first half of the spring semester, I plan to finish a "beta" version of Easy Android. The first step for that will be to finish implementation of saving and opening "project files." The current version keeps everything in memory (excluding the Android project when the user chooses to create one), which means that if the user were ever to exit the program, all of their progress could potentially be lost. The resulting implementation will most likely simply record the user's input in a file, and then pull from that file upon being opened in order to recreate the user's progress (could also be implemented as saving the "state" of the project).

Furthermore, while technically what I have right now could be deployed and be considered a beta version, I feel that a Graphical User Interface of some sort - as opposed to the current command line - is essential to beginner developers. That is, while an experienced programmer may be familiar and perhaps even more comfortable using the command line interface, someone new to programming would probably find the command line as confusing as typing in an editor. Implementing a GUI will not only remove the step where the user is forced to read through the "help" documentation, but also facilitate learning how to use Easy Android, as well as make the whole process much more intuitive and to some extent, more efficient. Thus, implementing a GUI to represent the functionality of the application will be my second priority, which I hope to complete well before the second half of the semester.

Assuming that all goes well, the following step will be to have users evaluate my application, in order to learn what kind of improvements are most critical for the user. My test group will most likely consist almost solely of users will little or no Android experience, since they are the target of my application, although I may ask for feedback from some more experienced users, simply for diversity. If time allows, I would hopefully make some improvements and have a second round of evaluations, but realistically, I will probably be finishing up the actual coding and moving on to the report at that point.

To translate this plan into concrete dates, the "saving/opening file" feature should be done within the next week, by February 15. Following that, the implementation of the GUI will hopefully take around two weeks, three weeks in the worst case (so by March 1, or March 8 at the latest). The first week will be spent on design decisions and solidifying how exactly the GUI will be implemented. The second, and third if necessary, weeks will be spent on implementation. Because the GUI is so central to the application - presentation is everything - it is more likely than not that this will be quite time-consuming. Depending on when the GUI is complete, evaluation and user testing will immediately follow, completed by March 15, right before Spring Break. The rest of March will be spent on improvements, and a second round of evaluations (more for the analysis than a second round of improvements) will take place in the first week of April. The rest of April will be spent on the writing the final report and preparing for my presentation. Note that this is a slightly optimistic timeline - as previously mentioned, delays would be handled by removing the second round of evaluations if necessary.