Nicolas Stanzione, Benjamin Chevoor, Josh Blanchette, Kyle Zhang
Ubicomp DropApp Report
4/23/2016

Goal:

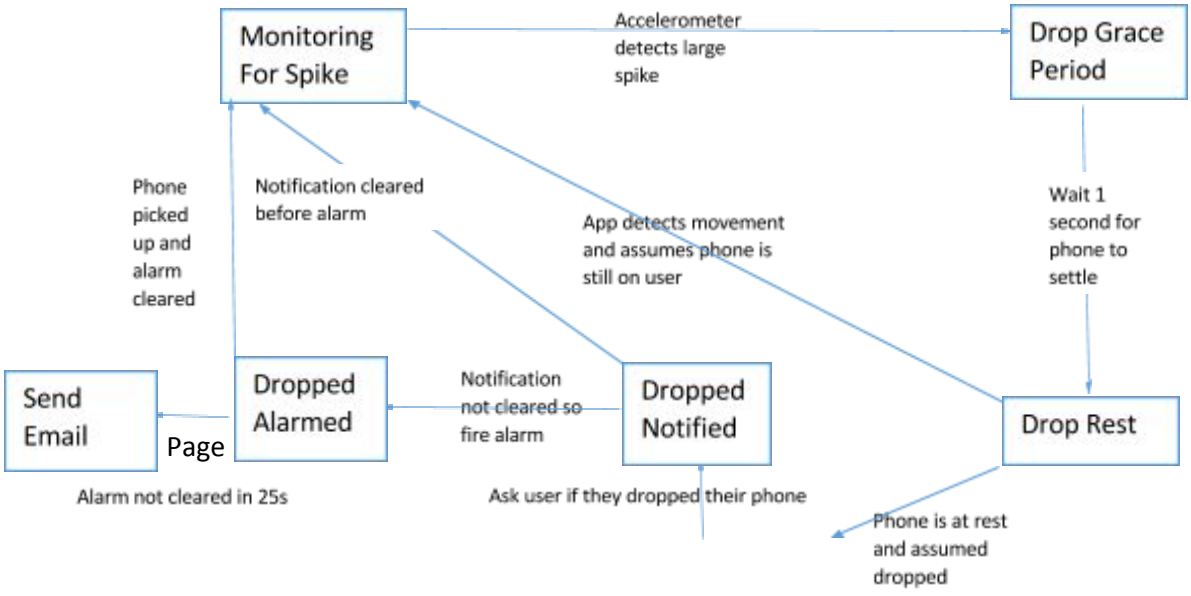
The purpose of this project was to take what we have learned in class and to use that knowledge to create a ubiquitous computing solution that could improve the lives of as many people as possible. The easiest way to reach out to a lot of people is to solve some problem involving cell phones since so many people are glued to these devices. Having your cell phone by your side is a certain must in today's society and even during the briefest moments without it can cause serious anxiety in some people. DropApp aims to put some minds at ease by ensuring people that there phone is much less likely to be lost using our product. Our main goal is to prevent the user's phone from ever being lost in the first place by alerting our users within the first ten seconds of the phone potentially being lost. This method eliminates the issue of the user not realizes there phone has been lost and potentially not start looking for it after a long amount of time where the phone could be damaged or stolen.

Features:

The main features of this app is not what the user does with it but what the app does for the user. The only thing the user needs to do is to initialize the app and then they can decide when they want the service running or not. This drop detection service is the main feature of the app. When the user has finished initializing and has hit the start button on the UI the drop detection service begins working in the background of your phone. When it detects a drop it waits to see if the user will pick the phone up and if they don't then it plays an alarm to alert the user that their phone has been dropped unintentionally. If the user hears the alarm and picks up the phone they can clear the alarm and the service will go back to detecting drops. If the user does not hear the alarm and does not pick up the phone then the application had a backup plan. The backup plan is an email that is sent to an address specified by the user during initialization and in this email contains the GPS coordinates of where the phone is. This is a last resort scenario, ideally the user should hear the alarm and pick their phone back up.

Design:

The real meat of this project is the background service, specifically the different states that it goes through in order to determine if the phone was dropped or not. Below is the state diagram for the background service.



File Structure:

Our project spans over many files and most of these files don't really do much so instead of going through each and every one I will go through the main files that have the most impact on the project. All of the files were created by Ben and he did most of the typing but we all worked together a lot and hashed out ideas together. Just because someone did all the typing doesn't mean they were responsible for all the work in that file. Ben had the most android experience so he handled a lot of the coding while Kyle, Josh, and I helped in the design and algorithm for finding the drop. It was truly a team effort across all fronts.

1. SetupActivity.java
    a. UI for the user to enter their email and other preferences
    b. Allows user to calibrate drop spike
2. EmailHandler.java
    a.
3. SettingsActivity.java
    a. Store the user's preferences from the settings page
4. MainActivity.java
    a. Main page of the UI
    b. Allows the user to...
        i. Start/stop app
        ii. Change settings
        iii. See data
5. DataViewActivity.java
    a. The graph showing the x y z and average values from the accelerometer
6. DropDetectionService.java
    a. Handles the scenario that checks for user interaction after a drop
    b. Second half of the state machine made above
    c. This is the algorithm for finding out if the user has picked up their phone or not
    d. This sends notifications to the user's phone
    e. Activates alarm if it's a drop
7. AccelerometerService.java
    a. First half of the state machine above
    b. Algorithm for finding out if the phone was dropped or not
    c. Activates dropdetectionservice when it thinks it's a drop
8. AlarmHandler.java
    a. Responsible for actually firing the alarm on the phone
9. EmailHandler.java
    a. Code to send the backup email if the phone is not recovered from the alarm
10. Settings.java
    a. Takes the info from the settings page and saves it

11. AccelerometerCalibrationFragment.java
    a. Used in initialization
    b. Finds out how sensitive the uses accelerometer is to determine could spike point

Project Evaluation

I had a hard time trying to find technical papers that sought the same goal as ours. I looked for a long time but most papers use the detection of the phone falling to make some other induction like an elderly person falling or a car accident. Most papers used a similar approach of using accelerometer data but their end goal was much different than ours. Most papers used more than one sensor, like electromagnetic sensors, compasses, or GPS, along with the accelerometer. This is useful in getting a more accurate reading, something we could use to handle our wide array of edge cases. Most papers also used more complex algorithms to determine drops like a discrete wavelet transform, filters (the car accident detection paper only looks for accelerations over 4G's), or using a high pass and low pass filter and then normalizing the data. Some of these methods are much more accurate than our algorithm so to upgrade our algorithm with some of these techniques would benefit us greatly. Judging by the other algorithms our algorithms is definitely novel compared to them since we really just apply a filter, not much math involved. The other papers also do much more data collection and testing than we did to really refine their algorithms, something we could have benefited from. One of the elderly fall detection papers only did tests with one type of phone which in my opinion seems like a bad idea because even in our group we did tests with three different phones and found all of our accelerometers be slightly different. It is naïve to think that all cell phones would have the same accelerometer data during the same action. Overall I think we created a great naïve algorithm for our application and given more time we could have made our algorithm more robust like the ones in the papers.

Paper References:

1. http://sensorlab.cs.dartmouth.edu/phonesense/papers/Yavuz-fall.pdf
2. http://www1.cse.wustl.edu/~schmidt/PDF/wreckwatch.pdf
3. http://www.asee.org/documents/zones/zone1/2014/Student/PDFs/171.pdf