CO$_2$ SOFTWARE

Design Document

February 21st, 2018

Sponsor

Dr. Andrew Richardson

Dr. Mariah Carbone

Mentor

Ana Paula Chaves Steinmacher

Team

Sam Beals, James Beasley,

Andrew Greene, Joseph Kelroy

1. Introduction

Global climate change is an issue that will affect every person on the planet. It is widely acknowledged that climate change is driven by rising levels of atmospheric carbon dioxide (CO2) resulting from fossil fuel burning. The importance of research into climate change cannot be understated, but when people think of data associated with climate change they typically think of the carbon footprint of our cities. They often forget that the environment and its carbon cycle hold data that is just as, if not more, valuable. Our goal is to ensure that ecologists can collect data as quickly, efficiently, and as easily as possible from the environment.

The Richardson-Carbone Lab is operated by Doctor Andrew Richardson and Doctor Mariah Carbone at Northern Arizona University. The Richardson-Carbone Lab studies carbon cycling in forest ecosystems. In a nutshell, the research is used to understand the balance between carbon uptake (photosynthesis by plants) and release (respiration both by living and growing plants, as well as by microorganisms decomposing dead organic matter in the soil). Through the use of modern technology, we aim to make the collection of CO_2 data from trees a streamlined process with minimal cost and intuitive user interaction, providing users the ability to collect, visualize, and interpret field readings.

Currently, the Richardson-Carbone Lab has to use a cumbersome set of equipment to conduct their research. This method of data collection is slow, not very portable/durable, and it is not the easiest to view out in the field. The proposed solution will make the entire process fast, portable, and easy to use while also providing important meta data to the researchers in real time. Currently, we are in the process of developing a mobile application for the Android platform. The app allows any Android device to communicate with the client’s LI840A gas analyzer. The data is streamed into the app and displayed in real time. Any of the data that is logged in the app can be saved and transferred out of the app via email. Additionally, the data saved by the app can be reviewed and deleted at any time by the user. This document will start with a brief explanation of our implementation, and how exactly we plan to solve the client's problems. It will then move on to an architectural overview, explaining how our solution will be structured. We will then give a more in depth analysis of each module in our architecture, before finally wrapping up the document with our implementation plan and conclusion.

2. Implementation Overview

In order to solve the issues listed above for our client, we are developing an application for mobile devices that can be used to measure and record data in conjunction with their LI-840A gas analyzer. The application's interface will be designed with a Lenovo Tab 4 in mind, as the large screen real estate allows for a higher level of detail. The application will use Android GraphView to render a series of graphs that displays the gas analyzer's reported data in real time. The software will also provide several means of statistical analysis, such as slope, standard error, and R^2, so the user can generate the information as needed with the convenience of staying within a single application. Finally the application will be able to transfer data out via email with convenient naming conventions of the files. With all of the tools being readily available on a tablet device, any researcher out in the field will no longer be burdened with having to carry a laptop or any other peripheral devices in order to collect and analyze data.

This implementation relies on the use of a few different existing technologies. We have chosen Android for our mobile operating system, mainly due to its popularity in the mobile device market and its wide range of libraries and tools it offers developers. Android Studio uses the Java programming language so all back end code will be using this. We will also be using the Android GraphView library as our data visualization tool because it allows for a large degree of customization in how the data is represented, as well as having support for many key features, such as live readout. Lastly, we chose email for our method of data transfer out of the application because of its ubiquity and easy integration with Android.

3. Architectural Overview

Java is an object-oriented language so our architecture can easily be organized into a series of objects. We have devised eight different objects that will be required to meet the requirements, with five being Android screen objects, and three being helper objects to either organize data, or assist in the management of that data.

Each screen object will be responsible for handling every aspect of how the user can interact with the software. For example, whenever the user presses a button on a particular screen, a corresponding function will be called from within the screen's object file to execute the designated response. These responses can be taking the user to another screen, changing the graph they are currently viewing, or starting to log a data subset.

Each helper object will have unique usage scenarios, but will overall be used for the easy manipulation of data within the application. For example, when a data reading needs to be loaded into the application so that it's information can be viewed again, we will be building a Data Reading Object using the information present in the saved files.

This will allow us to easily access and pass around values from inside of a Java object instead of having to pass around strings directly read in from Android file I/O. (See Module and Interface Design for exact use cases).

In the following section, we will go over in more detail the responsibilities of each object, what data they will store, and what methods will be necessary for them to fill their responsibilities.

4. Module and Interface Design

4.1. Home Screen

Overview

This home screen will act as a navigation point to screens in the application. The application has a forward thinking process. The Home screen contain some information about the application, as well as to points of access to other screens. The first point is to the Metadata screen. This will be used when a user would like to start a new data set. The other point is the file directory screen. Both of these points of access will be represented with buttons.

Methods

Name: goToMetaData
Parameters: none
Return value: none
Description: Takes the user to the metadata screen of the application.

Name: goToView
Parameters: none

Return value: none

Description: Takes the user to the view screen of the application.

4.2. Metadata Screen

Overview

On the metadata screen, the user will enter all of the information relevant to the specific instance of data collection. This includes the following data: Operator Name, Site Name, Plot Number, Temperature, Date, Time, and GPS. Once entered, the user will move on to start the data collection. The user may also take a picture for the dataset. There will be a button on the screen that will open the camera on the device. The picture, as well as the dataset and metadata, will all be saved together.

Methods

Name: goToGraph

Parameters: none

Return value: none

Description: Takes the user to the graph screen of the application.

Name: autoFill

Parameters: none

Return value: Integer values for GPS, Time, and Date

Description: Automatically fills in the GPS, Time, and Date for the metadata screen using given Java functions.

4.3. Graph Screen

Overview

The graph screen is the screen where the live graph readout can be observed, and data can be saved to the device. It contains several elements, including button to switch between graphs, start and stop logging buttons, and a button to finalize recording. In order to operate fully, this screen uses both the graph manager, and line graph objects.

Methods

Name: initialize

Parameters: All the various metadata values.

Return Value: none

Description: Constructor for the graph screen. Initializes a graph manager object, and tells it to start collecting information from the instrument.

Name: logButton

Parameters: none

Return Value: none

Description: Triggers when the user presses the start/stop logging button. Calls the graph manager function to start recording a new data set. All series data reading will be handled using the USB-serial android interface.

Name: nextButton

Parameters: none

Return Value: none

Description: Takes the user to the file directory screen. If a data log is still currently being recorded, ends it before proceeding.

4.4. File Directory Screen

Overview

The file directory screen is the screen in which the user can view all their previously recorded data sets in a list of the file names which are in the format: site name/sample ID/date/time. It contains only two elements: a list of data readings, and a button to take them to the home screen.

Methods

Name: initialize

Parameters: none

Return value: none

Description: Loops through the master application folder, builds an array of these files, and constructs the list of these files on the screen.

Name: goToHome

Parameters: none

Return value: none

Description: Takes the user to the home screen of the application.

Name: goToView

Parameters: Data Reading Object

Return value: none

Description: Takes the user to the data viewing screen. The data reading parameter corresponds to which data reading in the list the user chose.

4.5. Data Viewing Screen

Overview

The data viewing screen is the screen where previously recorded data sets can be loaded from the device to have their values reexamined, and chosen to be emailed.It contains buttons to switch between all the different graphs, several text fields to display metadata values, and and button for emailing.

Methods

Name: initialize
Parameters: Data Reading Object
Return Value: none
Description: Takes in a Data Reading Object, and constructs the screen based off the information in that object. This includes initializing four Line Graph Objects, which will be set as images on the screen, and setting all the appropriate text boxes with the metadata information like sight name and comments.

Name: email
Parameters: Data Reading Object
Return Value: none
Description: Builds an email using the Data Reading Object, and sends it from the google email on the device. This will be integrated with the tablets chosen email.

4.6. Graph Manager Object

Overview

The Graph Manager Object is the object that is directly responsible for communicating with the LI-840A and feeding information to the graphs. When initialized, it will establish communication with the instrument, and set the appropriate polling rate. On its own thread, it will continue to read in and store the information, before parsing it and sending the values to the corresponding graphs.

Methods

Name: initialize

Parameters: none

Return value: none

Description: Constructor for the Graph Manager Object. Starts by establishing communication with the LI-840A and setting the polling rate at twice per second. Then the object initializes all the graphs and starts the update loop on its own thread.

Name: run

Parameters: none

Return value: none

Description: Used to implement the runnable interface. Runs a continuous loop to wait half a second, then get data from the instrument and finally update the graphs.

Name: startLogging

Parameters: none

Return Value: none

Description: Tells the graph manager to start recording the inputs from the instrument to be saved as a data log.

Name: stopLogging

Parameters: none

Return Value: none

Description: Tells the graph manager to stop recording the inputs from the instruments in the current data log.

Name: toString

Parameters: none

Return Value: none

Description: Writes the entire current data log to a string and returns it. Used to write the log to a text file easily.

4.7. Line Graph Object

Overview

The line graph object is a way for us to easily interact with the Android GraphView library. Instead of having to interact with the methods of Android GraphView every time we want to display a graph, this object can be used. The Line Graph Object will have a simplified set of methods, tailored to our needs of a live updating line graph.

Methods

Name: initialize

Parameters:String graphName, String xLabel, String yLabel, String color

Return Value: none

Description: Constructor for the Line Graph Object. Sets class member variables for the graph name, x and y axis labels, line color, and other necessary values.

Name: initialize

Parameters: String graphName, String xLabel, String yLabel, String color,

Return Value: none

Description:

Name: addPoint

Parameters:Float value, Float time

Return Value: none

Description: Adds a point to the graph at the value and time specified. If the maximum x or maximum y values need to be adjusted to fit this new point, then it is done.

Name: toString

Parameters: none

Return Value: String

Description: Returns a string representation of the Line Graph Object. Used in writing to a file.

4.8. Data Reading Object

Overview

The Data Reading Object is used to represent a previously collected data reading. These objects will be constructed from the contents of a recorded readings folder, which contains three files: Metadata.txt, Data.txt, and Image.png. An array of these objects will be used to construct the list of readings on the file directory screen, and individual objects will be used for constructing the display of the data viewing screen.

Methods

Name: initialize

Parameters: String metadata, String data, Image image

Return value: none

Description: Constructor for the Data Reading Object. Takes in the content of the three files, parses them into individual values, and assigns those values to class member variables. These variables will be used in the getter methods of the object.

Name: getName

Parameters: none

Return value: String

Description: Returns the name of the data reading.

Name: getTime

Parameters: none

Return value: String

Description: Returns a string representation of the time the data reading was taken.

Name: getGps

Parameters: none

Return value: String

Description: Returns a string representation of the GPS location where the reading was taken.

Name: getComments

Parameters: none

Return value: String

Description: Returns a string of of the comments that were entered on the metadata screen. Returns null if no comments were entered.

Name: getPlotNum

Parameters: none

Return value: String

Description: Returns a string representation of the data reading's plot number. Returns null if no plot number was entered.

Name: getOpName

Parameters: none

Return value: String

Description: Returns the name of the operator entered on the metadata screen. Returns null if no operator was entered.

Name: getTemp

Parameters: none

Return value: String

Description: Returns a string representation of the temperature entered on the metadata screen. Returns null if no temperature was entered.

Name: getImage

Parameters: none

Return value: Java Image object

Description: Returns a Java Image object of the image taken on the metadata screen. Returns null if no image was taken.

Name: getGraph

Parameters: none

Return value: Line Graph Object

Description: Returns a Line Graph Object, constructed using the data found in Data.txt.

5. Implementation Plan

In order to go about fully implementing our product, we decided to split tasks and responsibilities for each member based on the screens assigned to each of them. The App consists of four main screens (and a basic home screen menu), with each screen being fully realized by an individual member. Of course, there are some aspects of the functionality that are not entirely contained to a single screen within the app. We have set up a few measures for the group to prevent issues arising from this.

In order to avoid any discrepancies that may arise from shared data and values between different screens, all of the work conducted by individual members is done in the same location. When any issues in the code or confusions in the design abound, it is discussed and settled in the same instance. This creates ease of communication and maximizes efficacy in regards to decision making and bug fixing. Prior any initiative being taken on a new aspect of the program, the parameters of its design is decided and agreed upon. Any work that is done outside of the group meetings is approved via online chat media on which the entire group is connected. This allows all members to be involved in decision making and prevents any members from being left in the dark in regards to new developments in our design.

As it can be seen, each team member is assigned at minimum one task per week. The app's navigation is on the forefront of the group's priorities, but tasks become more individualized as the project moves along in development. The File I/O system and the UI are among the first things to be completed. After they are developed, we move into the phase where most of the features the user interacts with will be implemented. After a few weeks, the graphing screen will be finished, and the user will be able to see data coming in from the gas analyzer in real time, and will also be able to adjust the look of the output. The Email integration will be completed within the same time frame, allowing the user the ability to transfer the data out of the app. A screen displaying all of the stored data will be developed, and will allow the user to perform various commands on them (e.g. email, delete, view). The app's functionality will be fully written and implemented by early March. After these features are implemented, the group will being the testing and debugging phase, where we will fix any issues occurring within the app as well as improving the polish on the UI.

6. Conclusion

The goal of our software is to aid in the gathering of field data for ecological research at Northern Arizona University. This plan has been written to ensure that our product is both comprehensive in how it solves problems for the client, and designed in a way that allows for simple implementation and maintenance. Through the use of this design, we plan to have a smooth and well defined implementation process that minimizes risk and maximizes team productivity.