Software Design Document

Version 1.0

Team Members: Joseph Griffith, Robert McIntosh, Brandon Samz, Corban Stevens

Project Sponsor: Gary Matsch and Harlan Mitchell

Faculty Mentor: Austin Sanders

Created: 2/07/18

Revised: 2/25/18

Table of Contents

1. Introduction

Every day, over one hundred-thousand flights are scheduled across the globe. With so many flights occurring everyday, accidents are inevitable. In 2016 alone, there were sixty five accidents on commercial airlines resulting in ten deaths. Accidents can happen for a number of reasons, most of which are out of the control of aircraft operators and engineers. However, it is the responsibility of the aircraft engineers and operators to minimize the risk of failure as much as possible. We are focusing our concerns on the needs of the engineer, and any engineer will inform the best way to see a problem in your system before it happens is to collect ample operation data.

Technology today is data driven, and, as software developers, data is paramount to building and maintaining successful solutions. Our team (BlueSky Group) has been tasked with assisting our client in their goal of obtaining more data. Gary Matsch and Harlan Mitchell with Honeywell Aerospace develop turbine engines and engine control systems for a myriad of private jets. These engines and their connected systems generate data every flight, data that is paramount to the reliability of their product. Currently, this data is downloaded from a computer that monitors the engine's performance through a wired connection. Honeywell Aerospace technicians periodically connect to this Engine Control Unit (ECU) and retrieve the data as often as they can. However, this process does not happen frequently enough. The cumbersome process of physically connecting to a computer and downloading this data on location greatly limits the amount of flight data that Honeywell can collect. Developing a wireless solution to this problem would allow Honeywell to collect ample amounts of data, but this solution would not bring immediate profit to their company. In other words, there is no business case to solve this problem. This is where our team comes in; our clients, Gary Matsch and Harlan Mitchell, see this as a very cumbersome and unnecessary process in an age when everything is wireless. To assist our client in proving to his superiors the importance of this solution, we at BlueSky Group will develop a prototype that solves this problem. With this problem solved Honeywell will be able to market their solution to anyone who owns a plane and sell this product. Currently there is already a wireless solution out there that performs a similar function, but it is bulky and still requires a computer and a lot of setup time. With our solution Honeywell will be able to complete with this other wireless model of data transfer and expand their already large client base.

Our prototype will take the form of a mobile app that uses Bluetooth to connect to the Engine Control Units, download their data, and process it. Currently, Honeywell technicians use an archaic Engine Management System called EEI to process flight data. Once they have downloaded the data from the ECU, this system displays the data in a way that can be analyzed by the technicians. Our prototype will emulate the functions of this system on a mobile platform, allowing for ease of access to this data. Our prototype will allow for a secure and easy way to download flight data on a much larger scale meaning greater reliability for the products Honeywell Aerospace creates.

We will also ensure that our solution will meet the requirements specified in our requirements document. The key functional requirements that our solution will be able to satisfy are allow for easy access with both a smartphone a tablet to ensure that data can be access on different platforms as long as they are all running android. The engine data should also be able to be downloaded any place the plane has landed with only a smartphone or tablet necessary for the engine download. Our application will also be able to download all of the data from the ECU to ensure they we get all of the data that we need to represent an accurate picture of how the engine is performing. Our application will then allow the technician to review all of the data that was collected in a presentable fashion. The application will also communicate with the ECU over a Bluetooth connection to ensure the the engine data can only be downloaded locally and transferred in a secure measure.

Our key performance requirements are as follows. Our application will download data with a lossless protocol. This will ensure that when we are downloading and displaying our data that it is accurate and the technicians are not going to make mistakes due to misinformation. Our application will also be able to do the download within 1 minute. With this goal we will make sure that our solution is much better than the current version of EEI in which the download currently takes around 10 minutes and requires a lot of setup. Our application will also not report incorrect data. Data that is shown will be displayed in correct formats as expected by technicians, and it will be displayed in a way that is readable in understandable. Lastly, our application will be able to establish a connection between the phone and laptop in under 15 seconds. This window of time will once again ensure that our application will be better then the current version of EEI as it is now where a technician has to walk into the plane and manually connect wired into the plane in order to start the download.

The environmental requirements for our application follow as such. Our application will run on android OS. We chose this platform because it has been the easiest platform for our team to develop on. This being said our application will be able to run on all android devices. This will ensure that no matter who is running our application then if they have an android device they will be able to run our application regardless of what actual device it is. As specified by our sponsor our application will also be able to mimic the basic functions of EEI but in a different layout that is much more readable than what is

currently available. This will allow technicians to read the data they collect much more easily than before. Our application must also be able to download at least 5 dummy data sets in order to compare the data on them. This would simulate the collection of multiple data sets over time, so being able to compare the data from these files is a must for technicians working on these aircraft.

2. Implementation Overview

Our primary goal with this project is to bring Honeywell's data acquisition methods into the 21st century. Their current wired-connection methods are much too cumbersome and deprive Honeywell of their full data acquisition potential. To solve this issue we will be making a mobile application that will utilize Bluetooth technology to access flight data wirelessly. This wireless functionality will provide Honeywell a number of benefits;

1. The ability to access flight data without a wired connection
2. The ability to access flight data anywhere in the world regardless of internet connectivity
3. The ability to download all flight data kept by the ECU
4. The ability to download flight data with only minor intrusion in the aircraft's cabin
5. The ability to view and process flight data on a mobile platform

These functionalities of our system will allow Honeywell to obtain vastly larger amounts of data as well to save large amounts of time and money. To implement these functionalities we will utilize five technologies: Linux Virtualbox, Python, Android Studio, Bluetooth, and MPAndroidChart.

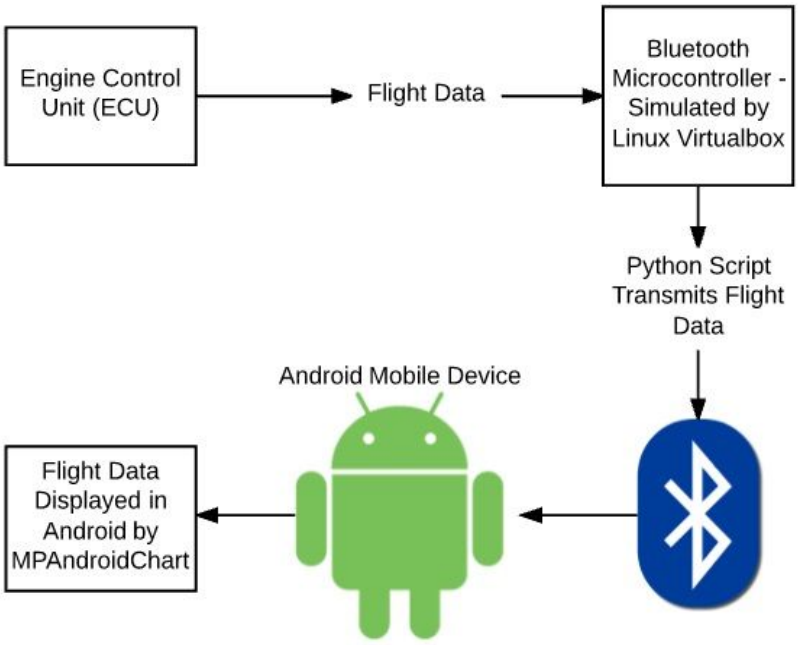

Figure 1: *Overview of System Functionality*

In order to test the previously listed functionalities in our system we have to ensure that we come as close as possible to reality. Honeywell turbine engines use Linux for their ECU's and so we must ensure that our system receives its flight data from a linux

system. To meet this demand, we will store and transmit our flight data from a Linux Virtualbox. To transmit the flight data wirelessly, we will use Bluetooth to form the connection and a simple Python script to initiate the data transfer. Once this process is completed, we will have met the first four previously mentioned functionalities. To achieve the fifth functionality, we must parse the data in the background of our Android application and display it in the foreground using the library MPAndroidChart. The following sections will expand upon the interaction of these technologies.

3. Architecture Overview

The architecture of our system consists of four main sections each utilizing the aforementioned technologies; the python script for sending files hosted in the Linux Virtualbox, the Bluetooth functionality modules, the file parser, and the display modules.

3.1. Key Requirements:

- *Linux Virtualbox Send File*: The only module in our Linux Virtualbox is responsible for awaiting a Bluetooth connection and sending the stored data file across the Bluetooth socket upon connection.
- *Bluetooth Functionality*: This module is responsible for displaying a list of available Bluetooth connections and previously paired devices. Upon selection of a device this module forms a connection between the two devices.
- *File Parser*: This module takes a file stream from the Bluetooth socket and parses it into a number of different data structures to be stored.
- *Display Modules*: These modules take in the corresponding data structures necessary for displaying their data using MPAndroidChart libraries.

The biggest architectural influence for this project is the software we are attempting to recreate, EEI. This software is old and cumbersome but the fundamentals of it are fairly simple. EEI allows the user to view a myriad of system info, and we intend to come as close to its functionality as possible without keeping its outdated appearance. That being said our application does follow the same architecture with the minor tweaks of how the data is collected and the platform the system is being developed for.

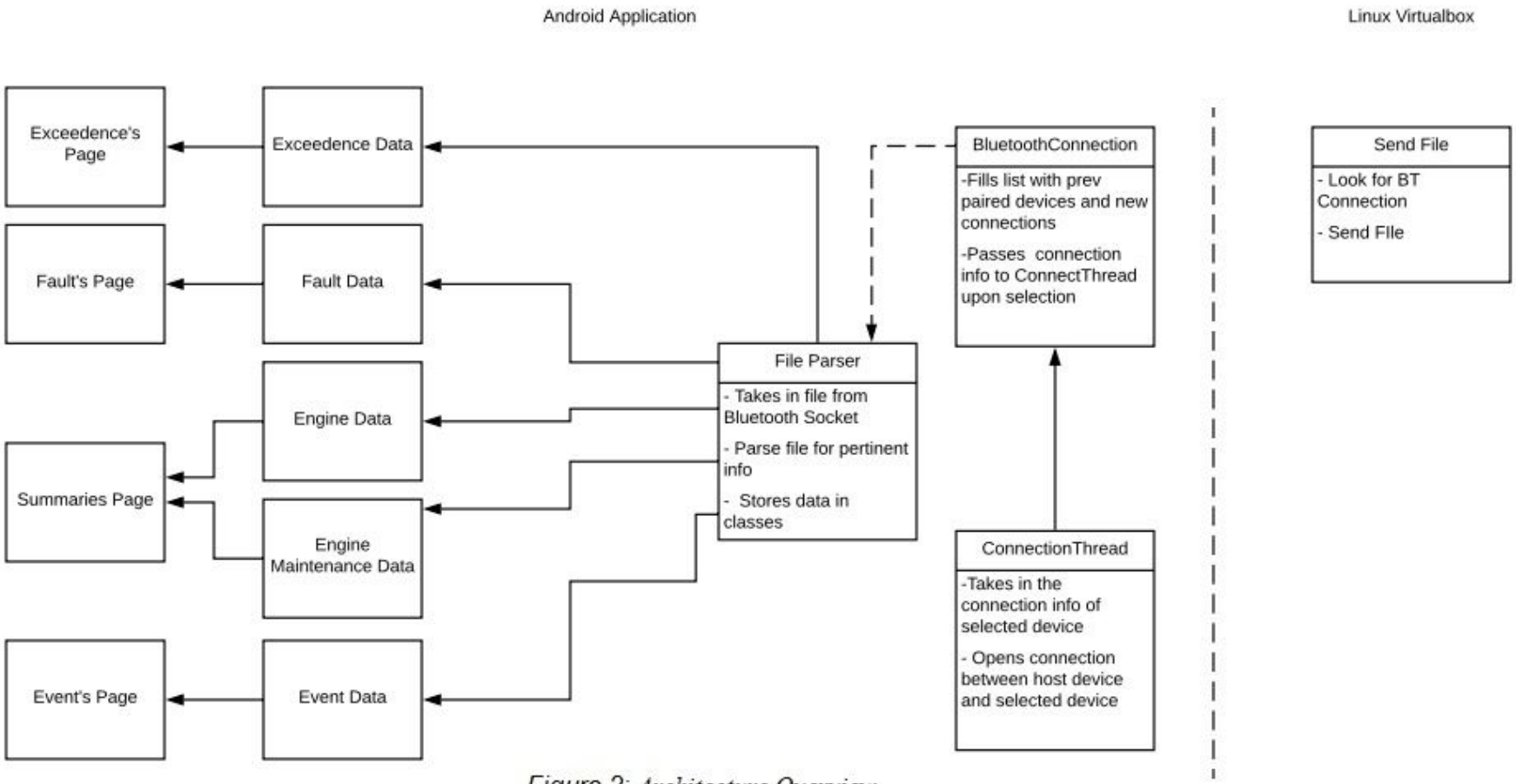

Figure 2: Architecture Overview

4. Module and Interface Descriptions

In this section we will illustrate the finer points of our systems architecture. The following sections are separated by the GUI pages within our application. Each module defines how the information is displayed and where its information is coming from. In this manner we will present the inner workings of our application

4.1. Module: Faults Page

The faults page is responsible for displaying faults read from the ECU data in an easy-to-read table format. This table should display the FF Fault ID, description, and time of each fault. Generally, this module interacts with the FileParser module by reading the list of faults that are stored into a Fault class. This list of faults is then used to build the table.

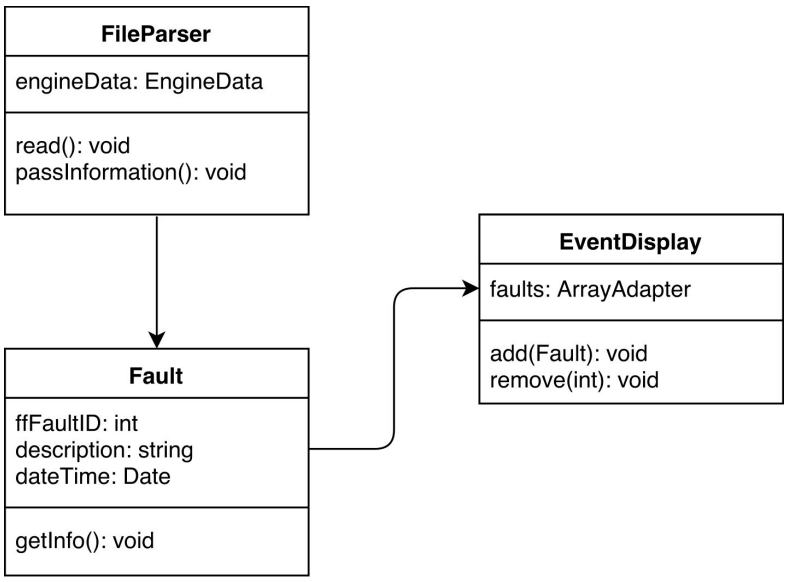

Figure 3: UML Class Diagram for Faults page.

Due to the nature of the faults screen, there is not much the user can interact with or modify. In general, the public interface is the EventDisplay, which displays fault data in a table format for the user to view. These faults are displayed using an ArrayAdapter, which will create the table that displays the FF Fault ID, description, and time of each fault. Should the list be long enough, users will be able to scroll through the list to view the full list of faults.

4.2. Module: Events Page

This component of the application will fill the role of displaying all of the events recorded by the ECU. These events are one of the major things that technicians look at when they are trying to diagnose what is wrong with an aircraft. These events are anything

that the ECU deems a significant change to the engine while it is running. This means that if there is a change in anything that is not recognized as “normal” by the ECU it will flag what is happening and start taking information down about what is happening the time it started and the time it ended. This component will fit in with the larger architecture because it is one of the main pages that we will have available for users to utilize while they are analyzing data from that they have just downloaded. This module will also receive data from the file parser class which will be given by way of passing a message and creating a new class every time a new one is read from the file.

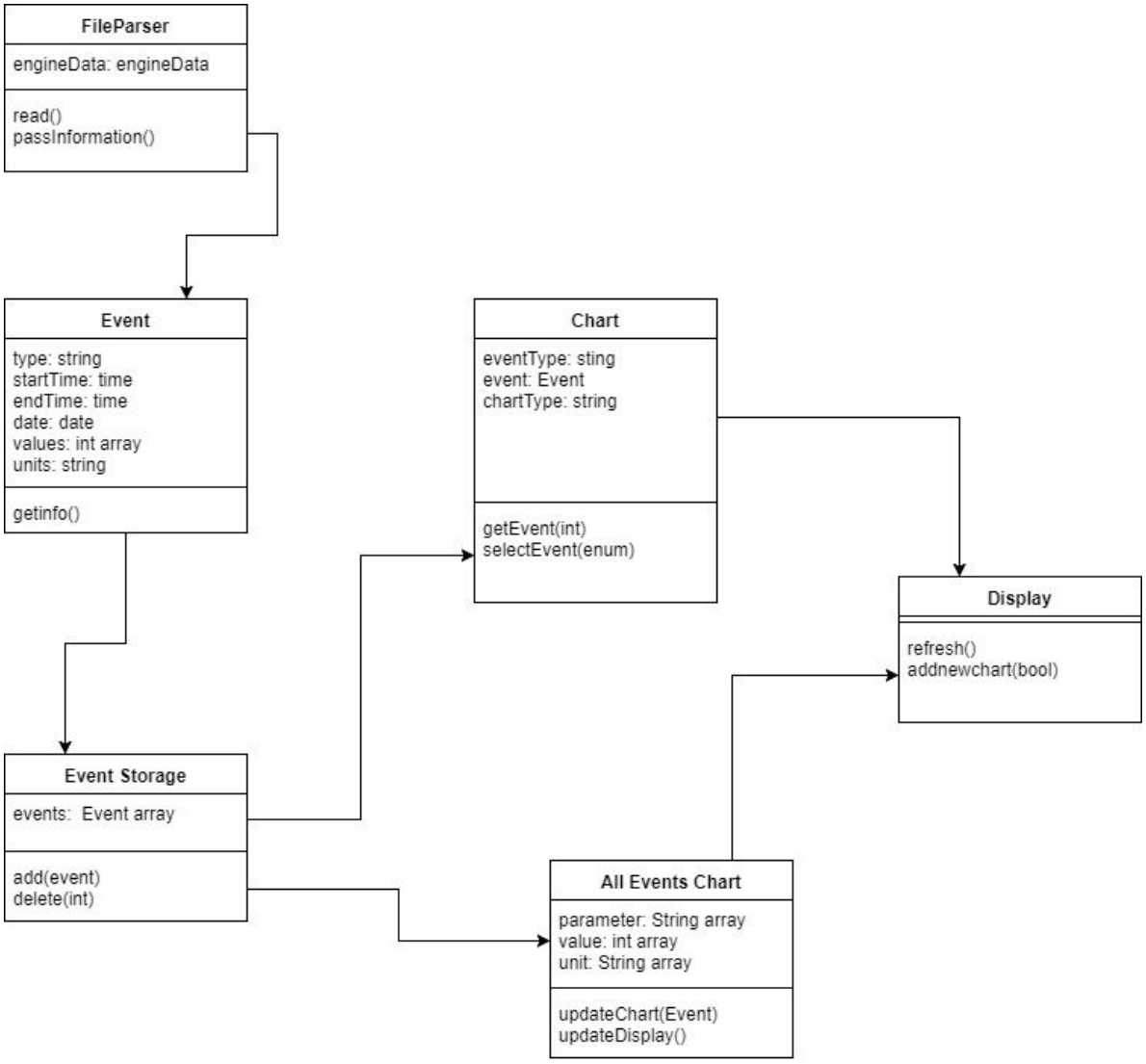

Figure 4: Events UML

Our public interface that the users interact with will be with the display, the chart, and the all events chart. When it comes to interacting with each of these, the user will not be inputting any information of their own on this page. The user will only be able to select what they want to do based on the options that they are given. This means that the parameters for any of the functions involving display will only have a few set options.

The only interaction with the All Events Chart will be a scroll bar. This chart just displays all of the events meaning what they are, the abnormal value received, and the units that go along with that value.

The next part of the module that the user will be able to utilize is the individual charts that display a graph of the event. These graphs will display the value in question over the time of the event occurring. The chart will start blank and the user will be able to select from the available events which one they want to display. This will use the selectEvent function. This function will have an enumeration of all the available events for the user to choose. Once the user chooses one the chart will then display this data as a graph to the user.

The last part of this module that the user will be able to utilize is the ability to select the chart currently being displayed. Since we do not know if the user running our application will be on a mobile phone or a tablet this will allow the user to select what information they need to view. The user will interact with the display class here and choose if they want to view a different chart if they do then the display will refresh bringing up that chart to the user.

4.3. Module: Summary Page

The summary page offers Honeywell technicians with a basic overview of engine data and engine maintenance data. The summaries page also allows the technicians to add comments to the data from this page,

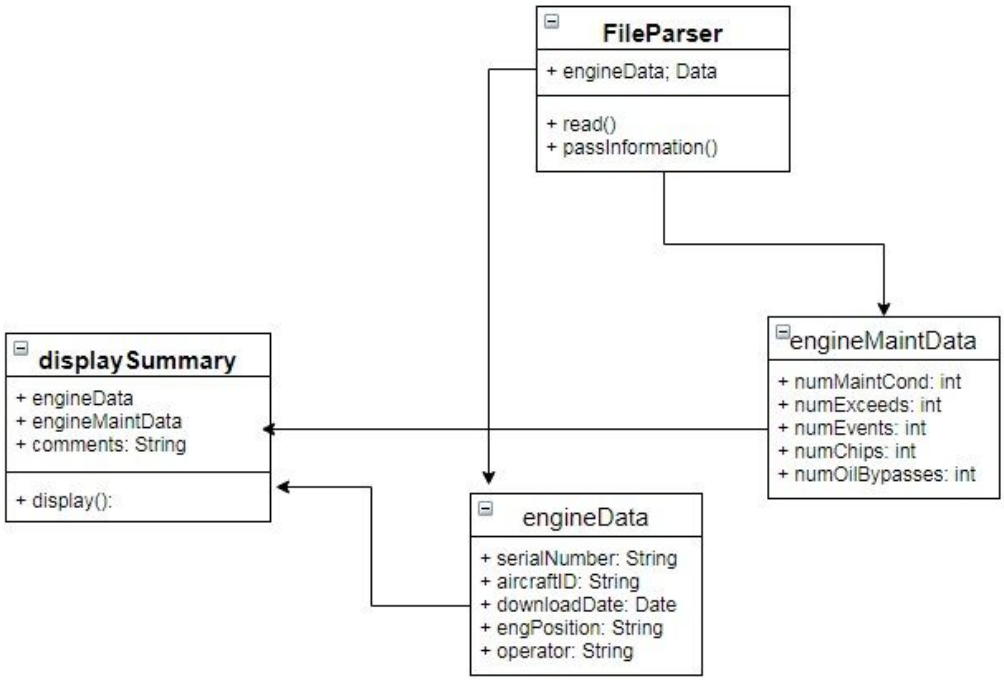

Figure 5: Summary UML

The summaries page contains two tables, one for engine data and one for engine maintenance. The engine data is akin to a configurations page, it contains info on the type of engine and aircraft the technician is analyzing. The engine maintenance table contains the real summary of the page. From this page the technician can instantly see if any serious problems occurred during the flight. If the technician notices that the number of exceedances and events is more than zero, he knows to visit those pages and investigate further

4.4. Module: Bluetooth Connectivity

The Bluetooth module for our app is responsible for connecting to the Linux Virtualbox as well as finding and displaying the local available Bluetooth connections. It uses one small utility file named ConnectionThread. When the user interacts with this page they see a list of available connections and connections remembered by their device. They can tap on any of the devices on the list to establish a connection.

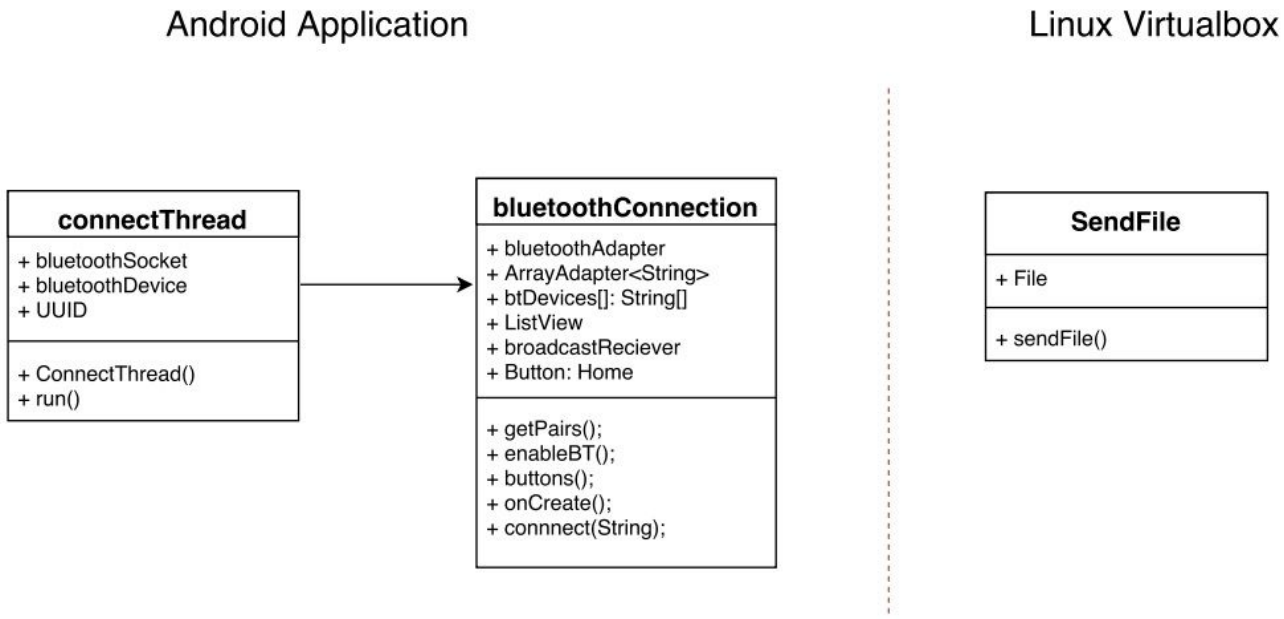

Figure 6: Bluetooth Connection Module

Before the BluetoothConnection class can initiate a connection there are a number of steps that need to be taken. First, the class calls the function enableBT() which prompts the user for the apps permission to use the host devices Bluetooth capability. The next step is to fill the list with the devices that the host device has paired with in the past. The function getPairs() accomplishes this task by querying the host device for an array of remembered devices, when the host device returns the device info the getPairs() function adds them to the ArrayAdapter which is then displayed in the list. The final step of setting up Bluetooth is to search for new devices, this is accomplished by the broadcastReciever. As the broadcastReciever finds new devices it passes them to the

onRecieve() function which adds them to the list in a similar fashion to the getPairs() function. When a user selects a device from the list the buttons() function passes the devices name and mac address to the ConnectThread() function. This function stores the host devices mac address and uses these two addresses to establish a connection by calling the run() function. Finally, when a connection is established the Linux Virtualbox automatically sends its file via the Bluetooth socket so it can be stored and parsed.

4.5. Module: Exceedances Page

The exceedances module in our app will show if a parameter for the engine reached above a certain threshold. Technicians will be able to see what these exceedances are and view the details regarding them. These details include the time and date that an exceedance occurred at and duration that exceedance occurred. The data for this page is obtained from the FileParser.

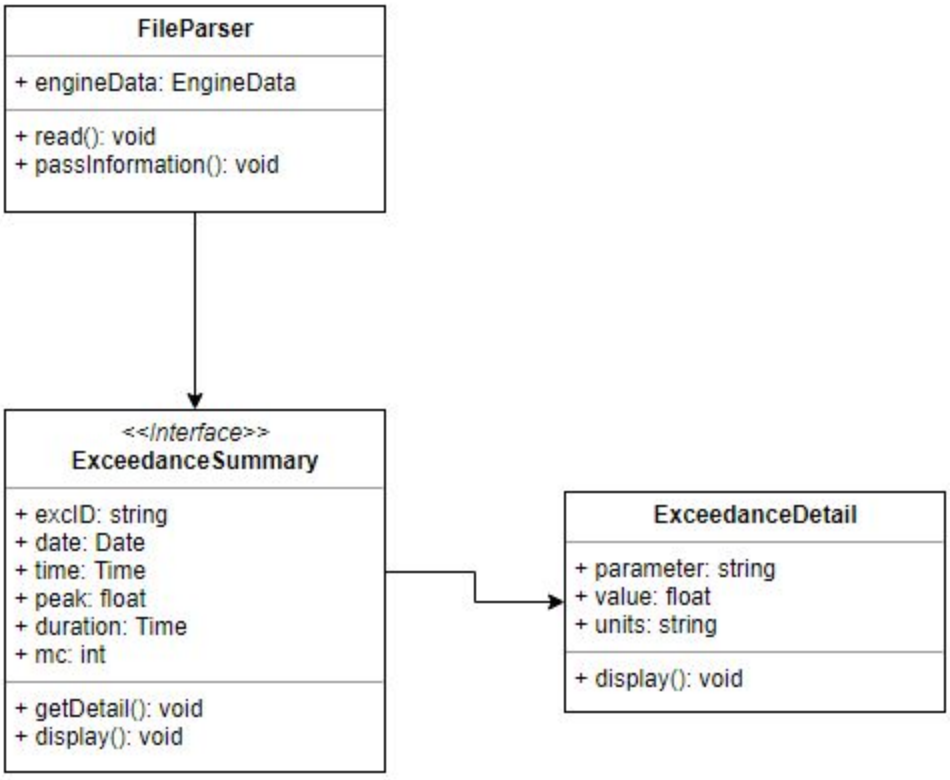

Figure 7: Exceedances UML

There is not that much that can be changed about this page. The only interactable object on this page will be for if a technician wants to view the details of an exceedance. There will be a scrollbar on the ExceedanceSumary and the ExceedanceDetail pages if the items for these lists become long enough.

5. Implementation Plan

With the module design in mind, the team was able to plan out the schedule for implementing each module. The focus was to ensure that important modules and modules that are relied upon by others were implemented first. The second focus was to schedule tasks that can be worked on in parallel as much as possible, in order to maximize group productivity. Below is the Gantt chart outlining the implementation plan and timeline.

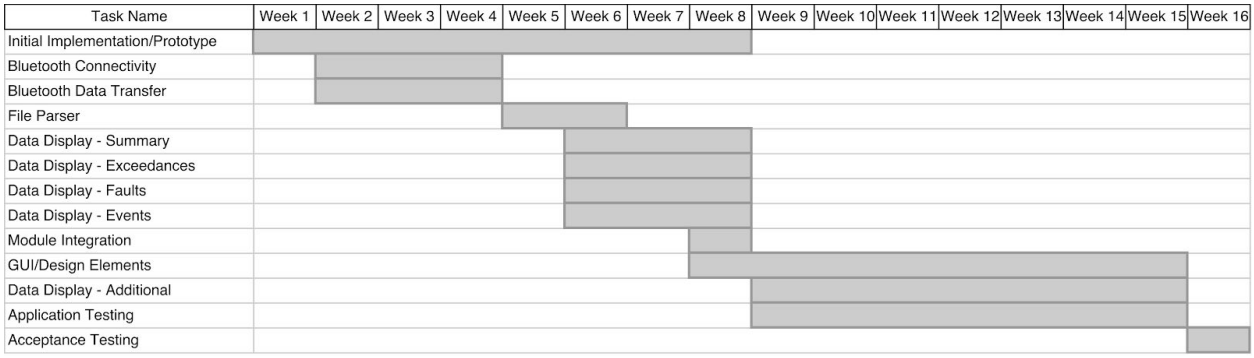

Figure 8: Gantt Chart for Implementation Plan Timeline

The initial eight weeks of the semester will consist of implementing the application into a working prototype. The initial focus (spanning Weeks 2-4) will be ensuring that Bluetooth connectivity and Bluetooth data transfer work correctly, as these modules cannot be tested via emulator. Once the Bluetooth functionality is working, the team will finalize a format for data files so that the file parsing module can be implemented starting in Week 5.

After work on the file parsing has begun, the team will be able to begin working in parallel on the various data display screens which will span Weeks 6-8. As these screens only receive data from the file parse module, and not each other, they can be implemented in parallel without any conflict. Once these various pages are working, we will begin to integrate the various modules in Week 8, which should consist of ensuring that each page is accessible and that data is being passed properly to each module.

Once basic functionality is implemented, the team will begin refining the GUI and other design elements beginning in Week 8 and continuing through Week 15. For the most part, these should be tweaks that make look application look and feel better, so they have a lower priority than modules that provide actual functionality. Similarly, the team will begin extending and adding functionality beginning in Week 9 and continuing through week 16. Functionality added in this stage is not the team's main focus and will be added (time permitting) throughout the testing stage.

In Week 9, the team will begin the testing phase, which will last through Week 15. The application will be thoroughly tested to ensure that all requirements are met. Each

module will also be tested to ensure that all output is accurate and provides expected results. Testing will be performed while keeping acceptance testing (Week 16) in mind, to ensure the final product passes the acceptance testing.

6. Conclusion

Many people rely on aircraft to get from one place to another in a timely manner, so making sure that they are safe to use is a critical task for many people. Engineers and maintenance personnel need to make sure that the airplanes that they are working on do not have any mechanical failures or faults that could result in an accident. To do this, technicians currently use a piece of software called EEI to help identify problem areas. Unfortunately, this piece of software is old and cumbersome. Blue Sky Group will provide the owners of EEI, Honeywell, with a mobile application prototype to aid with the modernization of EEI. The main task that Honeywell wants our team to accomplish is to be able to download a file from a Linux computer onto a mobile device using Bluetooth. To accomplish this task, our team will be using a python script on a Linux virtual machine that will send a file over Bluetooth. This file will then be received by a mobile device running android. This file will then be parsed and the data that it contains will be used to create graphs and to fill tables so that airplane technicians will be able to easily read the data and figure out what is wrong with the aircraft.