BlueSky Group

Technological Feasibility Analysis

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

Project Sponsor: Harlan Mitchell

Faculty Mentor: Austin Sanders

Created: 10/24/17

Revised: 10/26/17

Table of Contents

1. Introduction

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. Harlan Mitchell and Honeywell Aerospace develop turbine engines and engine control systems for a myriad of private jets. These engines and their connected systems generate ample 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 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 client Harlan Mitchell sees this cumbersome wired process as an embarrassment to the company 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 be developing a prototype that solves this problem.

Our prototype will take the form of a mobile app that uses bluetooth to connect to the Engine Control Units and 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 safe and easy way to download flight data on a much larger scale meaning greater reliability for the products Honeywell Aerospace creates

2. Technological Challenges

A majority of the challenges we face concern connecting to the Engine Control Unit and downloading its flight data. First, we need to determine what technology we will use to transmit this data. The long term plan for Honeywell is to install Bluetooth microcontrollers on all of their aircraft. So it is likely that our solution will use Bluetooth, however we intend to explore all possible options. Next, we need to determine how we are going to simulate the aforementioned process without direct access to an aircraft. For testing purposes, we need a solution that is accessible while still using a Linux OS. The Honeywell microcontrollers will use a Linux OS so it is imperative that our tests do as well. Once we have determined what Linux OS platform we will use we need to determine what programming language will be used to transmit the flight data. After overcoming the data transmission challenges, we can turn our focus to the functionality of our mobile application. The application we build needs to simulate Honeywell's Engine Management Software EEI. So, for our final challenge we have to choose a mobile platform and a means of displaying the flight data that is similar to EEI.

3. Technological Analysis

After reviewing the technological challenges, five key pieces of implementation have been identified, all of which have various options. These technological issues and their options for implementation are as follows:

- Bluetooth, WiFi or NFC for communication between the application and plane

- Raspberry Pi or Linux Virtualbox for the platform that the test program runs on
- Programming language the test script should be written in
- iOS or Android for implementation of the application
- Library to display data within the application

These five issues will be discussed further in the following sections, with various options being weighed.

3.1. Bluetooth, WiFi, and NFC

This section deals with what technologies we are going to use in our application that will allow for wireless communication between our app and the plane. This technology will need to be be able to download a 500MB file in around 4 or 5 minutes over a wireless signal. The current version of Honeywell's system takes much too long to download data over a wired connection, and we want to improve the download speed as well as make the connection wireless. The connection also needs to be secure. Our client has expressed the need for our connection to be secure so that other people will not be able send commands to the plane or retrieve data without the proper permissions. Lastly, the connection also needs to be usable no matter where the plane has landed therefore the technology needs to be able to be used anywhere in the world.

3.1.1. Bluetooth - The first technology that we looked into was Bluetooth. Bluetooth is a low power wireless connectivity technology used to transfer data and broadcast information between two devices. This option is appealing to us because we believe it meets all of our criteria that we need for our connection.

- *Speed -* Depending on the version of Bluetooth that we are able to use, Bluetooth has solid all around speeds. Bluetooth 3.0 and 4.0 both have a max speed of 25Mbit/s and Bluetooth 5.0 has a max speed of 50Mbit/s
- *Security -* Bluetooth uses encryption when transferring data from one device to another, this built in security is very appealing to us because it means that we will not have to add extra security measures into our application because the connection by itself is inherently secure.
- *Availability -* If we choose to use this technology, then the device that we put onto the plane will require Bluetooth capabilities along with our application. This means that we will be able to establish a connection anywhere in the world because the connection will only be between the device and the place as long as both the plane and the device have Bluetooth capabilities.

3.1.2. Wifi - This is a technology for wireless local area networking with devices that are able to have a wireless internet connection. WiFi is in almost every modern device and it is very commonly used when two devices need to communicate with each other like ours do for this project.

- *Speed -* When it comes to WiFi speeds there is a high degree of variation. The speed at which we can transfer data depends on the speed of the connection that the plane has with the outside world. From our research we have gathered the the max speed is around 15 Mbps

- *Security* - WiFi by itself is not secure at all. It can be intercepted by other people without too much trouble. If we were to use WiFi then we would have add in our own security measures which would add on more work to our project in total.
- *Availability* - In order to use WiFi the device needs a network signal from an internet provider. This means that our application would only be able to work in places where our device can access a network signal. Unfortunately, a network signal is not available everywhere on the planet. This means that our application would only be able to work in certain areas of the globe where there is a network signal.

3.1.3. NFC (Near Field Communication) - This technology is a set of communication protocols that enable two electronic devices to establish communication by bringing them close together.

- *Speed* - NFC provides a low speed connection. Through research we have found that the max NFC speed is around 424Kbit/s which is extremely slow compared to the other options.
- *Security* - NFC connection is based off of a contract-less encryption. This means that while the signal itself is secure between the two devices, anyone is able to communicate with that device. This means that someone could communicate with the device without expressed permission.
- *Availability* - NFC communication is always available as long as both devices have NFC capabilities. This also means that NFC will be available to use anywhere in the world.

	Max Speed (Mbytes /s)	*Security*	*Availability*
Bluetooth	3.125 – 6.25	Built in	everywhere
WiFi	15	none	Only where network signal is available
NFC	0.05	Built in	everywhere

Table 1: Technological feasibility of Bluetooth, Wifi, and NFC;

Chosen Approach - After looking at our options of available technologies our group has decided that we will be using **Bluetooth** because it is the most flexible option and it meets all of our needs. The speed it offers while it is not the best is still enough for us to use. It also is secure by default which means we do not need to add any extra layers of security although it may be something we explore if we have time. Lastly it also will be available anywhere on the planet because both the device and the plane will have Bluetooth capabilities. With Bluetooth meeting all of these requirements it is the obvious choice out of the three, and we think that it will be the best choice for us to use in our project.

Proving Feasibility - The team plans to prove the feasibility of using Bluetooth in combination with other necessary components. Since the team will be proving feasibility of writing a simple Bluetooth script to communicate with a test unit, a large (500mb or so) file will be sent as part of the process. This will prove that this amount of data is able to be sent without any issues and over a reasonable amount of time.

3.2. Raspberry Pi vs. Linux Virtualbox

An integral part of our prototype is emulating the microcontrollers that will be placed onboard the aircraft. Our mobile app will use bluetooth to connect to these microcontrollers and download the flight data. For the purposes of our prototype, we need to emulate this process in a way that can be demonstrated and tested without direct access to an aircraft. These microcontrollers will use Linux as their operating system, so it is imperative that our prototype does as well. Above all else, our solution to this challenge must be bluetooth enabled. The two solutions to this challenge that our team has considered are the Raspberry Pi and a Linux Virtualbox.

3.2.1. Raspberry Pi - The Raspberry Pi is a compact microcomputer very similar to the ones that would be installed on board the aircraft. The Pi has had three iterations each building on the functionality of the last. However, for our prototype we have narrowed down the options to the two latest Raspberry Pi releases.

- *Raspberry Pi 2* - The Raspberry Pi 2 does not come with Bluetooth built in but a "Bluetooth dongle" can be bought for $10- $13 and inserted into the usb ports. One can check if the device is being recognized by typing "lsusb" into the device terminal. If the raspberry pi is not recognizing the Bluetooth dongle use "update" in the terminal to update the system. From here all one need do is use the following command to install Bluetooth capability.
  ```
  "Sudo apt-get install bluetooth bluez-utils blueman bluez
  python-gobject python-gobject-2"
  ```

- *Raspberry Pi 3* - The Raspberry Pi 3 comes with wifi and Bluetooth 4.1 enabled. However some steps must be taken to enable its use on the device:
  1. From the Raspberry Pi desktop, open a new terminal window.
  2. Type "`sudo bluetoothctl`" then press enter and input the administrator password
  3. Next, enter "`agent on`" and press enter. Then type "`default-agent`" and press enter.
  4. Type "`scan on`" and press enter one more time. The unique addresses of all the Bluetooth devices around the Raspberry Pi will appear and look something like an alphanumeric `XX:XX:XX:XX:XX:XX`. If the device to be paired is set as discoverable (or put into pairing mode), the device nickname may appear to the right of the address. If not, a little trial and error or waiting to find the correct device will be necessary.
  5. To pair the device, type "`pair [device Bluetooth address]`". The command will look something like `pair XX:XX:XX:XX:XX:XX`.

3.2.2. Linux Virtualbox - The Linux Virtualbox would allow all of our team members to test our prototype without having to purchase a microcomputer such as the Raspberry Pi. Furthermore, this solution would provide ease of access to a Linux system as none of us currently own one. That being said it is imperative that the Virtualbox is capable of using its host machines bluetooth capability. The following instructions verify that the Virtualbox can achieve this requirement.

- *Enabling Bluetooth on Linux Virtualbox*
    1. Disable Bluetooth Adapter from Device Manager in Host Machine.
    2. Disable all services in Host Machine that are using Bluetooth. (TaskManager -> Services -> Press B to find Bluetooth Services & Stop).
    3. Start Virtualbox Ubuntu.
    4. Enable Bluetooth Adapter from Device Manager in Host Machine.
    5. In Virtualbox, go to Device -> USB -> Select Bluetooth Radio.
    6. Open terminal to check Bluetooth status: "`sudo/etc/init.d/bluetooth status`"

- *Sending A File to a Bluetooth Device*
    1. Open the Activities overview and start typing Bluetooth.
    2. Click on Bluetooth to open the panel.
    3. Make sure Bluetooth is enabled: the switch in the titlebar should be set to ON.
    4. In the Devices list, select the device to which to send the files. If the desired device is not shown as connected in the list, it needs to be connected (a panel specific to the external device appears).
    5. Check Send Files and the file chooser will appear.
    6. The owner of the receiving device usually has to press a button to accept the file. The Bluetooth File Transfer dialog will show the progress bar. Click "close" when the transfer is complete.

Chosen Approach - Since both of these solutions meet the requirements of Bluetooth capability and a Linux OS our decision ultimately comes down to what is the most accessible for our team. The Raspberry Pi is relatively affordable and closer to what will be installed on Honeywell's systems however the setup of a Raspberry Pi is a long and cumbersome process. Furthermore it would require us writing a script that connects to a Bluetooth device and sends the flight data file upon booting the system. We feel that for this reason as well as the cost of purchasing four Raspberry Pi's that the **Linux Virtualbox** would be much more accessible. The Linux Virtualbox would allow each of us to test our implementations from home and ultimately focus more time on adding functionality to our app. However, if we succeed in developing the main functionalities of EEI in our mobile app ahead of schedule then we would like to develop the Raspberry Pi approach in order to better demonstrate our prototype.

Proving Feasibility - Our team will test Linux Virtualbox's effectiveness by using the aforementioned steps to enable Bluetooth connection with a mobile device and send a basic file. This will demonstrate Linux Virtualbox's capability to emulate the Bluetooth Microcontrollers Honeywell will be installing in their aircraft.

3.3. Test Script Language

As we are not provided an actual Engine Control Unit with Bluetooth capability, it will be necessary to create something to simulate the engine. This test unit will need to be able to connect to our application via Bluetooth, receive a Bluetooth request for data, and send a stream of data back to the application, again via Bluetooth. Additionally this test unit will need to either generate test data in a format determined by the group or read test data from a file (in this same format determined by the group). This test data is the data that will be sent to our application. Ultimately this test unit will serve to demonstrate the capabilities of our application, but is not the main focus of the project. The decision for implementing the test unit comes down to which language a script will be written in to implement these Bluetooth commands and functionality, and the options are C, Python, and Java.

3.3.1. C

- *Dependencies* - BlueZ is required for programming a Bluetooth application in C in a Linux environment. As this test unit will be used for our own demonstration purposes, it will be our responsibility to set up the environment required for this approach.
- *Documentation and Examples* - Very little documentation is provided for the BlueZ library itself. However there are examples available for creating a Bluetooth connection using C.
- *Code Difficulty* - In order to create a connection, it is necessary to open a socket using the Bluetooth adapter which is done similarly to creating a TCP socket in C, along with the Bluetooth library itself. Once this socket is created, sending and receiving data is done identically to a TCP connection in C. It is not immediately clear how the UUID is set, which is used when connecting via the Android application. Overall fairly straightforward for those familiar with network programming, but has some confusing and involved pieces.

3.3.2. Python

- *Dependencies* - Python distutils, BlueZ, and Pybluez are required for programming a Bluetooth application in Python in a Linux environment. As this test unit will be used for our own demonstration purposes, it will be our responsibility to set up the environment required for this approach.
- *Documentation and Examples* - PyBluez provides documentation for the classes that it provides, as well as examples for creating a Bluetooth connection using Python.
- *Code Difficulty* - Again, a socket is created using functions provided by the PyBluez library. Once this socket is created, PyBluez provides a function to advertise the services to nearby devices, and a UUID can be provided. Once a connection is accepted there are simple functions to send and receive data. All functions are well named and simple, which means creating a connection and sending data is able to be done without a lot of code.

3.3.3. Java

- *Dependencies* - BlueZ and BlueCove are required for programming a Bluetooth application in Java in a Linux environment. As this test unit will be used for our own demonstration purposes, it will be our responsibility to set up the environment required for this approach.
- *Documentation and Examples* - The classes implemented within BlueCove have their own documentation, however examples using BlueCove are harder to find and don't have very much explanation.
- *Code Difficulty* - Similar to the other approaches, it seems a socket needs to be created and can then be used similarly to a TCP connection. There are, however, very few well documented examples, so figuring out the proper methods to use. Furthermore, as this

approach uses Java, everything needs to be encapsulated within a class which is then run a main() method, which adds an unnecessary layer of difficulty.

Chosen Approach - Of all these options, the best for our purposes will be writing the test script in **Python**. As our client wants this to run on a Linux environment, this will work well, as most Linux distributions come with Python as well. PyBluez provides simple interfaces and functions to establish Bluetooth connections, which means the team can focus on writing the application and not the test unit.

Proving Feasibility - We plan to test our choice by using the client and server Bluetooth example scripts provided. The team will run the server script on one machine running Linux and the client script on another. We then plan to send some random data between the two machines to show that the Bluetooth connection works and is able to transmit data as required.

3.4. Android vs. iOS

This project can be developed for one of two platforms - Android or iOS. Both of these platforms will do what we need to accomplish in order to satisfy the customer. Due to the limited scope of our project and our time constraints, we can only develop for one of these platforms. Each platform brings with it a different set of languages and tools.

3.4.1. Android - There are many devices around the world that use Android. It can be developed on any machine that can run Android Studio or Visual Studio. Developing for Android is something that our group is familiar with making this option the most appealing to us.

- *Accessibility* - Android applications are typically programmed in Java, which our group has ample experience working with it. This means that we will not have to waste time learning the ins and outs of a new language when we start prototyping and building our product.
- *Compatibility* - Android can be run on a variety of devices, meaning that our application can be used on many kinds of mobile devices. Unfortunately, this comes at a cost. Applications made for more recent versions of Android are sometimes incompatible with older versions. In order for our application to work on the wide range of devices that it will probably be deployed and used on, we will have to develop for the older versions of Android. This means that we will not be able to use some of the improvements that have been added to Android overtime. We will also not be able to support the users that have not yet updated their systems.
- *Cost* - One of the great things about developing for Android is the cost. It does not cost anything to deploy applications and no special hardware is required to begin development. All we need is a computer that is able run an IDE such as Android studio.

3.4.2. Apple iOS

iOS is somewhat similar to Android, but has some qualities that make it a more appealing development choice. The development process, however, is more difficult to begin.

- *Accessibility* - iOS applications are typically coded in Objective-C, which comes at the cost of time for our team considering we have no experience with this language. With this in mind we have elected to instead spend this time developing in a language familiar to all of us in order to implement more functionality.
- *Compatibility* - One of the biggest reasons for using iOS over Android is that the applications that we develop are probably going to work on older versions of iOS. This is mostly because Apple does not let other manufacturers use iOS for their own devices. This means that the programmer will not have to worry about it not working on their devices.
- *Cost:* A major drawback when developing for iOS, especially with small development teams, is the cost. To develop an iOS application, macOS, Apple's operating system that is used on their Macs, is required. There is no legal way to run macOS on a non-Apple computer, and doing so is a breach of their terms of service. This means that it is necessary to own a Mac to develop for iOS. Unfortunately, only one person in our group owns a Mac. Also, in order to deploy an application to the App Store, there is a $99 annual fee that one must pay.

Chosen Approach - Android will be the best development option for our group. The applications that we develop for it will be able to do what we need to do without us having to pay for computers or a license. Android development is already something that our group is familiar with, so we will not be surprised or confused while we develop our product. We will also not have to learn a new programming language. Developing for Android provides us with a low cost solution for our project needs and requirements.

Proving Feasibility - In order to show that Android is a feasible choice, the team will create a simple "Hello World" application. The team will then put the application onto an Android device in order to test it properly. This will demonstrate that the team is able to create an Android application and has the necessary environment in which to develop our final product.

3.5. Displaying Data Within Android

Once all the data is received via Bluetooth, it is important that the application displays that data in a format that is easily viewable for the user and highlights important data. Generally data should be able to be filtered on certain criteria such as type of data (i.e. which sensor the data came from), time of data (i.e. data between two dates), or any combination. Options we have found after research include two separate Android libraries (MPAndroidChart and Williamchart) or a basic table display within Android.

3.5.1. MPAndroidChart

- *Usage* - Add the library as a Gradle dependency to the project, which is standardly supported in Android Studio. This allows any updates to the library to automatically be used in the application without re-downloading the library. This library also provides support to download the .jar file which can be included in the application. This method will not break, but any updates to the library will have to be manually downloaded.

- *Documentation and Examples -* Detailed documentation via javadoc provided for the library as well as its own wiki provided with the library on Github. Additionally, the library provides source code for examples which is included in the library repository. All of these resources provide good documentation and serve to help figure out how to implement certain features that are necessary for our application.
- *Functionality Provided -* Provides 8 different chart types, value highlighting, and adjustable axes. All of these features allow for users to zoom in or out on certain displays to find data and read the data value at particular points. This functionality will be helpful for users to find the data that they need.

3.5.2. Williamchart

- *Usage* - Library can be added as a Gradle dependency to the project. This method is supported within Android Studio and allows any updates to the library be used automatically without re-downloading the library into the project.
- *Documentation and Examples -* This library provides a javadoc which documents the classes provided within the library. As this is the only documentation provided by the library, it would be necessary to experiment and read through thoroughly to implement necessary features in the project.
- *Functionality Provided* - Provides 5 different chart types. These charts provide various methods to modify the view, so features like zooming and scrolling could be implemented; however, would be more difficult to implement as they are not explicitly defined in the library.

3.5.3. Table in Android

- *Usage -* Table layouts are provided in Android and are generally used to format specific buttons and widgets on a page. This could be used to display our data by giving color to the grid lines and displaying the data on a standard table.
- *Documentation and Examples -* As this is implemented within Android, there is documentation provided by Android as well as examples from various users on the internet. As this display would be very basic, there are a large number of examples from users trying to create simple applications.
- *Functionality Provided -* Provides very basic functionality and would allow the data to be displayed without a lot of effort. The downside to this is that it is harder to format the data in a nice way that is easily readable by the user and not all information is well displayed in a table.

Chosen Approach - Of all these options, the best will be a combination of MPAndroidChart and a standard table layout within Android. The table will be necessary to display small pertinent information that may not necessarily translate well as a graph, whereas MPAndroidChart will provide more full-featured data visualization options to display the data in a timely manner to the app user.

Proving Feasibility - In order to test the chosen approach, the team plans on creating a simple test application using the example classes provided by the MPAndroidChart library. We will then modify the charts to demonstrate necessary features, such as highlighting particular data points, filtering data, and opening new screens after selecting certain data points. This will demonstrate that the MPAndroidChart library provides the functionality necessary to display data in a format that is easy for a user to understand.

4. Technology Integration

The biggest integration issue that we would face would be when the Android device has to receive data via Bluetooth from the Linux Virtualbox that is running our Python scripts. Fortunately, both of these devices will be using Bluetooth, so the transfer of data should be simple as long as we follow the Bluetooth protocol guidelines. The rest of the technologies that we are planning to integrate where meant to be used together. MPAndroidChart was built as an Android library, so having it interact with Android will be no problem. Android supports the use of Bluetooth natively. Unfortunately, many Linux distributions do not support Bluetooth out of the box. In order to use Bluetooth on a Linux machine, it is necessary to install the appropriate packages such as blueman or bluez. Python is easy to get working on Linux. In fact, it even comes preinstalled on many Linux distributions. If the distribution that we choose to use does not already have python installed, it can be easily installed using package repositories or from source.

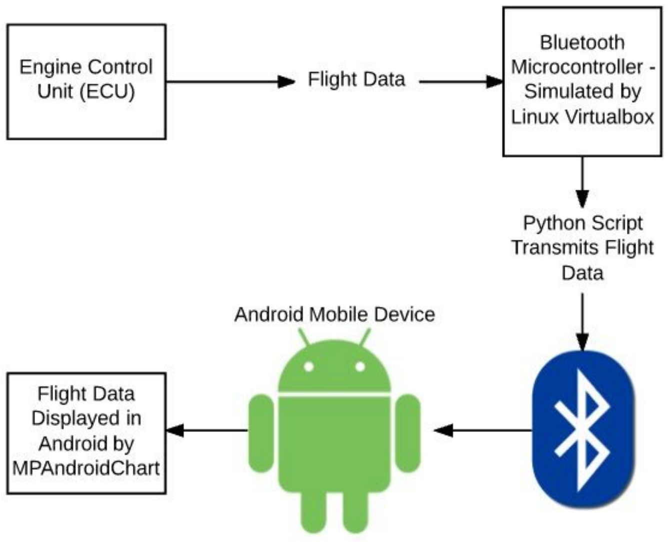
Engine Control Unit (ECU)
Flight Data
Bluetooth Microcontroller - Simulated by Linux Virtualbox
Python Script Transmits Flight Data
Android Mobile Device
Flight Data Displayed in Android by MPAndroidChart

Chart 1: Technology Integration

5. Conclusion

We are BlueSky Group, and we are creating a mobile app that will be able to wirelessly download engine data from an aircraft in a much simpler, faster, and more secure way than before. This will allow Honeywell to have a product that customers can use in that age of connected aircraft, and there will no longer be a need for bulk cables and time consuming downloads which is a current embarrassment to the company. Our application will allow Honeywell to get away from their old outdated technology and move into the new connected era of maintaining aerospace technologies. This document is intended to serve as the feasibility for our project and outline the different technological problems we plan to face and how we are going to overcome those problems.

Challenge	*Proposed solution*	*Confidence level*
Means of communication between a mobile device and the engines on board computer.	Use Bluetooth for wireless communication between our app the the plane	Strong
Determining a platform to run our tests on that simulate Honeywell's ultimate goal of installing Bluetooth microcontrollers on their aircraft.	Linux Virtualbox then if time permits Raspberry Pi	Strong
Choosing a language to write our Bluetooth data transfer script in.	Python	Strong
Picking a platform to create our mobile application in.	Android	Strong
Determining a way to display flight data that meets or exceeds EEI standards.	MPAndroidChart	Moderate

Table 2: Challenges and Proposed Solutions

Overall, we are confident in the solutions that we have chosen and I think that they will be the right choices for our project. We have put in enough hours of research that we believe each of these solutions is the best for our project. Some of the remaining open ended questions that we did not answer in this document are what kinds of libraries are we going to use for our app; however, many of the libraries that we are going to look at for the technologies we have listed are all well supported and well documented. This means that we are pretty much going to only have one choice when it comes to what kinds of libraries we will use for our project, and those are the officially supported ones.