SYSC 3010:

Final Project Report

Friday, April 7th, 2017

SYSC 3010 - Group T3

Patrick Perron

Brendan Lucas

Abdul Alnaaim

Suhib Habush

Contents

1.0 – Introduction

1.1 – Motivation

Anyone who cares about the safety of their personal property has had at least one moment where they've asked themselves "Did I leave my door unlocked?" But unless they have a trustworthy neighbor that can check it out, that thought will be nothing but paranoia for the rest of the day as they have no way of checking if they actually did. Even if they actually did leave their door unlocked, they do not have the means of actually locking it if they are sitting in their cubicle or idling in morning traffic.

In today's mobile world, people suffer from an incredible lack of control on the state of their door locks. While one has the technology to instantly speak to a relative living across the globe, they continue to use mechanical deadbolts that have been around for centuries to secure their front door. *Phantom Lock* aims to give people the mobile control they need for access to their homes. If one wants to let their mother from out of town into the house while they're at work, *Phantom Lock* provides the means of doing so. By giving a new layer of control to existing door locks, the *Phantom Lock* aims to push the common deadbolt lock into the 21st century.

1.2 – Problem Identification

Phantom Lock is an easy-to-install device that enables users to both control and monitor access to their house while on-the-go. Designed to be installed just over a deadbolt lock, this system allows users to lock and unlock doors in their home at any time from any mobile device, without affecting regular use of the door by turning the deadbolt or locking it with a key.

Phantom Lock's keypad attachment allows the use of a password to lock or unlock doors as an alternative to a key, but also provides the ability for any user to request access to a house they don't own, such as a friend's apartment. By pressing the request button on the keypad, the attached camera takes a photo of the user and automatically sends it to the owner's mobile device as a push notification, where the owner can either grant or deny access after examining the photo.

1.3 – User Interactions

The Phantom Lock system is interacted with mainly through the following interactions:

1. User locks or unlocks door with a passcode using the keypad
2. User locks or unlocks door with a key or turning the deadbolt
3. User requests access to door with keypad and camera. Door owner responds to request by providing or denying access
4. User checks state of lock while away from house
5. User lock or unlock door while away from house

2.0 – Design Solution

2.1 - System Overview

The Phantom Lock consists of 3 main components:

- An embedded system installed on a deadbolt-locked door
- A global server to monitor access to every door and store all user and current state information in a database
- A mobile application that allows users to access stored information on their doors and control the lock

Doors are also designed to be modular entities, meaning that not every door will contain the same attachments. Front doors contain a keypad and camera to allow requests to enter, while back doors simply contain the motor to turn the deadbolt and sensors to monitor the door and lock state. This gives users some flexibility in how they provide access to their home while still ensuring they can control it.

2.2 – System Architecture

Figure 1 shows the deployment diagram for the modules of Phantom Lock system.

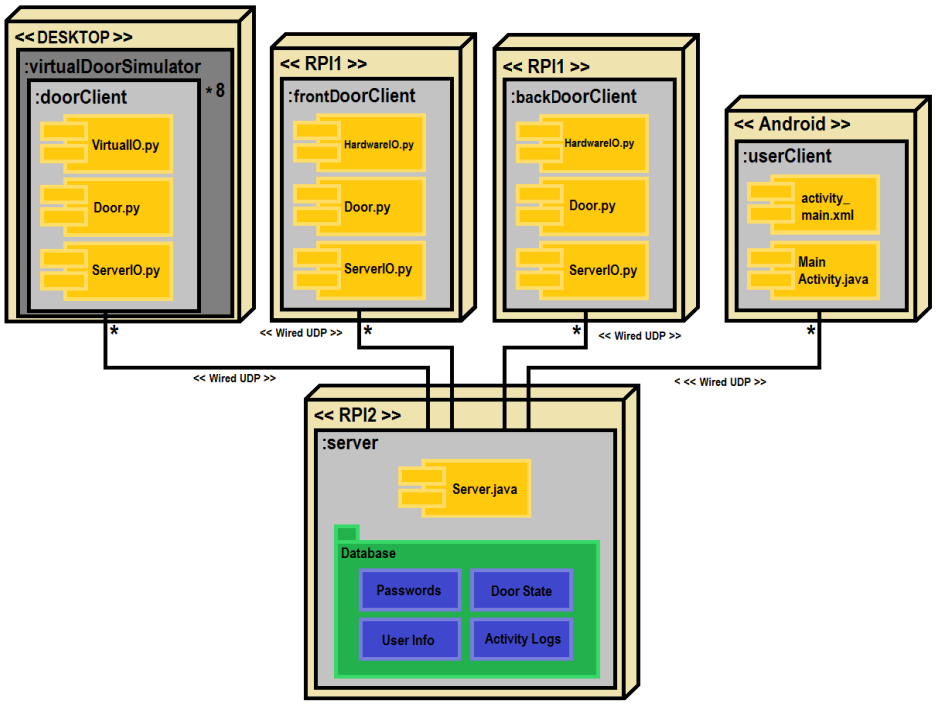

Figure 1: Phantom Lock Deployment Diagram

At its simplest, there are five different devices that make the Phantom Locks system. The server runs on a headless RPI2, and it stores the system's database that stores information on various doors, their states, their passwords along with a log of any activity in each. The Server RPI handles all the messages that are being sent to or received from clients, such as door state info or commands to lock or unlock a door.

As Phantom Lock has two types of doors, a front door and a back door, they will each be deployed differently. The front door runs on an RPI1, and contains software to interface with sensors that determine if the door is locked or open, a motor that controls the lock state, a camera to take pictures of visitors, and a keypad to take input for passwords. The back door also runs on an RPI2, but does not contain the keypad and camera interfaces. While it was originally planned to build the front door with the *GertBoard* and the back door with the *PiFace*, there was not enough time to develop this and most of the functionality worked adequately on just the *RPi1*. Phantom Lock needs to be able to handle communication from multiple doors and as only two are being built, a virtual door module is also deployed on a desktop. It contains a GUI that communicates with up to 8 instances of the door software. This will be described in greater detail in the testing section. All these doors were meant to be connected wirelessly, but due to material constraints, such as limited number of wireless adapters, and time constraints during development, all are connected via wired UDP.

Lastly, users can also use Phantom Lock with an Android application. It never communicates directly with any door, as all interactions are handled through the server. With this app, once can see the usage logs of their door and can also control the lock. This is the main way a user can control and monitor their doors. For both the Android app and Door Client connections, multiplicity is implied because the server is designed to be able to handle up to 256 doors in 256 households, and any number of clients connection with their phones.

2.2 – System Components

2.2.1 – Hardware

The following components were used while implementing the hardware:

- RPi1 and RPi3
- A 16x2 LCD
- 2-normally closed set of magnetic/door sensors.
- A Camera Module Board 5MP Webcam.
- A 3.5mm Portable Stereo Mini Headset Speaker
- A set of male-to-male, male-to-female and female-to-female cables.
- Wi-Fi adapter.
- Gertboard and PiFace.
- 2 Tower Pro SG90 Servo
- 4x4 Array Matrix Keypad
- Breadboard

These components are described as follows:

- Gertboard will be used with RPi1 to control the front door that will have a camera module, DC motor and a 4x4 Array Matrix Keypad while the PiFace will be used with RPi2 to control the back door that will have control only through the app.
- The 16x2 LCD is used to display messages and entry passcode
- Magnetic sensor is to determine the state of the door. Magnetic sensor detects the variations and disturbances in the magnetic field (Flex). Normally closed sensor is when the two parts apart, it creates a closed circuit which determines the door is open.

- The Raspberry pi camera module is used to request access by taking a picture of for, instance, the visitor and sending it to the mobile App for access validation.
- Cables are used to connect the hardware to the RPis through the breadboard that is used to allow different hardware components share the voltage source as well as the other pins.
- The servo has a neutral potion at 90° and will rotate to 180° counter clockwise representing the door's lock (deadbolt).
- The 4x4 Array Matrix Keypad is used to request access by entering the passcode.
- The portable stereo will be notify the user/visitor of whether the door being open or closed by playing sound.

The intent for LCD in this project is to display to the user the entered passcode. It requires 6 GPIO pins using male to female cables where a breadboard is used for this connection as shown in figure 1 and 2 below. This method will not steal the only serial port on the Pi.

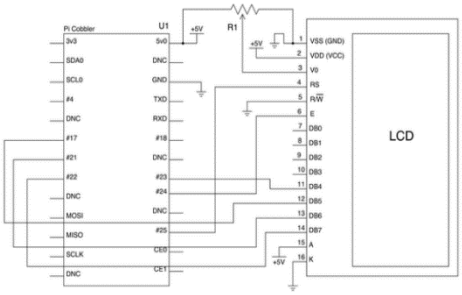

Figure 2: LCD connection

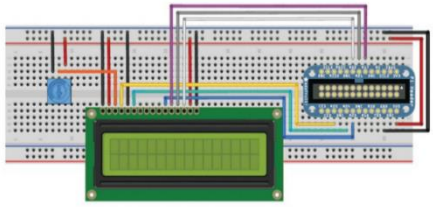

Figure 3: LCD wiring Diagram[1]

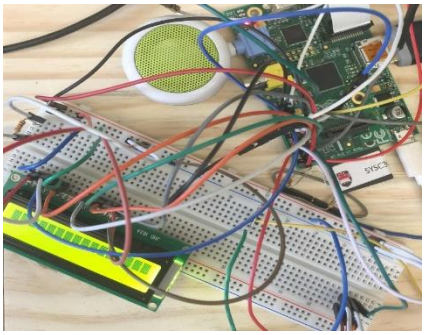

Figure 4: Actual LCD wiring [Alnaaim]

The camera module, both RPi1 and RPi3 have built-in camera port which allows us to plug and play as shown in figure 5 A and B below. Although the camera plug and play, the team is required to write code that meets the specification of the project. The intent for the camera is to validate the access request by taking a picture of the person who is in front of the door and send it to the householder.

Figure 5-A: Raspberry Pi Camera Port [2]

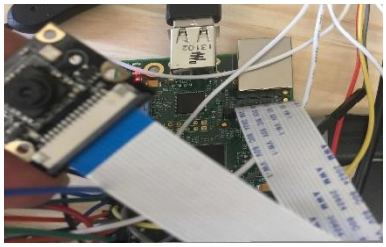

Figure 5-B: Actual Camera Plugged [Alnaaim]

The magnetic door sensor, used in this project, is a useful component in determining whether the door is opened or closed. It is made up of two components, a magnet with two metal terminals and a plain magnet. When the two magnets are close, the voltage difference between the terminals is zero, creating a closed circuit and allowing the current to flow. This is similar in operation to a switch, where the circuit is closed when the magnets are together refer to figure 6A below.

The magnetic door sensor requires three wires for the connection; one connected to the GPIO pin, one to the 5V Vout pin and the last connected to ground on the Raspberry Pi. A 1kΩ resistor was added between the voltage source and the GPIO pin on the Raspberry Pi. The resistor that was added

for such that when the circuit is closed, the voltage source is not shorted to ground, as this could damage the Raspberry Pi. The value of the resistor does not matter, any large resistor could do the job. A schematic of the resultant circuit can be seen in figure 6B below.

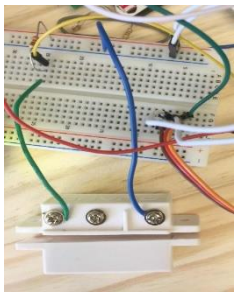

Figure 6-A: A Normally Closed Magnetic Sensor [Alnaaim]

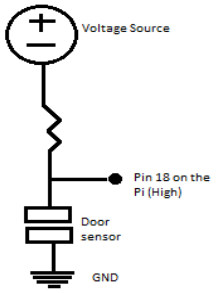

Figure 6-B: Magnetic Sensors Schematic [Alnaaim]

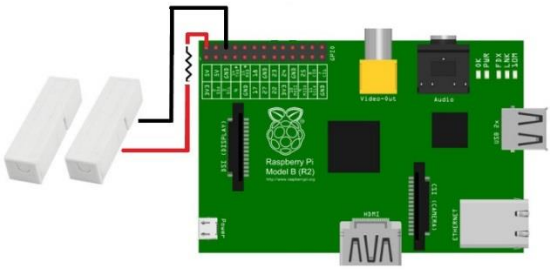

Figure 6-C: Magnetic Sensor connected to the Pi [Alnaaim]

A 3.5mm Portable Stereo Mini Headset Speaker is used for this project to notify the user of a certain event, such as access granted, by playing a sound. It is also plug and play through the use of AUX port as shown in figure 7 below.

Figure 7: A 3.5mm Portable Stereo Mini Headset Speaker [Alnaaim]

The Tower Pro Servo is used in this project to act like the door lock. It rotates clock and anti-clockwise representing the movement of the lock. The type of servo used in this project is Tower Pro S690 working at 5V DC. The servo has three wires where the orange represents the signal, red represents the ground and the brown wire is for the voltage source as shown in figures 8 & 9 below.

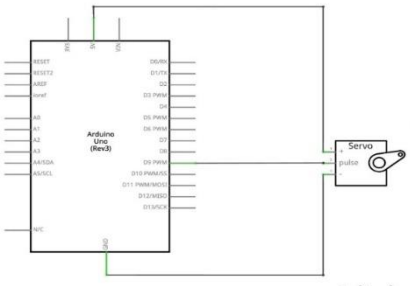

Figure 8: Tower Pro Servo circuit connection [3]

Fiaure 9: Tower Pro SG90 Servo [Alnaaim]

A 4x4 array matrix keypad (16 buttons) is a useful component that provides numeric interface for this project. It is used to enter the passcode by the user to request access to the home as shown in figures 10 & 11 below.

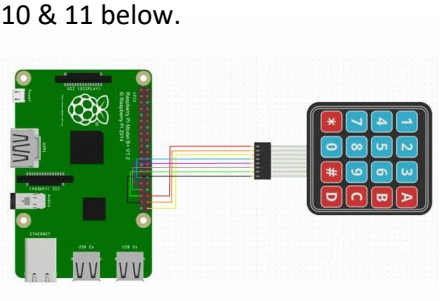

Figure 10: 4x4 Array Matrix Keypad Schematic [4]

Figure 11: 4x4 Array Matrix Keypad [Alnaaim]

The keypad utilizes 8 GPOI pins on the Raspberry Pi. As seen in figure 10 above, the connection was done through 8 male-to-female cables. 4 of them are input pins and the other 4 are output pins. It detects one entry key at a time by setting all of them low so the pressed button becomes high in the corresponding row and column. When the user passes a 4-digit passcode, a four-start digit will appear in the LCD for more privacy.

2.2.2 - Embedded Door Software

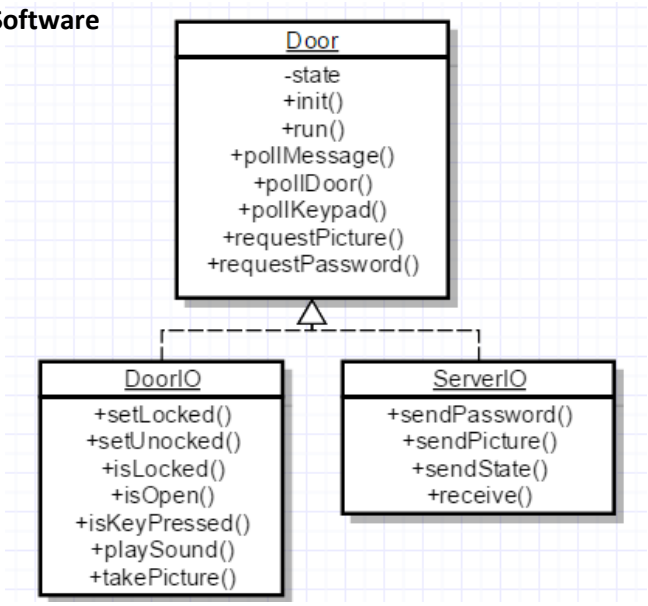

Figure 10: Class Diagram for Embedded Door Software

Figure 10 shows the UML class diagram for the core behavior of the embedded door system. It has been slightly abstracted to better showcase how it works. The Door class contains the core logic of the program. To get its input from the sensors, it relies on the methods in DoorIO, which interfaces directly to the hardware. To communicate with the server, the ServerIO module is used to send passwords, pictures, or door states to the server. For example, in a password sending transaction, isKeyPressed() in DoorIO will give Door the password it needs, then Door will process the transaction and finally send it to the server with ServerIO. Once a response is received with the function in ServerIO, pollMessage() will process the response and act accordingly. The embedded system running on the door regularly monitors input from each of its sensors as well as responds accordingly to any messages received from the server. In addition, it constantly updates the server with messages describing any changes made to the state of the door, such as someone locking or unlocking it.

In order to effectively monitor each sensor, multiple threads are used to poll each of the hardware components described in the section above. This means that each sensor can be checked concurrently and at specific and consistent intervals, which is useful since the state of the lock does not need to be checked as much as the buttons on the keypad being pressed. This will be done by running threads for pollDoor(), pollMessage(), and pollKeypad() in the background. Since Phantom Lock provides two types of doors: a front door and a back door, only the relevant polling threads need to be run on startup, meaning a back door will not need to run threads checking the state of the keypad since none is included.

Since the state of the door is stored on the database, the only function that has the power to modify the state of the door in the database is the thread that monitors the state of the lock and door.

Therefore, if a command to lock a door comes from a mobile client and passes through the server, the database entry will not be changed until the door receives the message, the door runs the command in DoorIO.py for the motor to turn the deadbolt, and the door and lock sensors determine that the door is properly secure, which will then send a message to the server that the state of the door is changed. This process is done to ensure the accuracy of values stored in the database.

2.2.3 – Server and Database

The server's requirements are to communicate and link the components together and save the states and attributes of the components of the door and android clients. The server is a multithreaded protocol that is continuously ready to receive messages. The server distinguishes between what to do with each message based on its library of opcodes and its database containing the components such as house and door numbers along with usernames and passwords, but also the network addresses of the various components.

The key functions that the server contains are to:

1. Accept Passcode message from door and respond with lock or unlock command (opcode = 0x00)
2. Begin image transfer from Door to server then from server to Android (opcode = 0x01)
3. Door state update from door, update database and send state to Android (opcode = 0x02)
4. Lock message from Android, Send Lock or unlock command to Door (opcode = 0x03)
5. Android address message, Store the android address (sent when android switches networks or network locations) (opcode = 0xFF)

With every request that the server receives, the server updates its database with a string representing the array of bytes that it received under the specific door that the request was directed towards (the special opcode 0xFF is so distant from the other codes for this reason, It does not involve a door and is not saved other than the address of the android user being updated).

This considered the full message send code (before the additional information section of a request) looks like: {houseNumberByte, doorNumberByte, opcodeByte, … additional information bytes}.

The server is required to be robust to preserve its integrity and can run continuously and resist hazardous requests and receivable packets that possibly do not pertain to it. Upon initial receive of a message, for the server to continue to process the information, the packet in question must have an accurate home and door number along with a valid opcode and additional data then must correspond with expected values. Any improper packets are ignored.

The Database stores a List of Houses which is effectively an object which contains everything else in the system. Each house contains a list of doors, a list of users, a passcode. Each door has a list of requests and a state(locked or unlocked). Each user has a username, a password, and a network address(which refers to the current network address of the android app that a user is logged into). With this information acquired from database, it is possible to run all protocol required by server, door, and Android.

2.2.4 - Android Application

The android application was created via AndroidStudio which is one of the convenient IDE used to create java based applications specifically for android and you can test it through one of the emulators that comes with it, or you can test on your android phone if you have one.

Android user will be able to control the door's lock from their phone. First, the user will be able to lock or unlock by simply pressing the button on the screen. The TextView underneath the button, which indicates the status of the door changes based on whether the door was locked or unlocked, it should also specify whether the door was locked through the app or by keypad. Once the user presses the button the recent activity's ListView will indicate which door was locked or unlocked. The recent activity's ListView, as the name indicates, chronologically lists recent interactions such as locking and unlocking doors or launching the application. Furthermore, each activity is appended to a time stamp indicating the time when the activity took place.

Communication between the server on the Raspberry Pi and the android application is done through UDP. Packets containing the opcodes of the request to the server are sent to the server over the internet. To connect to the server, use port number along with either the local IP of the Raspberry Pi if they are on the same network or public IP if connecting over the internet.

2.3 – User Interactions from a System Perspective

Since the components of the system have been described, the interactions of these components for the scenarios described in 1.3 can be demonstrated as UML Sequence diagrams.

1. User locks or unlocks door with a passcode using the keypad

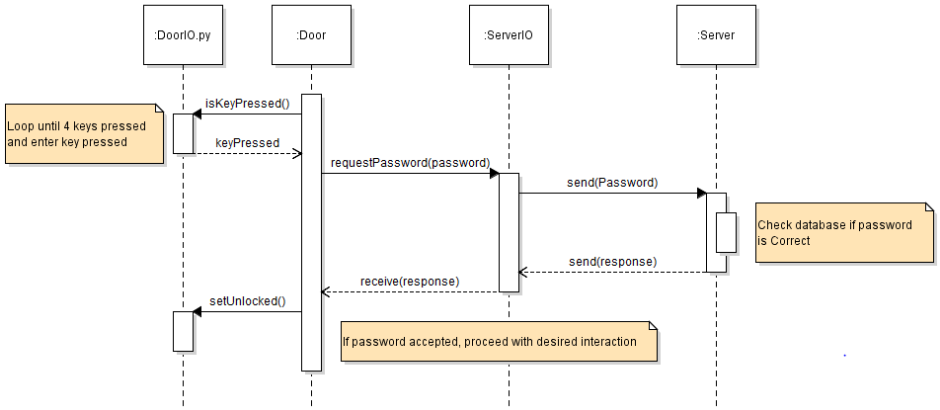

Figure 11: Sequence diagram for keypad scenario

In this scenario, once the password is entered using input from DoorIO, Door sends the password to server using ServerIO, where it compares the sent password to that in the database. It then sends a message back to the door saying whether the password was accepted or not, and the door will unlock if it has.

2. **User locks or unlocks door with a key or turning the deadbolt**

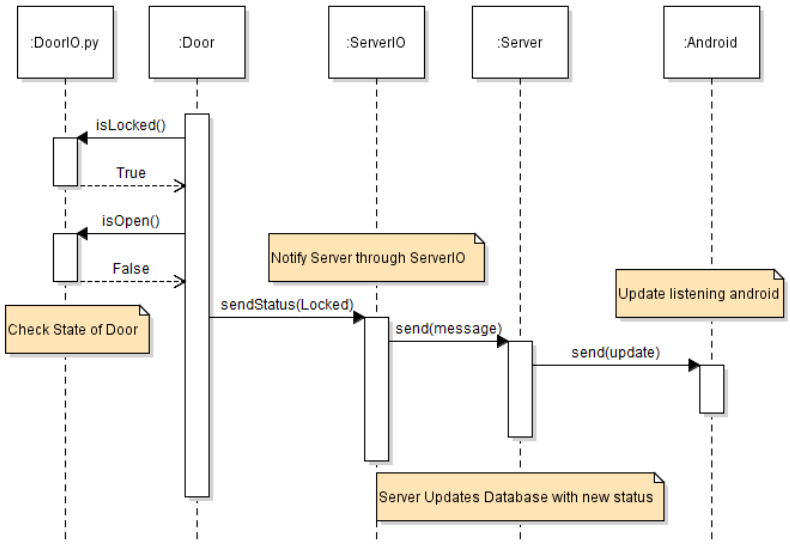

Figure 12: Sequence Diagram for manual door usage

In this case, the door checks the state of the door using isLocked() isOpen(), functions called in Door's pollDoor() thread. If a door state is changed, such as an unlocked door becoming closed and locked, Door will send an update message to the server, and the database and any listening android devices will be updated as well. It should also be noted that any time a door state changes, even if it is due to the motor turning it, this is the process used to update the database and listeners of any changes.

3. **User requests access to door with keypad and camera. Door owner responds to request by providing or denying access.**

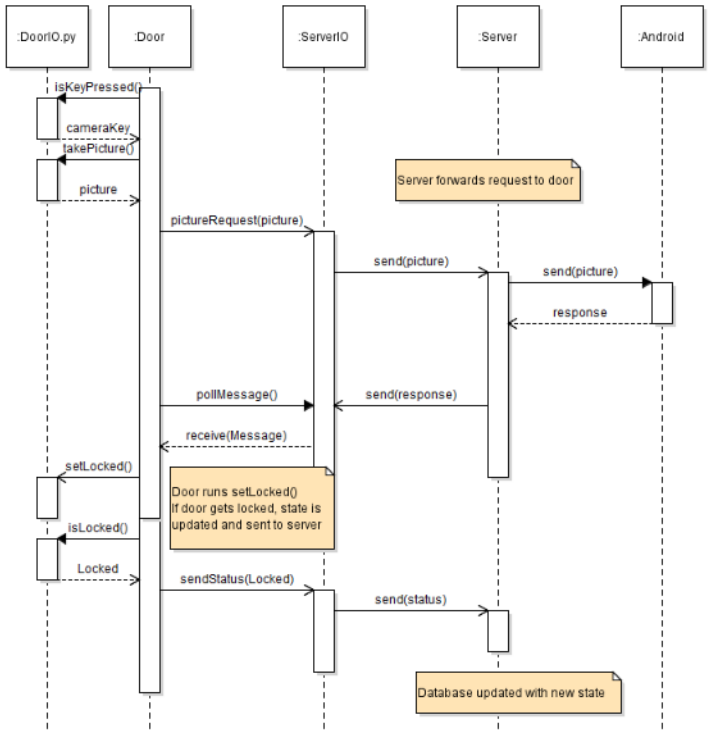

Figure 13: Sequence diagram for picture request

When the picture request is made, a picture is taken and is sent to the server. This takes place over several transfers as the picture to send is large. When receiving the picture on the android, the user can either accept or deny the request, and this response will be forwarded through the server back to the door. If accepted, the door will run the method to turn the lock, and if a state change is recorded, this status change will be passed to the server like in case 3 above.

4. User checks state of lock while away from house

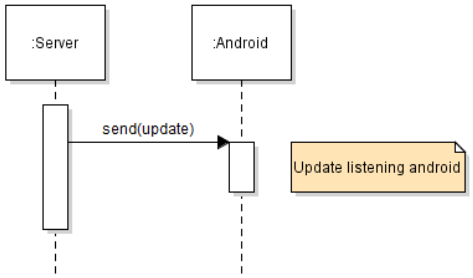

Figure 14: Sequence diagram for updating Android

If a user wants to check the status of their door from an Android, all they need to do is connect to the server and it will listen for any changes made to the state of the door.

5. User lock or unlock door while away from house

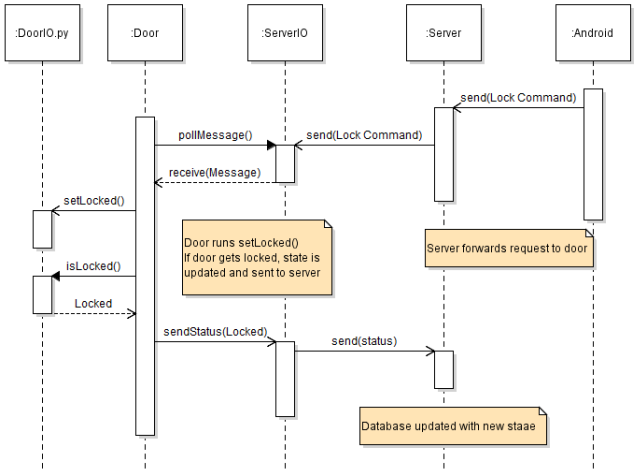

Figure 15: Sequence diagram for commands for Android

In this case, any command sent from the android will be forwarded to the Door, and the door will lock or unlock the door accordingly. Next, the same process as in case 2 will occur, where the door will send an update message to update the server and database. Any changes will then be passed to listening Android clients.

3.0 – Testing and Virtual Simulation

3.1 - Hardware Testing

Hardware testing was done manually. Code was written to create a desired response in the hardware output through software input, like with the motor turning, or to check software input after hardware input, such as pressing the keypad and seeing which key was pressed.

3.2 - Unit Testing

Testing for Phantom Lock was done using a series of test harnesses. As all pieces of code are treated as objects, a test file spawns an instance of the object, sets up initial conditions, and then runs a test, checking if the desired conditions were met after the test. A test case must return one of three values: 0 for pass, 1 for fail, and -1 for unresolved. All these tests are then run in one test harness to keep track of all the passes, fails, and unresolved results. Each module of the code will have its own respective test files and test harness, and each can be run all at once to determine if a module is working as expected. While the tests framework was written and all test cases were declared, there was not enough time to implement every test case. The cases that have not been implemented return an unresolved result.

3.3 - Distributed Server Testing

In order to effectively test the server, it had to be tested from multiple different network addresses and while receiving many packets at the same time in other words simultaneously. The server's test harness mad use of multiple threads so that it could send the server many messages and impersonate both door and android apps at the same time. The server was also tested by inundating it with many invalid requests while running the usual test harness. This effective "Garbage sender" was a generic local packet sender borrowed from GitHub. The server effectively managed all of its tests and did not observe significant decrease in speed when receiving many messages at the same time.

3.4 - Acceptance Testing through Virtual Simulation

As the server must work with multiple doors, yet only two will be made in actual hardware, a virtual door system is designed to simulate existing doors, shown in Figure 16. Through this interface, one can interact with the door as one would in real life, including opening the door, turning the deadbolt, or interacting with a keypad.

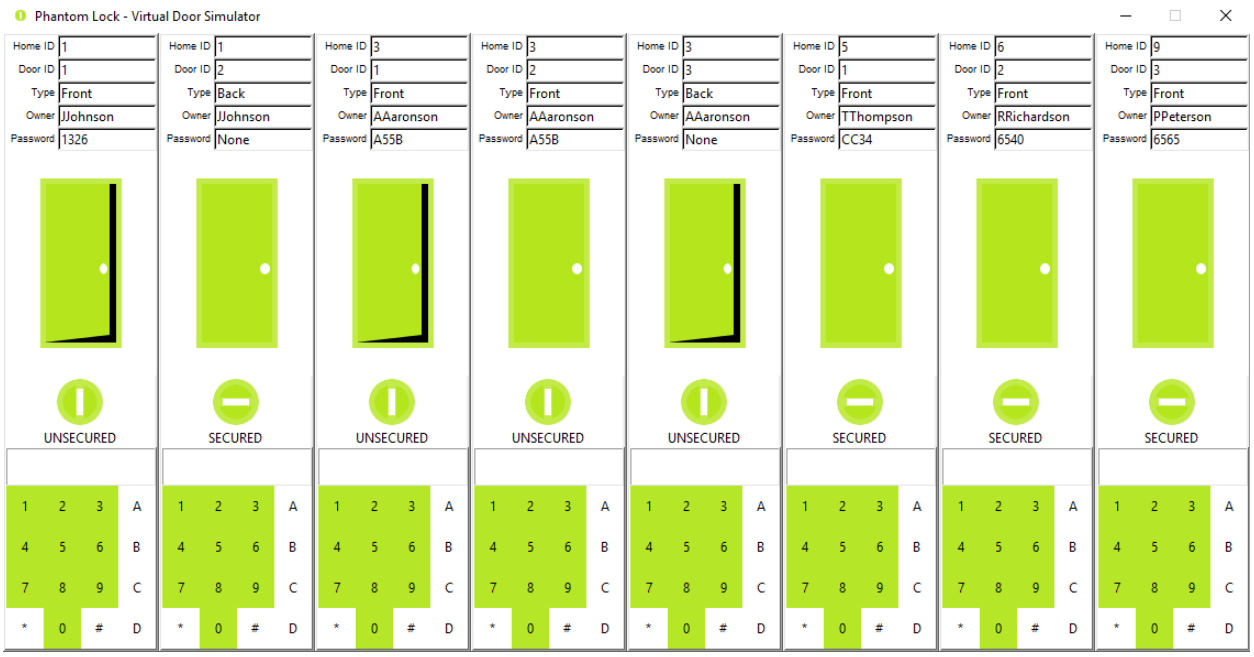

Figure 16: Virtual Door Simulator Interface

In order to have a single GUI control multiple doors at once, the main loops of each door are run as a separate thread. Instead of reading and writing to actual hardware, a new module called VirtualDoorIO is used. It implements the exact same functions to interact with hardware, meaning that the main code does not behave any differently regardless of which DoorIO module is used. In this module, there are variables that are shared between this module and the GUI, and this is how each module communicates. Therefore, if the loop want to read the state of the door, it will read the DOOR_BUFFER variable, which gets updated when the door button is pressed in the GUI. If the code wants to write to the LCD, the IO module writes to the LCD_BUFFER, which the GUI label for the LCD is listening to. This shared memory allows easy communication between the door and the GUI while not needing to change any of the door program's main code. Therefore, the GUI and Door are entirely separate entities communicating through shared memory.

Acceptance testing was done to prove the user interactions described in section 1.3 and 2.3 By using the virtual door, user interactions are simulated and it can be determined if the system behaves as it should during a series of different situations.

4.0 – Results and Discussion

4.1 – Use of MVC for Android and Virtual Door

As MVC is a fundamental part of an Android application, it was certainly used in the development of the application for Phantom Lock. The application is programmed to have separation between the layout and the control, with one handled in xml file and the other handled in java code. This represents the separation between the view and the controller. Likewise, the model is represented as the java code running in the background that is communicating with the server.

The Virtual Door GUI application used the spirit of MVC with a few modification. As the purpose of this program is to simulate the model, or program, as authentically as possible, the GUI had to be built somewhat around the model. Therefore, each component or widget of the GUI is separated into many different files, such as files for the keypad panel, door panel, and lock panel. Each of these widgets are designed to be independent, so they each have their own view and controller elements separate within their module. If a widget needs to output info from the Model, the controller within is used to extract that information and pass it to the View, or that actual widget within the GUI. Similarly, if a widget needs to pass information from the GUI to the program, the user uses the view to input information and the controller passes it to the model. Overall, in order to simulate the hardware interactions authentically the Model View Controller system was modified to have one view with multiple independent pairs of Views and Controllers for each hardware interaction.

4.2 – Changes Made during Term

Two major changes were made to the scope of project during the term, and they were entirely due to time constraints. The first major change was that the fact that the doors only use the Pi to interface with hardware and not the *PiFace* or *GertBoard*. This is because the hardware was originally tested on the Pi itself and as most of the functionality was working, there was not enough time to switch to the other hardware interfaces. This means that while it was originally in the design of the system, the project requirement of using a *GertBoard* and *PiFace* was not met.

The second change was the removal of the web application to be hosted on the server. This was also due to time constraints. However, the Desktop Application requirement for the project was still met due to the existence of the Virtual Door application.

4.3 – Challenges

The Following are challenges faced developing Phantom Lock

- When connecting the 16x2 LCD, it is recommended to use a potentiometer to control the brightness of the LCD; however, due the time limitation, the team decided to have the LCD at fixed brightness level without using the potentiometer and used a 1k ohm resistor instead.
- Another challenge faced was the magnetic sensors. The connection of the magnetic sensor was done through the use of previous electrical knowledge to come up with the output needed for the project. A schematic was created and a 1k ohm resistor was added between the voltage source and the GPIO pin on the Raspberry Pi prevent short circuit as shown previously in figure 6B above.
- The keypad is connected and programmed probably using 8 GPIO pin; however, it seems the keypad used in this project has a defect, which prevents the team from using it. Instead, the team decided to pass the entry passcode to the database through the sever assuming the entry passcode is entered correctly.
- Simulating the Door.py code virtually was a technical challenge as it was important to not change the logic of the code while still feeding it proper input. The virtual door software was the solution to this problem.
- Conducting the transfer of the photo taken on the RPi to the server. This process needed to take place over multiple packets with multiple acknowledgement packets sent back from the server. This process also needed to take place parallel to other regular door interactions. While many of the synchronization elements were successfully designed and implemented, the main problem was the fact that different imaging libraries were used to encode and decode the image. However, after much debugging this problem was fixed.

4.4 – Technical Recommendations

Overall, it is believed that the system is well designed all around, but time was the main constraint in developing it. Many challenges presented themselves along the way, as seen above, and these really slowed the progress of certain components of the program. This means that integration began far too late, and as integration itself provided its own challenges, it was hard to meet all the requirements by the deadline.

Many parts of the system are working correctly and are integrated. The integration of the door software with the server went relatively smooth as the proper message opcodes and format were discussed ahead of time. Similarly, the integration of the hardware and the door code went nearly flawlessly since the door hardware function used the exact same functions as those in the Virtual Door stub. The only shortcomings were individual component functionality that were not implemented, such as the physical keypad not working and the LCD not working.

In hindsight, it would have been better to dedicate more time into finishing the server earlier as it was the core component connecting the other modules of this project. Similarly, it would also have

been advisable to begin work on the Android application earlier as it was done at the end of March and integrating it with the system was a challenge. This meant that there was not enough time to work on fixing bugs that could have added additional functionality to the system, like finishing the image transfer. The essential problem that was faced was not having enough time to implement every component and this was mostly due to not starting early enough, which would have given time to deal with the unforeseen problems.

5.0 – References

[1] M. Sklar (2015). Adafruit Learning System. Drive a 16x2 LCD with the Raspberry Pi. https://cdn-learn.adafruit.com/downloads/pdf/drive-a-16x2-lcd-directly-with-a-raspberry-pi.pdf. Accessed [15 – Feb – 2017].

[2] Adafruit (2017). RASPBERRY PI. CAMERAS. RASPBERRY PI CAMERA BOARD. https://www.adafruit.com/products/1367l. Accessed [15 – Feb – 2017].

[3] Nodasystech (2017). Excel Industrial Electronics. Servo Wiring Diagram. http://nodasystech.com/design/servo-wiring-diagram.php. Accessed [25 – Feb – 2017].

[4] Hackster (2017). 4x4 Matrix Keypad with a Raspberry Pi and C#. https://www.hackster.io/lluppes/4x4-matrix-keypad-with-a-raspberry-pi-and-c-357ffe. Accessed [5 – Mar – 2017].