DAT110 Oblig 4 Part A-C

Joakim Johesan and Eirik Alvestad

May 5, 2019

Abstract

An Arduino access controll system with cloud based features, enabled by the use of a cloud server with a REST API. Designed and written with TinkerCad, Java, C++ and using Gson and the Spark framework.

1 Introduction

The Arduino design is here: https://www.tinkercad.com/things/kvRdZLvI430
All of the Oblig 4 is here: https://github.com/h180339/DAT110-Oblig4-DelB

The system consists of a simple Arduino unit, a Java Spark cloud service providing a REST API to be accessed by IoT devices, and a simulated Arduino in Java. The Arduino unit is programmed to act as a simple access control system. It is unlocked by pressing the right combination of the two input buttons. After being unlocked for a while it will go back to being locked.

The cloud service works by providing a message based REST API that enables IoT devices to save access logs and retrieve updated access code information. The device does this by connecting to the cloud service by using the HTTP protocol and sending HTTP GET, or POST requests. The messages it exchanges with the server are all serialized as JSON objects. We used the Spark framework to create simple ”servlets” that manage the incoming HTTP requests on the cloud service.

The Arduino is connected with a PIR detector, two input buttons, and 3 status LEDs. The design was made and simulated in TinkerCad and Java.

2 Access Control Design Model

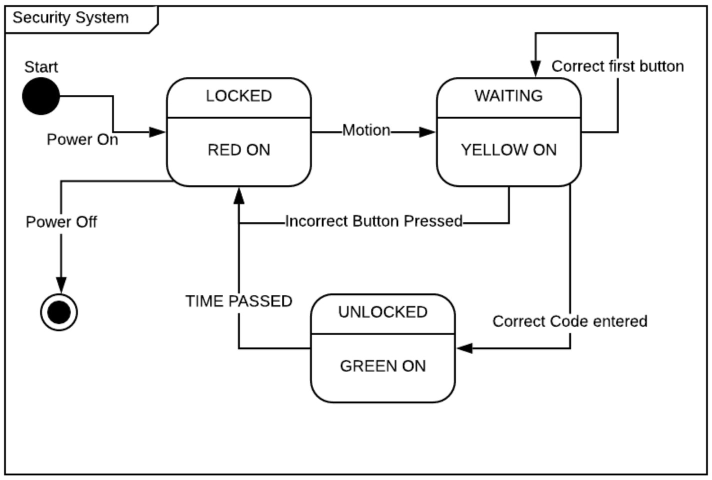

Figure 1: Access Control State Machine

The state machine models the functionality the final system has available. It does not actually have any locking or protection in place. Extending the system to accept longer and safer codes should not be hard, and connecting a solenoid or actuator to physically lock something would probably be easy.

The machine has 3 discrete states:

- **Locked**
  The system starts in this state, it also enters it 10 seconds after entering the unlocked state. In this state the Red LED light is on. And whatever the system is protecting should be unavailable.
- **Waiting**
  The system enters this state from the Locked state when the PIR detector senses movement. The Yellow LED will be lit while waiting. The system will wait in this state until a code sequence is entered. It will move the system to the Unlocked state when the two correct buttons are pressed in sequence. The yellow LED will blink whenever a button is pressed. It will move to the Locked state whenever a wrong key is pressed compared to what was required.
- **Unlocked**
  The system enters this state from the Waiting state after a correct code has been entered. In this state the Green LED will stay lit. The system will exit this state after 10 seconds and then enter the Locked state again.

As the state machine is a model to help with writing the code we didn't model the blinking of the Yellow LED light whenever buttons are pressed.

3 Access Control Hardware/Software Implementation

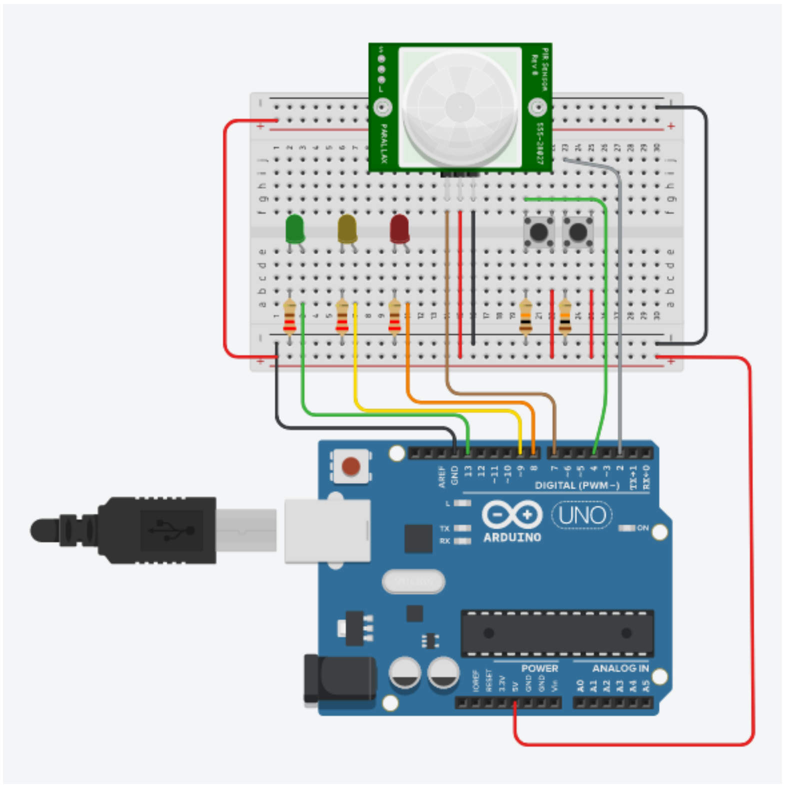

Figure 2: Arduino wiring diagram

Hardware:

The system has 3 signaling LEDs for showing the current system state. It also has two impulse push buttons used to input the access code. Finally it has a PIR detector that senses movement and signals this to the Arduino board. Everything is connected to the digital I/O.

The LEDs and input buttons all have current limiting resistors connected in series with them to prevent damaging them. This actually isn't necessary in the software simulation as they are indestructible.

Software:

The Arduino runs through the setup() function before running the loop() function until the device is turned off. The speed of the system makes the loop function complete several thousand times per second, because of this we need to save if we detected button presses and inputs as these can be pressed for several loops without an intended new input.

In the setup we set all the pin-modes for all the inputs and outputs we are using. For this task we only needed to use the digital inputs and outputs. These all receive/output either a HIGH or LOW signal. After the setup call, we create a few constants and variables to store information outside of the loop.

In the loop we first retrieve the states of the inputs. After that we check if the PIR detector input is HIGH and if we previously managed this event. If this is a new input we change the system to the waiting state, and save that we managed the input.

After this we enter a switch statement which performs different tasks depending on the set system state. In all the states, we refresh the outputs so they are correct for the systems current state. This is necessary because the system might have changed state.

In the waiting state we check if any of the buttons have been pressed, and if the last pressed button is a valid one. Doing this we can implement a simple code checking. We also pulse the Yellow LED so the user can see the input has been detected. The pulsing has to be done with a delay() call between the output states, as it would not be noticeable that the LED blinked by humans without it.

The code doesn't implement any actuator locking or unlocking, but this would simply be a line of code contained in each of the switch cases.

4 REST API cloud service

We implemented the cloud service using the spark/java framework, which makes it easy to implement a REST api. We used the framework so that the cloud service makes it possible for the access control device to register attempts to access the system in an access log, and to be able to change the access code and retrieve the current access code. The task was well written and we set up the routes the way the task specified.

The routes was set up using the spark framework which allows us to specify the operation type (get, post, put, etc..) and the url in a easy way. The access attempts is collected in memory, which means that the information is only stored for as long as the program is running. Each log message received via the POST HTTP operation is given a unique identifier. We used a Concurrent HashMap for storing the log-messages received from the device and an Atomic Integer for keeping track of the identifier like the task suggested. The hash-map uses the identifier as the key for the log-message. Like the task specified we used the Gson-library. We used two of the methods of this library(toJson, fromJson). toJson: converts a java object into a Json representation fromJson: converts a Json representation into a java object that the code specified.

5 Device Communication

We implemented the network communication in the access control device by using some of the services implemented in the cloud service.

The first task was to implement a way for the device to obtain the current access code. We did this by establishing a connection to the cloud-service and issue the appropriate HTTP GET request.
For this we used java socket to establish a connection to the cloud-service. We then constructed the get request consisting of the URL that matches the cloud-service route that returns the current access code, and write it to the ouput stream. We then take the response and iterate through it to get the body of the response, create a AccessCode object using the Gson-library and return that object.

The second task was to implement a way for the device to issue a HTTP POST request on the service in order to add a log access entry for the message. For this we used java socket again to establish a connection to the cloud-service. When using the cloud service it required us to create a post request where the body consists of a json represented object, we did this by first creting a java AccessMessage object from the parameter message, and convert it to a json representation using the Gson-library. We then constructed the post request consisting of the URL that matches the cloud-service route that allows us to save a message to the hashmap. We put the json representation of the message in the body of the request and wrote to the outputstream.

6 System Testing

While we implemented the operations the task required us to implement, it was important for us to be able to test what we just implemented to make sure it was correct. We did this mostly using the postman tool that the task suggested. The postman tool allowed us to easily paste in a the desired url and choose if we wanted to post/get/put to the specified url. The postman tool was the most useful when testing the post method because it allows us to insert a json representation of a object directly in the body of the request.
When we were done implementing the cloud service and started implementing the device network communication the testing consisted more of running the device and manually testing the different functions it should be able to perform. But here we also used the postman tool to test the doGetAccessCode() method, because the testing required us to change the access code, and this as easily done using the postman tool.

7 Conclusions

The project consisted of implementing a simulated access control device using an Arduino and a Java + Spark based cloud service.

We thought the project turned out good. The start code was well written, documented, and easy to work with. Learning how to write code for the Spark framework turned out to be uncomplicated. We both made separate implementations of the Arduino and its code for learning purposes. We also decided to write the report in LaTeX to get a sense of how to use it, having previously only used the math portions of it in a Markdown implementation.
We managed all the required given tasks, but did not have time to complete any extras. The overall experience was very insightful and we learned a lot.