Implementation Documentation

By Group 2

Anton Geneser Matzen
202008936@post.au.dk
Std.nr: 202008936
Au-id: 683185

Emil Hilligsøe Lauritsen
202004154@post.au.dk
Std.nr: 202004154
Au-id: 668867

Martin Michaelsen
202007433@post.au.dk
Std.nr: 202007433
Au-id: 672598

Kevin Vollesen Schønberg
202007282@post.au.dk
Std.nr: 202007282
Au-id: 674059

Table of contents

1. Version History

Test Specification Versioning Table			
Version	Date	Editor	Modifications
0.1	05-04-2022	All	Walkthrough of the current implementation
0.2	18-05-2022	All	Updating to match the final implementation
0.3	20-05-2022	All	Finalizing the implementation walkthrough
0.4	22-05-2022	All	Editing and finalizing document

Table 1: Implementation Document - Version History

2. Introduction

This document entails the implementation of our kitchen guard system. It will cover all methods and variables we have added to and modified from the Cep2 kitchen guard system received in tutorial 6[1]. The code blocks in this document will include line numbers on the left part of the code block, and will also show the file's name at the top of the code block.

2.1 Abbreviations

Abbreviation	Description
PHP	PHP: Hypertext Preprocessor
API	Application Programming Interface
LED	Light-emitting diode
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
GUI	Graphical User Interface
WAMP	Windows, Apache, MySQL (MariaDB), and PHP (Perl and Python)
MAMP	MacOS, Apache, MySQL (MariaDB), and PHP (Perl and Python)
LAMP	Linux, Apache, MySQL (MariaDB), and PHP (Perl and Python)

Table 2: Implementation Document - Version History

3. Implementation

The following will cover our developed Web Application programming interface or API for short, which handles processing events from the controller, storing them in a database and displaying them in a graphical user interface in the form of a web client all using the WAMP stack, which is an acronym for Windows, Apache, MySQL, and PHP.

[1] jmiranda-au, 2021: cep2app

3.1 The Kitchen Guard Controller

The implementation of our controller runs the logic() function once every second firstly checking if the stove is turned on. At the same time the __zigbee2mqtt_event_received() function will firstly monitor if there is any event received from the kitchen plug where "power > 0 and state == ON" indicating that the stove is turned on, if it is, an event is sent to the web API to store in the MySQL database. A timer will then be started to keep track of how long it has been since the last event received from the kitchen sensor with "occupancy == true". This timer will be reset every time this event is received from the kitchen sensor. If the timer reaches 20 seconds a global timer will start and an event, indicating that the user has left the kitchen, will be sent to the web API to store in the database using the SendEvent() function. If the global timer reaches 300 seconds or 5 minutes the controller will wait for an event with "occupancy == true" from a sensor in one of the other rooms in the house. If such an event is received the LED light in the kitchen will change color corresponding to the room. Originally it was intended for different LED lights corresponding to the different rooms to be turned on, however as we lacked LED components this was not possible to implement. An event will also be sent indicating that the user has left the stove on for more than 5 minutes. If at any point before the global timer reaches 20 minutes, the user reenters the kitchen the timer will be reset and if any alerting lights are on, they will be turned off. Here an event will also be sent to the web API. When the global timer reaches 20 minutes the stove will be shut off as well as any LED light and the kitchen LED light will light up in red and an event will be sent to the web API to indicate that the user left the stove on for more than 20 minutes. If at any point the kitchen plug is turned off manually an event will be sent to the web API to indicate this.

3.1.1 The Cep2Main.py file:

In this program the devices are added as seen in the following code block:

```
File: Cep2Main.py
if __name__ == "__main__":
    # Create a data model and add a list of known Zigbee devices.
    devices_model = Cep2Model()
    devices_model.add([Cep2ZigbeeDevice("Kitchen_Sensor", "pir"),
                       Cep2ZigbeeDevice("Kitchen_Light", "led"),
                       Cep2ZigbeeDevice("Kitchen_Plug", "power plug"),
                       Cep2ZigbeeDevice("Sensor_Room_1", "pir")
                       # The following sensors would be connected, however we only managed to connect 1 additional sensor
                       #Cep2ZigbeeDevice("Sensor_Room_2", "pir")
                       #Cep2ZigbeeDevice("Sensor_Room_3", "pir")
                       #Cep2ZigbeeDevice("Sensor_Room_4", "pir")
                       #Cep2ZigbeeDevice("Sensor_Room_5", "pir")

                       # The following LEDs would be connected, however we did not manage to add any additional lights
                       #Cep2ZigbeeDevice("Light_Room_1", "led")
                       #Cep2ZigbeeDevice("Light_Room_2", "led")
                       #Cep2ZigbeeDevice("Light_Room_3", "led")
                       #Cep2ZigbeeDevice("Light_Room_4", "led")
                       #Cep2ZigbeeDevice("Light_Room_5", "led")
                      ])
```

The commented devices can be added if they are connected and configured to have the listed friendly names in the confugartion.yaml zigbee2mqtt file. The following is a while loop that runs the logic function from our controller once every second.

```
File: Cep2Main.py
    while True:
        sleep(1)
        controller.logic()
```

3.1.2 The Kitchenguard.py file

In this python file the following global variables are created:

```
File: Kitchenguard.py
    timer = time.time() # This is the variable used to calculate time_sm and therefore global_timer.
    Kitchen_Light_State = 1 # Used to keep track of whether the kitchen light is on.
    stove_state = False # Used to indicate if the stove is on or off.
    time_sm = 0 # small timer, to keep track of if movement has been detected in the kitchen for the last 20 seconds.
    global_timer = 0 # Variable to keep track of how long no movement has been detected in the kitchen while the stove is on.
    idchain = 0 # Used to seperate the data in the database.
    Number_Of_Rooms = 1 # The number of rooms excluding the kitchen that have been configured.
    UserRoomState = 0 # Used to indicate if the user is in another room.
    temp = 0 # Used as a failsafe to handle delays caused by using HTTP to send events to the web API.
```

The first thing the controller waits for is for the stove_state to be True, as in the stove to be turned on. This is monitored in the __zigbee2mqtt_event_received() function as seen below:

```
File: Kitchenguard.py
        if device:
            try:
                #This checks if the event recieved includes the variable power, as in is the event from the plug.
                power = message.event["power"]
                state = message.event["state"]
                #If the device is not the kitchen plug then ignore the event. We only use that one plug.
                if device_id != "Kitchen_Plug":
                    pass
            except KeyError:
                pass
            else:
                #If the plug is powering something as in the stove is on, check if it was off previously, if it was do the following.
                if power>0 and state == "ON":
                    if self.stove_state == False :
                        self.stove_state = True
                        self.TimerStart()
                        self.idchain += 1
                        self.SendEvent("Stove Turned On")
                else: # Otherwise if the stove is off
                    if self.stove_state == True: #check if it previously was on, if it was do the following.
                        self.SendEvent("Stove Was Manually Turned Off")
                        if self.global_timer < 1200:
                            self.LightOff(0)
                    self.stove_state = False
```

The reason for both checking the power and state of the kitchen plug, is that when the stove is turned off it takes some seconds for the power output to reach 0. At the same the state of the kitchen plug can be ON but if nothing the stove is not powered on the power will remain 0. So the only time when the stove is truly turned on is when both the state is ON and the power is greater than 0. It can be seen that the TimerStart() function is called when the stove is turned on, it can be seen below:

```
File: Kitchenguard.py
    def TimerStart(self) -> None:
        """Updates the timer variable to the present time """
        self.timer = time.time()
```

When the stove is turned on the logic() function will start functioning fully, the logic() function can be seen below:

```
File: Kitchenguard.py
    def logic(self) -> None:
        if self.stove_state == False: # If the stove is off return
            return

        if self.temp != 0: # This is a failsafe used to handle the delay that sometimes comes with sending an event using HTTP
            self.TimerStart()
            self.temp = 0

        self.time_sm = time.time() - self.timer # The first timer
        self.global_timer = self.time_sm - 20 # The global timer
        """
        Used for testing purposes
        if round(self.global_timer) % 5 == 0 and self.global_timer >=0:
            print("User has left the kitchen for " + str(round(self.global_timer)) + " seconds.")
        """

        if self.time_sm >= 20:
            if  self.global_timer <= 1:
                self.SendEvent("User Has Left The Kitchen")
        """
        This part is for testing purposes
            if self.global_timer > 300 and self.Kitchen_Light_State != 1:
                self.LightOn(0)
        """
        if self.global_timer <= 300 and self.Kitchen_Light_State != 0: # This would be changed once we added more lights to the system
            self.TurnOffAllLights()

        if self.global_timer >= 1200: # If 20 minutes pass with no event recieved from the kitchen sensor with occupancy == true do the following
            self.TurnOffAllLights() # Turn off all alerting lights.
            self.LightOn(0) # Turn on the kitchen LED light.
            self.__z2m_client.change_color("Kitchen_Light",{"r":2,"g":0,"b":0}) # Make the light red to signify that the user left the stove on for more than 20 minutes.
            self.SendEvent("User Has Not Returned After 20 Minutes") # Send an event to the web API
            self.StoveOff() # Turn off the stove
        return
```

The logic() function is what changes the lights and kitchen plug depending on the value of the different timers. It also updates the values of the timers. It can be seen that when the first timer is above 20 seconds and if the global timer is less than 1 second an event is sent indicating that the user has left the kitchen. This could have been implemented in a prettier manner, however it has functioned as needed so far. Of course a situation where more than one second has passed since last the timers were updated could occur leading to the first timer jumping from less than 20 seconds to above 21 seconds meaning the global timer would be above 1 second and therefore never satisfy the condition required to send the event. This wasn't practically a problem but theoretically an issue nonetheless.

The if statement in the logic function line 125-126 is not a part of the final implementation with the devices in all the rooms in the house connected. However as we did not have more than 1 LED light and 2 sensors connected this is what is implemented. We could change this to what we think would work with all the devices. As we cannot test it, however, this would likely lead to many errors.

It can be seen that the logic() function uses the method TurnOffAllLights() often, this method can be seen below:

```
File: Kitchenguard.py
    def TurnOffAllLights(self) -> None:
        i = 0 # Variable used to shift through all rooms.
        new_state = "OFF"
        while i <= self.Number_Of_Rooms: # Shift through the number of rooms in the house
            #self.__z2m_client.change_state("Light_Room_"+str(i), new_state) # This should work however is untested
            i +=1
        self.LightOff(0)
```

It can be seen that the while loop does not do anything currently, this is because we have not connected additional LED lights, so we are unable to test if this piece of code works as intended.

The z2m_client change_state() method is called to change the state of a device, this is not something we have implemented. However we have implemented another method in the

Cep2Zigbee2mqttClient.py file called change_color(), as the name suggests this method can change the color of an LED light device. This method can be seen below:

```
File: Cep2Zigbee2mqttClient.py
    def change_color(self, device_id: str, color: str) -> None:
        if not self.__connected:
            raise RuntimeError("The client is not connected. Connect first.")

        self.__client.publish(topic=f"zigbee2mqtt/{device_id}/set",
                              payload=json.dumps({"color": color}))
```

Most of the function is simply a copy of the change_state() method where state has been changed to color. This method is used in the logic() function line 131 and also the __zigbee2mqtt_event_received() method line 235 as seen below:

```
File: Kitchenguard.py
try:
    # Check if the event includes the variable occupancy, as in is it from a sensor.
    occupancy = message.event["occupancy"]
except KeyError:
    pass
else:
    # If there is movement and the stove is on
    if occupancy and self.stove_state:
        #If the movement is in the kitchen
        if device_id == "Kitchen_Sensor":
            if self.time_sm > 20 and self.temp == 0:
                self.temp = 1 # Used to handle HTTP delay
                self.SendEvent("User Returned To The Kitchen")
            self.TimerStart() # Resets the timer
            self.UserRoomState = 0 # Used to indicate that the user is in the kitchen
            #self.__z2m_client.change_color("Kitchen_Light",{"r":2,"g":5,"b":2}) # Used for testing purposes

        elif device_id != "Kitchen_Sensor": # If the movement is not in the kitchen
            if self.global_timer > 300: # and 5 minutes have passed
                i = 1 # used to keep track of the rooms
                while i <= self.Number_Of_Rooms: # Check which room the movement occured in
                    if "Sensor_Room_"+str(i) == device_id and self.UserRoomState != i:
                        #self.LightOn(i)
                        #When the user enters another room the kitchen light will change colour. This is because we only have one light.
                        #self.TurnOffAllLights()
                        self.UserRoomState = i
                        self.LightOn(0) # Here we would light up room i, but as we only have 1 led light this has not been implemented
                        self.__z2m_client.change_color("Kitchen_Light",{"r":i+2,"g":i,"b":i+1}) # We instead change the color depending on i.
                        self.SendEvent("User is in Room "+str(i)) # Send an event to the web api.
                    i+=1
```

This is where events from pir sensors are handled. It can be seen that when the controller receives an event with occupancy as a variable it will check if it is true and if the stove is on. If these conditions are satisfied it will check where the movement has been detected. If it is in the kitchen it will check if the small timer has reached 20 seconds and the temp variable is equal to 0. If these are satisfied it will send an event indicating that the user has returned to the kitchen and set the temporary variable to 1 so as to avoid entering this if statement again. The reason why this can happen is that the controller can take multiple seconds to send an event using HTTP and therefore for some reason not finish running the if condition, instead running it from the beginning again without resetting the timer. This is made up for in the logic() function, where the timer is reset and the temporary variable set to 0. If the user is in the kitchen a variable called

UserRoomState is set to 0. This is a variable used when the event received is not from the kitchen sensor. As when that is true and the global timer has passed 5 minutes the controller will go through checking if the event received is from a sensor in another room. If the sensor in any of the rooms device id matches with the sender of the event the kitchen light will change color and an event will be sent to the web API to indicate that the user is in the corresponding room and that he has left the stove on for more than 5 minutes. Originally it was intended to light up the LED in the corresponding room however we did not have more LEDs so we could not implement this because of the inability to test it. The if statement in the while loop will also set the UserRoomState variable to i to indicate that the user is in room i, and so as to avoid continuously running this part of the code, everytime an event from that room's sensor is received.

It can be seen that the change_color() method is used and it takes a device id and an rgb in json format as an input. It also calls the function LightOn() which turns on an LED light of a specific room depending on the integer given as an input, the method can be seen below:

```
File: Kitchenguard.py
    def LightOn(self, room_number) -> None:
        new_state = "ON"
        if room_number == 0:
            self.Kitchen_Light_State = 1
            self.__z2m_client.change_state("Kitchen_Light", new_state)
            #if self.UserRoomState == 0: # Used for testing purposes
            #    self.__z2m_client.change_color("Kitchen_Light",{"r":2,"g":5,"b":2})
        else:
            #This line can be used if other light devices are configured. However it is untested.
            #self.__z2m_client.change_state("Light_Room_"+str(room_number), new_state)
```

It only works for the kitchen light currently as we have not tested it with multiple leds.
This method is very similar to the LightOff() method as seen below:

```
File: Kitchenguard.py
    def LightOff(self, room_number) -> None:
        new_state = "OFF"
        if room_number == 0: # If the room number is 0 indicating it is the kitchen, turn off the kitchen led


            self.Kitchen_Light_State = 0
            self.__z2m_client.change_state("Kitchen_Light", new_state)
        else: # Otherwise turn off the led in the room indicated by the room number
            #self.__z2m_client.change_state("Light_Room_"+str(room_number), new_state) # This should work however is untested
```

This also has the same problem as the LightOn() method. In the logic() function the method StoveOff() is called when the global timer has reached 20 minutes, this method can be seen below:

```
File: Kitchenguard.py
    def StoveOff(self) -> None:
        self.stove_state = False # set the stove state to False, to indicate that the stove is off
        self.__z2m_client.change_state("Kitchen_Plug", "OFF") # Turn off the kitchen plug
        self.SendEvent("Stove Was Turned Off By Controller") # Send an event indicating this to the web API
```

It uses the change_state() method and sends an event using the SendEvent() method we have implemented. The SendEvent() method can be seen below:

```
File: Kitchenguard.py
    def SendEvent(self, event) -> None:
        """
        This is for testing purposes
        print("Event("+event+") was sent to the database.")
        """
        # This is the code used to send a json package to the web API to store in the database.
        conn = http.client.HTTPConnection('kitchenguard.ddns.net')
        event_kitchen_occupancy = heucod.HeucodEvent()
        event_kitchen_occupancy.Timestamp = time.time()+7200
        event_kitchen_occupancy.Event = event
        event_kitchen_occupancy.IdChain = self.idchain
        data = event_kitchen_occupancy.to_json()
        conn.request('POST', '/a.php', data)
       """
        This is for testing purposes
        r1 = conn.getresponse()
        print(r1.status, r1.reason)
        while chunk := r1.read(200):
            print(repr(chunk))
       """
```

This method uses the heucod.py file, that was received in tutorial 9[2], to create a json string which contains three variables Timestamp, Event and IdChain. This Json string is then sent using HTTP to the webserver 'kitchenguard.ddns.net' using the HTTP request method. That is all for the implementation of the kitchen guard controller.

3.2 The Web API

The Web API is what handles storing events received from the controller in the MySQL database and displaying them on a web client as a GUI for the end user to see.

3.2.1 The a.php file

This file is what handles events received from the controller using HTTP. The important part of the file's functionality can be seen below:

[2] Jmiranda-au, 2022: cep2heucod

```
File: a.php
// Load the POST.
$data = file_get_contents("php://input");

// ...and decode it into a PHP array.
$jsondata = json_decode($data);

// Do whatever with the array.
//PrintObj($jsondata);

$servername = "localhost";
$username = "root";
$password = "cep2";
$dbname = "kitchenguearddb";

// Database connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Insert data Query
$sql = "INSERT INTO events ( Timestamp, Eventtype, IdChain)
VALUES ('$jsondata->Timestamp', '$jsondata->Event' , '$jsondata->IdChain')";
```

It starts off by receiving the json string sent through HTTP by the controller. It then decodes the json string. Sets up variable parameters that correspond to our kitchenguard database. Then it connects to the MySQL database, and using Structured Query Language, or sql for short, inserts the data into the database. It can be seen that we have three variables or columns in our database that are made from data received through the controller, those being Timestamp. Eventtype and IdChain. Timestamp is the number of seconds that have passed since 1970 the 1st of january 00:00 therefore it is an integer. Eventtype is a string that entails what event was received by the controller, it describes what the user has done. The IdChain variable is simply used to separate a 'session' of events, it is an integer that increments whenever the stove is turned off and on again.

3.2.2 The Index.php file

The index.php file is what handles both fetching from and displaying the database in the web client for the end user to see. The php part of the file can be seen below:

```
File: index.php
<?php
    $servername = "127.0.0.1";
    $username = "root";
    $password = "cep2";
    $dbname = "kitchenguearddb";

    // Database connection
    $conn = new mysqli($servername, $username, $password, $dbname);
    // Check connection
    if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
    }

    $sql = "SELECT Timestamp, Eventtype, IdChain, id FROM events ORDER BY id DESC";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        $time = gmdate("Y-m-d H:i:s", $row["Timestamp"]);
        echo "<tr>";
        echo "<td>" . $row['id'] . "</td>";
        echo "<td>" . $time . "</td>";
        echo "<td>" . $row['Eventtype'] . "</td>";
        echo "<td>" . $row["IdChain"] . "</td>";
        echo "</tr>";
    }

    }
    else {
    echo "0 results";
    }
    $conn->close();
?>
```

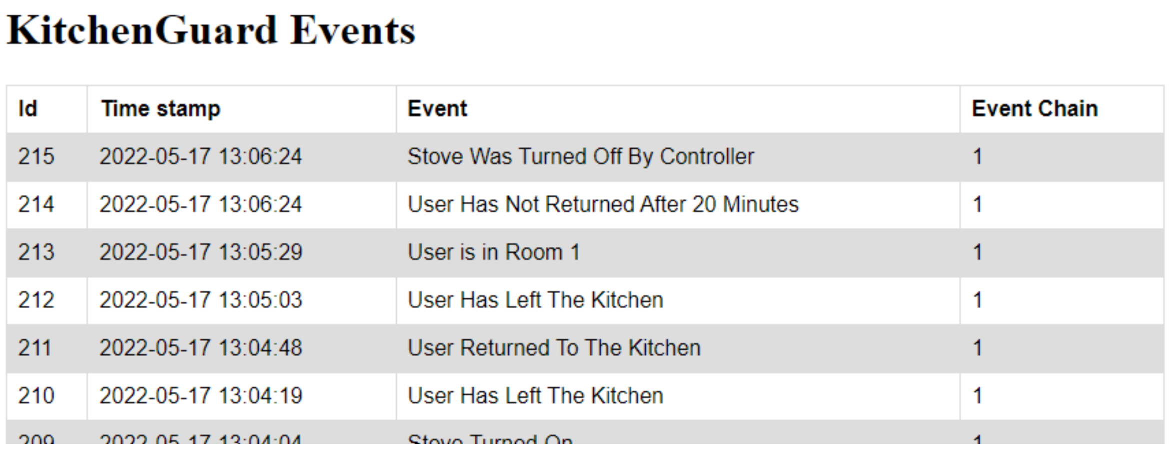
This php program also connects to the MySQL database using the database parameters. It then uses sql to select the variables Timestamp, Eventtype, IdChain and id, from the table events in the database, ordered by the id in a descending manner. Here the id is simply an integer given to all events stored in the database to help keep track of what event is which. The results from this query are then saved and placed in an HTML table created prior, to be seen by the end user in the web client.

3.3 WAMP

Our database and webserver are both run using WAMP which stands for Windows, Apache, MySQL, and PHP. Note that the 'M' can also refer to MariaDB and the 'P' can also refer to Perl and Python. For our implementation we have chosen WAMP since we are running the server on a Windows 10 system, but this is not a must. A similar server could be set up using LAMP, which is for Linux or MAMP which is for MacOS.

3.3.1 Database

Our database is an SQL database which runs on a MySQL server. The database is set up using the application ADMINER, which was used to set up the different columns of the database. This process will have to be repeated by a developer at a later point if the user wishes to host the server for themselves. The database data can be seen through the address Kitchenguard.ddns.net which is hosted using Apache. The database and web server are connected through the PHP scripts that are described in sections "3.2.1 The a.php file" and "3.2.2 The Index.php file". A view of the database can be seen in Figure 1:



KitchenGuard Events

Id	Time stamp	Event	Event Chain
215	2022-05-17 13:06:24	Stove Was Turned Off By Controller	1
214	2022-05-17 13:06:24	User Has Not Returned After 20 Minutes	1
213	2022-05-17 13:05:29	User is in Room 1	1
212	2022-05-17 13:05:03	User Has Left The Kitchen	1
211	2022-05-17 13:04:48	User Returned To The Kitchen	1
210	2022-05-17 13:04:19	User Has Left The Kitchen	1
209	2022-05-17 13:04:04	Stove Turned On	1

Figure1: View of Database

3.3.2 Web Server

The web server is used to store the data in the database and host our web page. This is hosted through the application Apache. The process of storing the data is done by the “a.php” which is called by the “Cep2Controller.py” file in the Send_Event() function, which is already described on sections “3.2.1 The a.php file” and “3.1.2 The Cep2Controller.py file”. The WebServer is also used to view the data from the database. This is again Apache which hosts our web page. The webpage is coded in the file “index.php”, this process is described in the section “3.2.2 The Index.php file”.

4. Sources

1. Jmiranda-au, 2021: cep2app, URL: https://github.com/jmiranda-au/cep2app
2. Jmiranda-au, 2022: cep2heucod, URL: https://github.com/jmiranda-au/cep2heucod