Architectural Design Specification

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 **2**

2. Introduction **2**

2.1 Abbreviations **2**

Table 2: Architectural Design Specification - Version History **2**

3. Architectural specification **3**

3.1 Physical View **3**

3.1.1 UML Deployment Diagram **3**

3.1.1.1 Sensors **5**

3.1.1.2 Actuators **5**

3.1.1.3 Zigbee Controller **5**

3.1.1.4 Raspberry Pi **5**

3.1.1.4.a Kitchenguard.py **6**

3.1.1.4.b Mosquitto server **6**

3.1.1.4.c Zigbee2MQTT **6**

3.1.1.5 Server Pc **6**

3.1.1.5.a WEB API **6**

3.1.1.5.b Database **7**

3.1.1.5.c web page **7**

3.2 Logical View **7**

3.2.1 UML Class diagram **7**

3.2.2 UML State diagram **9**

3.3 Process View **10**

3.3.1 UML Sequence diagram **10**

3.3.2 UML Activity diagram **14**

3.4 Development View **16**

3.4.1 UML Package diagram **16**

4. Sources **17**

1. Version History

Test Specification Versioning Table			
Version	Date	Editor	Modifications
0.1	28-02-2022	All	First iteration of the Architectural Design Specification
0.2	15-03-2022	All	Second iteration of the Architectural Design Specification
0.3	05-04-2022	All	Third iteration of the Architectural Design Specification
0.4	03-05-2022	All	Fourth iteration of the Architectural Design Specification
0.5	22-05-2022	All	Final iteration of the Architectural Design Specification

Table 1: Architectural Design Specification - Version History

2. Introduction

In this document we state the architectural design of the system derived from the use case driven requirements in the Requirement Specifications Document[1]. This document aims to give a clear definition of how the system should function both internally and externally with the environment.

2.1 Abbreviations

Abbreviation	Description
Pi	Raspberry Pi
UML	Unified Modeling Language
USB	Universal Serial Bus
HEUCOD	Healthcare Equipment Usage and Context Data
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
MQTT	Message Queuing Telemetry Transport
API	Application Programming Interface
LED	light-emitting diode
UC	Use Case

Table 2: Architectural Design Specification - Version History

[1] Lauritsen et al. 2022: Requirements Specification

3. Architectural specification

To develop the architecture of the system we use Krutchen's 4+1 model[2]. It contains a logical view, process view, development view, and physical view. The +1 is the scenarios, which are linked to the use cases created in the Requirement Specifications Document[3].

3.1 Physical View

The physical view is described in Sommerville's book Software Engineering as the view *"which shows the system hardware and how software components are distributed across the processors in the system. This view is useful for systems engineers planning a system deployment"*[4]. Thus diagrams suited for displaying this view would be a UML Deployment Diagram which can be seen in Figure 1.1.

3.1.1 UML Deployment Diagram

This diagram shows how the different software components are distributed between the different hardware components. We also see how the components share information by the lines or arrows connecting them. To differentiate between directed and non-directed connections the use of arrows is used for directed connections.

[2] Sommerville, 2016: p 174.
[3] Lauritsen et al. 2022: Requirements Specification
[4] Sommerville, 2016

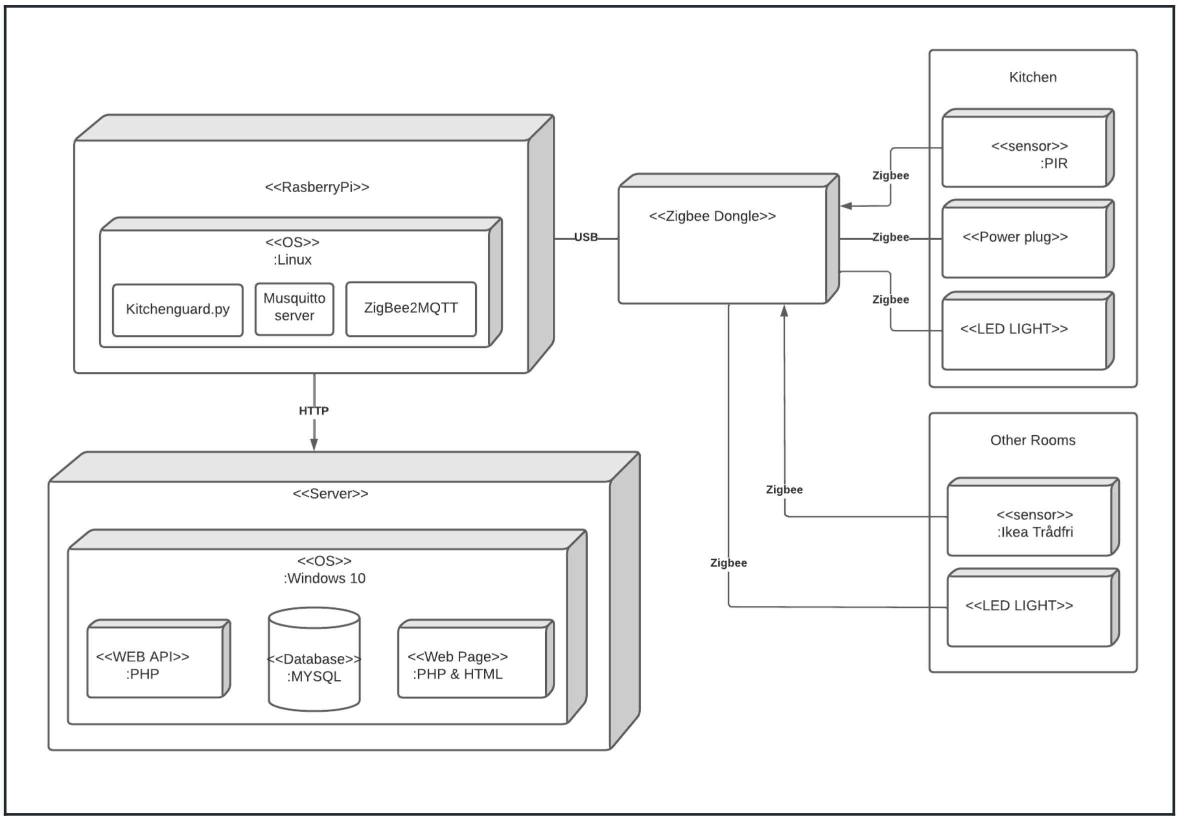

Figure 1.1 UML Deployment diagram

In the UML Deployment Diagram, we see the use of seven different components, which can be seen listed in table 1.3. We also see the use of three different connection types, which can be seen in table 1.4.

Components		
Component Name	Model	Quantity
Xiaomi Aqara Motion Sensor	RTCGQ11LM	Up to 5 (Collectively with Ikea sensor)
IKEA Trådfri Motion Sensor	E1525/E1745	Up to 5 (Collectively with Aqara sensor)
GLEDOPTO LED Light Strip	GL-MC-001PK	Up to 5
Immax NEO power unit Power Plug	07048L	1
Raspberry Pi4		1
Zigbee Controller USB Dongle	CC2531	1
Windows 10 Server	Any	1

Table 1.3 System components

Connections	
Name	Full name
HTTP	Hypertext Transfer Protocol
USB	Universal Serial Bus
Zigbee	

Table 1.4 System connections

3.1.1.1 Sensors

In the system we have three different sensors. We have two motion sensors, which are the Aqara and the Ikea motion sensors. These are used to detect the user's presence in a given room throughout the house. These sensors are interchangeable with each other and there can be used up to a collected number of five of the two sensors. The last sensor there is in use is the NEO power plug which is used to sense which state the stove is in as well as the power output of the plug. The states can be ON or OFF, while the power is used to determine if the stove is powered on.

3.1.1.2 Actuators

In the system we have two different Actuators. We have the GLEDOPTO LED Light Strips which are used to alert the user throughout his home if the stove has been left on. There can be used up to five of these light strips throughout the house. The other actuator in use is the NEO power plug which can be used to cut the power to the stove if it has been left unattended for too long.

3.1.1.3 Zigbee Controller

The Zigbee controller is used to receive the data from the sensors and actuators through a Zigbee signal and transfer it to the Pi using a USB connection.

3.1.1.4 Raspberry Pi

The Pi will be collecting the data from the Zigbee Controller and use the information to give commands to the actuators of the system. We can see in the deployment diagram that three different processes are executed on the Pi. These are the Kitchenguard.py, the Mosquitto server, and the Zigbee2MQTT interpreter. These will be elaborated on in the following sections.

3.1.1.4.a Kitchenguard.py

Kitchenguard.py is the program that contains most of the implementation of the requirements specified in the Requirement Specifications Document[5], and is written in the programming language Python. The Kitchen Guard program will take the events from the Zigbee sensors, determine which event is taking place, and decide which actions need to be taken. During this, the program will also establish which event from a premade list of events is happening and encode it using the HEUCOD python file given in tutorial 9[6], whereafter it will send it as a JSON string to the Server Pc via the HTTP protocol.

3.1.1.4.b Mosquitto server

The Mosquitto server is a broker which will allow communication to the system through the MQTT protocol standard. This will allow developers to review and send commands to the different Zigbee components from other devices.

3.1.1.4.c Zigbee2MQTT

The Zigbee2MQTT is tasked with converting the events given from the Zigbee controller into an MQTT format which can then be handled by an MQTT broker like our Mosquitto server.

3.1.1.5 Server Pc

The Server Pc is used to store our data in a database and host our Web Page. As seen on the HTML Deployment Diagram that three different processes are run on the server Pc. These are the WEB API, the Database, and the web page. These will be elaborated on in the following sections. The program used to run these are installed through Wamp.net[7]. Which is used to install the applications APACHE, MySQL, ADMINER, and the programming language PHP.

3.1.1.5.a WEB API

This is the program called by our Kitchen Guard program from the Pi. The connection is established using the Application APACHE. The program's functionality is that it receives a JSON string sent from the pi, decodes it, and then saves them to the database.

[5] Lauritsen et al. 2022: Requirements Specification
[6] Jmiranda-au, 2022: cep2heucod
[7] Wamp.net, 2022

3.1.1.5.b Database

This is the database used to store the events received from the Pi. It is created using the Application MySQL and is set up using the application ADMINER.

3.1.1.5.c web page

This is an HTTP web page hosted using the application APACHE and is written using the programming language PHP and HTML. This is the program that lets the user review the data sent to the Server Pc, and determine when the user has left the stove on.

3.2 Logical View

The logical view is described in Sommerville's book Software Engineering as the view *"which shows the key abstractions in the system as objects or object classes. It should be possible to relate the system requirements to entities in this logical view"*[8]. Thus diagrams suited for displaying this view would be UML Class Diagrams or UML State Diagrams.

3.2.1 UML Class diagram

To give an overview of the functions and attributes of the different classes we use class diagrams. They are a simple way to get an understanding of the different software components of the program.

We have based our code on the code given as part of tutorial 6[9]. This means that most classes have remained untouched and we will therefore focus on the one class we did change. In the diagram below we can see that the functions and attributes marked with green are what we have created or changed the functionality of. In other classes, we have changed minor things like variables for addresses of the devices and web server.

[8] Sommerville, 2016: p.174
[9] Jmiranda-au, 2021: cep2app

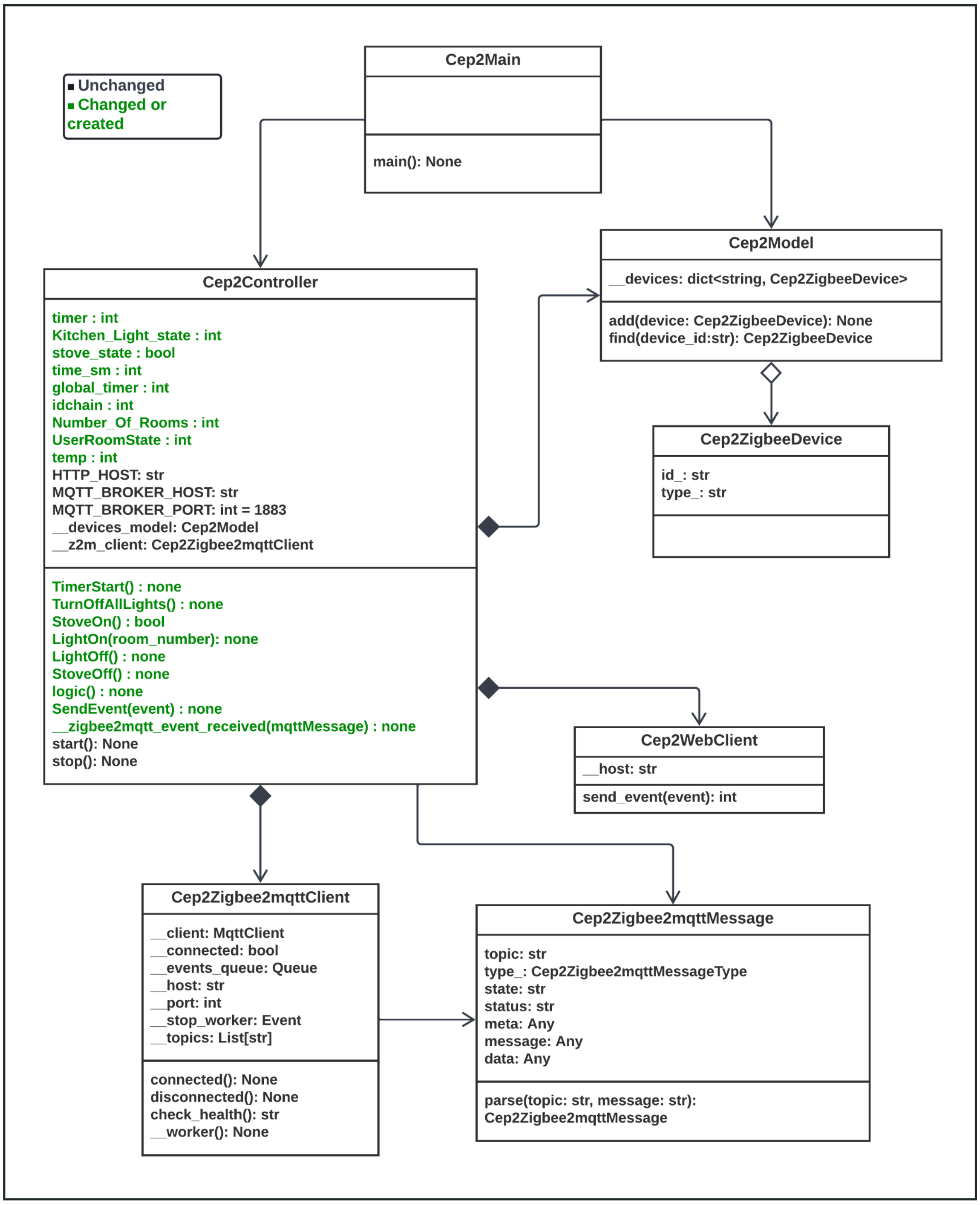

Figure 1.2 UML class diagram

The __zigbee2mqtt_event_received() function has been changed and the rest of the "changed or created" functions/attributes have been created by us.

3.2.2 UML State diagram

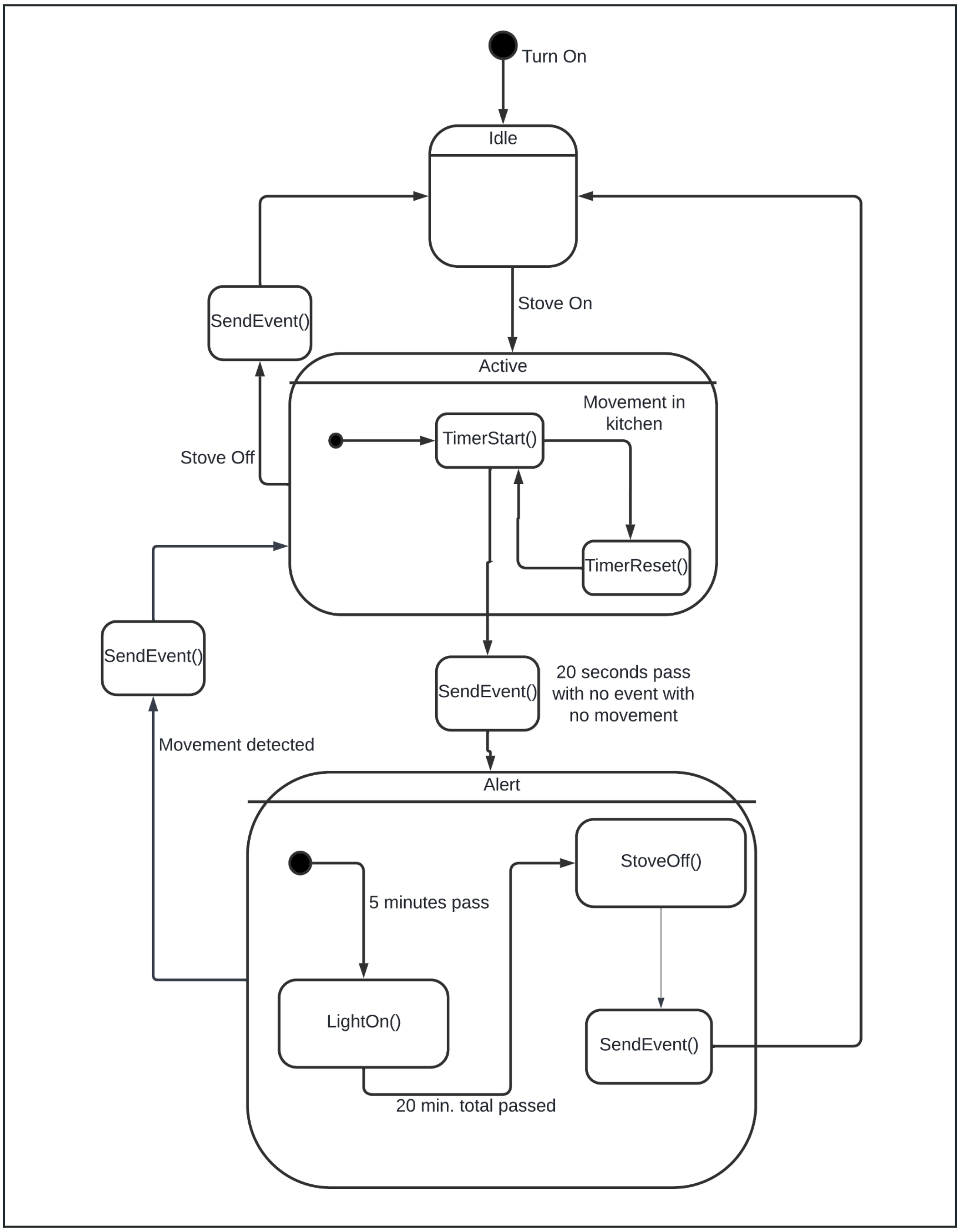

Figure 1.3 UML State Diagram

In this state diagram we see the three different states of the system. The system is idle when the stove is turned off. It then switches to active when the stove is turned on, from which it will monitor the kitchen for movement and reset a timer each time it receives confirmation that the user is in the kitchen. When the timer passes a certain threshold the system switches to alert state. Here the system thinks the user has left the kitchen and will alert the user with lights after 5 minutes and ultimately turn off the stove if the user does not return within 20 minutes. When leaving the alert state the system will log the data in the database for a third party to monitor.

3.3 Process View

The process view is described in Sommerville's book Software Engineering as the view *"which shows how, at runtime, the system is composed of interacting processes. This view is useful for making judgments about non-functional system characteristics such as performance and availability"*[10]. Thus diagrams suited for displaying this view would be UML Sequence Diagrams and UML Activity Diagrams.

3.3.1 UML Sequence diagram

To show the design of key functions in the program, we will use sequence diagrams, which show how the functions execute and how they call other functions depending on the state of the program. In the following sequence diagrams, we are seeing functions where classes would normally be in a sequence diagram. This is because the main functionality of the system is contained within the cep2controller class. So to illustrate the actual functionality we have changed the scope of the diagrams to focus on two functions within the cep2controller class and how they interact with other functions in the system.

We see in Figure 1.4 a sequence diagram of the logic and __zigbee2mqtt_even_received functions. The boxes tagged with "Alt" are alternative routes the function can take depending on the conditions stated in the top left of each "Alt" box.

The lines indicate actions and only the ones named "function call" are function calls. This means that the text next to the curved lines is the actions performed by the function.

[10] Sommerville, 2016: p.174

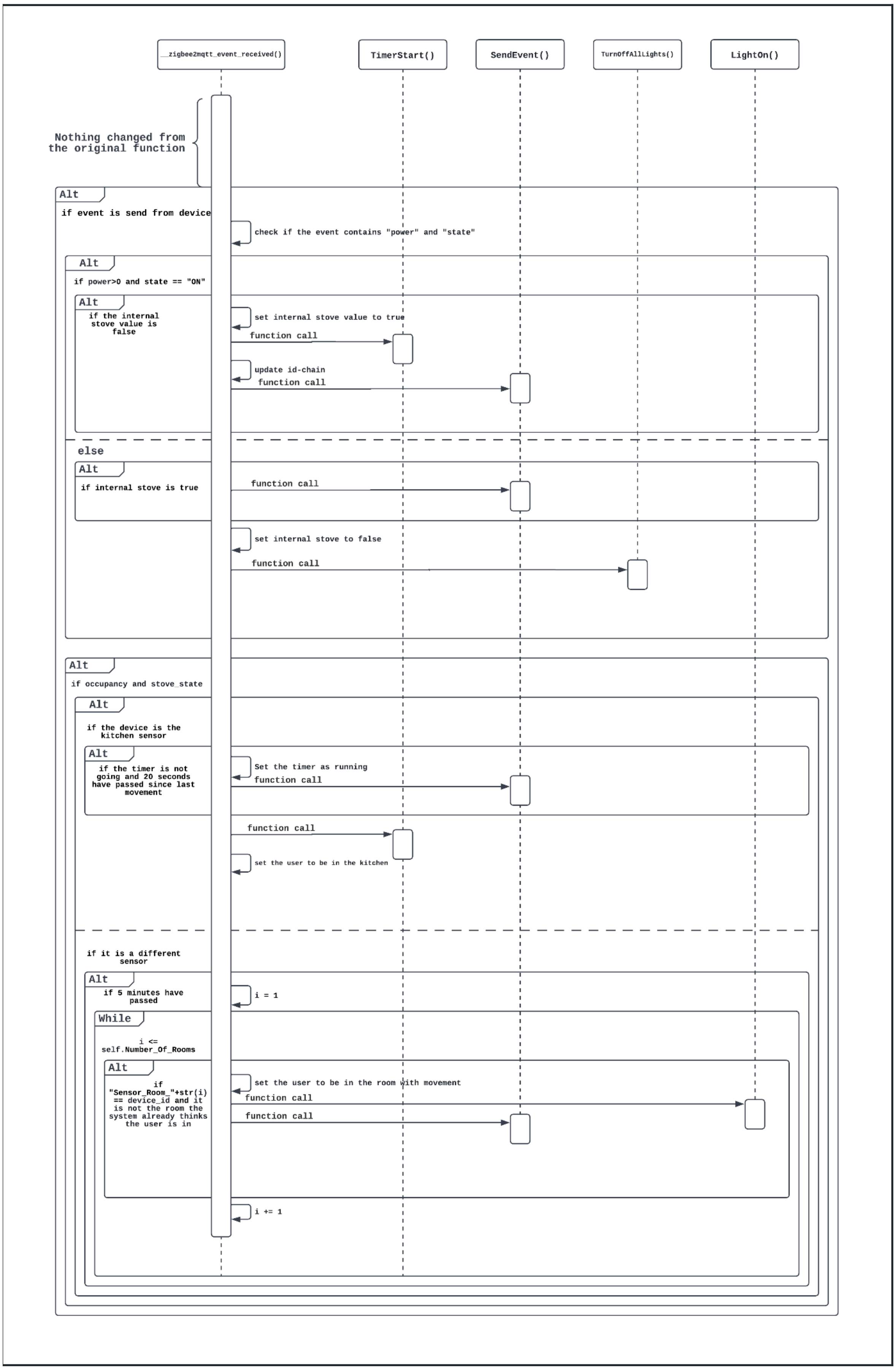
__zigbee2mqtt_event_received()
TimerStart()
SendEvent()
TurnOffAllLights()
LightOn()
Nothing changed from the original function
Alt
if event is send from device
check if the event contains "power" and "state"
Alt
if power>0 and state == "ON"
Alt
if the internal stove value is false
set internal stove value to true
function call
update id-chain
function call
else
Alt
if internal stove is true
function call
set internal stove to false
function call
Alt
if occupancy and stove_state
Alt
if the device is the kitchen sensor
Alt
if the timer is not going and 20 seconds have passed since last movement
Set the timer as running
function call
function call
set the user to be in the kitchen
if it is a different sensor
Alt
if 5 minutes have passed
i = 1
While
i <= self.Number_Of_Rooms
Alt
if "Sensor_Room_"+str(i) == device_id and it is not the room the system already thinks the user is in
set the user to be in the room with movement
function call
function call
i += 1

Figure 1.4 UML Zigbee2mqtt_event_recived Sequence diagram

In Figure 1.5 we see the sequence diagram for the __zigbee2mqtt_event_received function. This function was part of the given code and has been changed. The unchanged part is shown in the beginning and only the last part of the function, which we have changed, is shown.

In the diagram, we see how the function determines what component sent the event and determines what to do accordingly.

Next is the sequence diagram of the logic function. This handles the information the __zigbee2mqtt_event_received function has interpreted and is responsible for turning off the stove and turning on the lights when needed. It also handles two timers, one small timer to determine if the user has left the kitchen and a larger timer to determine the time the user has spent away from the stove.

This function is called periodically once every second, which means that every second it uses the latest information and can therefore make decisions on updated information.

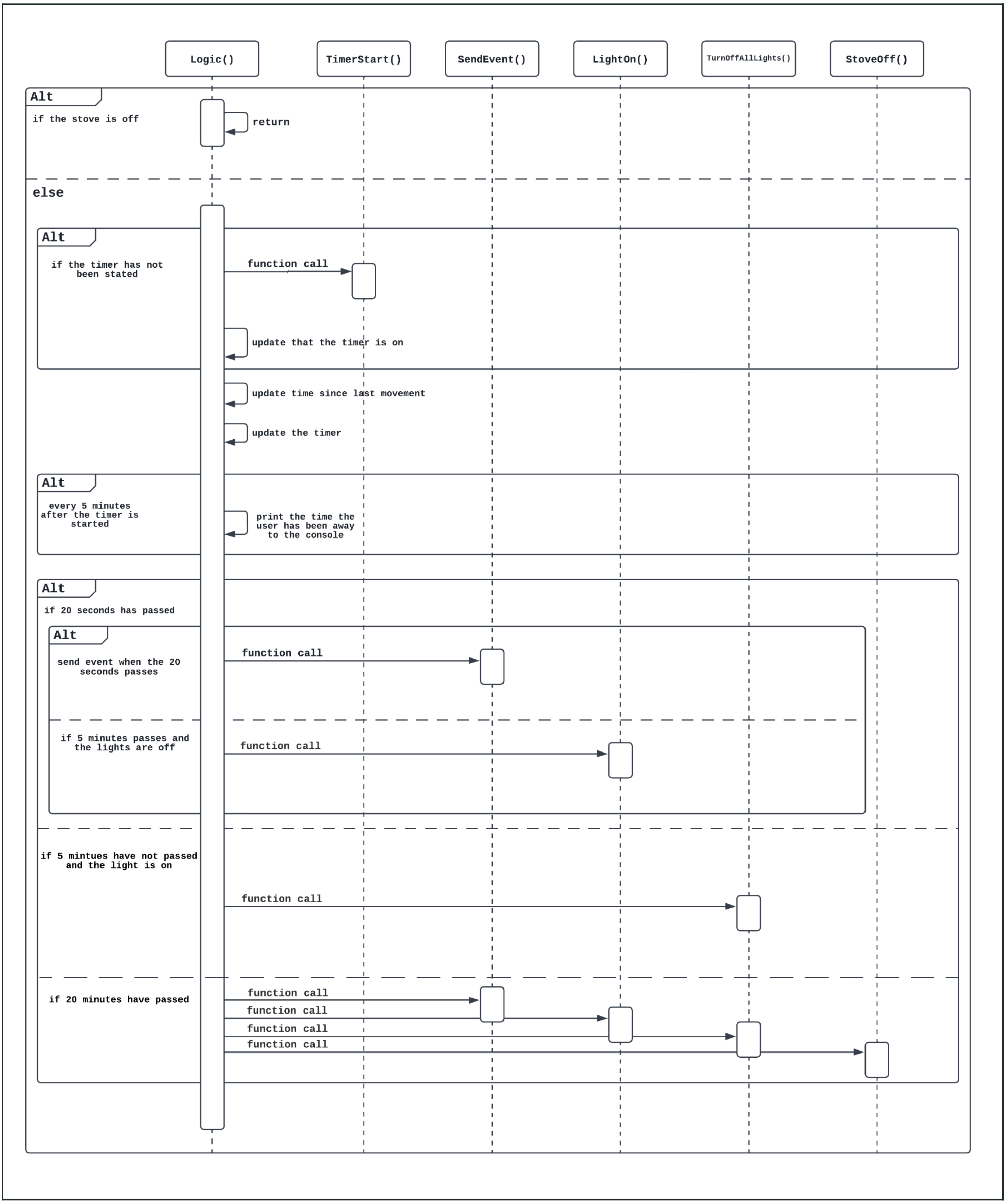
Logic()
TimerStart()
SendEvent()
LightOn()
TurnOffAllLights()
StoveOff()
Alt
if the stove is off
return
else
Alt
if the timer has not been stated
function call
update that the timer is on
update time since last movement
update the timer
Alt
every 5 minutes after the timer is started
print the time the user has been away to the console
Alt
if 20 seconds has passed
Alt
send event when the 20 seconds passes
function call
if 5 minutes passes and the lights are off
function call
if 5 mintues have not passed and the light is on
function call
if 20 minutes have passed
function call
function call
function call
function call

Figure 1.5 UML sequence diagram of the logic function

3.3.2 UML Activity diagram

An activity diagram that describes how the system responds to different actions of the user can be seen below.

For an activity to begin it is required that the stove is turned on. When this precondition has been satisfied the controller will start monitoring for movement in the kitchen, see UC1. What this means is that the controller will wait for events received from the sensors. If it receives an event from the kitchen sensors within 20 seconds it will continue monitoring. If it has not received an event from the kitchen sensor indicating user presence for a period of more than 20 seconds, however, it will start a timer, see UC2. If the user returns to the kitchen before the timer reaches 5 minutes it will go back to monitoring the stove. If the user does not return, however, the controller will start waiting for other sensor events indicating which room the user is in, see UC4. If it does not receive any event it simply does nothing, however, if it receives an event from one of the sensors in one of the rooms, the controller will alert the user by turning on an LED in the corresponding room, see UC3. If movement is detected in the kitchen before 20 minutes pass, the controller will go back to step 1 and continue monitoring the stove. If not, however, the stove will be turned off, see UC6.

Throughout this activity flow chart, the controller will at some points send events to the web API, this is indicated by the hexagon figure, see UC7.

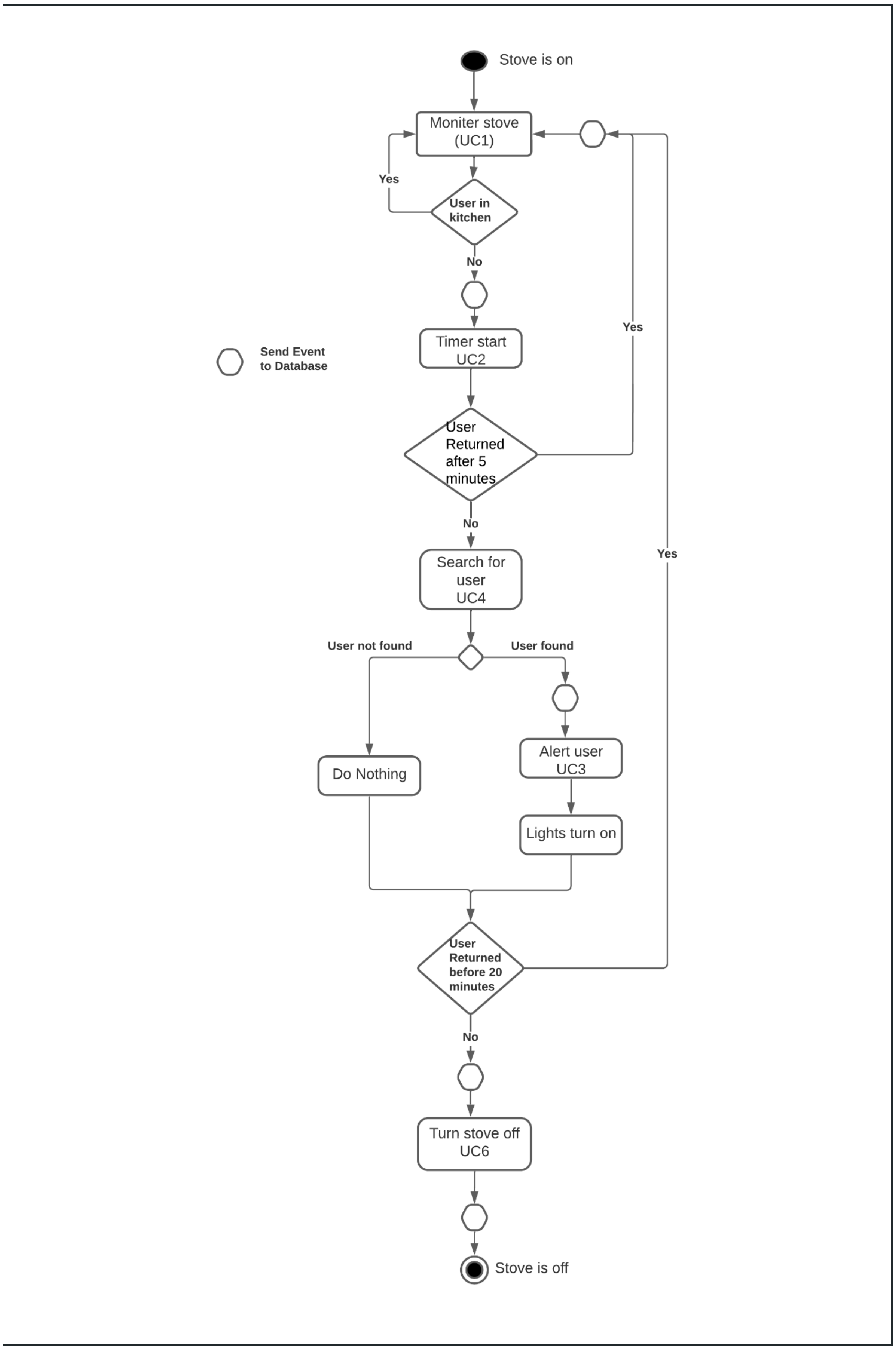

Figure 1.6 UML Activity diagram

3.4 Development View

The development view is described in Sommerville's book Software Engineering as the view *"which shows how the software is decomposed for development; that is, it shows the breakdown of the software into components that are implemented by a single developer or development team. This view is useful for software managers and programmers* "[11]. Thus a diagram suited for displaying this view would be UML Package Diagrams.

3.4.1 UML Package diagram

Figure 1.7 shows a UML package diagram of the software of the whole system. The dotted lines symbolize communication. The arrow shows directed communication. That means that Kitchenguard.py can communicate with the Web API but the Web API cannot communicate with Kitchenguard.

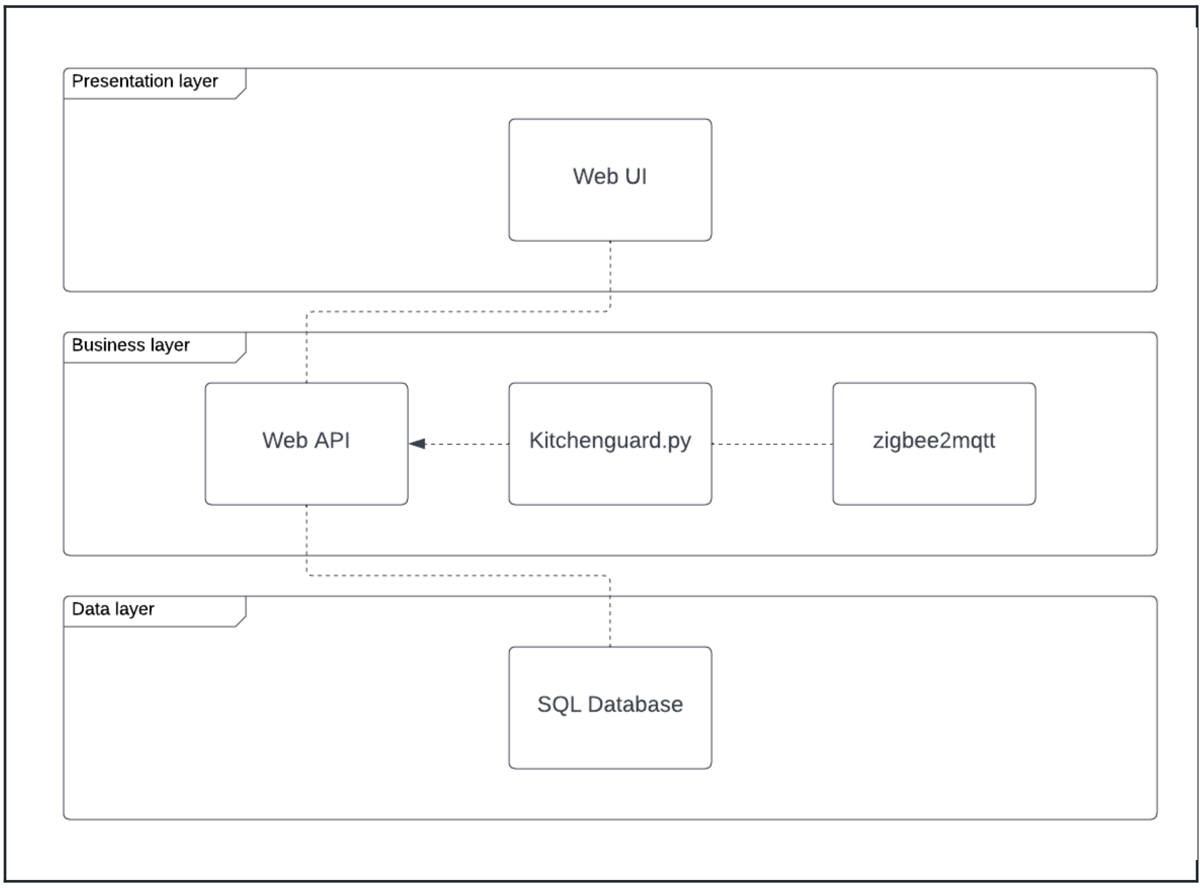

Figure 1.7 UML package diagram

[11] Sommerville, 2016: p.174

4. Sources

1. Eclipse, Mosquitto documentation, July 2020. URL: https://mosquitto.org/documentation/.
2. Wamp.net, Wamp.net Documentation, February 2022: https://wamp.net/docs
3. Jmiranda-au, 2022: cep2heucod, URL: https://github.com/jmiranda-au/cep2heucod
4. Jmiranda-au, 2021: cep2app, URL: https://github.com/jmiranda-au/cep2app