Smart and Secure Home

Sheikh Muhammad Adib bin Sh Abu Bakar
TuringJuniors

Soest, Germany
sheikh-muhammad-adib.bin-sh-abu-bakar@stud.hshl.de

Zafirul Izzat bin Mohamad Zaidi
TuringJuniors

Soest, Germany
zafirul-izzat-bin.mohamad-zaidi@stud.hshl.de

Ammar Haziq bin Mohd Halim
TuringJuniors

Soest, Germany
ammar-haziq.bin-mohd-halim@stud.hshl.de

Hadi Imran bin Md Radzi
TuringJuniors

Soest, Germany
hadi-imran.bin-md-radzi@stud.hshl.de

Leuteu Feukeu Evrard
TuringJuniors

Hamm,Germany
evrard.leuteu-feukeu@stud.hshl.de

***Abstract*—The construction and conversion of ordinary homes into "smart homes" has seen an unprecedented growth in recent years. This can be ascribed to technologies such as the Internet of Things, sensors, smart phones, smart appliances, cloud computing, and digital assistants that we have nowadays such as Siri made by Apple, Alexa made by Amazon, Cortana for Microsoft and so on. At the outset, smart homes were designed to enhance the quality of life for us human beings. Impressively, we have seen people that utilize smart home reaping the benefits of security, energy conservation, and the ability to monitor their lighting, door locks and also window while they are in their space of comfort by using only their fingertips, for example in bed or siting on a couch. Of course, being able to monitor home devices using smart technologies could be a tremendous benefit and a great help to anyone, particularly when you are sure that your home is safe and sound even though you are far from home. This paper presents a smart home system that will focus more on security. In the concept part, all the diagrams starting from class diagram, use case diagram and every single sequence diagram of each device will be clarified. This post would also show a source code and a simulation in the evaluation to assist the reader get a basic and a rough understanding. Furthermore, the reader may read our goals and what we have accomplished during the preparation of this paper on the outlook and conclusion. Last but not least, at the end of the article, the reader can find additional detail about our smart home project in the appendix section.**

Keywords—secure home, smart home

I. MOTIVATION

We will start by defining smart home and describing the problem with today's smart home system. So, what is a smart home? A smart home is a home, which uses smart technology to do certain tasks and operate certain devices, many of which involves:

- Lighting
- Heating and cooling
- Door locks
- Home security
- Home entertainment
- Kitchen and laundry appliances.

But what makes it "smart"? The "smart" technology can sense what is happening in its surrounding using a particular sensor or device and perform tasks based on the information that it collects. The simplest example is where a person walks into a room, and the smart technology will sense the person entering and will automatically turn the light on or off, depending on the instruction programmed into the device.

Nowadays smart home has been evolving to do more tasks to make homes smarter. However, there are still disadvantages to the current smart home systems available to the consumer, some of which are:

- Privacy issues
- Security issues
- Safety issues
- High installation cost
- Requires an expert when a problem arises

Different protocol required for different smart devices from different companies.

We strive as a group to reduce as much disadvantage as possible and improve on the current smart home system in our project. However, we figured that safety is the most important aspect in a home. It does not matter how smart the home is, if it is not safe for the inhabitants, then it does not feel like home. As a result of this philosophy, we decided to put our full focus on one aspect, the security of the home.

As a result of our development, we have managed to solve some of the disadvantages that the current smart home systems have.

To solve the problem of different protocols required for different smart devices from different companies, we decided to use a centralised system. With this method, instead of the devices communicating with each other to create a sequence of action, the devices only have to communicate to what we called a control hub, thus creating a centralised system with the control hub being the so called "brain" of the system.

As we mention before that our system is focusing on the security of the home, we have planned a scenario where the system tries to prevent a thief that try to enter the house. It starts with the smart door asking the user for the pin to verify whether it is the user or an intruder trying to enter the home.

If the user enters the pin successfully, obviously the door will unlock and let the user enter the home.

But if the wrong pin is entered, the door will communicate with the control hub and the control hub will send a signal to application to send a notification to the user's phone to verify one more time to make sure that the user did not enter the wrong pin on purpose. The user can choose to either press the "It is me" button or "It is not me". If the user pressed the "it is me button", the smart door will unlock the door. However, when the other button is pressed, the smart door will be locked, while the smart alarm will ring, and the smart lamp will blink to attract attention. The application in the user's phone will initiate an emergency call to alert the authorities.

Note that every single communication of the devices must go through the control hub so that everything is centralised. This enables the system to add or remove devices without any faults that cannot be resolved by the user.

II. Approach

To develop the system in more effective and efficient way, we first plan the process. The reason behind this, is we can produce a complete system in a short time, because the change in requirement can be done at any time since this model allow iteration. So, the system produced can be much better than using waterfall model and the tension during the development can be reduced. The incremental development model is illustrated on Figure 1[1].

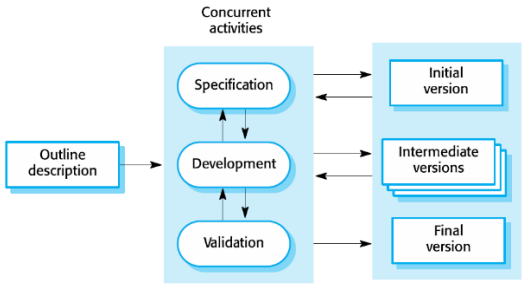

Figure 1[1]

To produce an organized system, a concrete system architecture must be carefully planned which is done through analysis and design process. First, requirements that contain the solutions to the discovered problem and new functionalities are listed forming the requirements. This process called specification process and can be modelled as shown in figure 3[1]. Next, we go to development process. This is where design and implementation process occur. Design process can be visualized as shown in figure 3[1]. Based on the requirements the raw system architecture that so called system prototype and class diagram are sketched. After that number of scenarios are created to develop clear sequence and use case diagram that are needed to produce concrete system architecture. So, in this smart home project, more than one smart device is being develop. This is because the system architecture must fulfil the interaction that occurs in each smart devices and scenarios. With clear sequence diagrams and concrete system architecture, State machine can be easily built for implementation. The last process steps are validation and defect test to check whether the system fulfil the requirements. This is where the iteration could be happened because the created system may contain fault. The detail for each process steps, e.g., sequence diagram for each scenario and source code for the implementation, are discussed in further subtopics. After the analysis and design processes have been completed, implementation is our next step. The programming language that being used for implementation is C++. As mentioned before, we do validation and defect test after implementation. This process flow is shown in figure 4[1].

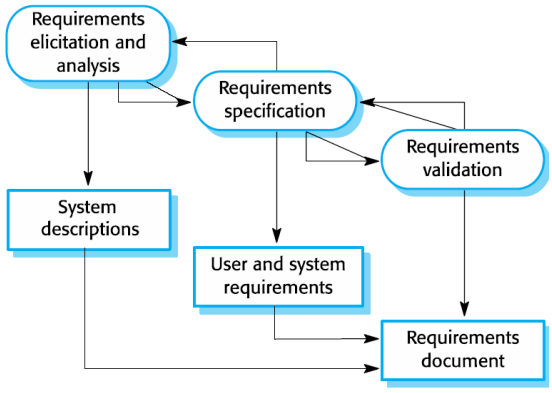

Figure 2[1]

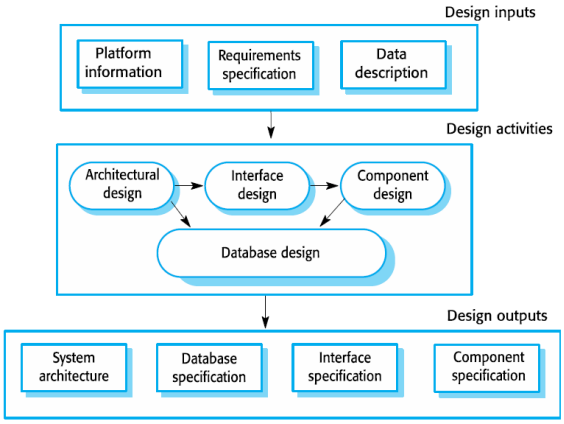

Figure 3[1]

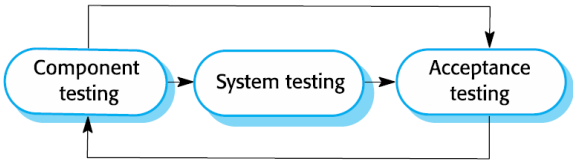

Figure 4[1]

To achieve a system that run smoothly and remove as much fault as we can, we did simulation for this project and Arduino is choose as platform for simulation.

III. Concept Part

A. *prototype*

The first phase of our project began with a pre-study where basic knowledge on Smart Homes was gathered. The aim of the pre-study was to get a summary of smart home technologies in general. The pre-study also enabled the search for content to be used as a guide in this research, such as details on smart home technologies.

Therefore, we now have understood the application domain and problem domain as mentioned before. Using these understanding, we documented the requirements for this project with the new functionalities as discussed before. In said requirements is a textual scenario that we developed for each smart device which is the solution to the problem domain

Since we want to make it clear what our project is all about, the first solution is to develop our first prototype or know as raw architecture as shown in figure 5 after the information have been compiled. One of the key factors is that the raw system architecture itself is a static diagram which is usually used to model the static view of the structure.

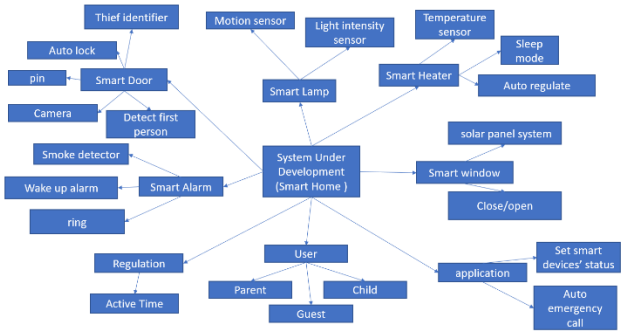

Figure 5 : prototype of system architecture

In the upcoming section, we will slowly introduce our carefully planned precise steps into building our final product. Having already identified all the components that we wanted to integrate into our system, our next step is to construct formal specifications through UML diagrams. Firstly, we needed a class diagram to further, not only add details to the elements but also refine their interactions in our smart home system. Everything regarding class diagrams, its definition, usage and more will be explained shortly in the next section.

B. *Class diagram*

Class diagram is not only used for visualizing, defining and recording various facets of the model, but also for creating the executable code of the software application. You can find the code below located in the evaluation part. This diagram defines the characteristics and operations of the class, as well as the restrictions placed on the system. By developing the class diagram, we can make the environment entities and the relations between the elements much clear.

After Class diagram was established, the attributes and functionality of each object were figured out by low fidelity sketches. The drawings represented how the objects and its functions would work internally. By using drawings that could quickly and easily be modified or redrawn. Since the project is done separately, every part of the object has its own class diagram. After finish with sketches, we finally have come out with complete Class diagram. As seen in figure 6.

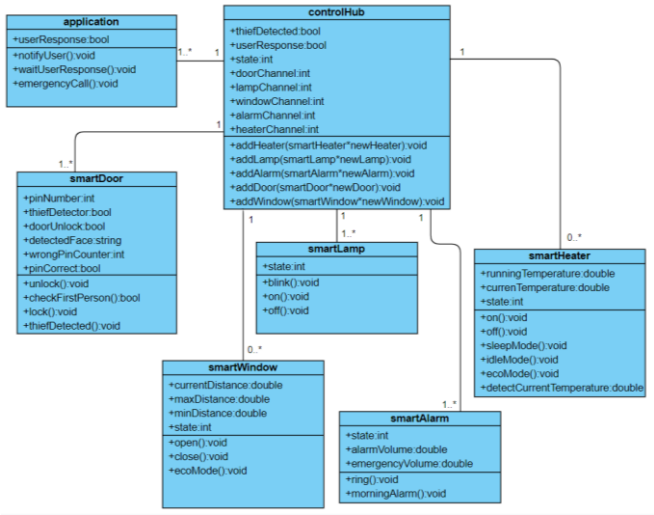

Figure 6 : class diagram for the system

Based on the diagram, we can see that the Control Hub is the centre between all devices in the system. The Hub contains features like thief detected which is to detect either the person is a thief or not. Next, the attribute of the Control hub also contains user response. This attribute will automatically link to the application. So, the owner of the house or to be exact the person that has an application in their phone can easily know and verify the person that want to enter the house. As you can see on the diagram, our Control hub also has their own functions such as addHeater, addLamp, addWindow, addDoor and addAlarm. To conclude our initial statement, we can say that the Control Hub is the brain of our Smart home project. It will connect the application and also smart devices such as smart window, smart door, smart alarm, smart lamp and smart heater. Since every smart device in our system undergoes the same steps which are getting input from environment, then the information is passed through to the control hub to take further actions, we can generalise the interactions and come up with the overall system architecture. Further explanations regarding the overall system architecture will be explained in detail in the upcoming subtopics of our paper.

Other than the control hub, we have also defined all the entities of each smart device. As you can see, to make the connection work very well, each smart device and also application has the same attribute which is controller. In application, it consists of three functions. The first one is notify. This function will notify user if anything happens inside or outside the house. Then it also has emergency call to make sure the user alert 24/7 if there are unexpected things occur. Next, in smart door. The functions are unlock and check first person. The door will unlock after the first person is check and enter correct pin number, but if the pin number is wrong, the door will remain lock and the user will receive a notification for a verification either the person is a stranger or not. Furthermore, we also made smart lamp. There are a few entities as you can see in the diagram such as blink and on. The lamp used movement sensor to detect if there is a person in that compartment or not. So, when the person enters the bed room as example, the lamp will automatically on. So, when will the lamp blink? This will occur when there is an emergency situation occur such as when thief is detected or when there is a smoke in the house. Now, we move on to smart window, we included three functions which are close, open and eco mode. To help the reader understand on how it will

work especially for an eco-mode, the rough idea is a tiny transparent layer of solar panel will be located on the window. Therefore, the user can use solar energy when it is available instead of main power source. Then, in smart alarm we have ring as a function. As an additional information, the alarm itself not only ring when there is an emergency, for example when there is a smoke or a thief that want to break the house. You can also set an alarm to help you wake up more easily in the morning. Not to forget, the tone will be set by user through our application. With variety of tone, it can help user to differentiate when there is emergency or just a normal alarm. Last but not least, in smart heater we also made on, off, idle mode and also sleep mode function. The uniqueness of smart heater is that it has sleep mode function. So, whenever you want to sleep at night, the heater will automatically set to your requested temperature. As a consequence, it will help you get a deep and beauty sleep.

In order to get a clearer picture on what is needed for the system to run based on the contextual scenario that we have chosen earlier, we need to further refine the exact requirements. We accomplish this goal by modelling the scenario using use case and sequence diagrams. Specifically, the use case diagram's main function is to make the boundary between each device in the system and their respective environments much clearer. On the other hand, the sequence diagram is used to identify any flaws and deficits of our system in the scenario.

C. Usecase diagram

With both of the information that we gathered from said diagrams, we have a clearer vision and better understanding of the overall system. This is important as we can now develop a more concrete system architecture with refined details. That is exactly what we did on this research paper for the reader to easily digest our overall system architecture. This is also further explained at the later subtopic section in the paper.

We first make the use case for the control hub since it is the centre of our system and then followed by all the other smart devices. The use case that we have built for the control hub is based on our main integrated scenario where the system gives a notification to the house owner when a person tries to enter the house. If the owner gives a positive response, the person is safe and known, he or she is free to enter. Otherwise, the system tries to prevent the unknown person or thief from entering the house without the permission of the owner. We made a use case diagram to summarise the scenario above so that the reader finds it easier to distinguish the boundaries between the system and the environment. The use case diagram for the complete system is shown in figure 7.

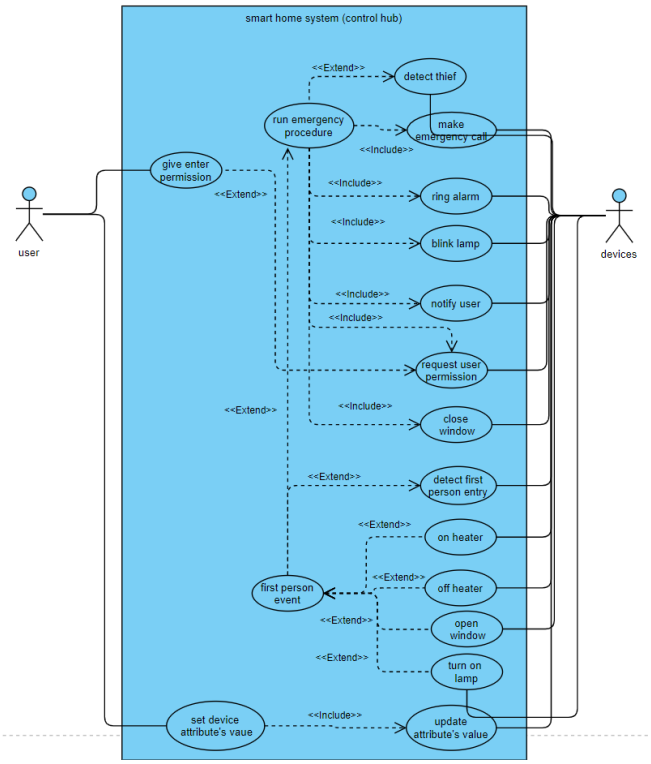

Figure 7 : use case diagram for control hub

Other than that, we also made individual use case diagrams for each device and those are shown below:

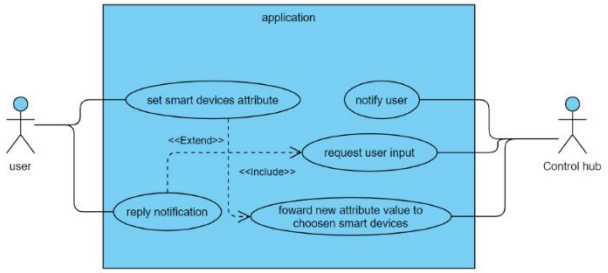

Figure 8 : use case diagram for application

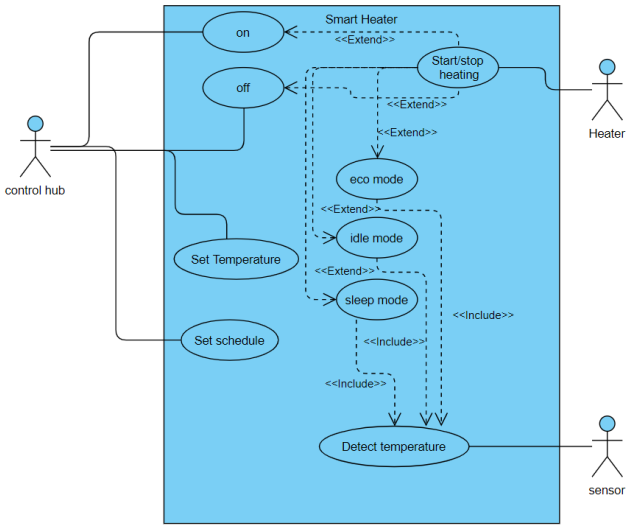

Figure 9 : use case diagram for Smart Heater

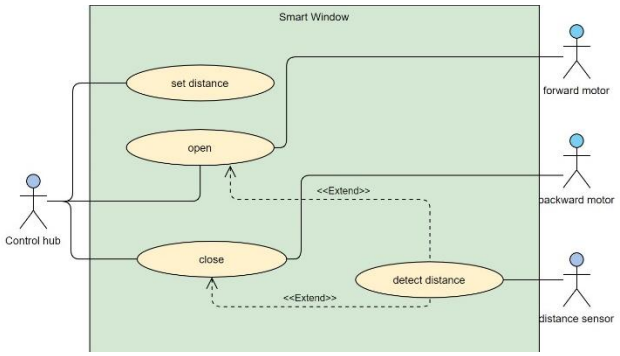

Figure 10 : use case diagram for Smart Window

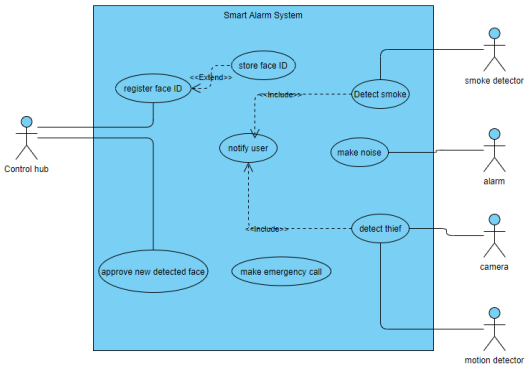

Figure 11 : use case diagram for Smart Alarm

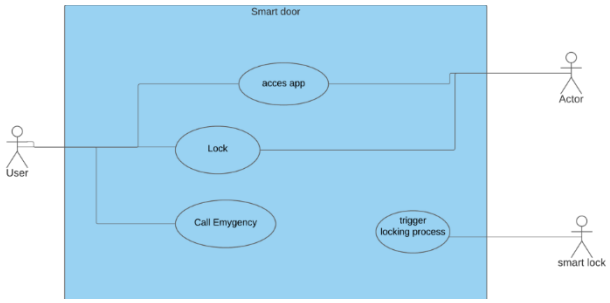

Figure 12 : use case diagram for Smart Door

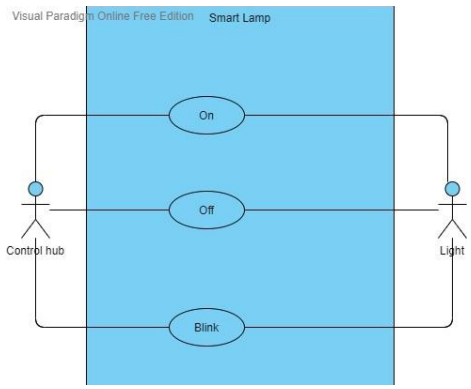

Figure 13 : use case diagram for Smart lamp

D. *Sequence Diagram*

In the following section, we continue our development process with the addition of a sequence diagram. With the sequence diagram, we can see the flow of use of the system given the context situation. More importantly, we again use the analysis of the requirement and all of the preceding diagrams that we have built so that we can design a sequence diagram for the overall system with control hub as the system centre. This is shown in figure 14 .We now have the ability to make the interface clear, because, as mentioned before, the sequence diagram is used to identify any flaws and deficits of our system in the scenario.

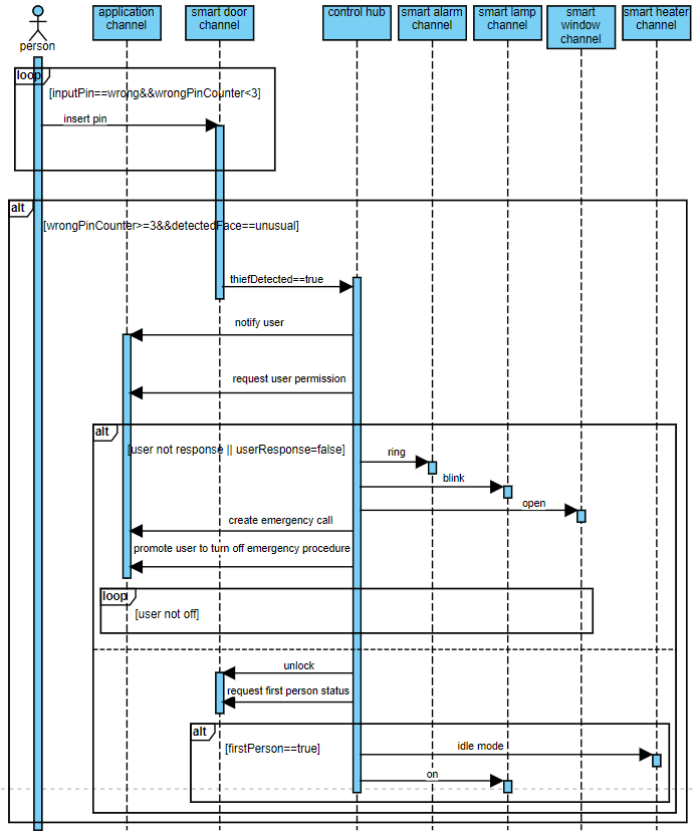

Figure 14 : Sequence diagram for control hub

After that, we made multiple sequence diagrams for each device, so that way, we can obtain the detailed analysis for the whole system. All of the sequence diagrams for each smart device are shown below:

1) *application*

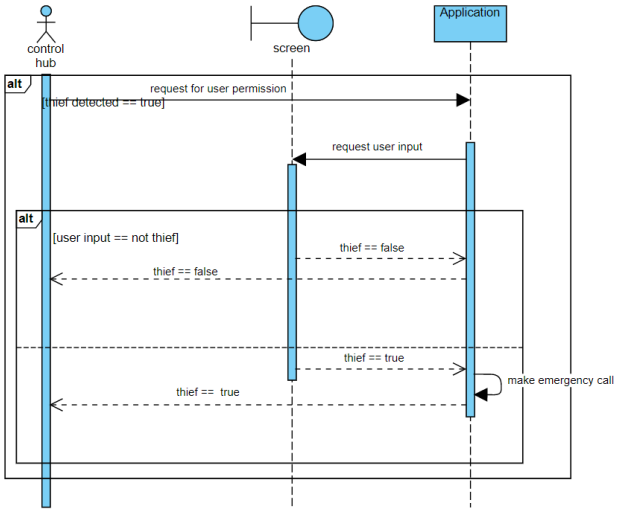

Figure 15 : Sequence diagram for application

2) smart heater

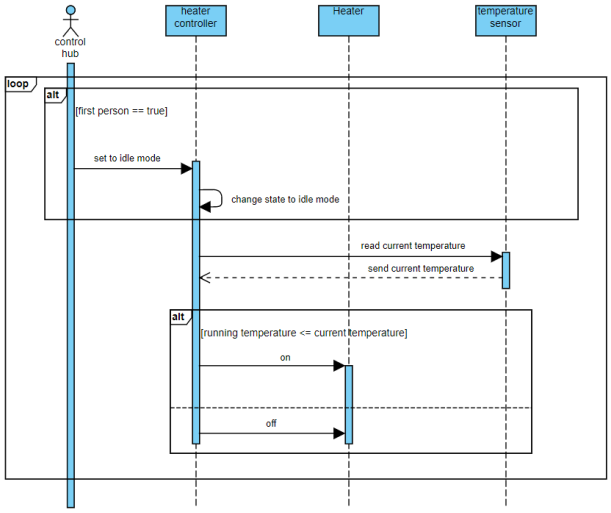

Figure 16 : Sequence diagram forSmart Heater

3) Smart alarm

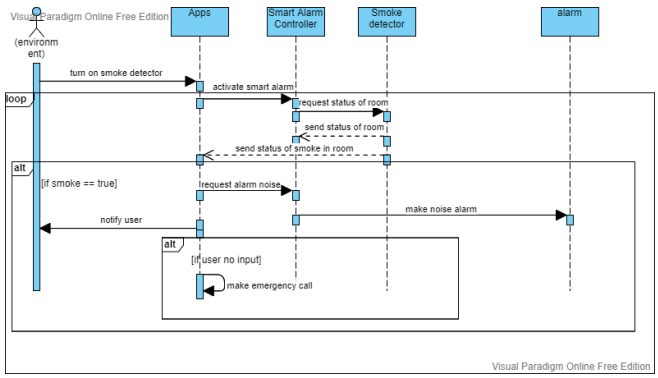

Figure 17: Sequence diagram for Smart Alarm

4) Smart lamp

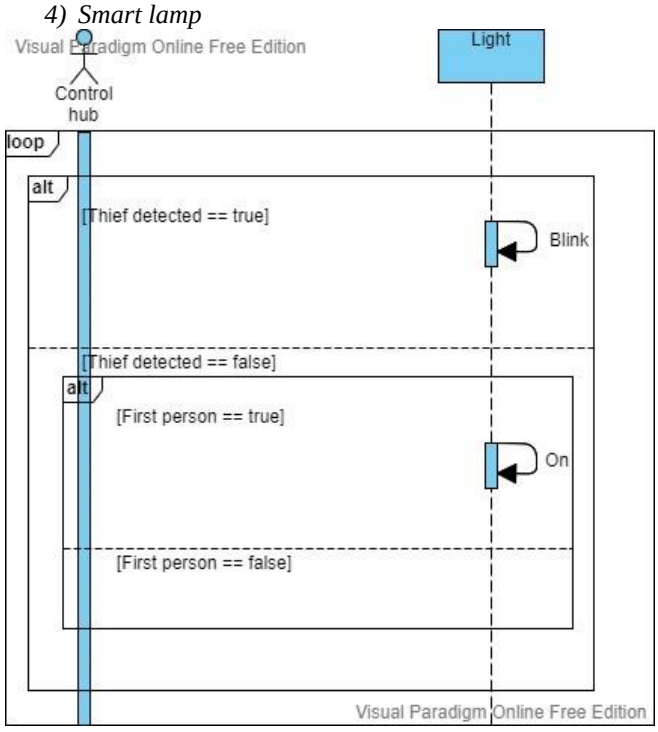

Figure 18 : Sequence diagram for Smart Lamp

5) Smart window

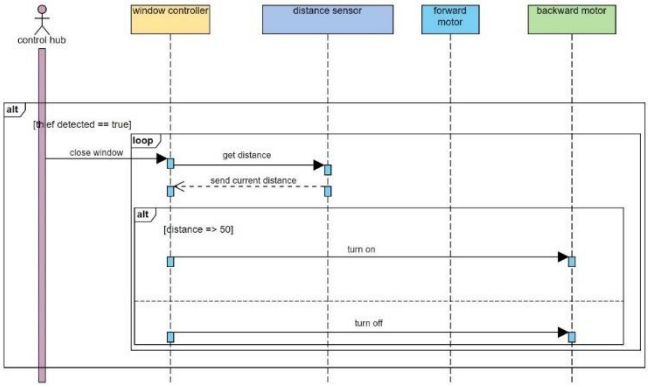

Figure 19 : Sequence diagram for Smart Window

6) Smart Door

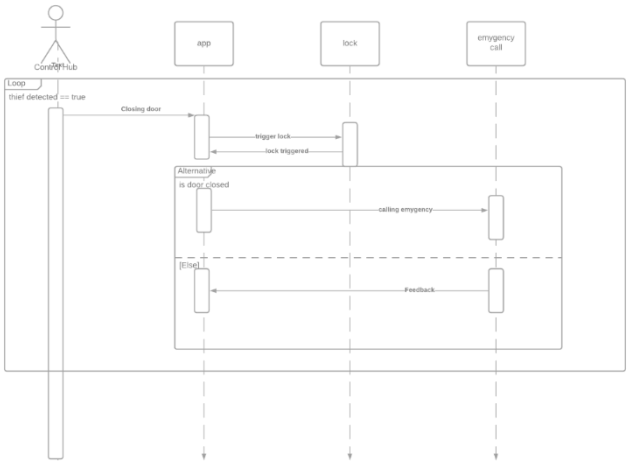

Figure 20: Sequence diagram for Smart Door

The development process of the use case diagrams and also the sequence diagrams in the preceding sections are done through many iterations and have undergo many different version updates until we obtained the finalised requirements for the system. The requirements are then later concretely modelled with fine interaction within the system. As shown in our development model before, inspections are carefully carried in each iteration.

E. System architecture

We are now at the next process stage and we are almost near to define the concrete system architecture. In this stage, we identify the required system components from each sequence diagram that we have did before so that we can focus on the general structure of the system architecture. As we go through all the sequence diagram that are shown previously, we can see that they have a same pattern. Using that discovery, we then extract the pattern and generalize the information in order to sketch roughly our nearly concrete system architecture. After that we later refine again what we have

sketched to produce our final concrete system architecture. The final diagram produced can be seen in figure 21.

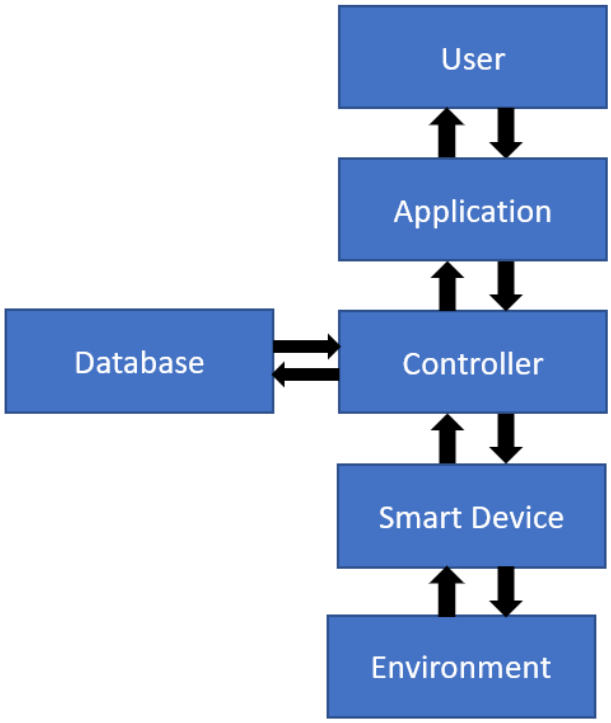

Figure 21 : System architecture

F. State machine

Next, we will dive into the implementation process. But before that, we need to plan precisely how to implement all of our findings before we produce the source codes. The planning is very crucial so that the so called "spaghetti code" can be avoided all together and allow us to make modifications on the source code during future version updates with minimal to no problems. Carefully planned implementation process also enables us to develop testing methods to test the implementation with ease and no complications.

We, first gather all the diagrams that we have created including system architecture and sequence diagrams. On this stage we focus on interaction between system entities or simply the behaviour. The product in this stage is state machine where it will be used for the further implementation of our system. The real time implementation will be explained in the next topic which is evaluation. The state machine for the main integrated scenario that we have built for control hub can be seen in figure 22.

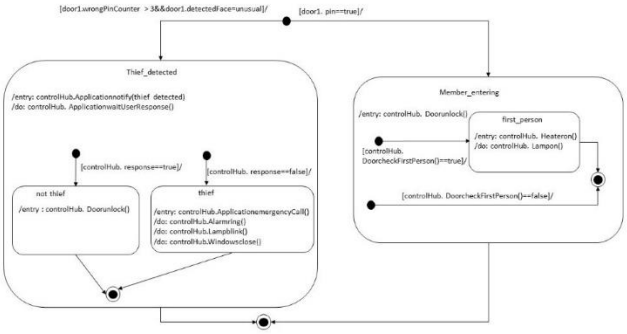

Figure 22 : State machine for control hub

Note that the state machine of other smart devices are located in appendix to assist the reader have more understanding.

IV. Evaluation

With already defined state machine that we have explain in preceding topic, we now can proceed to the implementation stage. In this topic we just focus on the implementation of the state machine made in this project's implementation, we use C++ language because it is an object-oriented programming language and the language that used by almost all IDE for microcontroller. The detail of the implementation is attached in appendix part. After that, again, we go through validation test and defect test to identify hidden faults and make sure that the code run accordingly. Now we go through the partly implementation that we put in main function for our main state machine which is belong to control hub:

```
int state;
cin >> state;  //input from smart door

switch (state)
{

case THIEF_DETECTED:
controller1.Applicationnotify(1);// 1 means thief detected
bool response =
Controller1.ApplicationwaitUserResponse();
if (response == true)
{
controller1.Doorunlock();
break;
}
else if (response == false)
{
controller1.ApplicationemergencyCall();
controller1.Alarmring();
controller1.Lampblink();
controller1.Windowclose();
break;
}

case MEMBER_ENTERING:
controller1.Doorunlock();
If (controller1.DoorcheckFirstPerson() == true)
{
controller1.Heateron();
controller1.Lampon();
}
break;

}
```

Now we are ready to bring everything that we have planned to the real world. This stage is known as simulation. For the purpose of this project, we have chosen Arduino as our simulation platform since the characteristic of the microcontroller fulfil almost all the requirement for each device. The connection for the simulation is shown in figure 23. The communication protocol between the application and control hub in this simulation is UART and the protocol between the control hub and smart devices is I2C.

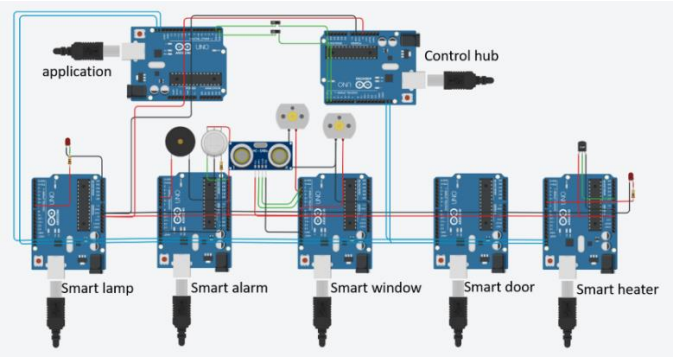

Figure 23

For the implementation in Arduino, just few alterations on the previous source code are made, because Arduino IDE also use C++.The implementation for each device is shown in appendix. The link of the simulation for each device is given below:

A. Smart Heater

https://www.tinkercad.com/things/lAD4ZoHsiDf-copy-of-smartheater/editel?sharecode=8vV4TOWRMg2UaGrVHB98qfqQal2hi4VgY8FHAXQHseU

B. Smart Lamp

https://www.tinkercad.com/things/jbvB8ZBRjG4-smartlampsimulation/editel?sharecode=SIB-ZBEgPfQ3aBtXoVPkvR-HUBKJs-RNI8HQN_RWD1Q

C. Smart Window

https://www.tinkercad.com/things/gkzD85yWREY-brilliant-jarv/editel?sharecode=OyS7AOpYOTNWKkxoVgYZ8xF2RgvWd3ADcVgCZ-ZUBYY

D. Smart Alarm

https://www.tinkercad.com/things/hoJD9PJmdnZ-magnificent-gaaris/editel?sharecode=UHSXnEBCIvDgAqeyeY0S0l9xBgdQhS-Jpz3NQEMTmQU

V. Summary and Outlook

In this paper, we presented what we have developed which is a smart home system with the main focus design is to tackle the problems regarding the security faults. We have explained in detail how we achieved the final product through multiple incremental steps of analysis and design during the last few months of official academic hours. We also included details, like our source code and many different diagrams to help the reader better understand what is being explained.

With the help of multiple UML diagrams, like class, use case and sequence diagrams, we managed to create the main components of the system. After developing the individual components of our smart home system, we generalized the pattern of interaction between the environment and the devices, and came up with the overall system architecture. Then, we focused on a specific scenario which is when a thief is entering the house. Finally, we developed the use case diagram for the given scenario to better help us implement the code to C++ and simulation using Arduino. The smart home system has accomplished by this paper only has constructed a framework, has realized partial function of a specific scenario of the Smart Home system.

Although we made a huge step in the right direction, we feel as though there are many more improvements that can be made in the future. Improvements that we wanted to make are further implementation of multiple special features for each device in our system. For example, our smart alarm will have the ability to set a morning alarm that goes off during the specified time. Another example would be our smart window will have solar panels so that the house will have the ability to switch to an eco-mode that uses less electricity and utilizes solar energy. Nevertheless, we are grateful to have been guided to have some experience in developing our own product for the future.

In the future, we hope that our product can be more refined and finally realized to the world and introduce the product to the main public. We also hope to be given more opportunities like this to have a better understanding on our study course.

VI. Appendix

A. State machine

For better understanding of the implementation for each device we first should look the state machine for each device. The state machine for each device is shown below.

1) Application

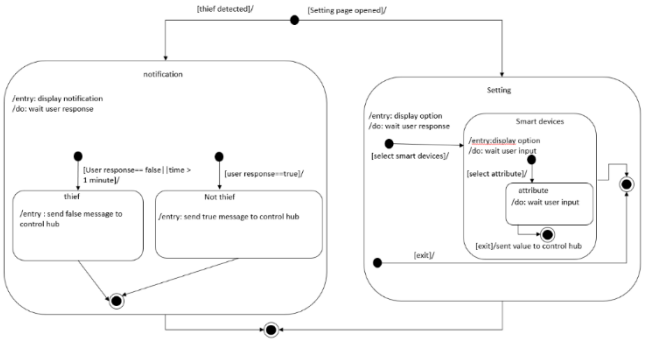

Figure 24: State machine for application

2) Smart heater

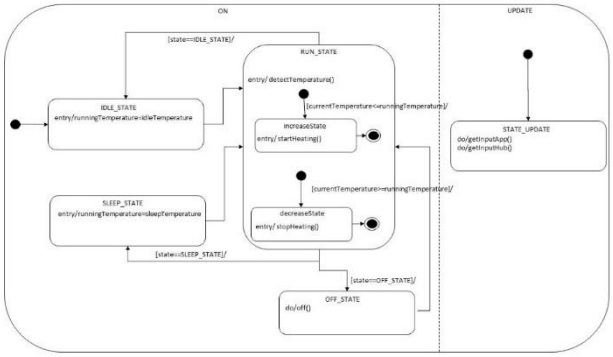

Figure 25 : State machine for Smart Heater

3) Smart window

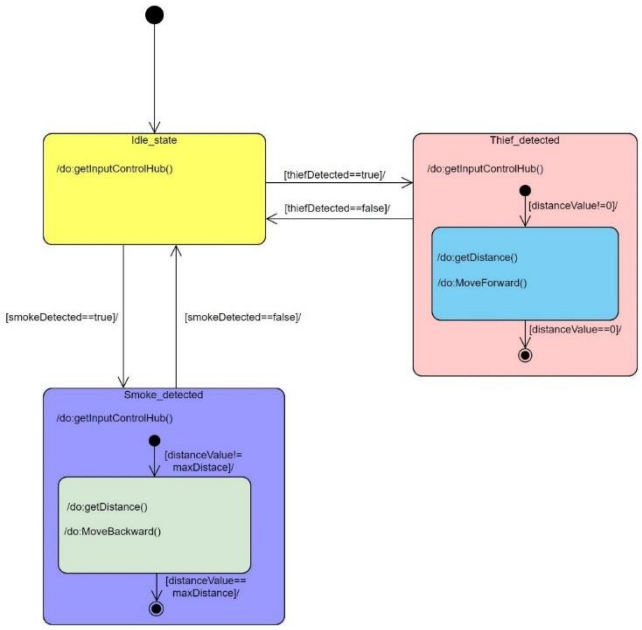

Figure 26 : State machine for Smart Window

4) Smart alarm

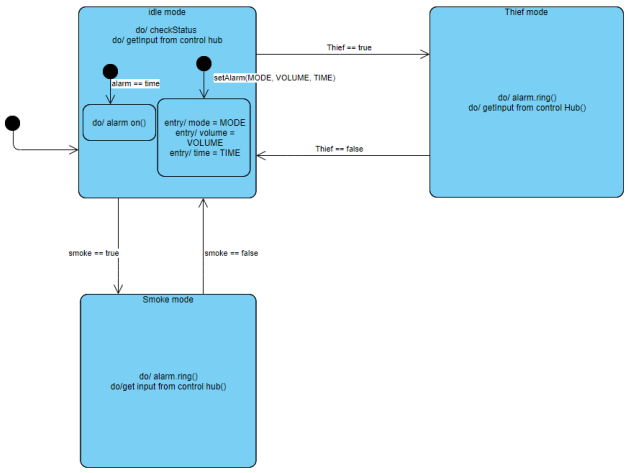

Figure 27: State machine for Smart Alarm

5) Smart Lamp

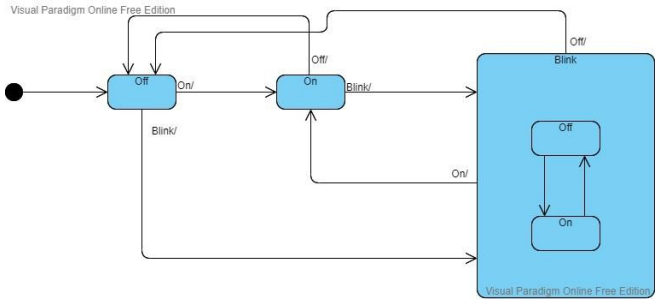

Figure 28: State machine for Smart Lamp

6) Smart Door

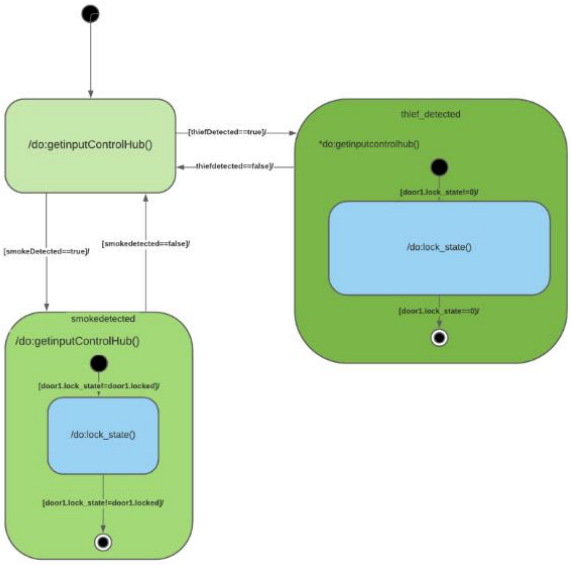

Figure 29 : State machine for smart door

Now we will go through the header files and source code for each devices

B. Source file and header

1) Control Hub

a) Header file

```
#ifndef CONTROL_HUB
#define CONTROL_HUB

class controlHub
{
public:

//to communicate with smart heater
void Heateron();

//to communicate with application
void Applicationnotify(int code);
bool ApplicationwaitUserResponse();
void ApplicationemergencyCall();

//to communicate with smart door
void Doorunlock();
bool DoorcheckFirstPerson();

//to communicate with smart Alarm
void Alarmring();

//to communicate with smart lamp
void Lampblink();
void Lampon();

//to communicate with smart window
void Windowclose();
void Windowopen();
void smokeDetected();
void thiefDetected();
};
#endif
```

b) Souce file

```
#include <iostream>
#include "controlHub.h"
using namespace std;

//to communicate with smart heater
void controlHub::Heateron()
{
   cout << "set heater to " << "on" << endl;   // send code to
heater to turn on the heater
}

//to communicate with application
void controlHub::Applicationnotify(int code)
{
   cout << "thief detected" ; //output to application
}
bool controlHub::ApplicationwaitUserResponse()
{
   int response;
   cin >> response;   //input from application(by user)
   return response;
}
void controlHub::ApplicationemergencyCall()
{
   cout << "emergency call"; // output tu application
}

//to communicate with smart door
void controlHub::Doorunlock()
{
   cout << "door lock"; //to lock door(output to door)
}
bool controlHub::DoorcheckFirstPerson()
{
   bool result;
   cin >> result; //input from door to check either the person
that come in is the first person
   return result;
}

//to communicate with smart Alarm
void controlHub::Alarmring()
{
   cout << "ring alarm";// to ring the alarm(output to smart
alarm)
}

//to communicate with smart lamp
void controlHub::Lampblink()
{
   cout << "blink lamp";// to blink the lamp(output to smart
lamp)
}
void controlHub::Lampon()
{
   cout << "lamp on";// to on the lamp(output to smart lamp)
}

//to communicate with smart window
void controlHub::Windowclose()
{
   cout << "window close";// to close the window(code :6)
}

//to communicate with smart window
void controlHub::Windowopen()
{
   cout << "window open";// to open the window(code :7)
}

void controlHub::thiefDetected()
{
   cout << "window thief Detected" << endl; // (code :4)
}

void controlHub::smokeDetected()
{
   cout << "window smoke Detected" << endl; // (code :5)
}
```

c) Source file

```
#include <iostream>
#include "controlHUb.h"

#define THIEF_DETECTED 0
#define MEMBER_ENTERING 1

using namespace std;
bool enterHouseScenario = true; // event 1

int main()
{
        controlHub controller1;
        while (enterHouseScenario)
        {

                int state;
                cin >> state;  //input from smart door

                switch (state)
                {
                case THIEF_DETECTED:

        controller1.Applicationnotify(1);// 1 means thief
detected
                        bool response =
controller1.ApplicationwaitUserResponse();
                        if (response == true)
                        {

        controller1.Doorunlock();
                                break;
                        }
                        else if (response == false)
                        {

        controller1.ApplicationemergencyCall();
                                controller1.Alarmring();

        controller1.Lampblink();

        controller1.Windowclose();
                                break;
```

```
                }
            case MEMBER_ENTERING:
                controller1.Doorunlock();
                if
(controller1.DoorcheckFirstPerson() == true)
                {
                    controller1.Heateron();
                    controller1.Lampon();
                }
                break;
            }
        }
}
```

2) Smart Heater

a) Header file

```
#include<iostream>
using namespace std;

#ifndef SMART_HEATER
#define SMART_HEATER

class controlHub {};
class smartHeater
{
public:
    smartHeater(controlHub* newController);
    int state = 0;
    double detectTemperature();

    void getInputApp();  //get input from application
    void getInputHub();  //get input from control hub

    void setIdleTemperature(double temp);
    void setSleepTemperature(double temp);

    double currentTemperature = 0;
    double runningTemperature = 0;      //temperature
need to be mantained
    double idleTemperature = 0;         //default
temperature set by owner
    double sleepTemperature = 0;        //temperature on
sleep mode is on
    const double ecoTemperature = 20.0;   // room
temperature. used to reduce energy conservation

    void on();
    void off();
    void startHeating();
    void stopHeating();

    bool heating =false;                // heater status.
either heating or cooling(stop heating)

    void sleepMode();
    void idleMode(void);
    void ecoMode();
private:

    controlHub* controller;
};
#endif
```

b) Souce file

```
#include<iostream>
using namespace std;
#include "smartHeater.h"

smartHeater::smartHeater(controlHub* newController)
{
    controller = newController;
}

double smartHeater::detectTemperature()
{
    cin >> currentTemperature;     // get input from
temperature sensor
    return currentTemperature;
}

void smartHeater::getInputApp()
{
    int Input;
    cin >> Input;    //input from application
    if (Input == 0)  //code-0 -> do nothing
    {
    }
    else if (Input == 1)  //code-1 -> to set idle
temperature
    {
        double temp;
        cout << "idle temperature value :";
        cin >> temp ;
        setIdleTemperature(temp);

    }
    else if (Input == 2) //code-2 -> to set sleep
temperature
    {
        double temp;
        cout << "sleep temperature value :";
        cin >> temp;
        setSleepTemperature(temp);
    }
    else if (Input == 3) //code-3 -> to change the state
to idle state
    {
        state = 0;//idle state
    }
    else if (Input == 4) //code-4 -> to change the state
to sleep state
    {
        state = 1;//sleep state
    }
}
void smartHeater::getInputHub()
{
    int Input;
    cin >> Input;    //input from control Hub
    if (Input == 0)  //code-0 -> do nothing
    {
    }
    else if (Input == 1)  //code-1 -> on the smartHeater
```

```
	{
		state = 0; //start the smart heater in idle
state
	}
	else if (Input == 2)  //code-1 -> on the smartHeater
	{
		state = 2; //turn off the heater
	}
}

void smartHeater::setIdleTemperature(double temp)
{
	idleTemperature = temp;
}
void smartHeater::setSleepTemperature(double temp)
{
	sleepTemperature = temp;
}

void smartHeater::startHeating()
{
	heating = true;		//turn on heater
}
void smartHeater::stopHeating()
{
	heating = false;		//turn off heater
}

void smartHeater::sleepMode()
{
	runningTemperature = sleepTemperature;
}
void smartHeater::idleMode(void)
{
	runningTemperature = idleTemperature;
}
void smartHeater::ecoMode()
{
	runningTemperature = ecoTemperature;
}

void smartHeater::on()
{
	state = 0; // start the smart heater in idle state
}
void smartHeater::off()
{
	runningTemperature = -100;
}
```

c) Source file

```
#include<iostream>
#include "smartHeater.h"
#include <pthread.h>

#define NUM_THREADS 2
#define IDLE_STATE 0
#define SLEEP_STATE 1
#define OFF_STATE 2

controlHub mainController;
smartHeater heater1(&mainController);
using namespace std;

void* ON(void* threadid)
{

	while (1)
	{
		switch (heater1.state)
		{
		case IDLE_STATE:
			heater1.idleMode();
			break;
		case SLEEP_STATE:
			heater1.sleepMode();
			break;
		case OFF_STATE:
			heater1.off();
			break;
		}

		heater1.detectTemperature();
		if (heater1.currentTemperature <=
heater1.runningTemperature)
		{
			heater1.startHeating();
		}
		if (heater1.currentTemperature >=
heater1.runningTemperature)
		{
			heater1.stopHeating();
		}
	}
	pthread_exit(NULL);
}

void* UPDATE(void* threadid)
{
	while (1)    //keep update
	{
		heater1.getInputApp();
		heater1.getInputHub();
	}
	pthread_exit(NULL);
}

int  main()
{
	pthread_t threads[NUM_THREADS];
	{
		pthread_create(&threads[0], NULL, ON,
(void*)0);
		pthread_create(&threads[1], NULL,
UPDATE, (void*)1);
	}
	pthread_exit(NULL);
}
```

3) Smart lamp

a) Header file

```
#include <iostream>
using namespace std;
```

```
#ifndef SMART_LAMP
#define SMART_LAMP

class controlHub{
};
class smartLamp
{
   public:

   smartLamp(controlHub* newController);
   void getInputControlHub();
   int state;
   void blink();
   void on();
   void off();

   private:

   controlHub* controller;
};

#endif
```

b) Souce file

```
#include <iostream>
#include "smartLamp.h"

using namespace std;

void smartLamp :: blink(){
   for(int x=0;x<=50;x++){
            on();
            off();
            }
}

void smartLamp :: on(){
   cout<<"lampOn";
}

void smartLamp :: off(){
            cout<<"lampOff";
}

void smartLamp::getInputControlHub()
{
   int input;
   cin >> input; //from control hub
   if (input == 0)
   {

   }

   else if (input == 1)
   {
      state = 0;
   }
   else if (input == 2)
   {
      state = 1;
   }
   else if (input == 3)
   {
      state = 2;
   }
}

smartLamp :: smartLamp(controlHub* newController){

   controller = newController;
}
```

c) Source file

```
#include <iostream>
#include "smartLamp.h"
using namespace std;

#define ON 0
#define OFF 1
#define BLINK 2

int main(){

            controlHub mainController;
            smartLamp lamp1(&mainController);

            switch(lamp1.state){
                        case ON:
                                    while(lamp1.state==ON){
                                    lamp1.getInputControlHub();
                                    lamp1.on();
                                    break;
                                    }

                        case OFF:
                                    while(lamp1.state==OFF){

            lamp1.getInputControlHub();
                                                lamp1.off();
                                                break;
                                    }

                        case BLINK:
                                    while(lamp1.state==BLINK){

            lamp1.getInputControlHub();
                                                lamp1.blink();
                                                break;
                                    }
            }

}
```

4) Smart window

a) Header file

```
#include <iostream>
using namespace std;

#ifndef SMART_WINDOW
#define SMART_WINDOW

class controlHub;
class smartWindow
{
```

```
public:

    void open();
    void close();
    void ecoMode();
    void getDistance();
    void moveFoward();
    void moveBackward();
    void getInputControlHub();
    double distanceValue =0;
    bool thiefDetected =0;
    bool smokeDetected =0;
    int state =0;
    int maxDistance = 500;
    smartWindow(controlHub* newController);

private:
    controlHub* controller;

};

#endif
```

b) Souce file

```
#include "smartWindow.h"
#include <iostream>
using namespace std;

void smartWindow::open() {
    getDistance();
    while (distanceValue!= maxDistance)
    {
        getDistance();
        moveBackward();
    }
}

smartWindow::smartWindow(controlHub* newController)
{

    controller = newController;
}

void smartWindow::close() {
    getDistance();
    while (distanceValue != 0)
    {
        getDistance();
        moveForward();
    }

}

void smartWindow::getDistance() {
    cin >> distanceValue;
}

void smartWindow::moveForward() {
    cout << "motor_move_foward" << std::endl;
}

void smartWindow::moveBackward() {
    cout << "motor_move_backward" << std::endl;
}

void smartWindow::getInputControlHub() {
    int input;
    cin >> input;
    if (input == 1)
    {
        thiefDetected = true;
    }
    if (input == 2)
    {
        smokeDetected = true;
    }
     if (input == 3)
    {
        state = 0; //idlestate
    }

    if (input == 4)
    {
        state = 1;//thiefDetected
    }

    else if (input == 5)
    {
        state = 2;//smokeDetected
    }
}
```

c) Source file

```
#include <iostream>
#include "smartWindow.h"
using namespace std;

#define idleState 0
#define thiefDetected 1
#define smokeDetected 2

int main()
{
        controlHub mainController;
        smartWindow window1(&mainController);

        switch (window1.state) {
        case idleState:
                while (window1.state == idleState) {
                        window1.getInputControlHub();

                }
                break;

        case thiefDetected:
                while (window1.state == thiefDetected) {
                        window1.getInputControlHub();

                        while (window1.distanceValue!=
0)
                        {
                                window1.close();
                        }
```

```
                }
                break;
        case smokeDetected:
                while (window1.state == smokeDetected)
{
                        window1.getInputControlHub();


        while(window1.distanceValue!=window1.maxDist
ance)
                        {
                                window1.open();
                        }

                }
                break;
        }
}
```

5) Smart Alarm

a) Header file

```
#include <iostream>
using namespace std;

#ifndef SMART_ALARM
#define SMART_ALARM

class controlHub;
class smartAlarm
{
public:

   bool thiefDetected;
   int MODE; //music
   int state;
   void ring();
   void on();
   double volume;
   double emergencyVolume=255;
   double alarmVolume;
   void getInputHub();
   void getInputApp();
   void setMorningAlarm(int MODE,double volume, double
time);

   smartAlarm(controlHub* newController);

private:

   controlHub* controller;
};

#endif
```

b) Souce file

```
#include "smartAlarm.h"
#include <iostream>

void ring()
{
   volume = emergencyVolume;
   on();
}

void on()
{
   cout << "ON"; //output to bell
}

void getInputHub()
{
   int input;
   cin >> input; //from control hub
   if (input == 0)
   {

   }
   int input;
   cin >> input;
   else if (input == 1)
   {
      ring();
   }
   int input;
   cin >> input;

   else if (input == 3)
   {
      theifDetected = true;
   }
   else if (input == 4)
   {
      state = 0;
   }
   else if (input == 5)
   {
      state = 1;
   }
   else if (input == 6)
   {
      state = 2;
   }
}

   void getInputApp()
   {
      int input;
      cin >> input; //from app
      if (input == 0)
      {

      }

      else if (input == 2)
      {
         setAlarm(int mode, double Volume, double time);
         cin >> mode;  //from app
         cin >> volume;  //from app
         cin >> time;  //from app
      }
   }

void setAlarm(int mode, double Volume, double time)
{
```

```
   MODE = mode;
   alarmVolume = Volume;
}

smartAlarm :: smartAlarm(controlHub* newController){

   controller = newController;
}
```

c) Source file

```
#include <iostream>
#include "smartWindow.h"
using namespace std;

#define idleMode 0
#define thiefMode 1
#define smokeMode 2

int main()
{
        controlHub mainController;
        smartAlarm alarm1(&mainController);

        switch (alarm1.state) {
        case idleMode:
                while (alarm1.state == idleMode) {
                        alarm1.getInputControlHub();
                        break;
                }

        case thiefMode:
                while (alarm1.state == thiefMode) {
                        alarm1.getInputControlHub();
                        alarm1.ring()

                        break;
                }
        case smokeMode:
                while (alarm1.state == smokeMode) {
                        alarm1.getInputControlHub();
                        alarm1.ring()

                        break;
                }
        }


}
```

6) Application

a) Header file

```
#include <iostream>
using namespace std;

#ifndef APPLICATION
#define APPLICATION

class controlHub;
class application
{
   public:

   bool userResponse;
   void notify(bool thiefDetected);
   void waitUserResponse(int time);
   void emergencyCall();

        //to communicate with smart heater
        void HeatersetIdleTemperature(double temp);
        void HeatersetSleepTemperature(double temp);
        void Heateron();
        void Heateroff();
        void HeatersleepMode();
        void HeateridleMode(void);
        void HeaterecoMode();
};

#endif
```

b) Souce file

```
#include "application.h"
#include <iostream>

void application :: notify(bool thiefDetected){

}
void application :: waitUserResponse(int time){

}
void application :: emergencyCall(){

}

void application::HeatersetIdleTemperature(double temp)
{
   cout << "set idle temperature to " << temp << endl; //
send code and value to heater to set idle temperature
}
void application::HeatersetSleepTemperature(double temp)
{
   cout << "set sleep temperature to " << temp << endl; //
send code and value to heater to set sleep temperature
}
void application::Heateron()
{
   cout << "set heater to " << "on" << endl;   // send code to
heater to turn on the heater
}
void application::Heateroff()
{
   cout << "set heater to " << "off" << endl; // send code to
heater to turn on the heater
}
void application::HeatersleepMode()
{
   cout << "heater mode chang to" << "sleep mode" <<
endl; // send code to heater to change state to sleep state
}
void application::HeateridleMode(void)
{
   cout << "heater mode chang to" << "idle mode" << endl;
// send code to heater to change state to idle state
}
void application::HeaterecoMode()
{
```

```
    cout << "heater mode chang to" << "eco mode" << endl;
// send code to heater to change state to eco state
}
```

c) Source file

```
#include <iostream>
#include "application.h"

#define CONTROL_HUB 0
#define SMART_HEATER 1

int state = 0;
int inputCode = 0;
int inputValue = 0;

using namespace std;

int main()
{
        cout << "code for devices" << endl << "control
Hub 0" << endl << "smart heater 1" << endl;
        cout << "communicate with :";
        cin >> state;

        cout << "code :";
        cin >> state;

        cout << "value :";
        cin >> state;

        switch (state)
        {
        case CONTROL_HUB:

                //give  output to controlhub
                //get input from controlhub
                break;
        case SMART_HEATER:
                //give  output to smart heater
                break;
        }

        state = 0;
}
```

7) Smart Door

a) Header file

```
#include <iostream>
using namespace std;

#ifndef SMART_DOOR
#define SMART_DOOR

class controlHub;
class smartdoor
{
public:

    void open();
    void close();
    void lock();
    void unlock();
    void getInputControlHub();
    double lock_state =0;
    bool thiefDetected =0;
    bool smokeDetected =0;
    int state =0;
    int locked = 500;

    smartdoor(controlHub* newController);

private:

    controlHub* controller;

};

#endif
```

b) Source file

```
#include <iostream>
#include "smartdoor.h"
using namespace std;

#define idleState 0
#define thiefDetected 1
#define smokeDetected 2

int main()
{
        controlHub app;
        smartdoor door1(&app);

        switch (door1.state) {
        case idleState:
                while (door1.state == idleState) {
                        door1.getInputControlHub();

                }
                break;

        case thiefDetected:
                while (door1.state == thiefDetected) {
                        door1.getInputControlHub();

                        while (door1.lock_state!= 0)
                        {
                                door1.close();
                        }

                }
                break;
        case smokeDetected:
                while (door1.state == smokeDetected) {
                        door1.getInputControlHub();


        while(door1.lock_state!=door1.locked)
                        {
                                door1.open();
                        }

                }
```

```
                    break;
            }
}
```

C. Implementation in specific microcontroller

The chosen microcontroller for simulation is Arduino.

1) Control Hub

```
#include <Wire.h>

#define MEMBER_ENTERING 1
#define THIEF_DETECTED 0
#define IDLE_STATE 2

#define APPLICATION_ADDRESS 3
#define SMART_HEATER_ADDRESS 4
#define SMART_DOOR_ADDRESS 5
#define SMART_ALARM_ADDRESS 3
#define SMART_LAMP_ADDRESS 1
#define SMART_WINDOW_ADDRESS 2

int output[2]; //{adress,code,value //i2c
double value;
int serialComunicationInput;
int serialComunicationOutput;
int state=2;//start with idle state

void wireCommunicationOuput();
int wireCommunicationInput();

class controlHub
{
public:
   //to communicate with smart heater
   void Heateron();
   //to communicate with application
   void Applicationnotify(int code);
   bool ApplicationwaitUserResponse();
   void ApplicationemergencyCall();
   //to communicate with smart door
   void Doorunlock();
   bool DoorcheckFirstPerson();
   bool DoorcheckThief();
   //to communicate with smart Alarm
   void Alarmring();
   //to communicate with smart lamp
   void Lampblink();
   void Lampon();
   //to communicate with smart window
   void Windowclose();
   void smokeDetected();
   void thiefDetected();
   void Windowopen();

};

//to communicate with smart heater
void controlHub::Heateron()
{
   output[0] = SMART_HEATER_ADDRESS; //heater
address
   output[1] = 6; //code to turn on the heater
   value = 0;//no value needed
   wireCommunicationOuput();
}

//to communicate with application
void controlHub::Applicationnotify(int code)
{
   if(code==1); // code to notify the user
  {
   Serial.println(" Thief detected ");  //print on application
using UART protocol
  }
}
bool controlHub::ApplicationwaitUserResponse()
{
   bool response = false;
   Serial.println("\n thief? (no-> 1, yes-> 0) : "); //print on
application using UART protocol
   while(!Serial.available()){}
   response = (bool)Serial.parseInt();   //input from
application(by user) using UART protocol
   delayMicroseconds(10);
   return response;
}
void controlHub::ApplicationemergencyCall()
{
   Serial.println("\n creating emergency call "); //print on
application using UART protocol
}

//to communicate with smart door
void controlHub::Doorunlock()
{
   output[0] = SMART_DOOR_ADDRESS;//door address
   output[1] = 0;//to lock door(output to door)
   value = 0;
   wireCommunicationOuput();
}
bool controlHub::DoorcheckFirstPerson()
{
   bool firstPerson = 0;
   output[0] = SMART_DOOR_ADDRESS;//door address
   output[1] = 0;//code to request input from door to check
either the person that come in is the first person
   value = 0;//value recieved

   wireCommunicationInput();
   if ((int)value)
   {
      return true;
   }
   else
   {
      return false;
   }
}
bool controlHub::DoorcheckThief()
{
   bool thief = 0;
   output[0] = SMART_DOOR_ADDRESS;//door address
   output[1] = 0;//code to request input from door to check
either there is thief or not
```

```
   value = 0;//value recieved

   wireCommunicationInput();

   if ((int)value!=1)
   {
      state = THIEF_DETECTED;
   }
   else
   {
      state = state;
   }
}

//to communicate with smart Alarm
void controlHub::Alarmring()
{
   output[0] = SMART_ALARM_ADDRESS;//alarm
address
   output[1] = 1;// to ring the alarm(output to smart alarm)
   value = 0;
   wireCommunicationOuput();
}

//to communicate with smart lamp
void controlHub::Lampblink()
{
   output[0] = SMART_LAMP_ADDRESS;//lamp address
   output[1] = 3;// to blink the lamp(output to smart lamp)
   value = 0;
   wireCommunicationOuput();
}
void controlHub::Lampon()
{
   output[0] = SMART_LAMP_ADDRESS;//lamp address
   output[1] = 1;// to on the lamp(output to smart lamp)
   value = 0;
   wireCommunicationOuput();
}

//to communicate with smart window
void controlHub::Windowclose()
{
   output[0] = SMART_WINDOW_ADDRESS;//windows
address
   output[1] = 6;// to close the lamp(output to smart
window)
   value = 0;
   wireCommunicationOuput();
}

void controlHub::Windowopen()
{
   output[0] = SMART_WINDOW_ADDRESS;//windows
address
   output[1] = 7;// to open the lamp(output to smart window)
   value = 0;
   wireCommunicationOuput();
}

void controlHub::thiefDetected()
{
  output[0] = SMART_WINDOW_ADDRESS;//windows
address
   output[1] = 4;// thief detected(output to smart window)
   value = 0;
   wireCommunicationOuput();
}

void controlHub::smokeDetected()
{
  output[0] = SMART_WINDOW_ADDRESS;//windows
address
   output[1] = 5;// smoke detcted (output to smart window)
   value = 0;
   wireCommunicationOuput();
}



void wireCommunicationOuput()
{
  int address=output[0];
  Wire.beginTransmission(address);
  Wire.write(output[1]);
  Wire.write((int)value);
  Wire.endTransmission();
}
int wireCommunicationInput()
{
  Wire.beginTransmission(output[0]);
  Wire.write(output[1]);
  Wire.endTransmission();
  delay(10);
  Wire.requestFrom(output[0],1);
  if(Wire.available())
  {
   value=Wire.parseInt();
  }
}

void setup()
{
   Wire.begin();
   Serial.begin(9600);
}

void loop()
{

   controlHub controller1;
   controller1.DoorcheckThief();
   bool response=false;

   switch (state)
   {
   case THIEF_DETECTED:
      controller1.Applicationnotify(1);// 1 means thief
detected
      response= controller1.ApplicationwaitUserResponse();
      if (response == true)
      {
         controller1.Doorunlock();
         state=MEMBER_ENTERING;
         break;
```

```
        }
        else if (response == false)
        {
            controller1.ApplicationemergencyCall();
            controller1.Alarmring();
            controller1.Lampblink();
            controller1.Windowclose();
            break;
        }
        break;

    case MEMBER_ENTERING:
        controller1.Doorunlock();
        if (controller1.DoorcheckFirstPerson() == true)
        {
            controller1.Heateron();
            controller1.Lampon();
        }
        break;
   case IDLE_STATE:
      //controller1.Heateroff();
      //controller1.Heateron();
        break;
    }
}
```

2) Smart Heater

```
#include <Wire.h> //communication using i2c protocol

#define IDLE_STATE 0
#define SLEEP_STATE 1
#define OFF_STATE 2

#define temperatureSensorPin A0
#define heaterPin 6

short int inputCode;  //where the code should be store
int inputValue;  //store any value from application and
control hub

class controlHub {};
class smartHeater
{
public:
    smartHeater(controlHub* newController);
    int state = 2;
    double detectTemperature();

    void getInputApp();  //get input from application
    void getInputHub();  //get input from control hub

    void setIdleTemperature(double temp);
    void setSleepTemperature(double temp);

    double currentTemperature = 0;
    double runningTemperature = 0 ;   //temperature need to
be mantained
    double idleTemperature = 0;         //default temperature
set by owner
    double sleepTemperature = 0;        //temperature on sleep
mode is on
    const double ecoTemperature = 20.0;   // room
temperature. used to reduce energy conservation

    void on();
    void off();
    void startHeating();
    void stopHeating();

    bool heating = false;                 // heater status. either
heating or cooling(stop heating)

    void sleepMode();
    void idleMode(void);
    void ecoMode();
private:

    controlHub* controller;
};

smartHeater::smartHeater(controlHub* newController)
{
    controller = newController;
}
double smartHeater::detectTemperature()
{
    int sensorValue = digitalRead(temperatureSensorPin);
// get input from temperature sensor
    //translating the input to temperature value
      sensorValue = analogRead(A0);    //read the analog
sensor and store it
  currentTemperature = (double)sensorValue / 1024;
//find percentage of input reading
  currentTemperature = currentTemperature * 5;
//multiply by 5V to get voltage
  currentTemperature = currentTemperature - 0.5;
//Subtract the offset
  currentTemperature = currentTemperature * 100;
//Convert to degrees
    return currentTemperature;
}

void smartHeater::setIdleTemperature(double temp)
{
    idleTemperature = temp;
}
void smartHeater::setSleepTemperature(double temp)
{
    sleepTemperature = temp;
}
void smartHeater::startHeating()
{
    digitalWrite(heaterPin, HIGH);  //turn on heater
}
void smartHeater::stopHeating()
{
    digitalWrite(heaterPin, LOW);             //turn off heater
}
void smartHeater::sleepMode()
{
    runningTemperature = sleepTemperature;
}
void smartHeater::idleMode(void)
{
    runningTemperature = idleTemperature;
```

```
}
void smartHeater::ecoMode()
{
   runningTemperature = ecoTemperature;
}
void smartHeater::on()
{
   state = 0; // start the smart heater in idle state
}
void smartHeater::off()
{
   runningTemperature = -100;
}

controlHub mainController;
smartHeater heater1(&mainController);

void setup()
{
   Wire.begin(4); //heater addess is 4
   Wire.onRequest(output);
   Wire.onReceive(input);
   Serial.begin(9600);
   pinMode(heaterPin, OUTPUT);
   pinMode(temperatureSensorPin, INPUT);
}

//void* UPDATE(void* threadid)  replaced as interupt
void input(int howMany)
{
   Serial.print("\nreceive input : ");
   if (Wire.available())
   {
      inputCode = Wire.read();
   }
   while (Wire.available())
   {
      inputValue = Wire.read();
   }

   //void smartHeater::getInputApp()
   if (inputCode == 0)  //code-0 -> do nothing
   {
   }
   else if (inputCode == 1)  //code-1 -> to set idle
temperature
   {
      heater1.setIdleTemperature(inputValue);
   }
   else if (inputCode == 2) //code-2 -> to set sleep
temperature
   {
      heater1.setSleepTemperature(inputValue);
   }
   else if (inputCode == 3) //code-3 -> to change the state to
idle state
   {
      heater1.state = 0;//idle state
   }
   else if (inputCode == 4) //code-4 -> to change the state to
sleep state
   {
      heater1.state = 1;//sleep state
   }

   //void smartHeater::getInputHub()
   if (inputCode == 5)  //code-0 -> do nothing
   {
   }
   else if (inputCode == 6)  //code-1 -> on the smartHeater
   {
      heater1.state = 0; //start the smart heater in idle state
   }
   else if (inputCode == 7)  //code-1 -> on the smartHeater
   {
      heater1.state = 2; //turn off the heater
   }
}
void output()
{

}

//void* ON(void* threadid) as viod loop()
void loop()
{
   while (1)
   {
      switch (heater1.state)
      {
      case IDLE_STATE:
        heater1.idleMode();
         break;
      case SLEEP_STATE:
         heater1.sleepMode();
         break;
      case OFF_STATE:
         heater1.off();
         break;
      }

      heater1.detectTemperature();
      if (heater1.currentTemperature <=
heater1.runningTemperature)
      {
         heater1.startHeating();
      }
      if (heater1.currentTemperature >=
heater1.runningTemperature)
      {
         heater1.stopHeating();
      }
   }
}
```

3) Smart alarm

```
#include <Wire.h> //communication using i2c

#define idleMode 0
#define thiefMode 1
#define smokeMode 2
#define buzzerPIN 5
#define smokeSensorPIN A5

class controlHub {};
```

```
class smartAlarm
{
public:

   bool thiefDetected = 0;
   int MODE = 0; //music
   int state = 0;
   void ring();
   void on();

   double volume = 0;
   double emergencyVolume=255;
   double alarmVolume = 0;

   void setAlarm(int MODE, double volume, double time);

   smartAlarm(controlHub* newController);

private:

   controlHub* controller;
};

controlHub mainController;
smartAlarm alarm1(&mainController);

void smartAlarm :: ring()
{
   volume = emergencyVolume;
   on();
}

void smartAlarm :: on()
{
   pinMode(buzzerPIN, HIGH); //output to bell
}

void smartAlarm :: setAlarm(int mode, double Volume,
double times)
{
   MODE = mode;
   alarmVolume = Volume;
}

smartAlarm :: smartAlarm(controlHub* newController){

   controller = newController;
}

int inputCode;
int inputValue;

void input (int A)
{
  if (Wire.available())
  {
    inputCode = Wire.read();
  }
  if(Wire.available())
  {
    inputValue = Wire.read();
  }
  if (inputCode == 0)
    {

    }

    else if (inputCode == 1)
    {
       alarm1.ring();
    }

    else if (inputCode == 3)
    {
       alarm1.thiefDetected = true;
    }
    else if (inputCode == 4)
    {
       alarm1.state = 0;
    }
    else if (inputCode == 5)
    {
       alarm1.state = 1;
    }
    else if (inputCode == 6)
    {
       alarm1.state = 2;
    }
}

void setup()
{
  Wire.begin(3); //alarm address is 3
  Wire.onReceive(input);
  pinMode(buzzerPIN, OUTPUT);
  pinMode(smokeSensorPIN, INPUT);

}

void loop()
{
  switch (alarm1.state)
  {
  case idleMode:
    while (alarm1.state == idleMode)
    {

      break;
    }

  case thiefMode:
    while (alarm1.state == thiefMode)
    {

      alarm1.ring();

      break;
    }
  case smokeMode:
    while (alarm1.state == smokeMode)
    {

      alarm1.ring();
```

```
    break;
   }
  }

}
```

4) Smart window

```
#include <Wire.h>
#define idleState 0
#define thiefDetect 1
#define smokeDetect 2

#define motorForwardPin 3
#define motorBackwardPin 8
#define trigPin 4
#define echoPin 2

class controlHub{};
class smartWindow
{
public:

   void open();
   void close();
   void ecoMode();
   void getDistance();
   void moveForward();
   void moveBackward();
   double distanceValue =0;
   bool thiefDetected =0;
   bool smokeDetected =0;
   int state =0;
   int maxDistance = 500;

   smartWindow(controlHub* newController);

private:

   controlHub* controller;
};

controlHub mainController;
smartWindow window1(&mainController);

void smartWindow::open() {
   getDistance();
   while (distanceValue!= maxDistance)
   {
     getDistance();
     moveBackward();
   }
}

smartWindow::smartWindow(controlHub* newController)
{
   controller = newController;
}

void smartWindow::close() {
   getDistance();
   while (distanceValue != 0)
   {
     getDistance();
     moveForward();
   }
}

void smartWindow::getDistance() {
   double duration;

   digitalWrite(trigPin, LOW);
   delayMicroseconds(2);
   digitalWrite(trigPin, HIGH);
   delayMicroseconds(10);
   digitalWrite(trigPin,LOW);
   duration = pulseIn(echoPin,HIGH);
   distanceValue =duration * 0.034/2;
}

void smartWindow::moveForward() {
   digitalWrite(motorForwardPin, HIGH);
   delay(3000); //3second delay
   digitalWrite(motorForwardPin, LOW);

}

void smartWindow::moveBackward() {

   digitalWrite(motorBackwardPin, HIGH);
   delay(3000); //3second delay
   digitalWrite(motorBackwardPin, LOW);
}

void InputHub(int a) {
   int input;
   if(Wire.available())
   {
    input=Wire.read();
   }

   if (input == 1)
   {
      window1.thiefDetected = true;
   }
   if (input == 2)
   {
      window1.smokeDetected = true;
   }
    if (input == 3)
   {

      window1.state = 0; //idlestate
   }

   if (input == 4)
   {

      window1.state = 1;//thiefDetected
   }

   else if (input == 5)
   {
      window1.state = 2;//smokeDetected
   }
```

```
}

void setup() {
  Wire.begin(2);
  Wire.onReceive(InputHub);
  pinMode(motorForwardPin, OUTPUT);
  pinMode(motorBackwardPin, OUTPUT);
  pinMode(trigPin,OUTPUT);
  pinMode(echoPin,INPUT);
}

void loop() {
    switch (window1.state) {
  case idleState:
    while (window1.state == idleState) {

    break;
    }

  case thiefDetect:
    while (window1.state == window1.thiefDetected) {

      while (window1.distanceValue!= 0)
      {
        window1.getDistance();
        window1.moveForward();
      }
      break;
    }
  case smokeDetect:
    while (window1.state == window1.smokeDetected) {

      while(window1.distanceValue!=window1.maxDistance)
      {
        window1.getDistance();
        window1.moveBackward();
      }
      break;
    }
  }

}
```

5) Smart lamp

```
#include <Wire.h>
using namespace std;
#define ON 0
#define OFF 1
#define BLINK 2
#define LED 7

class controlHub{

};
class smartLamp
{
    public:

    smartLamp(controlHub* newController);

    int state=1;
    void blink();
    void on();
    void off();

    private:

    controlHub* controller;
};

 controlHub mainController;
  smartLamp lamp1(&mainController);

void smartLamp :: blink(){
    double starttime=millis();
    while((millis()-starttime)<=50000){
      on();
      delay(1000);
      off();
      delay(1000);
    }
}

void smartLamp :: on(){
    digitalWrite(LED,HIGH);
}

void smartLamp :: off(){
  digitalWrite(LED,LOW);
}

void InputHub(int a)
{
    int input;
    if(Wire.available())
    {
      input=Wire.read();
    }
  int buff=Wire.read();

    if (input == 0)
    {

    }

    else if (input == 1)
    {
       lamp1.state = 0;
    }
    else if (input == 2)
    {
       lamp1.state = 1;
    }
    else if (input == 3)
    {
       lamp1.state = 2;
    }
}

smartLamp :: smartLamp(controlHub* newController){

    controller = newController;
}

void setup() {
```

```
Wire.begin(1);
Wire.onReceive(InputHub);
pinMode(LED,OUTPUT);
  Serial.begin(9600);

}

void loop() {

  switch (lamp1.state){

    case ON:
    lamp1.on();
    break;

    case OFF:
    lamp1.off();
    break;

    case BLINK:
    lamp1.blink();
    break;
  }
}
```

6) Application

```
#include <Wire.h>

//interaction state
#define CONTROL_HUB 0
#define SMART_HEATER 1

#define SMART_HEATER_ADDRESS 4
#define SMART_DOOR_ADDRESS 5
#define SMART_ALARM_ADDRESS 6
#define SMART_LAMP_ADDRESS 7
#define SMART_WINDOW_ADDRESS 8

int inputDevice = 0;
int inputAction = 0;
int output[2];
int value;
bool response;

void controlHubAction(); //list of control hub action(list of
function)
void smartHeaterAction();//list of smart heater action(list of
function)

class application
{
public:

    bool userResponse;
    void notify(bool thiefDetected);
    bool waitUserResponse();
    void emergencyCall();

    //to communicate with smart heater
    void HeatersetIdleTemperature();
    void HeatersetSleepTemperature();
    void Heateron();
    void Heateroff();
    void HeatersleepMode();
    void HeateridleMode();
    void HeaterecoMode();
};


void application::notify(bool thiefDetected)
{
    if (thiefDetected)
    {
        Serial.println("thief detected\n");
    }
}
bool application::waitUserResponse()
{
    Serial.println("is this your member?\n");
    while(!Serial.available()){}
    response = '0'-Serial.read();

    return response;
}
void application::emergencyCall()
{
    Serial.println("creating emergency call\n");
}

//i2c communication protocol

void wireCommunicationOuput()
{
    int address = output[0];
    Wire.beginTransmission(address);
    Wire.write(output[1]);
    Wire.write(value);
    Wire.endTransmission();
}
int wireCommunicationInput()
{
    Wire.beginTransmission(output[0]);
    Wire.write(output[1]);
    Wire.endTransmission();
    delay(10);
    Wire.requestFrom(output[0], 1);
    while(!Serial.available() ){
  }
    value = Serial.parseInt();
}

void application::HeatersetIdleTemperature()
{
    output[0] = SMART_HEATER_ADDRESS;
    output[1] = 1;//code to set idle temperature in heater
    Serial.println("Value for idle temperature :");
    while(!Serial.available() ){
  }
    value = Serial.parseInt();
    wireCommunicationOuput();

}
void application::HeatersetSleepTemperature()
{
```

```
    output[0] = SMART_HEATER_ADDRESS;
    output[1] = 2;//code to set sleep temperature in heater
    Serial.println("Value for sleep temperature :");

    while(!Serial.available() ){
   }
    value = Serial.parseInt();
    wireCommunicationOuput();
}
void application::Heateron()
{
    output[0] = SMART_HEATER_ADDRESS;
    output[1] = 3;//code to on heater (start with idle mode)
    value = 0;//no value needed
    wireCommunicationOuput();
}
void application::Heateroff()
{
    output[0] = SMART_HEATER_ADDRESS;
    output[1] = 7;//code to off heater (stop heating)
    value = 0;//no value needed
    wireCommunicationOuput();
}
void application::HeatersleepMode()
{
    output[0] = SMART_HEATER_ADDRESS;
    output[1] = 4;//code to change heater state to sleep mode
    value = 0;//no value needed
    wireCommunicationOuput();
}
void application::HeateridleMode(void)
{
    output[0] = SMART_HEATER_ADDRESS;
    output[1] = 3;//code to change heater state to idle mode
    value = 0;//no value needed
    wireCommunicationOuput();
}
void application::HeaterecoMode()
{
    output[0] = SMART_HEATER_ADDRESS;
    output[1] = 8;//code to change heater state to eco mode
    value = 0;//no value needed
    wireCommunicationOuput();
}


application app1;

void controlHubAction()
{
    //what to do to control hub
}
void smartHeaterAction()
{
    //what to do to smart heater
    Serial.println("code action :");
    Serial.println("set idle temperature 1");
    Serial.println("set sleep temperature 2");
    Serial.println("on 3");
    Serial.println("off 4");
    Serial.println("sleep mode 5");
    Serial.println("idle mode 6");
    Serial.println("eco mode 7");

    Serial.println("");
    Serial.print("action :\n");
    while(!Serial.available() ){
  }
    inputAction = Serial.parseInt();

    switch (inputAction)
    {
    case 0:
      break;
    case 1:
      app1.HeatersetIdleTemperature();
      break;
    case 2:
      app1.HeatersetSleepTemperature();
      break;
    case 3:
      app1.Heateron();
      break;
    case 4:
      app1.Heateroff();
      break;
    case 5:
      app1.HeatersleepMode();
      break;
    case 6:
      app1.HeateridleMode();
      break;
    case 7:
      app1.HeaterecoMode();
      break;
    }
}
void setup()
{
    Wire.begin();
    Serial.begin(9600);
}
void loop()
{

    delayMicroseconds(100);
    Serial.println("");
    Serial.println("code for devices :");
    Serial.println("");
    Serial.println("control hub 0");
    Serial.println("smart heater 1");
    Serial.println("");
    Serial.print("communicate with :");
    while(!Serial.available() ){
  }
    inputDevice = Serial.parseInt();

    switch (inputDevice)
    {
    case CONTROL_HUB:
      controlHubAction();
      break;
    case SMART_HEATER:
      smartHeaterAction();
```

```
        break;
    }

}
```

D. Github overview

1) Lines of code

TABLE I. LINES OF CODE

Lines of code	2027

Each tasks was equitably divided to every members of the group who work actively to program and execute each codes. Theses codes can be found on the link below.

https://github.com/TuringJuniors/Fortran

2) Number of submits per person

TABLE II. NUMBER OF SUBMITS PER PERSON

Member	Number of Submits
Sheikh Muhammad	355
Zafirul Izzat	121
Hadi Imran	97
Ammar Haziq	157
Evrard	66

3) Structure (folder hierarchy)

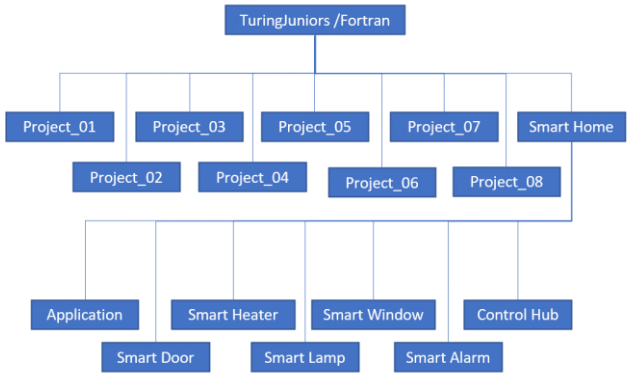

Figure 30 : folder hierarchy

E. Contribution

TABLE III. CONTRIBUTION IN PERCENTAGE AND WORKING HOUR

Member	Contribution in Percentage (%)	Working hours (hours)
Sheikh Muhammad	39	30
Zafirul Izzat	16	30
Hadi Imran	14	30
Ammar Haziq	20	30
Evrard	11	30

VII. AFFIDAVIT

We (Sheikh Muhammad, Zafirul Izzat, Ammar Haziq, Hadi Imran, Evrard) herewith declare that we have composed the present paper and work ourselves and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form has not been submitted to any examination body and has not been published. This paper was not yet, even in part, used in another examination or as a course performance.

ACKNOWLEDGMENT

We are heartily and most grateful to our beloved professor, Prof. Stefan Henkler, whose inspiration, encouragement, guidance, and support in the beginning towards the final level enabled us to develop awareness and also open the eyes of how important security in every smart home. We would also like to express our respects and regards and benefits to any or all of individuals who supported us whatsoever throughout the completion of the work. For additional information, we would like to express that in this paper, every person in our group is responsible for each smart device, starting from developing the class diagram until writing the documentation and not to forget the implementation code. Every creative decision was discussed and agreed with all team members during our team meeting. We tried our best to make everyone feel involve in the project. For more information, please refer table below:

TABLE I. TASK

Member	Contribution		
	Topics	***Subtopics***	***Developed devices***
Sheikh Muhammad	Approach		Control Hub
	Concept part	prototype	Application
			Smart Heater
Zafirul Izzat	Abstract		Smart Windows
	Acknowledgment		
	Concept part	Class diagram	
Hadi Imran	Motivation		Smart Lamp
	Concept part	Use case diagram	
Ammar Haziq	Summary		Smart Alarm
	Concept part	Sequence diagram	
Evrard	Appendix		Smart Door
	Evaluation		

REFERENCES

[1] Stefan Henkler, “µC_04_StandardEng” , pp. 7–29, October 2020. *(unpublished)*

[2] Miller, “M. My smart home for seniors”, June 2017.