



Design of Embedded Systems

ESSTA, Energy Saving Smart-home distributed
Temperature control Application

Falzone Giovanni

jointly M.Sc Embedded Computing Systems

Sant'Anna School of Advanced Studies

University of Pisa

June 9, 2019

Contents

1	Introduction	2
1.1	Central Unit	2
1.1.1	Graphical user interface	2
1.1.2	Communication	3
1.2	Room Module	3
1.2.1	Energy Saving mode	3
1.2.2	Valve control	3
1.2.3	Communication	4
1.2.4	Errors	4
2	SySML Functional model	5
2.1	Central Unit	6
2.2	Room module	7
3	Implementation, Central Unit module	10
3.1	Wiring	11
3.2	Graphic Interface	11
4	Implementation, Room module	13
4.1	Components description	13
4.1.1	ServoMotor	13
4.1.2	Valve position switches	14
4.1.3	Temperature-Humidity Sensor	14
4.2	Wiring	15
5	JSON messages implementation	16
5.1	Request Message	16
5.2	Status Message	16
6	CUnit tests and simulation of the system	17
6.1	Variables tests	17
6.2	JSON Compliance tests	17
6.3	Network noise tests	18
7	Future Work	19

1 Introduction

The purpose of this project is to realize a smart-home application to control the heating system of a building based on the temperature of each room, in order to minimize the consumption of the building each room apply an energy saving function reducing the desired temperature when it is not needed. The system is composed by two differ modules

- central unit module
- room module

1.1 Central Unit

The *Central Unit* has the role of coordinator that retrieve the status of each room and compute the average values for the building.

1.1.1 Graphical user interface

Using a graphical User Interface the module represents the average values of the building and the values for each room, the graphical User Interface is composed by:

- *Main page* to represent the overview of the building
- *Room page* to represent the status of each room
- *Setting page* to set the desired temperature

In the *Settings page* the module shall allow the user to set the desired temperature shared with the rooms. The minimum and maximum allowed temperatures are 15 C° and 30 C°.

When the *Main page* is selected the module shall represent the average values among all the rooms for *Temperature*, *Humidity* and *Usage*. When the *Main page* is selected the module shall represent the *Energy Saving* if at least one room is set to *Energy Saving mode*. When the *Main page* is selected the module shall represent the *Warning* if at least one room is set to *crashed*.

The graphical user interface shall represent the following information as reported in the table 1.

Information	Format
Temperature	C°
Humidity	%
Usage	%
Energy Saving	boolean
Warning	boolean

Table 1: Display Information

1.1.2 Communication

Whenever a *Room Request message* is sent and the *Room Status message* is not received within 20s the module shall mark the room as **crashed**. The module shall send the *Room Request message* for each room at least every 30s.

1.2 Room Module

The purpose of this module is to control the temperature of the room acting on a valve in order to reach and maintain the *GoalTemperature*.

1.2.1 Energy Saving mode

In order to minimize the consumption the module keep tracks of the presence of motion inside the room using a motion sensor.

If a motion is detected in the last 30s, the module shall set the *GoalTemperature* to the one set by the user, otherwise the module shall set the *GoalTemperature* to:

$$GoalTemperature = DesiredTemperature - EnergySavingTemperatureOffset \quad (1)$$

Whenever the module is in *Energy Saving mode* it shall notify through the *Interface*.

1.2.2 Valve control

In order to control the heating of the room the valve is moved to different positions based on the temperature error (*ActualTemperature* - *GoalTemperature*) as in the following table, whenever one of these rules is valid the module shall move the valve in the correspondent position described in the third column of the table 2 as percentage of maximum flow.

rule	valve position	Flow in %
<i>error < -HIGH</i>	OPEN_POSITION	100
<i>error ∈ [-HIGH, -APPROCHING]</i>	HIGH_POSITION	75
<i>error ∈ [-APPROCHING, +APPROCHING]</i>	MIDDLE_POSITION	50
<i>error ∈ (APPROCHING, HIGH]</i>	LOW_POSITION	25
<i>error > HIGH</i>	CLOSED_POSITION	0

Table 2

Whenever the valve is in *OPEN_POSITION* or *CLOSED_POSITION* the module shall check the position and set **Valve Error** if it is not valid.

In the following table3 are reported the temperature thresholds:

HIGH	2 C°
APPROCHING	1 C°

Table 3

1.2.3 Communication

Whenever a *Room Request message* is not received within 60s the module shall send the *Room Status Message* and set the **Communication error**.

1.2.4 Errors

Whenever one of the errors are set, the module shall notify it through the *Interface*. In the following table 4 are reported the possible errors.

Valve error
Communication error
Sensor error

Table 4

2 SySML Functional model

In the picture 1 is reported the functional Block Definition Diagram that describes the composition of the system, composed by one Central Unit and up to eight Rooms, the two modules are connected via two FlowPort as shown in 2. The Central Unit send a *RoomRequest* message composed as follows:

parameter	type	[Min,Max]
Id	Natural	[1,8]
DesiredTemperature	Float	[15.00, 30.00]

Table 5: Room Request variables

The Room module send a *RoomStatus* message composed as follow:

parameter	type	[Min,Max]
Id	Integer	[1,8]
Eco	Boolean	[0, 1]
Temperature	Float	[15.00, 30.00]
Humidity	Float	[0.00, 100.00]
Valve	Integer	[0, 100]

Table 6: Room Status variables

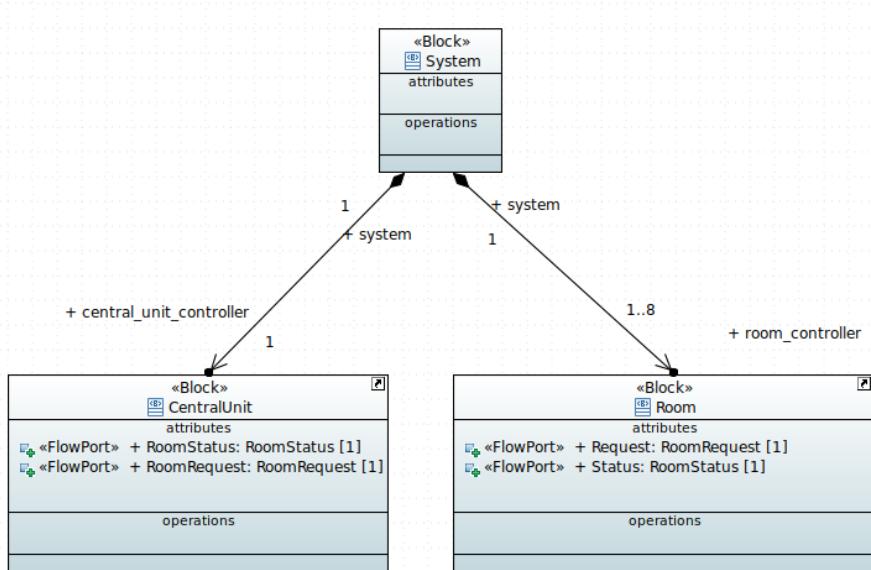


Figure 1: System Components

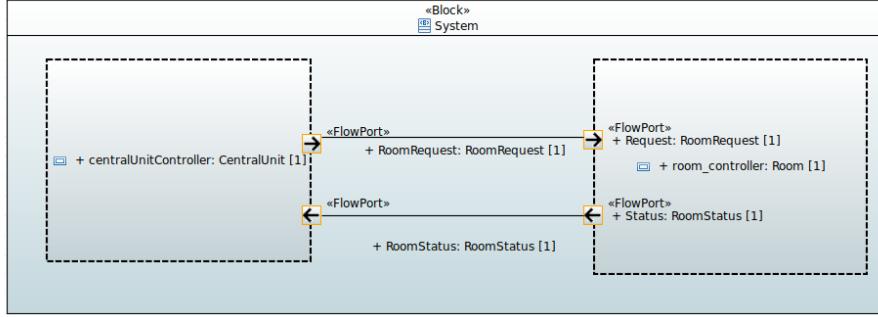


Figure 2: System Internals

2.1 Central Unit

The *Central Unit* is composed by two modules, the *RoomsManager* and the *UserInterfaceManager*. The *RoomsManager* implements the functionalities related to the status of each room. The *UserInterfaceManager* that implements the functionalities related to represent the status of the system. The two components exchange data as shown in 4.

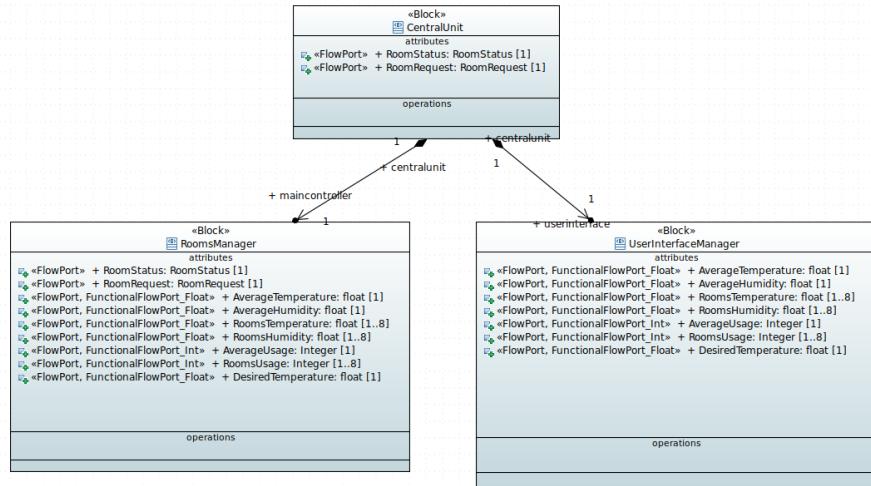


Figure 3: Central Unit components

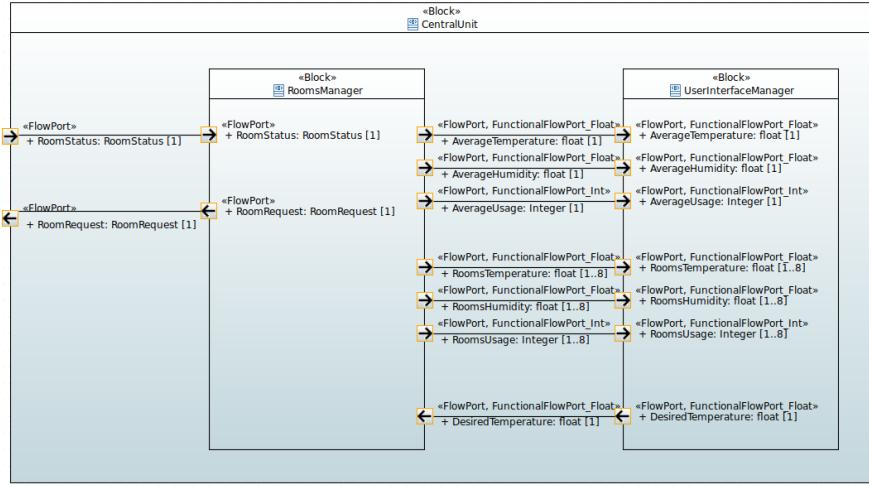


Figure 4: Central Unit internals

2.2 Room module

The main component of this module is the *MainController* composed by different functions as shown in 6.

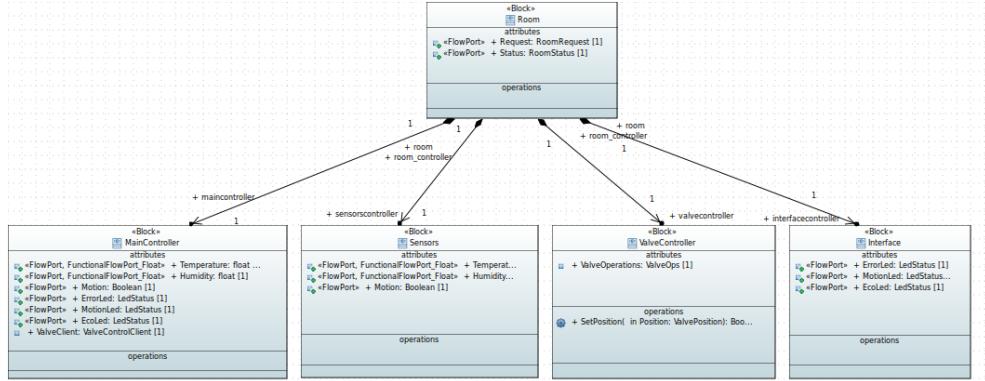


Figure 5: Room Components

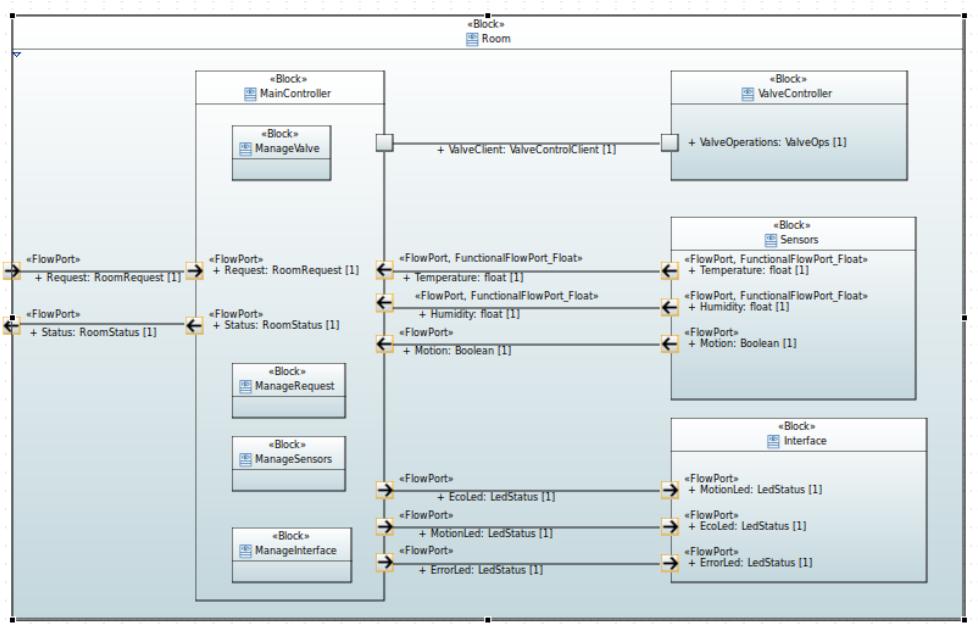


Figure 6: Room Internals

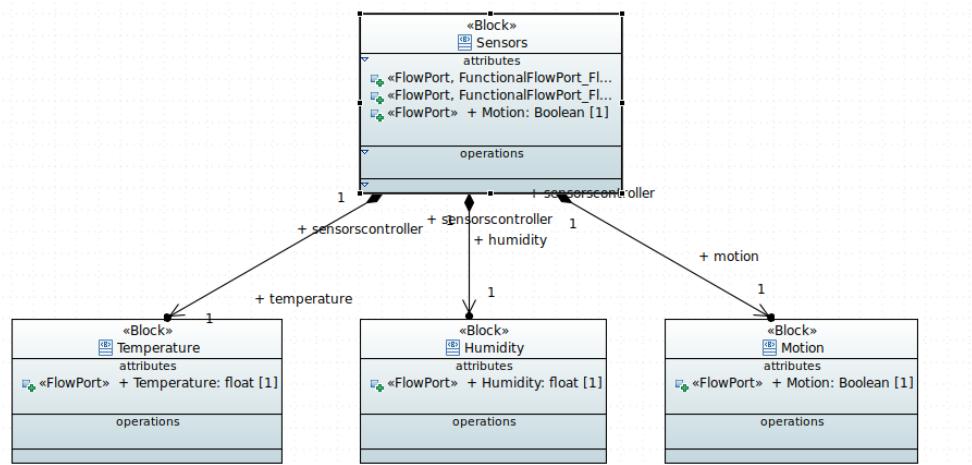


Figure 7: Room sensors components

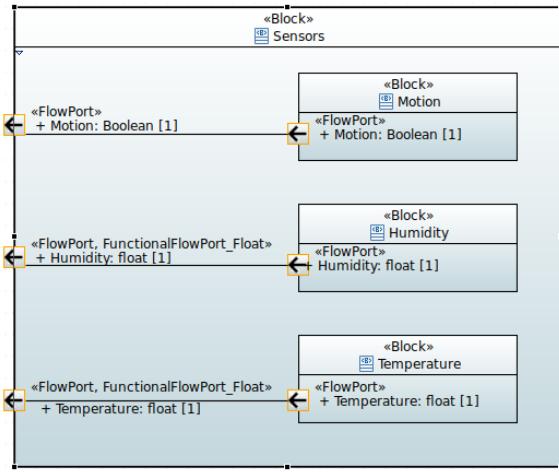


Figure 8: Room sensors internals

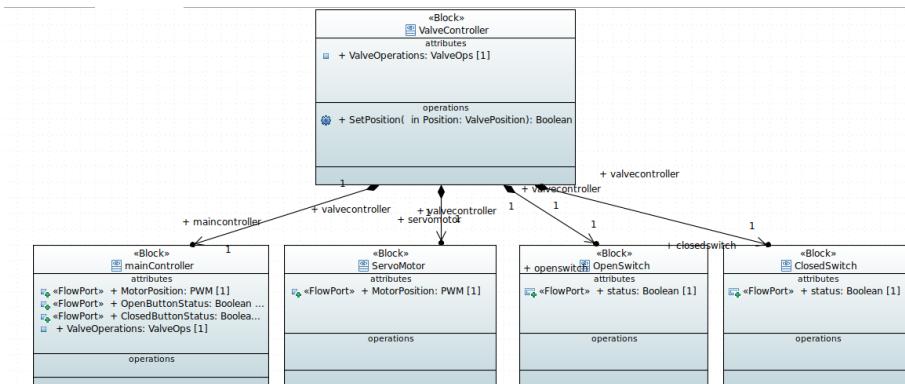


Figure 9: Valve Controller components

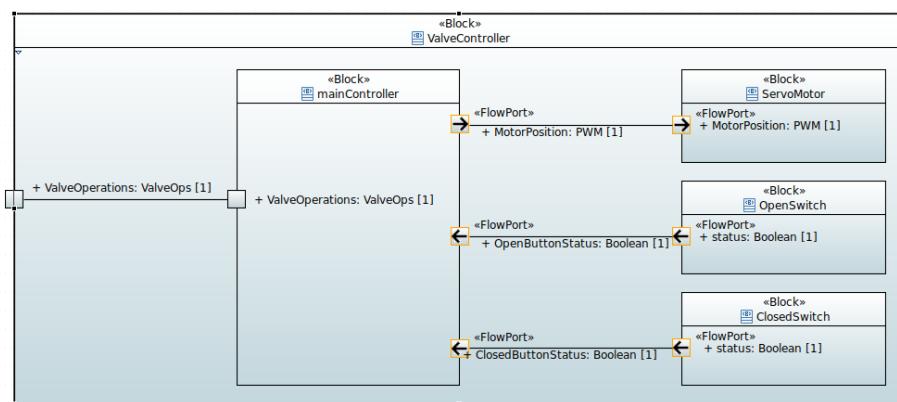


Figure 10: Valve Controller internals

3 Implementation, Central Unit module



Figure 11: Central Unit module

The module is composed by:

- STM32F407VG DISCOVERY powered by ARM Cortex-M4 32-bit
- LCD board Touchscreen 3"5

In the discovery board are running the *Central Unit's* tasks on top of Erika RTOS, the system is composed by three different tasks reported in the table ???. Whenever a *Room Status* message arrives through the *USART* interface, the *Check message* aperiodic task is activated in order to check the message and execute the *Room's* functionalities.

The *Check message* task apply a simple JSON compliance test and check each field name and value of the message, then the message is converted in a *Room structure* and the Room's status and *Building* status are updated.

While the *Check message* task has the role of processing the incoming *Room Status*, the *Polling* task has the role of managing the rooms sending a check message periodically one room after the other, if the response *Room Status* message doesn't arrive within the next task's job it's considered as an error and the same *Room Request* message is sent until a maximum of 3 resend, then the room is marked as crashed.

The *Graphic* task has the role to represent the data of the building and for each room depending on the selected page by the user, in the settings page the user is able to set the desired temperature that will be sent starting from the next job of the *Polling* task.

Name	Frequency	Priority
Check message task	Aperiodic	3
Polling task	0.2Hz	2
Graphic task	10Hz	1
Receiver	USART background	/

Table 7: Tasks running on Central Unit

3.1 Wiring



Figure 12: Discovery board wiring

The module is using the *USART6* of the discovery board to communicate with the *room* modules, in the following table is reported **USART**'s configuration.

USART	TX pin	RX pin	baud-rate	Parity Bit	Stop bits	Control Flow
6	PC6	PC7	9600	No	1	No

Table 8: USART configuration

3.2 Graphic Interface

The graphic interface is composed by three different screens:

- Home screen
- Settings screen
- Room screen



(a) Home screen

(b) Settings screen



(c) Room screen

The main page is the **home screen** reported in 13a, in this page are reported the average values of the building. If at least one room is marked as *crashed* then a warning appear on the home screen. If at least one room is in *eco mode* then an *eco* icon is displayed on the home screen. If the difference between the desired temperature and the actual average temperature is less then 0.5C° , the shown thermometer icon is hot otherwise cold. In the **settings screen** reported in 13b it is possible to set the desired temperature of the building that is the same for each room. In the **room screen** reported in 13c is displayed the status of the selected room, status composed by:

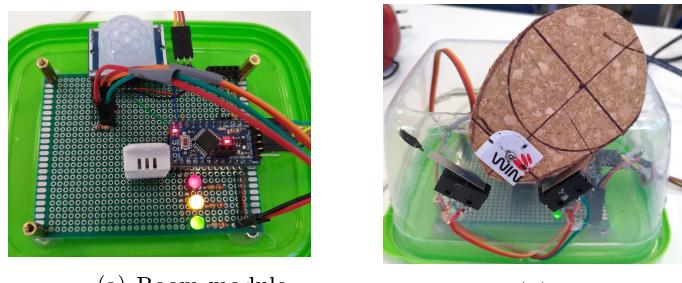
- Eco mode (the icon on the top-left of the screen)
- Temperature
- Humidity
- Valve position
- Warning (Warning icon between the eco icon and the icon in the top-center)

4 Implementation, Room module

The *Room* functionalities are running on a ATM328P, the tasks of the module are implemented as pseudo-periodical tasks using the *timer0* to check the activation time of each task, the tasks are reported in the following table.

Name	Frequency
Main task	0.5Hz
Control valve task	0.25Hz

Table 9: Tasks running on the Room module



(a) Room module

(b) Valve

The module is composed by:

- Arduino pro mini, Atmega328P (8MHz, 3.3v logic)
- DHT22 Temperature and Humidity sensor
- PIR motion sensor
- Red led
- Green led
- Yellow led
- Servo Motor tower pro SG90
- 2 switch to check the open and closed position of the valve

4.1 Components description

4.1.1 ServoMotor

The chosen Servo motor is a Tower Pro SG90 reported in 15, As reported in the 15, the signal used to move the servomotor in the correct position is a PWM and the duty-cycle describe the position that the motor has to maintain. The signal to control the motor position is managed by the Servo library using the **timer1** a 16-bit timer. The digital circuit inside the servomotor will adjust the position of the rotor using a potentiometer attached to that. In the following table are reported the characteristics of the servomotor:

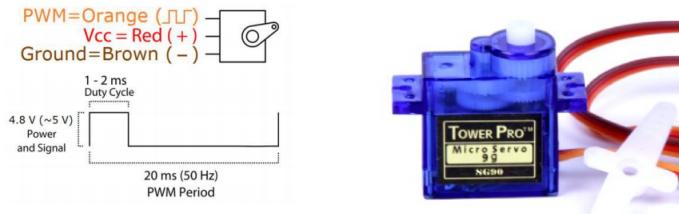


Figure 15: Servo Motor

Voltage(V)	4.8 - 6
Torque(Kg-cm)	2.5
Speed(sec)	0.1
Weight(g)	14.7

Table 10: Servo motor characteristics

4.1.2 Valve position switches

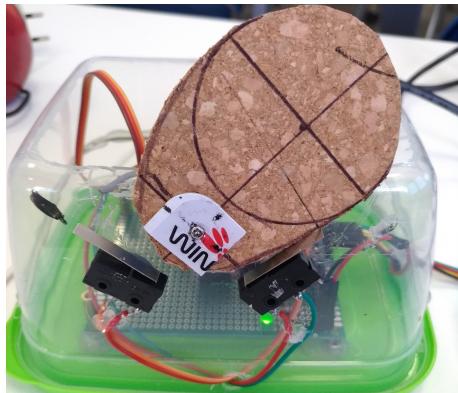


Figure 16: Discovery board wiring

In order to check the position of the valve, two switches are fixed in the open and closed position, during the *valve initialization phase* the possible positions are computed starting from the open and closed position. Whenever the valve is in the open or closed position a check is done using the switches.

4.1.3 Temperature-Humidity Sensor

The sensor DHT22 have been used for the project, in the following tables are reported the features of the sensors. The sensor has a dedicated one-wire communication protocol implemented by the DHT adafruit library, a pull-up resistor is used in order to maintain the line clear when no one is transmitting.

Voltage(V)	3 - 5
Current(mA)	2.5
Humidity(%)	0 - 100
Humidity Accuracy(%)	2 - 5
Temperature(C°)	-40 - 80
Temperature Accuracy(C°)	+/- 0.5
Sampling rate(Hz)	0.5

Table 11: DHT22 characteristics

4.2 Wiring

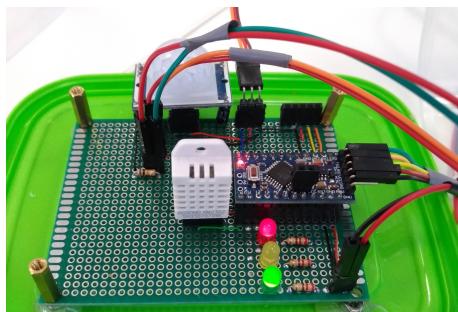


Figure 17: Room board wiring

The board is powered by 5V external power attached to the *Raw pin*, this source is shared with the servomotor and the PIR sensor. In 17 is reported a picture of the implementation, in the following table the pins' configuration.

Red led	13	D OUTPUT
Yellow led	12	D OUTPUT
Green data	11	D OUTPUT
DHT data	10	D INPUT
PIR data	9	D INPUT
ServoMotor PWM	8	D OUTPUT
Open Switch	7	D INPUT
Closed Switch	6	D INPUT
USART TX	0	D OUTPUT
USART RX	1	D INPUT

Table 12: Atmega328P Wiring

5 JSON messages implementation

As described in the SyML internal model of the system 4, the modules exchange information as *Room request* and *Room status*. These messages have been implemented using the *JSON* format, in the following are reported two messages exchanged during execution. In the following version of the project the field names and the order of the variables must follow this example otherwise the simple JSON parser will consider the message as corrupted.

5.1 Request Message

```
|| {  
||   "Id": "01",  
||   "DTemp": "30.00"  
|| }
```

5.2 Status Message

```
|| {  
||   "Id": "02", "Eco": "0",  
||   "sens": [  
||     {"Nm": "Tmp", "Val": "23.70", "Fmt": "C"},  
||     {"Nm": "Hum", "Val": "051.90", "Fmt": "\%"}  
||   ],  
||   "acts": [  
||     {"Nm": "Vlv", "Val": "100", "Fmt": "\%"}  
||   ]  
|| }
```

6 CUnit tests and simulation of the system

In order to check the behaviour of the system a sequence of tests have been executed on the *ESTA_JSON_lib* functions, that is used to convert the incoming JSON string from the module *Room*.

6.1 Variables tests

For each variable present in the JSON message a boundary test is executed.

Variable	Min	Max	1°	2°	3°	4°	5°	6°	7°
Id	1	2	0	1	2	3	/	/	/
Eco	0	1	-1	0	1	2	/	/	/
Temperature	15.0	30.0	14.99	15.0	15.01	29.99	30.00	30.01	22.5
Humidity	0.0	100.0	-0.01	0.0	0.01	99.99	100.00	100.01	50.00
Valve	0	100	-1	0	1	99	100	101	50

Table 13: Variables tests

6.2 JSON Compliance tests

In order to check the *JSON compliance code checker* the following tests are applied modifying a correct message.

Number	Description
1	Missing odd number of {
2	Missing odd number of "
3	Missing odd number of /
4	Check field name <i>Id</i>
5	Check field name <i>Eco</i>
6	Check field name <i>Nm</i>
7	Check field name <i>Val</i>
8	Check field name <i>Fmt</i>
9	Check field name <i>Tmp</i>
10	Check field name <i>Hum</i>
11	Check field name <i>Vlv</i>
12	Check Temperature format value <i>C</i>
13	Check Humidity format value <i>%</i>
14	Check Valve format value <i>%</i>

Table 14: JSON compliance tests

In the following picture 18 is reported the result of the tests executed using the framework *CUnit* Version 2.1.3.

```

Suite: Suite_1
  Test: testJSON_check_message ...passed
  Test: testJSON_ID ...passed
  Test: testJSON_eco ...passed
  Test: testJSON_temperature ...passed
  Test: testJSON_humidity ...passed
  Test: testJSON_valve ...passed
Completing suite1

Run Summary:    Type   Total     Ran Passed Failed Inactive
                 suites      1       1     n/a      0       0
                  tests       6       6       6      0       0
                 asserts     43      43      43      0     n/a

Elapsed time =    0.000 seconds

```

Figure 18: CUnit results

6.3 Network noise tests

In order to check the correctness behaviour of the *Central Unit* module a python script that simulates the rooms has been created and used to test the module simulating two different rooms. The **Room simulator** creates the correct JSON string and send it, in order to simulate the **noise** of the network this cases have been considered:

Number	Description	Probability
1	Message lost, the response message is discarded	0.1
2	Message corrupted, 3 random characters are replaced with ?	0.1

Table 15: Network noise tests

```

Receive: {"Id": "01", "Dttemp": "24.00"}
-----Corrupted-----
Message lost

Receive: {"Id": "01", "Dttemp": "24.00"}
Send: {"Id": "01", "Eco": "1", "sens": [{"Nm": "Tmp", "Val": "22.37", "Fmt": "C"}, {"Nm": "Hum", "Val": "88.27", "Fmt": "%"}]}, "acts": [{"Nm": "Vlv", "Val": "056", "Fmt": "%"}]}
-----Corrupted-----
Receive: {"Id": "02", "Dttemp": "24.00"}
Message lost

Receive: {"Id": "02", "Dttemp": "24.00"}
-----Corrupted-----
Send: {"Id": "02", "Eco": "0", "sens": [{"Nm": "Tmp", "Val": "27?26", "Fmt": "C"}, {"Nm": "Hum", "Val": "11.02", "Fmt": "%"}]}, "acts": [{"Nm": "Vlv", "Val": "046", "Fmt": "%"}]}
Receive: {"Id": "01", "Dttemp": "24.00"}
Message lost

Receive: {"Id": "01", "Dttemp": "24.00"}
-----Corrupted-----
Send: "?Id": "01", "Eco": "1", "sens": [{"Nm": "Tmp", "Val": "26.65", "Fmt": "C"}, {"Nm": "Hum", "Val": "18.88", "Fmt": "%"}]}, "acts": [{"Nm": "Vlv", "Val": "044", "Fmt": "%"}]}
Receive: {"Id": "01", "Dttemp": "24.00"}
Send: {"Id": "01", "Eco": "0", "sens": [{"Nm": "Tmp", "Val": "25.51", "Fmt": "C"}, {"Nm": "Hum", "Val": "22.18", "Fmt": "%"}]}, "acts": [{"Nm": "Vlv", "Val": "072", "Fmt": "%"}]}
Receive: {"Id": "01", "Dttemp": "24.00"}
Message lost

```

Figure 19: Room simulator

7 Future Work

This kind of **energy saving** application strictly depends on the usage of the building and the building itself.

In order to maximize the effectiveness of the heating system and minimize the consumption, different parameters play an important role, I report some of them in the following list

- room characteristics (dimension, orientation, furniture, etc.)
- usage of the room strictly depends on the users' life style
- external weather condition
- period of the year

These parameters have to be taken in consideration to compute the **energy threshold** of each room in such a way that the single room can reach the desired temperature in the shortest time.

One of the possible future work could be to integrate a *Machine Learning* functionality in order to adjust this *energy threshold* parameter considering all the information related to the construction of the building and the usage coming from the sensors in the rooms.