

SMART LIGHTING AND HEATING SYSTEM – SECOND DELIVERY

André Filipe Almeida Gonçalves 1210804
Jorge Rafael Novais Moreira 1201458

ISEP INSTITUTO SUPERIOR
DE ENGENHARIA DO PORTO

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

2024

This report meets the requirements outlined in the Course Unit Description of the Critical Systems Laboratory, in the 1st year of the Master's in Critical Computer Systems Engineering.



Departamento de Engenharia Informática

Instituto Superior de Engenharia do Porto

19 de Dezembro de 2024

1. INTRODUCTION

This document underlines the development process of the chosen project for the subject Laboratório de Sistemas Críticos. It provides an overview of the problem context and details the components and technologies chosen for the project's implementation.

1.1. CONTEXT

In modern buildings like the one housing the CISTER unit, smart management of lighting and heating is essential for promoting environmentally friendly practices, reducing operational costs, and enhancing the well-being of researchers. This project aims to develop an automated climate and lighting control system tailored to the unique characteristics of the CISTER building, particularly focusing on its 3rd floor.

The floor comprises seven individual offices, each equipped with one smart blind and one smart heater, four double-occupancy offices with two independent smart blinds and heaters each, and the director's office, which features three smart blinds and four smart heaters. The building layout is illustrated in the accompanying diagram.

All offices are equipped with smart devices, including blinds represented in green, heaters in red, and ceiling lights in orange, which can be monitored and controlled via wireless sensor networks (WSN) as shown in the Figure 1 .

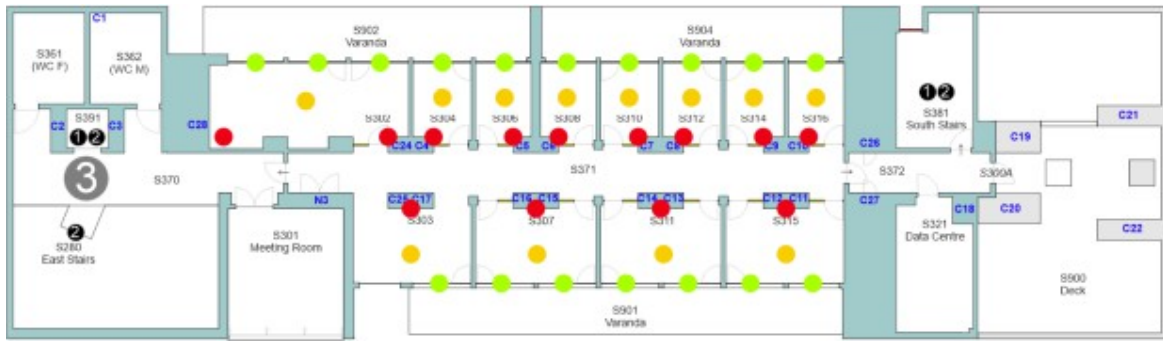


Figure 1- Floor 3 of CISTER building.

1.2. OBJECTIVES

The focus of this project is to automate the lighting and temperature control on the 3rd floor of the CISTER building, specifically targeting the individual offices due to the variety of office types present on the floor.

The first task involves automating the control of office lighting. Priority is given to natural sunlight, with the blinds automatically adjusting to maximize natural light entry. When natural light is insufficient, artificial lighting will compensate to maintain the desired illumination level.

The second task addresses the control of heating through the smart heater installed in the office.

Additionally, the envisioned automated climate control system will include a user interface that allows office occupants to configure their preferences for lighting and temperature, as well as manually adjust the blinds, lights, and heater. The system will also notify occupants if their preferences cannot be met, potentially indicating an operational issue.

The objective is to design and implement a prototype system that fulfils these requirements. Due to limited access to the building (in line with current government guidelines) and the potential risks of interacting with real infrastructure—such as damaging the blinds, which could result in costly repairs or user inconvenience—the implementation will primarily rely on simulated behaviours. However, integrating physical sensors or actuators, such as representing blind movements with different LED colours, is strongly encouraged. When such actuators are included, their status (enabled/disabled) must be clearly indicated.

2. DESIGN

This chapter outlines the understanding of the problem and the design of the solution that will guide the project's development.

2.1. PROBLEM OVERVIEW

The project involves managing various actuators based on environmental parameters. Since temperature and light intensity are the key factors, it is essential to use sensors capable of accurately measuring these values.

2.1.1. Components Used

Component	What we would like to use	What we will be using
Temperature Sensor	DHT11	DHT11
Luminosity Sensor	TSL2561	LDR5539
Blinds Mechanism	SG90	SG90
Heater Mechanism	Heating Plate	LED
Smart Lights	LED	LED

Table 1- Components

As mentioned above, there are alternative components we would prefer to use instead of the ones currently chosen. We will now explain the reasons for selecting these components and explain why we would choose some others over the ones initially planned.

Temperature Sensor:

We have decided to keep the DHT11 sensor for temperature measurement in our project. The DHT11 operates within a voltage range of 3V to 5.5V, and we will power it using a 3.3V supply. Since the ESP32 operates at 3.3V logic, we will handle the level shifting for the data line to ensure proper communication between the ESP32 and the DHT11 sensor. Despite its limitations in accuracy compared to other sensors like the DHT22, the DHT11 is sufficient for

our project's requirements, providing reasonably accurate temperature readings within the desired range while keeping the project cost-effective.

The DHT11 will be connected to **pin 18** of the ESP32 and is mounted securely on a breadboard for easy prototyping and wiring. Its three pins (VCC, GND, and DATA) are connected as follows:

- **VCC** is connected to the 3.3V power rail of the ESP32.
- **GND** is connected to the ground rail.
- **DATA** is connected to ESP32 **pin 18**.

This setup ensures reliable temperature readings while keeping the design simple and robust.

Luminosity Sensor:

The TSL2561 is a digital light sensor capable of measuring illuminance with higher precision and a wider dynamic range, making it suitable for applications requiring accurate light intensity measurement in varying conditions. Despite its advantages, the TSL2561 requires more complex interfacing and is generally more expensive than analogue alternatives.

For this project, we have chosen the LDR5539, a light-dependent resistor (LDR), which, while less precise and lacking the dynamic range of the TSL2561, is both cost-effective and simple to integrate. The LDR5539 provides adequate performance for our requirements by detecting light intensity changes effectively. Additionally, its straightforward analogue output simplifies the circuit design and reduces the need for additional hardware or complex programming, aligning well with our budget constraints and project scope.

The LDR is mounted on the breadboard and forms a voltage divider circuit with a 10k Ω resistor. This configuration allows the ESP32 to measure the changing resistance of the LDR in response to light levels by reading the corresponding voltage change. The centre of the voltage divider is connected to **pin 34**, an analogue input pin on the ESP32.

The connections are as follows:

- One end of the LDR is connected to the **3.3V power rail**.
- The other end of the LDR is connected to one side of the 10k Ω resistor.
- The opposite side of the resistor is connected to **GND**.
- The centre tap of the voltage divider (between the LDR and resistor) is connected to ESP32 **pin 34**.

Blinds Sensor:

We have chosen to use the SG90 motor for the blind's mechanism, as it is a low-cost, widely available, and reliable option for small projects. It offers sufficient torque and precision for controlling small blinds, and it is easy to integrate with microcontrollers like Arduino. In our project, we will use the SG90 motor to simulate the blinds mechanism by moving it forward to open the blinds, which will trigger one LED to light up, and moving it backward to close the blinds, which will light up a different LED. This setup will simulate the opening and closing of blinds.

The SG90 servo motor is powered directly from the ESP32, which provides adequate power for this low-torque motor. The connections are as follows:

- **VCC** is connected to the **3.3V power rail**.
- **GND** is connected to the **ground rail**.
- **Control signal** is connected to ESP32 **pin 19**.

Heater Mechanism:

We initially planned to use a heating plate to control temperature more directly. However, due to power constraints and the nature of the project, we decided to substitute the heating plate with an LED, which will allow us to simulate the heating process more efficiently without requiring additional power resources. In our simulation, using an LED is simpler because it provides a straightforward way to visually represent the heating process without the complexity of managing actual temperature control. The LED will be turned on to simulate heating, offering a clear and easy-to-implement solution for our needs.

The LED is mounted on the breadboard and connected in series with a 220Ω resistor to limit the current and protect the LED. The ESP32 toggles the LED on or off to simulate the heater being activated or deactivated.

The connections are as follows:

- The positive leg (anode) of the LED is connected to **pin 4** of the ESP32.
- The negative leg (cathode) is connected to one side of the 220Ω resistor.
- The other side of the resistor is connected to **GND**.

Smart Lights:

We have chosen an LED because it fits the functional requirements of the smart lights in our project. The LED provides a simple and effective way to simulate the behaviour of a real smart light. When the ambient brightness, measured by the LDR5539, falls below a defined threshold, the LED automatically lights up, mimicking how smart lights respond to low light conditions.

The system dynamically monitors light intensity through the LDR5539 sensor, connected to an analogue input pin of the ESP32. The ESP32 processes the sensor readings and compares them to the predefined threshold. If the measured brightness is lower than the threshold, the ESP32 sends a signal to activate the LED.

The LED is mounted on the breadboard and connected in series with a 220Ω resistor to limit the current and protect the LED. The ESP32 toggles the LED on or off to simulate the heater being activated or deactivated.

The connections are as follows:

- The positive leg (anode) of the LED is connected to **pin 5** of the ESP32.
- The negative leg (cathode) is connected to one side of the 220Ω resistor.
- The other side of the resistor is connected to **GND**.

2.1.2. Technologies

Technologies
MQTT
TCP/IP
NODE-RED
ARDUINO (ESP32)

Table 2 - Technologies

Arduino:

The system incorporates two Arduino boards, each serving a distinct role to ensure modularity and efficiency. The Sensor Arduino is tasked with collecting data from the DHT11 temperature and humidity sensor and the LDR5539 light-dependent resistor. This Arduino processes the sensor data and transmits it to the TCP server for validation and further handling. Initially, the

data is sent in an unstructured format tailored to the project's immediate needs. Although Smart Data Models are not yet implemented, they are planned for future integration to standardize the data format and enhance compatibility with other systems.

The Actuator Arduino, on the other hand, manages the system's actuators, including the SG90 servo motor and LEDs. It receives validated commands from the TCP server and executes actions such as adjusting the blinds mechanism, turning on or off the simulated smart lights, or activating the heating simulation. Both Arduinos connect to the same Wi-Fi network, enabling reliable communication with the TCP server and ensuring smooth operation within the system's architecture.

MQTT:

MQTT is planned for use in the future to facilitate communication between the control unit (TCP server) and the user interface, which is built using Node-RED. Once developed, MQTT will allow the control unit to publish sensor data received from the Sensor Arduino to an MQTT broker, making it accessible to Node-RED for real-time visualization and monitoring. Additionally, MQTT will be used to send user commands from Node-RED to the TCP server, which will relay them to the Actuator Arduino. This bi-directional communication will help streamline integration and data flow across the system.

At present, MQTT is still in the development phase, and data transmission is handled using basic unstructured communication between the components. Smart Data Models will later be incorporated to ensure standardized data formatting for MQTT communication.

TCP/IP:

TCP/IP is the primary protocol used for communication between system components, ensuring reliable and connection-oriented data exchange. The Sensor Arduino sends sensor readings to the TCP server over TCP/IP, where the data is validated and processed. The server also sends commands to the Actuator Arduino through the same protocol, ensuring that actuator actions are carried out without any loss of data. This reliable communication protocol is essential for maintaining the integrity of the system.

NODE-RED:

While Smart Data Models are not yet implemented, they are an important planned feature for the system. These models will provide standardized data formats for sensor readings and actuator

commands, ensuring consistent communication across components. Once integrated, Smart Data Models will improve the system’s compatibility with third-party IoT platforms and streamline data management. The Sensor Arduino will format its sensor data according to these models before transmitting it to the server, and user commands from Node-RED will also be structured accordingly. This future enhancement will ensure the scalability and interoperability of the system.

This combination of technologies ensures that the project is built on a solid foundation, with immediate functionality in place and a roadmap for future improvements as seen in the Figure 2.

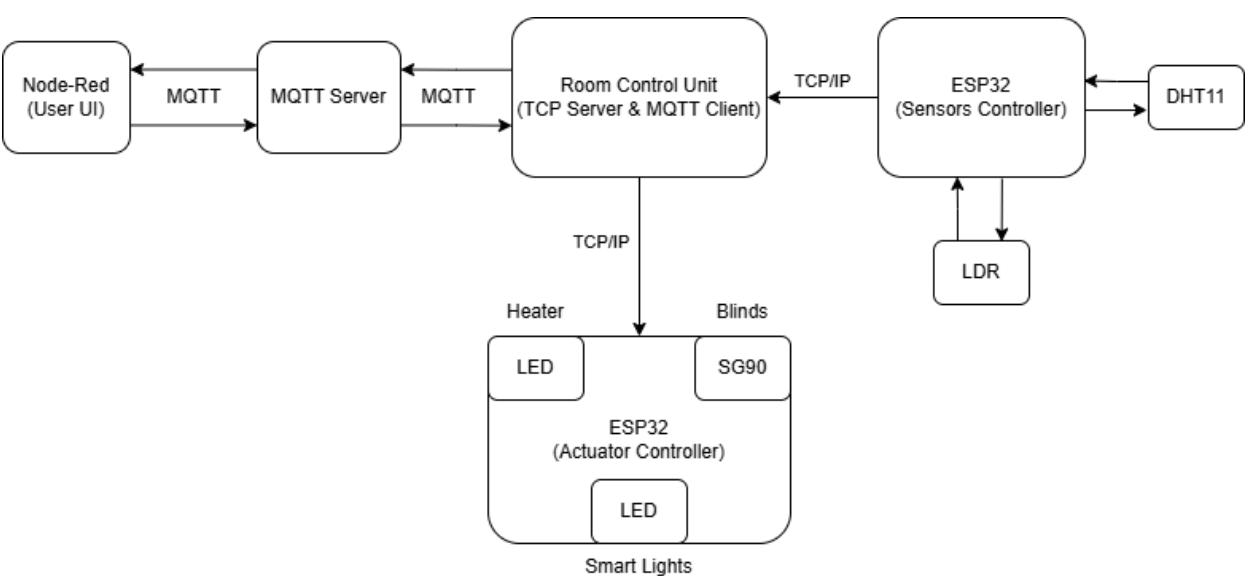


Figure 2 - Communication scheme.

2.2. Problem Domain Design

In this section, it will be discussed the problem domain design created to answer the problem posed in this assignment.

2.2.1. Black Box

First, we identified the Stakeholder Needs, the requirements that the stakeholders have for the assignment. The identified Stakeholder Needs were:

	Id	text
User Needs	SN-1	
Automated Lighting Control	SN-1.1	When natural sunlight is available, the system shall adjust blinds to allow the required amount of light. If natural sunlight is insufficient, the system shall activate artificial lighting to compensate.
Natural Light Adjustment	SN-1.1.1	When natural sunlight is available, the system shall adjust blinds to allow the required amount of light.
Artificial Light Compensation	SN-1.1.2	When natural sunlight is insufficient, the system shall activate artificial lighting to compensate.
Automated Heating Control	SN-1.2	The system shall control the heating in the office using smart heaters.
User Interface	SN-1.3	The system shall provide a user interface that allows office occupants to configure their preferences for lighting and temperature, and manually change the status of blinds, lights, and heaters.
Hazard Mitigation	SN-1.4	The system shall include measures to prevent or mitigate failures by identifying malfunctioning components, providing temporary fixes, and providing long-term fixes.
Notification System	SN-1.5	If the occupant's preferences are not being met, the system shall notify the occupant and indicate a potential problem with the system.
WSN Network	SN-1.6	Where possible, the system shall utilize Wireless Sensor Networks (WSNs) for communication and control.

Figure 3 Stakeholder Needs

The next step was creating the system context, where we represented the system needed and the

interaction between these systems.

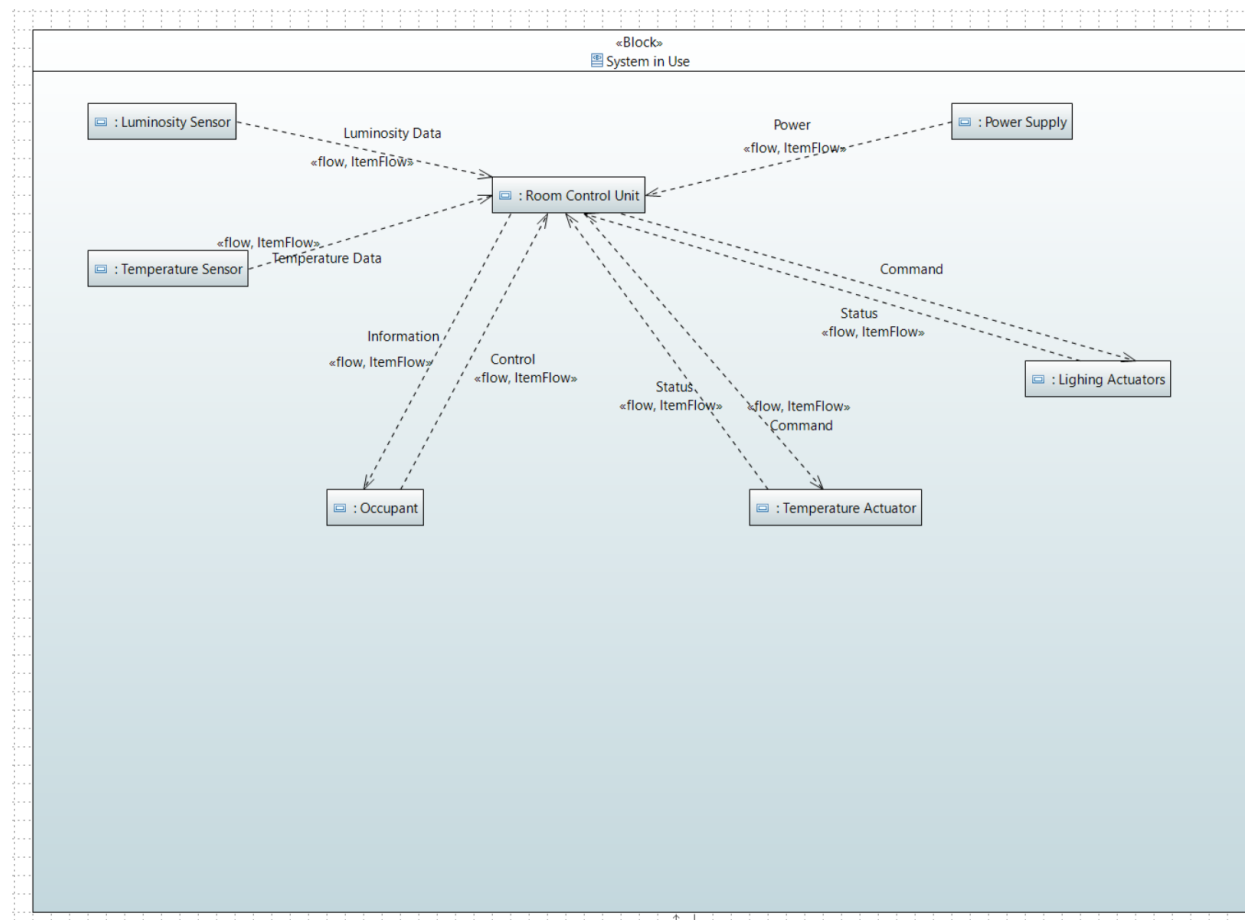


Figure 4 System Context

Following the creation of the system context, we created a use case diagram for the use cases that we found.

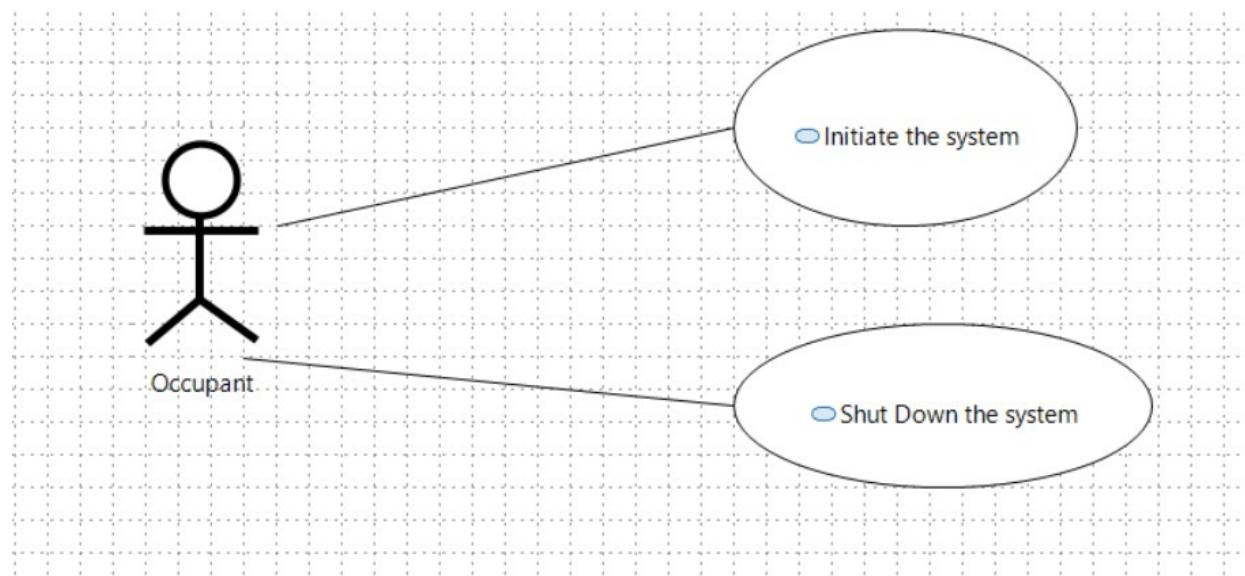


Figure 5 Use Cases

Using the use cases that we found, we created an activity diagram for these use cases, showing the flow of the systems during execution, and further explaining what happens between these systems.

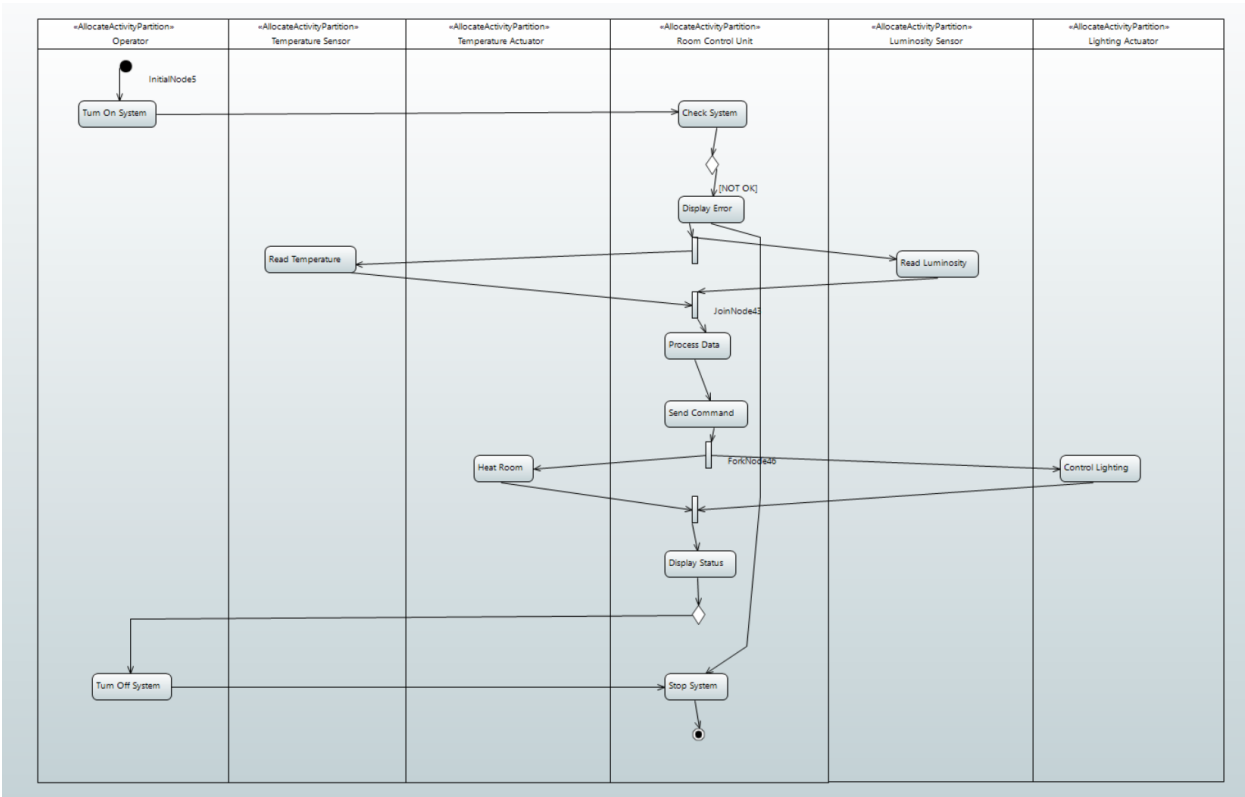


Figure 6 Activity Diagram

To finish the black box design, the measures of effectiveness, the metric that ensure the system is working correctly, are created.

We measure the power needed for the system to work and the response time between the system receiving the data and sending the proper command.

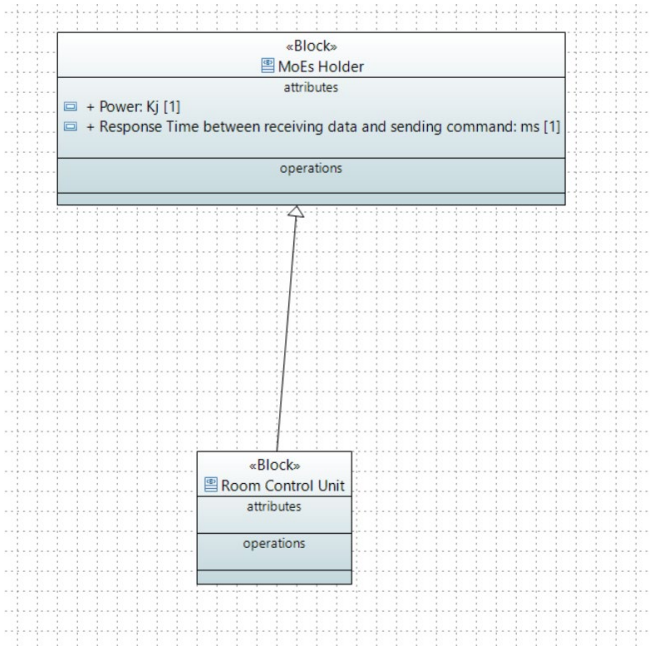


Figure 7 Measures of Effectiveness

2.2.2. White Box

The first thing done for the white box model was figure out what were the conceptual systems that existed. The following figure is the representation of these conceptual systems.

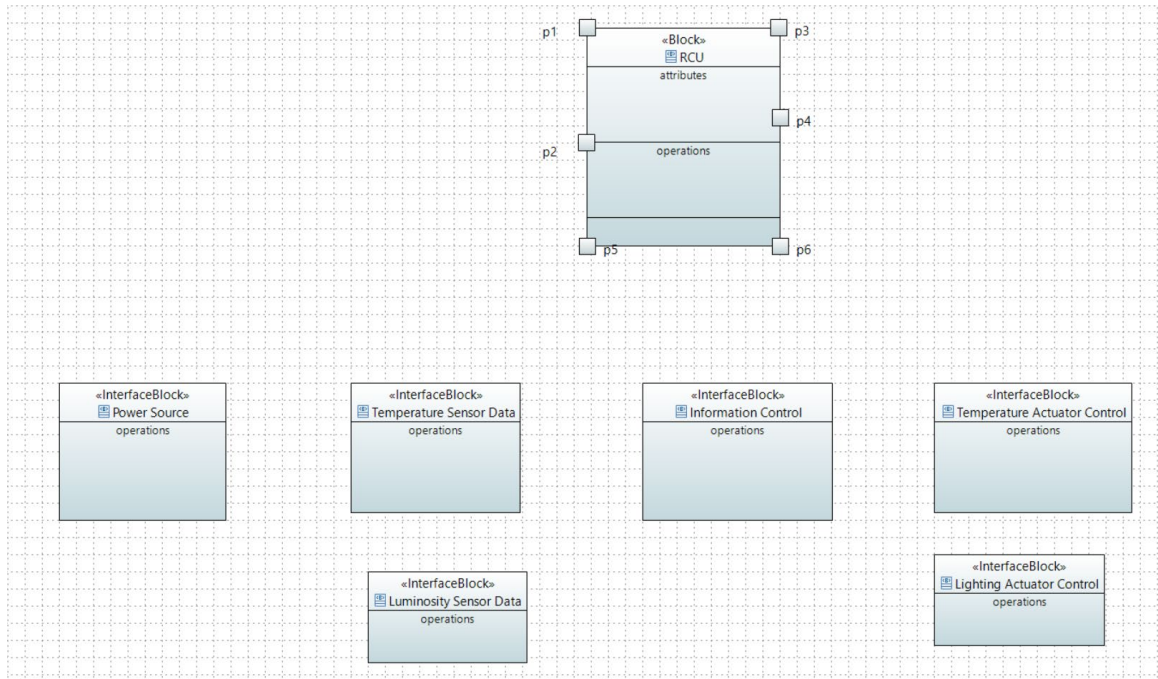


Figure 8 Conceptual Systems

The next step of the modelling process was to represent the conceptual subsystems envisioned for the room control system and the interaction between these conceptual subsystems and with the outside systems.

To figure out what were the subsystems needed for the system, we did a functional analysis of said system, being the main functionalities of the system reading the data from the sensors, processing said data, sending commands to the temperature and lighting actuators, and interacting with the user.

So, we used these functionalities to create each subsystem.

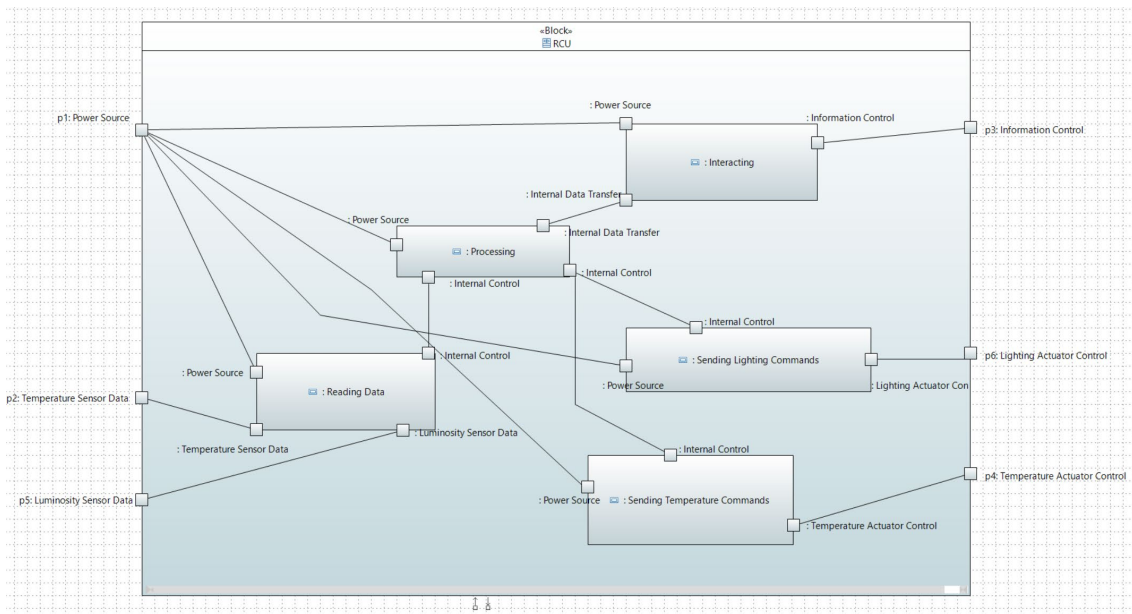


Figure 9 Conceptual Subsystems

Like in the black box model we created an activity diagram to represent the flow of execution of these conceptual subsystems.

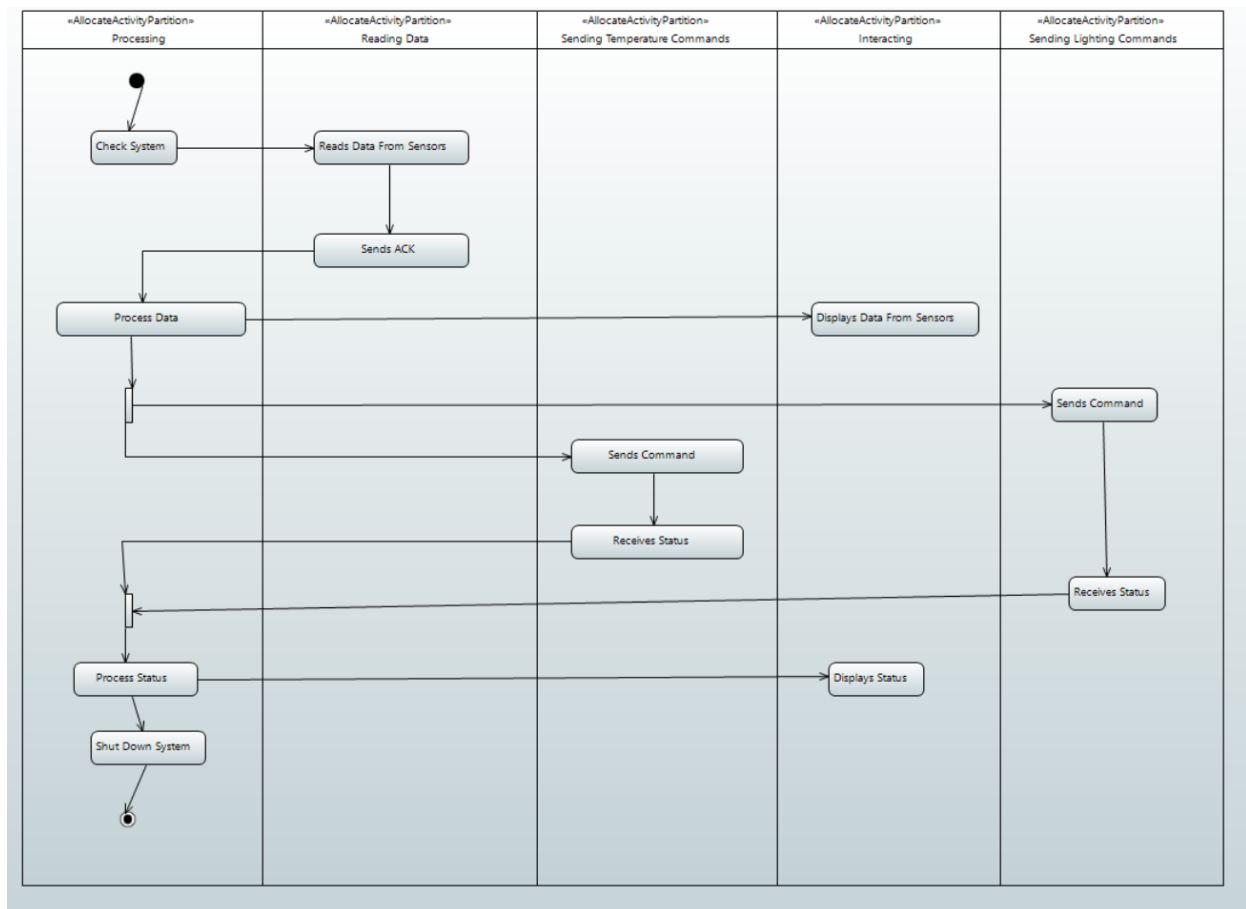


Figure 10 Conceptual Subsystems Activity Diagram

Lastly, we represented the measure of effectiveness used by these subsystems.

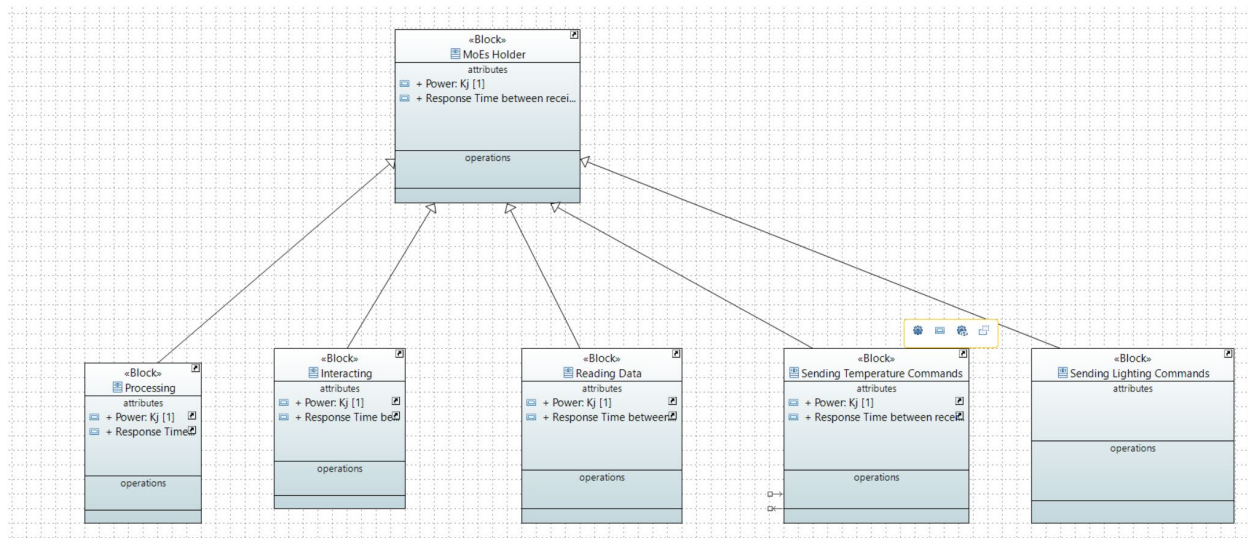


Figure 11 Measures of Effectiveness for the Conceptual Systems

2.2.3. Traceability

The last part of the problem domain is the traceability between the Stakeholder Needs and the systems that are represented in the system context, explaining what each system does in relation to the Stakeholder Needs, and subsystems representing also the relation between these subsystems and the Stakeholder Needs.

		A	B	C	D	E	F
		Automated Lighting Control	Automated Heating Control	User Interface	Hazard Mitigation	Notification System	WSN Network
0	System In Use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	Property1 : Room Control Unit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	Property2 : Luminosity Sensor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
4	Property3 : Lighting Actuators	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
5	Property4 : Power Supply	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
6	Property5 : Occupant	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7	Property6 : Temperature Sensor	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
8	Property7 : Temperature Actua...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
9	Sending Temperature Commands	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
10	Reading Data	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
11	Interacting	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
12	Processing	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
13	Sending Lighting Commands	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 12 Traceability

2.3. Solution Domain Model

In this section, it will be discussed the solution domain design created to answer the problem posed in this assignment.

2.3.1. System Requirements

The system requirements are directly correlated to the Stakeholder Needs represented in the problem domain section. These requirements are the minimum applications that the system needs to function correctly as envisioned by the stakeholders.

		A	B	C	D	E	F	G	H
		☒ Natural Light Adjustment	☒ Artificial Light Compensation	☒ Heating Control	☒ Preference Configuration	☒ Preference Notification	☒ Failure Prevention	☒ Temporary Fix	☒ Wireless Sensor Network
0	☒ Subsystem Requir...	☐	☐	☐	☐	☐	☐	☐	☐
2	☒ User Interface	☐	☐	☐	☑	☐	☐	☐	☑
3	☒ Feedback Display	☐	☐	☐	☐	☑	☐	☐	☑
4	☒ User Input Vall...	☐	☐	☐	☑	☐	☐	☐	☐
5	☒ Interface Availa...	☐	☐	☐	☐	☐	☐	☑	☐
6	☒ Error Notification	☐	☐	☐	☐	☐	☑	☐	☑
7	☒ Real-time Inter...	☐	☐	☐	☑	☐	☐	☐	☑

Figure 13 System Requirements

2.3.2. High Level Architecture

After the requirements, the high-level architecture was modelled, where all the subsystems that describe the system, room control, are captured. The subsystems are as follows, the data reading system, processing system, command system and user interacting system.

In comparison with the conceptual subsystem in the problem domain, the command system is a junction of the two subsystems created for sending commands for each type of actuators.

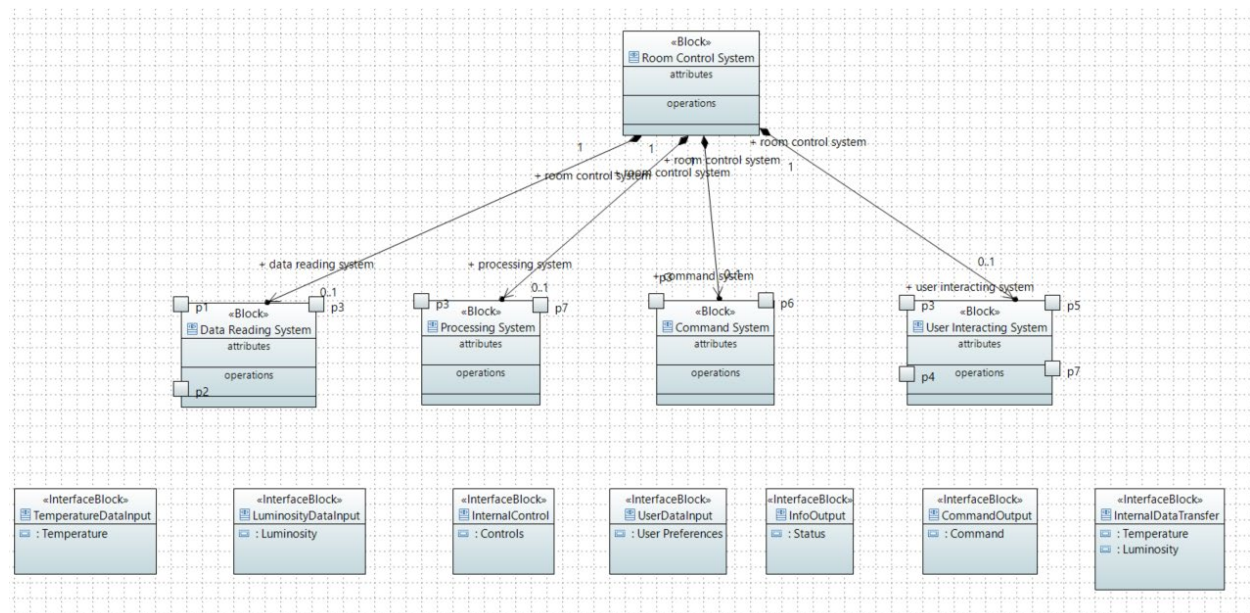


Figure 14 High Level Architecture

2.3.3. Subsystem Requirements

The subsystem requirements specify the requirements for the individual subsystems within the larger system. These requirements outline what is necessary for a part of the system to function properly and integrate seamlessly with the rest of the system.

As we have four subsystems, we will have one table for each subsystem.

	id	text
Subsystem Requi...	SSR-1	
Data Processing	SSR-1.1	The subsystem shall process data from sensors, user inputs, and external systems to control the office environment.
Decision-making	SSR-1.2	The subsystem shall make decisions based on predefined rules and occupant preferences to optimize office conditions.
Data Integrity	SSR-1.3	The subsystem shall validate incoming data to ensure it is free from corruption or malicious interference.
Fault Isolation	SSR-1.4	If a fault occurs in the processing subsystem, it shall isolate the fault to prevent impact on other subsystems.
Safe State	SSR-1.5	If the subsystem fails, it shall transition the system into a safe state to avoid harm to occupants or equipment.
Error Handling	SSR-1.6	The subsystem shall detect errors during operation and log them for diagnosis while attempting to recover automatically.

Figure 15 Processing System Requirements

The following table represents the subsystem requirements for the data reading system.

	id	text
Subsystem Requi...	SSR-1	
Sensor Data C...	SSR-1.1	The subsystem shall collect data from all connected sensors, including lighting, temperature, and occupancy sensors.
Data Accuracy	SSR-1.2	The subsystem shall ensure that sensor data is accurate to within the tolerances specified by the sensor manufacturer.
Real-time Data	SSR-1.3	The subsystem shall read data from all sensors at intervals of no more than one second.
Sensor Validati...	SSR-1.4	The subsystem shall validate sensor readings to detect anomalies that may indicate sensor malfunction or tampering.
Data Overload ...	SSR-1.5	The subsystem shall implement safeguards to prevent data overload from impairing system performance.
Safe Failure	SSR-1.6	If a sensor fails to provide data, the read data subsystem shall notify the processing subsystem and use the last known good value as a fallback.
Error Logging	SSR-1.7	The subsystem shall log errors, including missing or out-of-range sensor data, for further analysis.

Figure 16 Data Reading System Requirements

The last tables represent, respectively, the command system requirements and the user interacting system requirements.

	id	text
Subsystem Requi...	SSR-1	
Command Tra...	SSR-1.1	The subsystem shall transmit commands to the actuators, including blinds, lights, and heaters, based on instructions from the processing subsystem.
Real-time Com...	SSR-1.2	The subsystem shall ensure that all commands are transmitted to the actuators within 100 milliseconds of receiving instructions from the processing subsystem.
Command Con...	SSR-1.3	The command subsystem shall confirm the successful execution of commands by receiving status updates from actuators and sending these confirmations to the processing subsystem.
Failure Notifica...	SSR-1.4	If an actuator does not respond to a command within a specified time frame, the command subsystem shall notify the processing subsystem of the failure.
Fallback State	SSR-1.5	If an actuator fails to execute a command, the command subsystem shall revert the actuator to its last known safe state.

Figure 17 Command System Requirements

	id	text
Subsystem Requi...	SSR-1	
User Interface	SSR-1.1	The subsystem shall provide a user interface that allows occupants to configure preferences for lighting, temperature, and blinds.
Feedback Disp...	SSR-1.2	The subsystem shall display feedback on the current system status, including lighting, temperature, and actuator conditions.
User Input Vall...	SSR-1.3	The subsystem shall validate user inputs to ensure they are within acceptable operational ranges before transmitting them to other subsystems.
Interface Avail...	SSR-1.4	The human interacting subsystem shall ensure the user interface remains accessible even during partial system failures.
Error Notification	SSR-1.5	The subsystem shall notify occupants of errors, such as actuator malfunctions or unmet preferences, and provide suggested actions.
Real-time Inte...	SSR-1.6	The subsystem shall process user commands and provide feedback within one second of interaction.

Figure 18 User Interacting System Requirements

2.3.4. Traceability

Like it was done in the previous subsection, traceability was done for each set of requirements of each subsystem with the system requirements, ensuring that each subsystem requirement can be traced to one or more system requirements.

The following matrices were created to help visualize the traceability between each set of subsystem requirements and the system requirements.

		A	B	C	D	E	F	G	H
		Natural Light Adjustment	Artificial Light Compensation	Heating Control	Preference Configuration	Preference Notification	Failure Prevention	Temporary Fix	Wireless Sensor Network
0	Subsystem Requi...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	Data Processing	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	Decision-making	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4	Data Integrity	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	Fault Isolation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	Safe State	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7	Error Handling	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 19 Processing Subsystem Requirements to System Requirements

		A	B	C	D	E	F	G	H
		Natural Light Adjustment	Artificial Light Compensation	Heating Control	Preference Configuration	Preference Notification	Failure Prevention	Temporary Fix	Wireless Sensor
0	Subsystem Requirements	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	Sensor Data Collection	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	Data Accuracy	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4	Real-time Data	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
5	Sensor Validation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6	Data Overload Prevention	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7	Safe Failure	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
8	Error Logging	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 20 Data Reading Requirements to System Requirements

		A	B	C	D	E	F	G	H
		Natural Light Adjustment	Artificial Light Compensation	Heating Control	Preference Configuration	Preference Notification	Failure Prevention	Temporary Fix	Wireless Sensor
0	Subsystem Requirements	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	Command Transmission	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	Real-time Command	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4	Command Confirmation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
5	Failure Notification	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6	Fallback State	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Figure 21 Command Requirements to System Requirements

		A	B	C	D	E	F	G	H
		Natural Light Adjustment	Artificial Light Compensation	Heating Control	Preference Configuration	Preference Notification	Failure Prevention	Temporary Fix	Wireless Sensor Network
0	Subsystem Requirements	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	User Interface	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	Feedback Display	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
4	User Input Validation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5	Interface Availability	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	Error Notification	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
7	Real-time Interaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 22 User Interacting Requirements to System Requirements

2.4. Safety and Reliability Analysis

In terms of safety and reliability there were some additions that had in mind the creation of a system which was both safe and reliable.

Not only the addition of requirements like hazard mitigation in the Stakeholder Needs, or the fallback state in the command subsystem requirements allow the system to be more reliable, limiting the fails and its damages, but also the addition of checks when starting the system allows the better execution of said system.

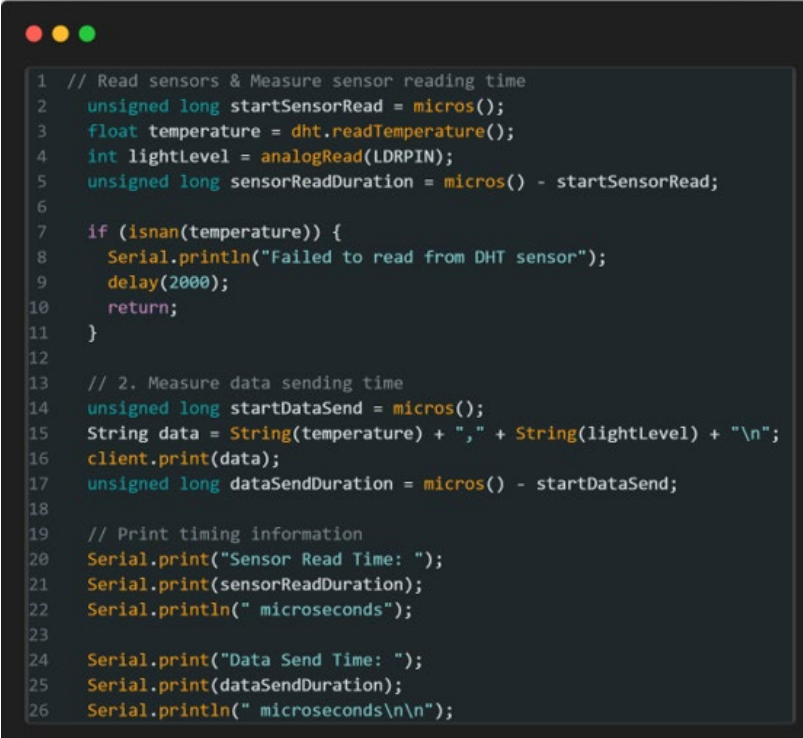
With the addition of error handling, we allow the user to, in real time, check the system and its execution, and in case of error the occupant can promptly verify the system and correct any errors or failures in said system.

3. Development

This section outlines the development process of the project, which integrates sensor readings, actuator execution, server communication, and low-level assembly code to perform led control.

3.1. ESP32 Sensor Reading

The ESP32 collects data from various sensors, including a DHT11 temperature sensor and an LDR (Light Dependent Resistor) and then sends the data retrieved from the sensors to the server TCP as seen in Figure 23.



```
1 // Read sensors & Measure sensor reading time
2 unsigned long startSensorRead = micros();
3 float temperature = dht.readTemperature();
4 int lightLevel = analogRead(LDRPIN);
5 unsigned long sensorReadDuration = micros() - startSensorRead;
6
7 if (isnan(temperature)) {
8     Serial.println("Failed to read from DHT sensor");
9     delay(2000);
10    return;
11 }
12
13 // 2. Measure data sending time
14 unsigned long startDataSend = micros();
15 String data = String(temperature) + "," + String(lightLevel) + "\n";
16 client.print(data);
17 unsigned long dataSendDuration = micros() - startDataSend;
18
19 // Print timing information
20 Serial.print("Sensor Read Time: ");
21 Serial.print(sensorReadDuration);
22 Serial.println(" microseconds");
23
24 Serial.print("Data Send Time: ");
25 Serial.print(dataSendDuration);
26 Serial.println(" microseconds\n\n");
```

Figure 23 – Sensors

The ESP32 continuously reads the temperature and light levels from the sensors and sends them to the server via TCP. The data is sent in a comma-separated format, and the server can then process and act on the information.

3.2. ESP32 Actuator Execution

The ESP32 controls actuators based on the commands received from the server. For example, it can turn on or off an LED or control a motor depending on what the server commands. This part of the system is shown below, where the server reads the sensor values, analyses the data, and sends the correct commands to the ESP32. Below is an example of the esp32-side logic executing commands:

```
1 // Wait for server's response
2 unsigned long startCommandReceive = micros();
3 if (client.available()) {
4     String commands = client.readString();
5     unsigned long commandReceiveDuration = micros() - startCommandReceive;
6
7     Serial.print("Command Receive Time: ");
8     Serial.print(commandReceiveDuration);
9     Serial.println(" microseconds\n\n");
10
11     // Parse and execute commands
12     int index = 0;
13     while ((index = commands.indexOf('\n')) != -1) {
14         unsigned long startCommandExecution = micros();
15
16         String command = commands.substring(0, index);
17         command.trim();
18
19         if (command == "HEATER_ON") {
20             digitalWrite(HEATERPIN, HIGH);
21         } else if (command == "HEATER_OFF") {
22             digitalWrite(HEATERPIN, LOW);
23         } else if (command == "LIGHT_ON") {
24             lightOnAssembly();
25         } else if (command == "LIGHT_OFF") {
26             lightOffAssembly();
27         } else if (command == "BLINDS_OPEN") {
28             for (pos = 0; pos <= 180; pos++) {
29                 blindsServo.write(pos);
30                 delay(15);
31             }
32         } else if (command == "BLINDS_CLOSE") {
33             for (pos = 180; pos >= 0; pos--) {
34                 blindsServo.write(pos);
35                 delay(15);
36             }
37         }
38     }
39 }
```

Figure 24- Actuators

3.3. Server TCP

The server handles the sensor data received from the ESP32, ensuring its validity, generating the corresponding commands, and sending them back to the ESP32 to control the actuators. Upon receiving the data, the server first validates its format and extracts the temperature and light-dependent resistor (LDR) values. If the data is valid, the server generates commands based on predefined conditions, as shown in the table below. For example, it adjusts the blinds, lights, and heater according to the LDR value and temperature. These commands are sent back to ESP32 if there are any changes in state, ensuring smooth communication.

Furthermore, the server tracks the time taken for different tasks, such as receiving the data, validating it, generating commands, and sending them to the client. These timing metrics assist in performance monitoring and troubleshooting. The server continues processing data and commands in a loop until the client disconnects.

Condition	Blinds	Lights	Heater
Light < 2000	Open (BLINDS_OPEN)	On (LIGHT_ON)	Based on Temp
2000 <= Light < 3000	Open (BLINDS_OPEN)	Off (LIGHT_OFF)	Based on Temp
Light <= 3000	Close (BLINDS_CLOSE)	Off (LIGHT_OFF)	Based on Temp
Temp < 20°C	No Change	No Change	On (HEATER_ON)
Temp >= 20°C	No Change	No Change	Off (HEATER_OFF)

Table 3- System Behaviour

3.4. Assembly

To implement this project, it was necessary to include a piece of code developed in Assembly. After evaluating the requirements, it was decided to implement two Assembly functions to control the state of an **LED**, which represents one of the actuators in the system. These functions directly manipulate the ESP32's GPIO registers, ensuring efficient control of the hardware.

The first function, **lightOnAssembly**, turns on the **LED** by setting **GPIO5** to high (on), while the second function, **lightOffAssembly**, turns off the **LED** by setting **GPIO5** to low (off). Here's the code for both functions:

A screenshot of a code editor with a dark background and light-colored text. The code is written in C with inline assembly blocks. It defines two functions: `lightOnAssembly` and `lightOffAssembly`. Both functions use `asm volatile` to execute assembly instructions. The `lightOnAssembly` function uses `movi`, `l32i`, `or`, and `s32i` instructions to set bit 5 of the GPIO output register. The `lightOffAssembly` function uses `movi`, `l32i`, `and`, and `s32i` instructions to clear bit 5 of the GPIO output register. The code is numbered from 1 to 23.

```
1 // Inline assembly function to turn on light
2 void lightOnAssembly() {
3     asm volatile (
4         "movi a2, 0x3FF44000\n"
5         "movi a3, (1 << 5)\n" /
6         "l32i a4, a2, 0x4\n"
7         "or a4, a4, a3\n"
8         "s32i a4, a2, 0x4\n"
9         ::: "a2", "a3", "a4", "memory"
10    );
11 }
12
13 // Inline assembly function to turn off light
14 void lightOffAssembly() {
15     asm volatile (
16         "movi a2, 0x3FF44000\n"
17         "movi a3, ~(1 << 5)\n"
18         "l32i a4, a2, 0x4\n"
19         "and a4, a4, a3\n"
20         "s32i a4, a2, 0x4\n"
21         ::: "a2", "a3", "a4", "memory"
22    );
23 }
```

Figure 25 - Assembly Code

The Assembly code begins by loading the base address of the GPIO control registers into a register, which provides the starting point for accessing the GPIO registers on the ESP32. Then, a bitmask is created for GPIO5, targeting pin 5 on the ESP32. The current state of the GPIO output register is read, and depending on the desired action, the corresponding bit for GPIO5 is set or cleared. Specifically, in the **lightOnAssembly** function, a bitwise OR operation is performed to set GPIO5 to high (turning on the LED), and in the **lightOffAssembly** function, a bitwise AND operation is used to clear the bit and turn off the LED. Finally, the updated value of the GPIO output register is stored, applying the change to the GPIO pin.

3.5. Real Time Scheduling

To address our real-time scheduling requirements, we began by analysing the problem and dividing it into distinct subsystems: the Room Control Unit, the Arduino for sensors, and the

Arduino for actuators, as illustrated in Figure 2.

To maximize the efficient use of computing time, we researched various scheduling algorithms to determine which would best suit our needs. We concluded that pre-emptive Fixed Priority Scheduling (FPS) is the most appropriate approach because it ensures tasks are executed in a specific order, which aligns with our system's requirements.

Room Control Unit:

The Room Control Unit manages the following tasks:

- **Verify_Data_Sensors:** This periodic task continuously receives data from the sensors. The data collected by this task serves as input for other system operations.
- **Generate_Commands:** This aperiodic task executes only when an action is required for the actuators. For instance, if the temperature exceeds the predefined threshold, this task will send a command to turn off the heater (LED). Additionally, a sporadic server will need to be implemented to handle command generation and transmission, ensuring a given bandwidth is allocated for efficient communication with the actuators.
- **MQTT_Data:** This periodic task still to be implemented sends data to the interface via MQTT at regular intervals. It relies on shared resources, such as temperature and luminosity data, which must first be updated by the **Verify_Data_Sensors** task. To ensure **MQTT_Data** uses the most recent data, a mutex will be implemented to manage access to these shared resources, preventing it from transmitting outdated information.
- **Sensor_Verification:** This periodic task will be implemented to periodically verify the proper functioning of the sensors. For example, the DHT11 sensor will read the temperature and compare it with the previously recorded value. If the temperature shows a significant variation, it may indicate a malfunction in the sensor. Similarly, if the sensor stops providing readings altogether, it may also be broken, prompting the need for a technician to inspect and verify the component.

Arduino (Sensors):

This Arduino is responsible for managing the following tasks:

- **Read_Data_Sensors:** This periodic task continuously reads data from its respective components. For example, the LDR reads luminosity, and the DHT11 reads temperature. The collected data is then stored in variables for further use.
- **Send_Data_Sensors:** This periodic task continuously sends the sensor data to the Room Control Unit. To ensure that the data sent is up-to-date and consistent, a mutex will be

implemented, preventing outdated or unordered data from being transmitted.

Arduino (Actuators):

This Arduino is responsible for managing the following tasks:

- **Receive_Commands:** this Aperiodic task will only be executed when the Room Control Unit sends commands.

Task Sets:

	C(ms)	D(ms)	T(ms)	Priority
Verify_Data_Sensors	10	35	35	4
MQTT_Data	10	70	70	2
Sensor_Verification	15	100	100	1

The server will have priority 3 to be able to send the command as soon as possible. In order for the task set to schedulable, and at the same time the server be able to execute when needed with the right priority, we gave the server a bandwidth of 15%. So, for this sporadic server we gave it a period of 40, and a capacity of 6.

	C(ms)	D(ms)	T(ms)	Priority
Read_Data_Sensors	10	20	20	2
Send_Data_Sensors	7	30	30	1