Service Architecture Project

Introduction

For our Service Architecture course at INSA Toulouse, we implement a Proof-of-Concept for managing INSA's room. This application must have multiple functionalities: automatic closing windows, turning on heating, turning off lights …

By using RESTful web service, the application retrieves data from sensors like temperature or movement detectors and according to the values of the retrieved data, actions on actuators can be triggered.

I: Using Agile methodology with Jira

For this project, we want to apply the Agile Scrum methodology and we used Jira for its intuitive user interface and also because it's free.

- **Product backlog**

Our Scrum workflow starts with Product Backlog where the product owner (in this case, it's our professor and us) builds the ideas and features that could go into our project. We implement different feature sets as different user stories with the format: "As a …, i need …, so that…". We also specify each user story's priority.

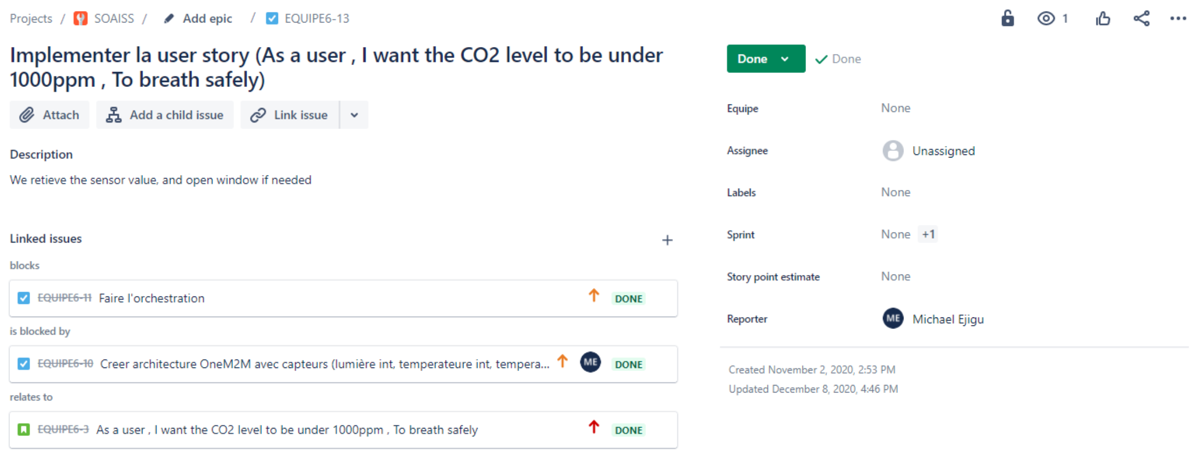
- **Sprint planning**

This is where we discuss the top priority user story and determine what can go to the next sprint. By grouping tasks in sprints we have an easier common goal to achieve.

- **Sprint Backlog**

Sprint backlog is a list of user stories that have been committed to the next sprint. We want to make sure that the entire team has a solid understanding of what each of the user stories involves, based on the discussion from the sprint planning.

We also associate tasks to members of our team, so we create tasks which need to be done. Each task groups different top items in our user stories, the criteria is that every user story that has similar functionality needs to be associated to the same task. One user story can be related to one or multiple tasks and one task can be related to one or multiple user stories.



Implementer la user story (As a user , I want the CO2 level to be under 1000ppm , To breath safely)

Description

We retieve the sensor value, and open window if needed

Linked issues

blocks

~~EQUIPE6-11~~ Faire l'orchestration DONE

is blocked by

~~EQUIPE6-10~~ Creer architecture OneM2M avec capteurs (lumière int, temperateure int, tempera... ME DONE

relates to

~~EQUIPE6-3~~ As a user , I want the CO2 level to be under 1000ppm , To breath safely DONE

Equipe	None
Assignee	Unassigned
Labels	None
Sprint	None +1
Story point estimate	None
Reporter	Michael Ejigu

Created November 2, 2020, 2:53 PM
Updated December 8, 2020, 4:46 PM

Figure I-1: Implement of Sprint backlog using User Story and Task

- **Sprint**

We finish our sprint table. Each sprint table has a determined time frame for each grouped user story. The sprint duration varies with the difficulty of the tasks to achieve. That's how we plan and finish each Sprint. We repeat this workflow for each sprint until we accomplish the final product.

- **Git**

For version control, we use Git as every member of our team already had experience with this tool.

II: Developing our OM2M architecture

To model our system, we have designed an architecture using OM2M. The Infrastructure Node (IN) represents the work office in which is located a Middle Node (MN) representing the room where we are deploying our smart system. We have 13 devices in the MN (a button to switch on/off the smart system, 6 sensors and 6 actuators) which are all represented in OM2M by Application Entities (AEs). Each of these AEs contains a Container (CNT) named "DATA" in which are stored Content Instances (CINs). Each time a sensor (or actuator) value (or status) changes, a new CIN appears in the CNT "DATA". It is important to notice that the latest CIN can be retrieved easily from OM2M as it is linked with the attribute "la" (stands for "last").

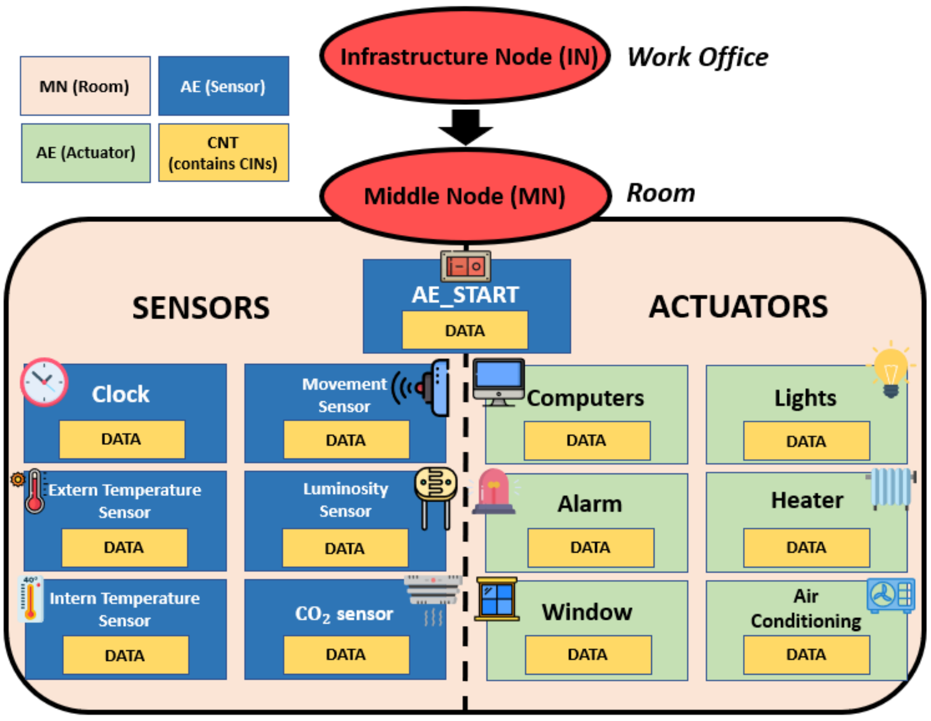


Figure II-1: Diagram showing our OM2M architecture

III: Developing some useful methods *(RoomManagementUsefulMethods.java)*

Once we have set up our OM2M architecture, we must now interact with it.

To test our OM2M system we begin by using POSTMAN to create requests to retrieve and post values on our sensor and actuator architecture. For the input and output format we decide to go with Json for its simplicity.

We then develop some java methods to be able to do the same thing. We use the open source library Jackson to simplify this process.

- ***retrieveOm2mData***

It takes an URL (identifying the sensor or actuator on our system) as input and returns a Json Object which is the content of the response of our OM2M architecture.
For example if we wish to get the latest state of the window (1 for open or 0 for closed), we call the following method as such:

retrieveOm2mData("http://localhost:8282/mn-name/Window/DATA/la")

- ***setOm2mValue***

It takes three inputs: URL, and 2 strings (the variable type and the content value).
For example to update the state of the Window as closed, we proceed as such:

setOm2mValue("http://localhost:8282/mn-name/Window/DATA", "Integer", "1")

Moreover to update our system instead of calling each of our methods, the method ***update()*** calls all our services.

Finally the method ***getAllValues()*** is able to print out in the Console all the latest values of all the sensors and actuators.

IV: Services Implementation

According to our user demands, we have implemented 6 services using the methods presented in the previous part to set and retrieve values from OM2M. It is important to note that the 6 services of the smart system will be operational only if the AE_START button is switched on (value = 1).

1. *Make sure the CO_2 concentration in the room is below the danger level*

If the value measured by the CO_2 sensor exceeds 1000ppm, the air quality is not good enough in the room. If the window is not open yet, it is automatically opened to purify the air.

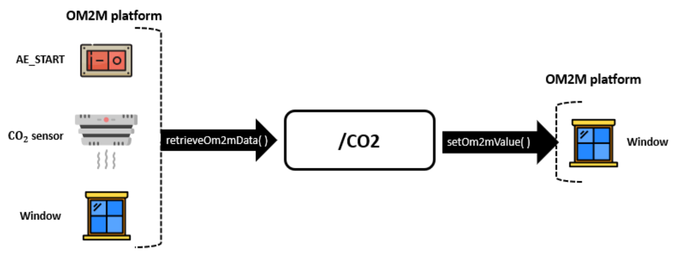


Figure IV-1: Diagram showing /CO2 service

2. *Disconnect the computers during lunch time*

For the workers in this room, it has been decided that lunch time was everyday between 12:00pm and 1:00pm. In this time period, they are not supposed to work. Thus, if their computer is on, it is automatically switch off and they can not switch it on until the end of lunch time. To simulate a clock, we have attributed an integer between 0000 and 2400 with the first two digits corresponding to the hour and the last two digits corresponding to minutes (in the French hour system).

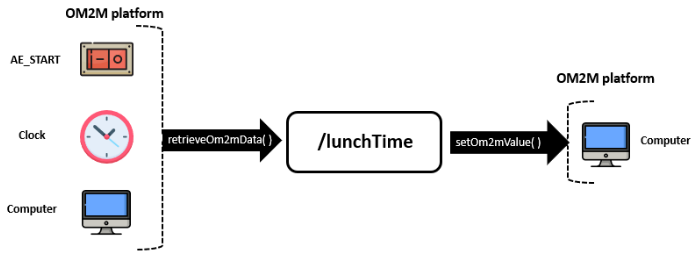


Figure IV-2: Diagram showing /lunchTime service

3. *Secure the room from intrusion during night*

Each night (between 11:00pm and 7:00am), a security system is started to prevent the room from intrusions. In that time period, when the movement sensor detects a movement in the room, the alarm rings (if it is not ringing yet). If no more movement is detected or if we are no more in this time period, the alarm stops.

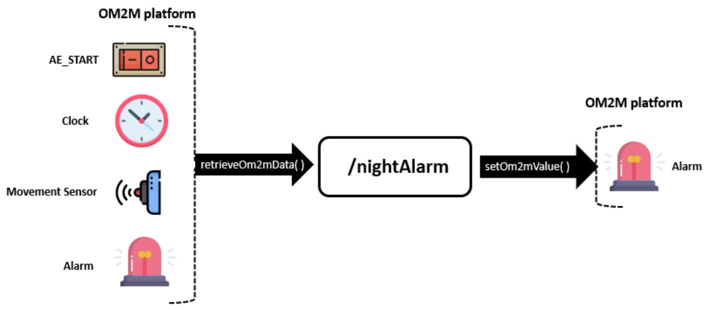


Figure IV-3: Diagram showing /nightAlarm service

4. *Switch off the lights out of the working hours*

The aim of this functionality is to save energy (and money) by switching off the lights in the room when nobody is supposed to work. It means that, between 11:00pm and 7:00am (night) and between 12:00pm and 1:00pm (lunch time), the lights are switched off (if they are not off yet).

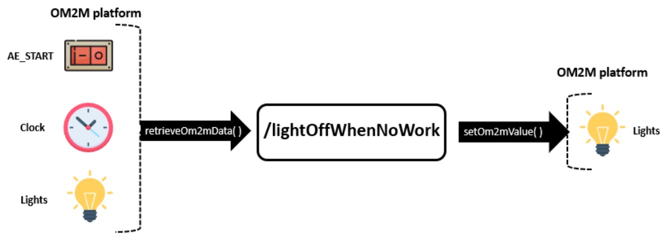


Figure IV-4: Diagram showing /lightOffWhenNoWork service

5. *Optimize the lighting level when the external light is enough*

The aim of this functionality is to save energy (and money) by switching off the lights when the external luminosity is enough. The luminosity sensor measures the luminosity coming into the room. If this luminosity is higher than 500lux and if the lights are already on, we automatically switch them off. This functionality is active only during working hours (7:00am – 12:00pm and 1:00pm – 11:00pm) to avoid conflicting with the /lightOffWhenNoWork service.

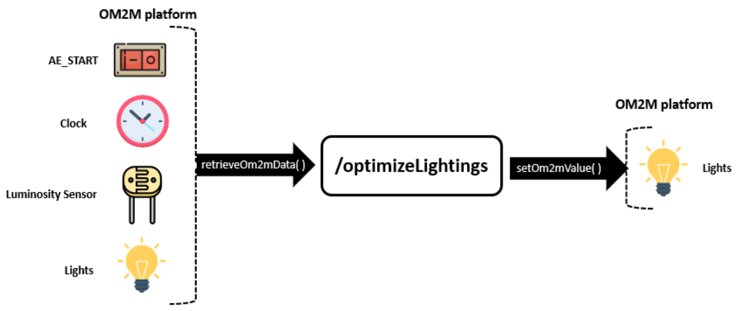


Figure IV-5: Diagram showing /optimizeLightings service

6. *Regulate the temperature to reduce the thermal losses*

This functionality is the most complex of our smart system and is used to regulate the temperature in the office room. It involves Window, Air Conditioning and Heater actuators.

However, as the CO_2 detection is the priority, it will have the priority over regulating temperature by opening the window if needed.

We have different cases:

- If the temperature is cold in the room (below 15°C), we have two solutions:
  - If external temperature is higher than 15°C, then we open the window (and make sure the Air Conditioning and the Heater are off)
  - If external temperature is lower than 15°C, then we put the Heater on (and make sure the Air Conditioning is off and the window is closed)
- If the temperature is hot in the room (higher than 25°C), we have two solutions:
  - If the external temperature is higher than 25°C, then we put the Air Conditioning on (and make sure the Heater is off and the window is closed)
  - If the external temperature is lower than 25°C, then we open the window (and make sure the Air Conditioning and the Heater are off)

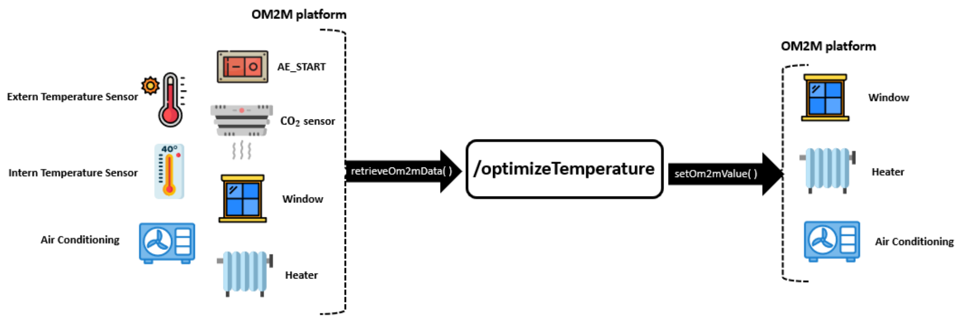


Figure IV-6: Diagram showing /optimizeTemperature service

V: Use Scenario

To demonstrate that our system works, the main method in our RoomManagementUsefulMethods narrates different scenarios.

One by one to test our services, we simulate the sensor values, tell our system to update itself and print out all the states of all sensors and actuators.

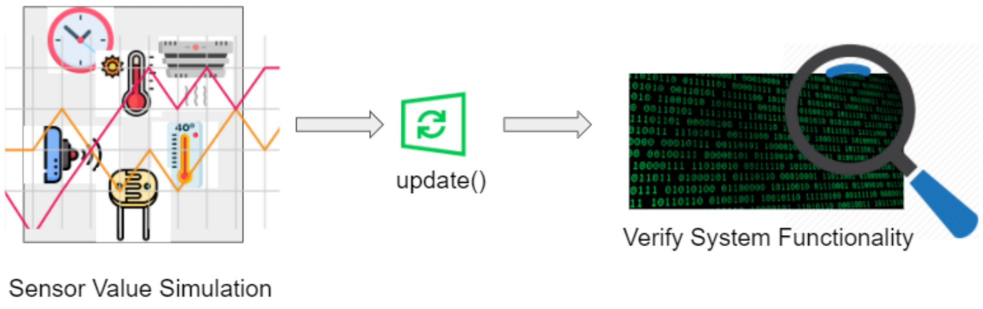
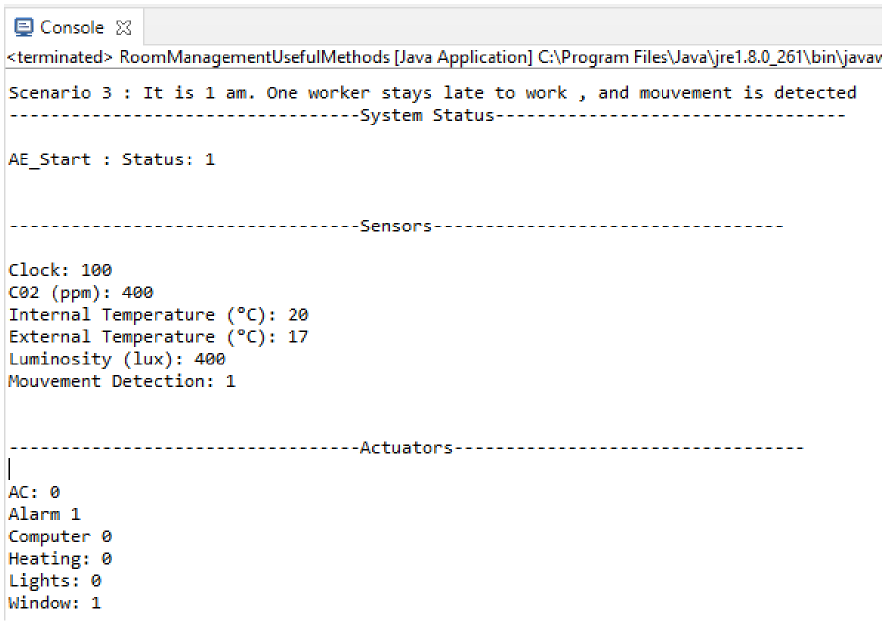


Figure V-1: Diagram showing how we call our list of services

For example to test our night alarm functionality (Scenario 3), we set the Clock to 1 am and the Movement Detector to 1, our alarm turns on. You will find the screenshot of the output for scenario 3 in Figure V-2 below:



```
<terminated> RoomManagementUsefulMethods [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javav
Scenario 3 : It is 1 am. One worker stays late to work , and mouvement is detected
---------------------------------System Status---------------------------------

AE_Start : Status: 1


---------------------------------Sensors---------------------------------

Clock: 100
C02 (ppm): 400
Internal Temperature (°C): 20
External Temperature (°C): 17
Luminosity (lux): 400
Mouvement Detection: 1


---------------------------------Actuators---------------------------------

AC: 0
Alarm 1
Computer 0
Heating: 0
Lights: 0
Window: 1
```

Figure V-2: Screenshot of console output for test of scenario 3

Conclusion

Our main objective was achieved. We have managed to develop services that can orchestrate the automation of a working room. We used Java to develop the services and OM2M to organise our resources. We used Git for version control and Jira for agile project management. To improve this project we can develop a UI to monitor all actuator values and history. We can also find a better way to simulate sensors.