SOA Report

Léo Picou - Victor Maupu - Mohanprabhu Selvaraj

January 2020

1 Introduction

We made this project for the SOA course. The purpose of this project was to develop an OM2M architecture and an application to manage the GEI building.

In order to change from the traditional building management where the sensors are always the same (temperature, luminosity, etc...), we decided to have some fun and to develop a highly critical building management application, with highly sensitive actuators and sensors.

You can find all our work hosted on Gitlab here `https://gitlab.com/victor.maupu/geiproject`.

2 Abstract architecture

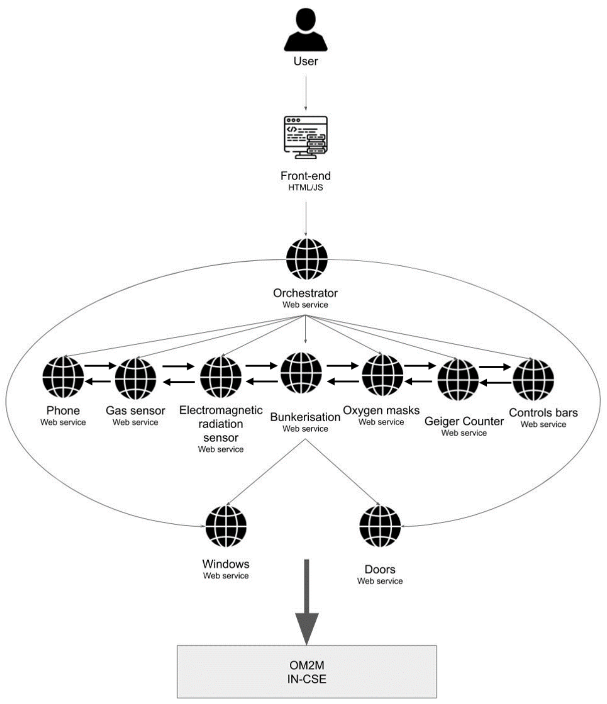


Figure 1: Architecture

Simplified user stories :

- **Too much toxicity ?**
  Bunkerize the GEI and call the emergencies, don't let dangerous gaz particules escape from the building !

- **Too much electromagnetic field ?**
  Call the emergencies and play the Ghostbuster jingle !

- **Too much radioactivity ?**
  It means that the nuclear reactor of the GEI is melting, needs to make the control rods fall in order to stop the nuclear reaction ! Also call the emergencies.

- **It's night/daytime ?**
  Close/open windows and doors !

Since everything is virtual on OM2M, we had some fun implementing a completely virtual environment. The user stories has been simplified in order to improve the readability of the report, but there are more that are actually nested in the ones above.

3 User interface

The user interface is the interface between our system and users. We tried to develop a user-friendly interface that should make easy to control the system. Data from our system are fetch with asynchronous GET requests in Javascript and are displayed in a table. We can modify the state of an element (door, window) with buttons next to current state.

We hosted on Github Pages an artificial static version of our interface, only the 'Update' and 'Print Graph' buttons works (other buttons can't work since we can't host java micro-services on github). It allows you to browse through a live version and marvel at our interactive plots : `https://lpicou.github.io/SOA_interface/`. Screenshot of the interface can be also found in annexes, figure 6 and 7.

There is also six general buttons at the head of the page :

- **Call 911**
  Just call the police
- **Update**
  Remove data from the table and fetch them again to update values
- **Print Graph**
  Print some graph at the end of the page
- **Disable CB**
  Disable controls bars
- **Unbunkerisation**
  Disable bunkerisation of GEI
- **Test SMS**
  Test SMS function. Please do not abuse of this function because real SMS is send.

4 OM2M architecture

Our OM2M architecture is as follow :

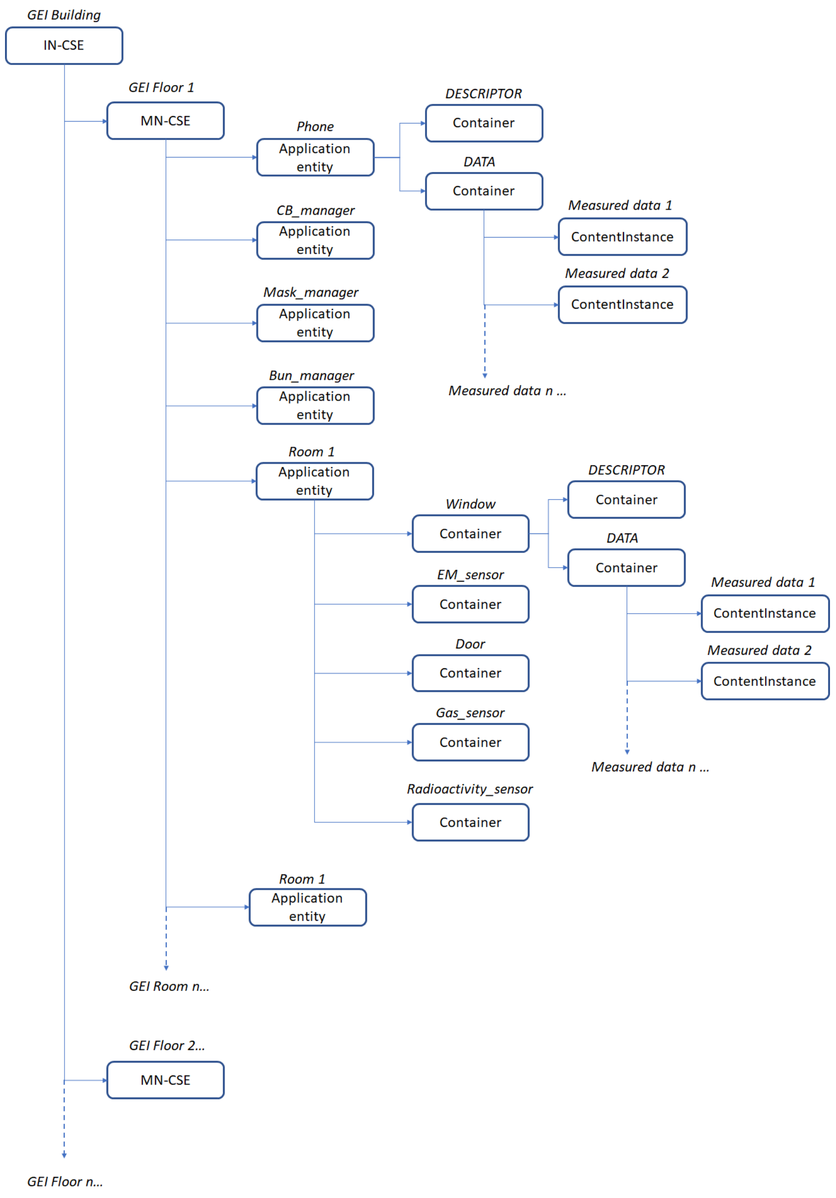


Figure 2: Architecture

We managed to create a randomly iniated OM2M database, which means that it creates for example 10 rooms, and fill each sensors and each actuators of each rooms with random data. We took some time to implement this random initialization, but it allowed us to have the really nice data plots that you can see in annex 7. Furthermore, it led to increased development speed, since we didn't have to delete/create the database with Postman, just run our Java code to entirely automate the process.

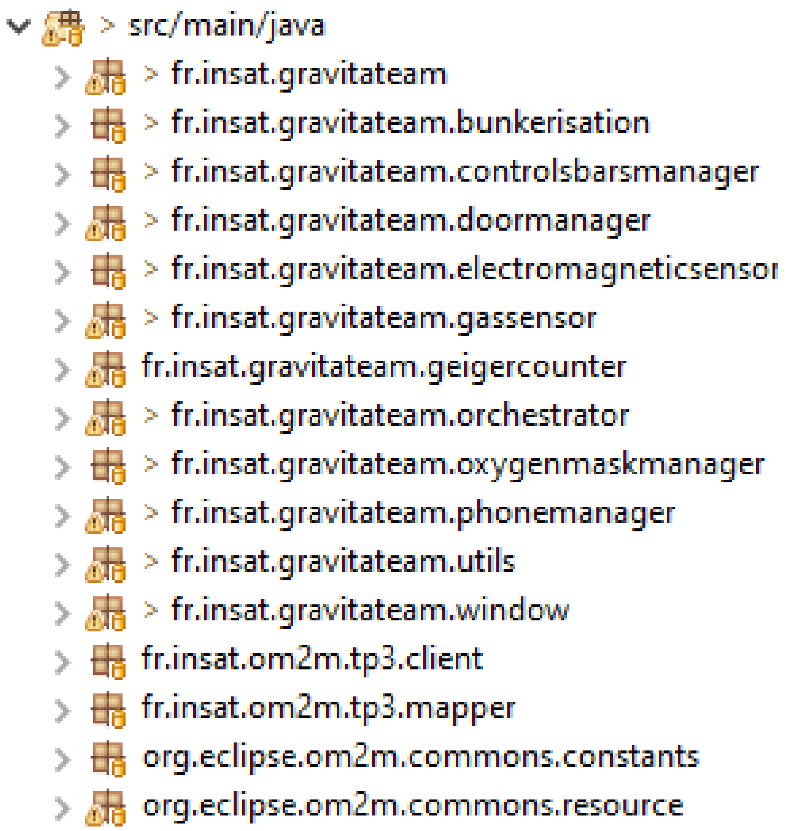
Each contentInstance holds 3 generics fields :

- **location** Room 2, or GEI for example
- **unit** Celsius, Watt, state (for actuators)
- **value** A number, a boolean state (open/close, bunkerized/not_bunkerized..)

Each actuators also have a POA, to be able to enable/disable it from outside. To be sure that our requests have been correctly forwarded by OM2M to the POA, we implemented a simple Flask server which listen on the POA and answer back when it receives a request. With this setup, we were able to know exactly the route of our request through the network.

5 Java architecture

In order to increase our development speed, we decided to make only one Spring-Boot application but separated into **different** packages. For each micro-service, we have one package :



```
src/main/java
    fr.insat.gravitateam
    fr.insat.gravitateam.bunkerisation
    fr.insat.gravitateam.controlsbarsmanager
    fr.insat.gravitateam.doormanager
    fr.insat.gravitateam.electromagneticsensor
    fr.insat.gravitateam.gassensor
    fr.insat.gravitateam.geigercounter
    fr.insat.gravitateam.orchestrator
    fr.insat.gravitateam.oxygenmaskmanager
    fr.insat.gravitateam.phonemanager
    fr.insat.gravitateam.utils
    fr.insat.gravitateam.window
    fr.insat.om2m.tp3.client
    fr.insat.om2m.tp3.mapper
    org.eclipse.om2m.commons.constants
    org.eclipse.om2m.commons.resource
```

Figure 3: Java packages

This way, we can easily start the all the micro-services very easily. We also decided to use this development configuration to avoid having 15 different eclipse project with each time the same libraries which are used nearly everywhere (org.eclipse.om2m.commons.resource, org.eclipse.om2m.commons.constants, Obix/Marshal mapper and Om2m client).

We completely understood the concepts of micro-services, that's why **there are no** Java calls between each micro-services, only HTTP requests. If for example the Orchestrator decides to close the GEI's windows because it's 22:00, it makes a GET request on http://IP:PORT/windows/all/close . It does not call the window package by doing a sort of 'Window.close()'. All the micro-services are deployed therefore on the same local IP, but could be without **any** problems deployed in a separate SpringBoot application, actually we just have to copy/paste the package and it's ready to use.

5.1 Routine of orchestrator

At the launch of the application, we decided to make two orchestrator threads :

- One for managing sensors. This thread loop over each room, and check each sensor values. Depending if the value is above a certain threshold, it can either :
  - **Threshold 1** : Send a simple warning SMS by using our phone micro-service.
  - **Threshold 2** : Call emergencies by using our phone micro-service. It will moreover call a special actuator micro-service which match the situation, for example bunkerization if there is too much toxicity, or using bar manager if there is too much radioactivity.
- One for managing actuators. This one checks in a regular basis the time, and close the windows and the doors accordingly.

Since these two threads have a frequency of checking really different, it made sense to separate them. We understand that from the orchestrator, each micro-service of our application is called.

6 Method of development

We used the agile method for this project. It means that we divided this project into different functionality that we developed one after the other. Because it was our first project with this method, we did not used this method as much as we should. We were not used to so it was not easy for us to respect the due time of sprint for example. Furthermore, it was a little project with only three members so we did not feel the need to use this method fully even if it would probably have optimized our time and our organization.

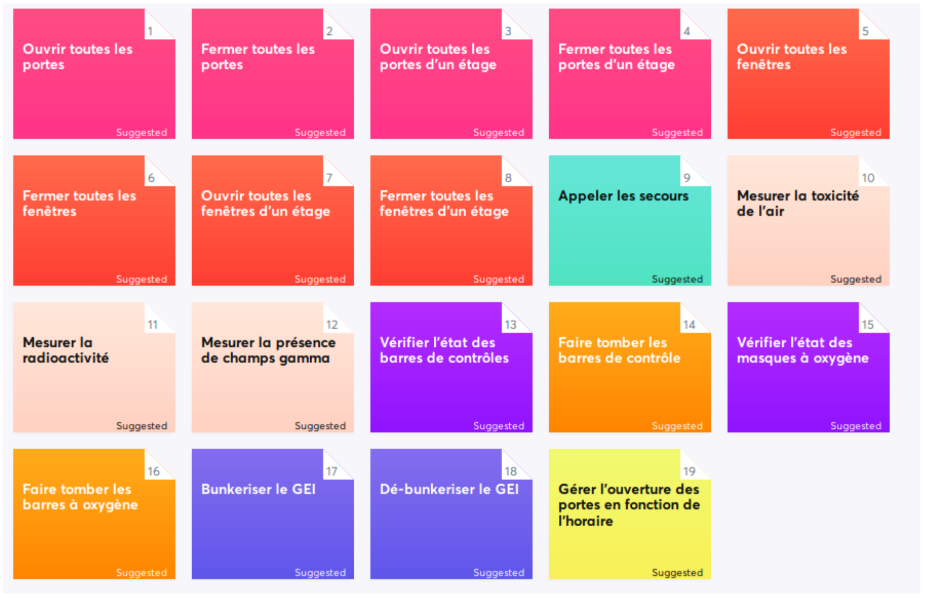
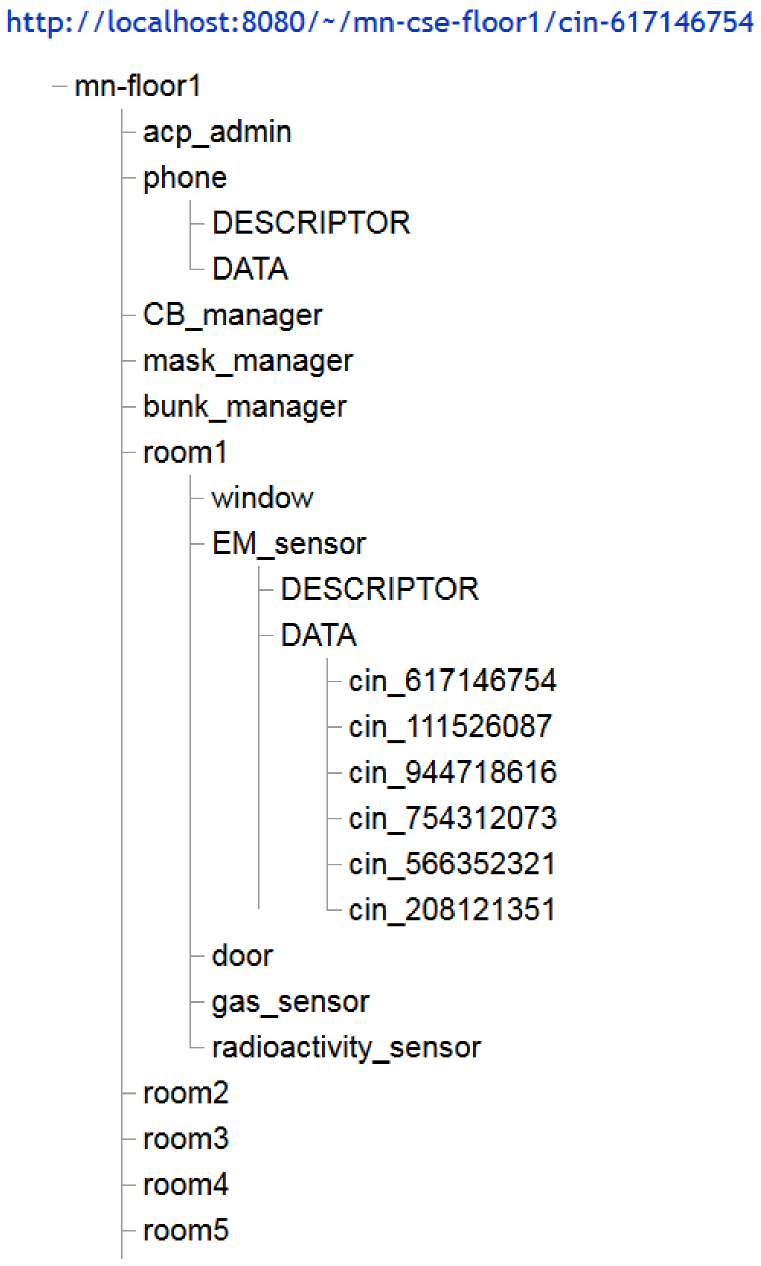

1 Ouvrir toutes les portes Suggested
2 Fermer toutes les portes Suggested
3 Ouvrir toutes les portes d'un étage Suggested
4 Fermer toutes les portes d'un étage Suggested
5 Ouvrir toutes les fenêtres Suggested
6 Fermer toutes les fenêtres Suggested
7 Ouvrir toutes les fenêtres d'un étage Suggested
8 Fermer toutes les fenêtres d'un étage Suggested
9 Appeler les secours Suggested
10 Mesurer la toxicité de l'air Suggested
11 Mesurer la radioactivité Suggested
12 Mesurer la présence de champs gamma Suggested
13 Vérifier l'état des barres de contrôles Suggested
14 Faire tomber les barres de contrôle Suggested
15 Vérifier l'état des masques à oxygène Suggested
16 Faire tomber les barres à oxygène Suggested
17 Bunkeriser le GEI Suggested
18 Dé-bunkeriser le GEI Suggested
19 Gérer l'ouverture des portes en fonction de l'horaire Suggested

Figure 4: Stories

7 Conclusion

This project allowed us to acquire a good understanding of web services and OM2M architecture. We really liked to use SpringBoot to deploy micro-services because it's really fast to deploy, and it abstracts all the intermediates steps of servlet programming which can sometimes be harsh. The main difficulty of this project was to schedule the three parts of the projects : the OM2M architecture, web services and the web front-end. The organisation with agile method makes easier for us to organize who has to do what.

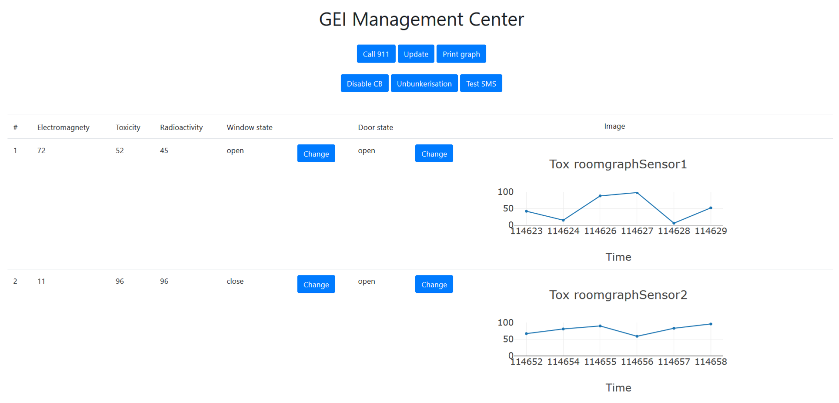
8 Annexes



http://localhost:8080/~/mn-cse-floor1/cin-617146754

```
- mn-floor1
    ├ acp_admin
    ├ phone
    │   ├ DESCRIPTOR
    │   └ DATA
    ├ CB_manager
    ├ mask_manager
    ├ bunk_manager
    ├ room1
    │   ├ window
    │   ├ EM_sensor
    │   │   ├ DESCRIPTOR
    │   │   ├ DATA
    │   │   │   ├ cin_617146754
    │   │   │   ├ cin_111526087
    │   │   │   ├ cin_944718616
    │   │   │   ├ cin_754312073
    │   │   │   ├ cin_566352321
    │   │   │   └ cin_208121351
    │   ├ door
    │   ├ gas_sensor
    │   └ radioactivity_sensor
    ├ room2
    ├ room3
    ├ room4
    ├ room5
```

Figure 5: Screen from OM2M web client



GEI Management Center

#	Electromagnety	Toxicity	Radioactivity	Window state		Door state		Image
1	72	52	45	open	Change	open	Change	
2	11	96	96	close	Change	open	Change	

Figure 6: Main Interface

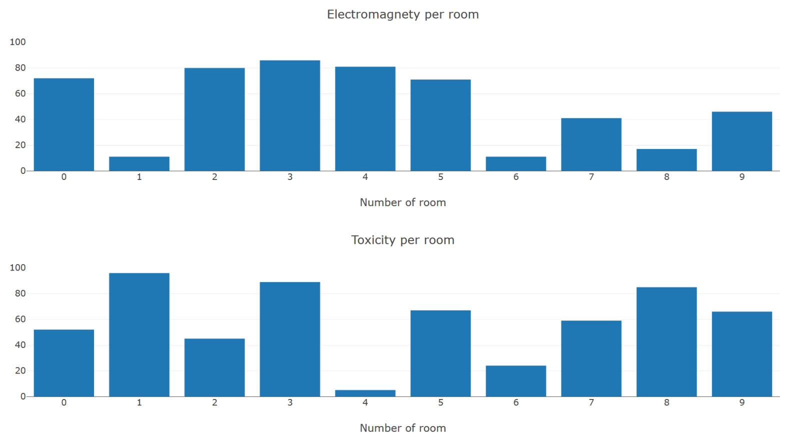


Figure 7: Plots of various data in the front-end