INNOVATIVE SMART SYSTEMS

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# SERVICE-ORIENTED ARCHITECTURE

*GEI ROOMS MONITORING*



Sophie Rougeaux
Agathe Limouzy
Elie Taillardat
Jonathan Malique

PTP ISS
Group A1
27/01/2019

# SUMMARY

# INTRODUCTION

During our last semester at INSA, in ISS, we are studying service-oriented architecture (SOA). In this architecture, services are provided to components by application components. The project linked to this module is about managing INSA GEI class rooms.

For this project, we first defined some scenarios depending on what we want to control and know in each room (scenarios could be about turning on lights, alarm or everything). Also, to retrieve informations and modify it in the room, we needed some sensors and actuators. After, we had to implement those scenarios using several components: web services (REST and SOAP), web interface (for the user) and OM2M (to store resources).

This report presents the choices we have made about the scenarios, how we managed to create this service-oriented architecture and the organization we have choosen with the IceScrum software to plan it.

# I. SCENARIOS AND SPECIFICATIONS

## A. Scenarios

We could imagine everything, numerous sensors and actuators, with numerous data and resources. Each room has two main informations: the opening hours and the closing hours. So, we have decided o created 3 different scenarios:

### 1. The Heating regulation

In this case, if the interior temperature is below the threshold control temperature and if the presence sensor says that there is someone inside the room (during the opening hours), the heating actuator will turn on the heater.

### 2. The Night regulation

We are watching if the current time is inside the opening hours or not, if not we will declare a Boolean isNight as true. Moreover, if it is the night and that the presence sensor feels someone, the alarm actuator will turn on the alarm.

During the night, the light will be turned off by the lamp actuator and the doors will be closed.

### 3. The Light regulation

The system must watch if the illuminance of the room is inferior to the illuminance threshold, and if the presence sensor feels someone inside the room, the light has to be turned on.

## B. Sensors and actuators of our architecture

For every room that we can define in OM2M (right now only reading room) we have:

- The exterior temperature
- The Interior temperature
- A presence sensor
- An illuminance sensor and threshold control
- A heater actuator and threshold control
- A lamp actuator
- An alarm actuator
- A doors actuator
- Opening hours and closing hours control

# II. CHOSEN ARCHITECTURE

To implement those scenarios with the different sensors and actuators, we managed to think about a great architecture. Here is this architecture:

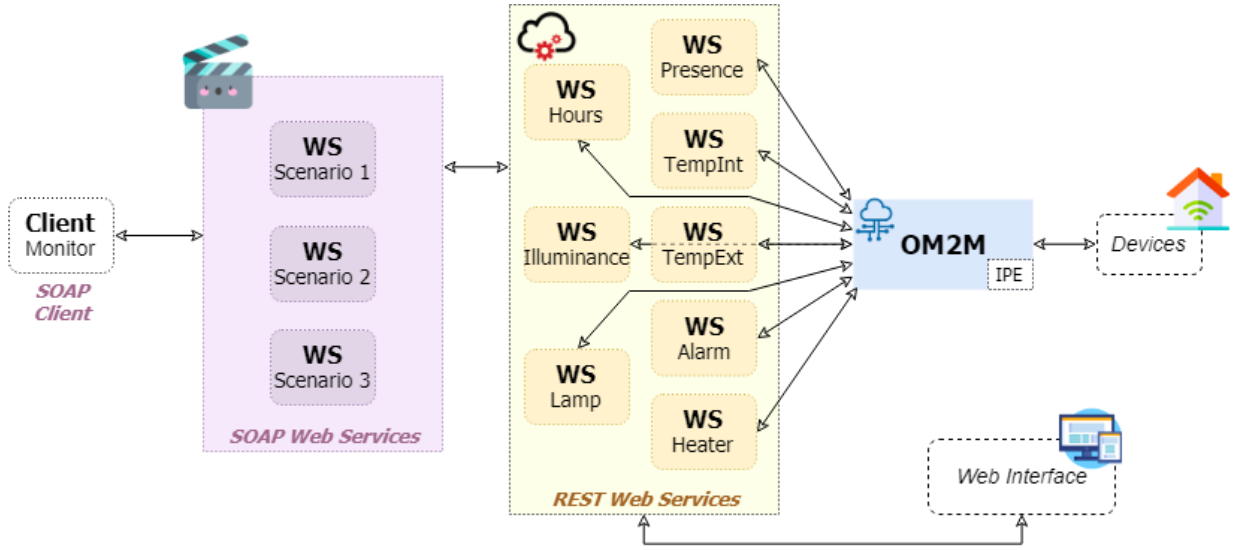



*Figure 1 – Architecture for GEI rooms monitoring*

## A. OM2M

We have choosen to use this middleware to store and access to all the data from sensors and actuators. It is a useful standard for IoT, using REST architecture.

We also have created a **plugin** to easily create a new resources allocation for a new room.

## B. Web interface

This interface manages to retrieve data from OM2M database using REST Web Services. We can display all data that we want to show to the user. The user can also send data to OM2M using the web service (for example the hours of alarm turned on).

## C. Web services REST

The web services give us the opportunity to write data in OM2M in a simpler way. In fact, we have created a Java class for each sensor and actuator that allow to send GET or POST requests to OM2M automatically.

## D. Web Services SOAP - Scenarios

This part of the architecture is useful to take care about the scenarios defined before. In fact, the SOAP web services will manage data collected by REST web services to verify and test each scenario and send some new data to OM2M (via the REST Web Services) if there is

something to change depending on the scenarios. We chose the SOAP protocol because it is better for calling services one after another, like we want to do.

## E. SOAP Client – Monitor

The Monitor App is a client which has to call Scenarios - SOAP Web Services. It is running a loop (each few seconds) and calls the SOAP Web Services to launch each scenario regularly.

# III. CREATED ARCHITECTURE

## A. OM2M

The OM2M architecture allows to define each room of INSA with **REST architecture**.

The plugin created and integrated to OM2M gives us the opportunity to create an entire Room in the OM2M architecture. When we are launching this plugin, one room is created, and all the necessary resources are allocated.

You can find an example of resources allocated by the creation of a room (Reading Room) on the following screenshot of OM2M platform:

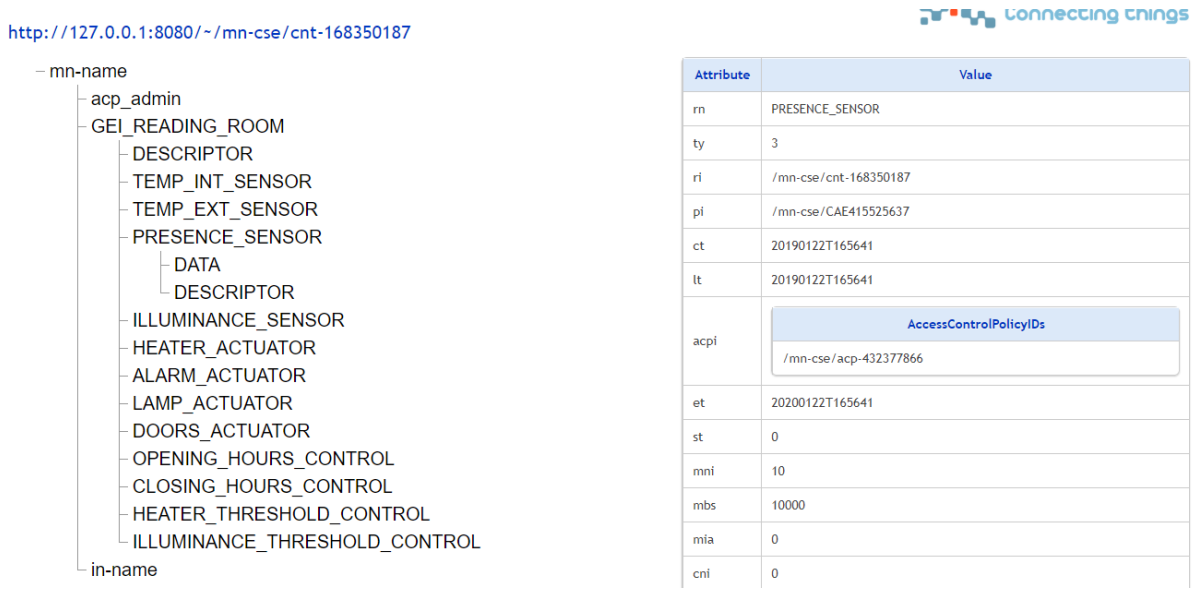



*Figure 2 - OM2M architecture with resources allocation*

For each room, we had to create Application Entities (AE), Containers (CNT) and Content Instances (CI) in DATA Containers.

## B. Web interface

The web interface that we created is coded with **AngularJS** and allows to retrieve the sensors values from the REST Web Services. It also allows the user to set different thresholds to adapt the scenarios : the temperature threshold when the heater must be set on, the illuminance threshold when the lamps must be switched on, and the opening and closing hours of the room for the night scenario.

This interface looks like this:

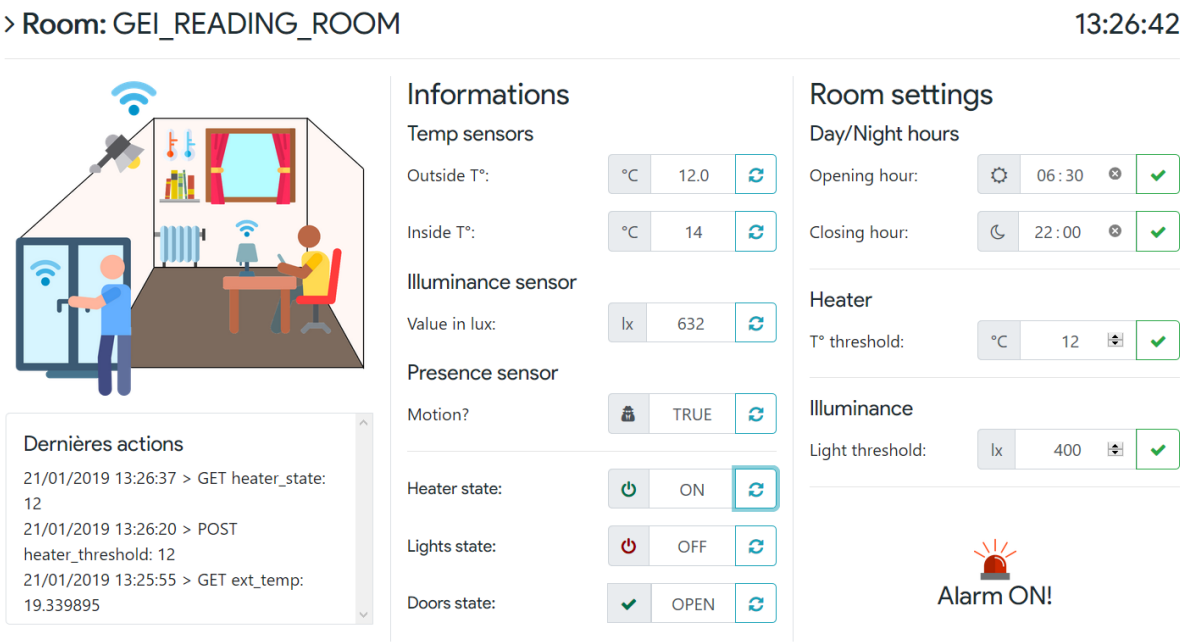



*Figure 3 – Web interface (for users)*

## C. Web services REST

We have created as much Web services as the sensors and actuators number. They can be accessed via the path :

http://<ip>:4200/REST_Smart_Rooms_Project/webapi/{roomId}/<name>

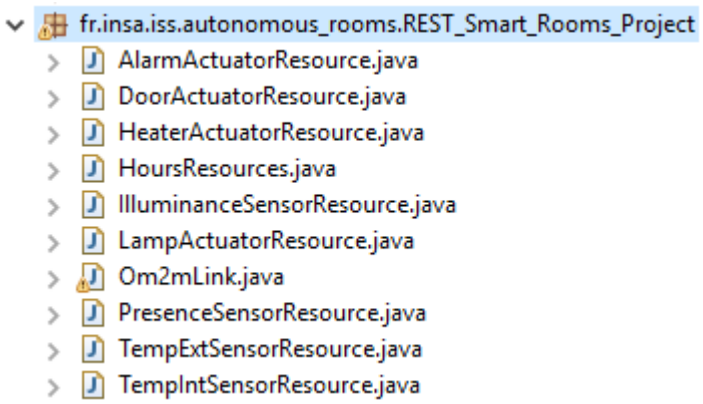



*Figure 4 : One REST Web Services per actuator/sensor*

These Web Services are clients of OM2M. We created an Om2mLink.java class to facilitate the requests to send to OM2M, using the Jersey client because we had a problem with the Java REST client at the end of the project.

With the sensors Web Services (Interior Temperature, Exterior Temperature, Illuminance, Presence), we answer to the GET requests. In our GET functions, we call a REST client to ask OM2M for the latest value.

With the actuator Web Services (Alarm, Door, Heater, Hours, Lamp), we also answer to POST requests to allow the user and the Monitor to change the states of the actuators and the threshold values.

An example of the HeaterActuator resource is shown on next page.

```
@Path("{roomId}/heater-actuator")
public class HeaterActuatorResource {
	private Om2mLink client = new Om2mLink();

	@GET
	@Produces(MediaType.TEXT_PLAIN)
    public String getHeaterStateObj(@PathParam("roomId") String roomId) throws
IOException {
		// get valeurs from OM2M
		String payload = client.retrieveHeater(roomId);
		//System.out.println(payload);
		// unmarshalling the notification
		Mapper map = new Mapper();
		ContentInstance cin = null;
		cin = (ContentInstance) map.unmarshal(payload);
		return String.valueOf(cin.getContent());
    }

	@GET
	@Path("/threshold")
	@Produces(MediaType.TEXT_PLAIN)
    public String getHeaterThreshold(@PathParam("roomId") String roomId) throws
IOException {
		String payload = client.retrieveHeaterThreshold(roomId);
		Mapper map = new Mapper();
		ContentInstance cin = null;
		cin = (ContentInstance) map.unmarshal(payload);
		return String.valueOf(cin.getContent());
    }

	@POST
	@Path("/threshold/{number}")
	@Consumes(MediaType.APPLICATION_XML)
    public void setHeaterThreshold(@PathParam("roomId") String roomId,
@PathParam("number") String threshold) throws IOException {
		client.postHeaterThreshold(roomId, "<m2m:cin
xmlns:m2m=\"http://www.onem2m.org/xml/protocols\"><cnf>message</cnf><con>" + threshold +
"</con></m2m:cin>");
		System.out.println("The threshold has been successfully changed to " +
threshold);
		return ;
    }

	@POST
	@Path("/trigger/{bool}")
	@Consumes(MediaType.APPLICATION_JSON)
    public void triggerHeater(@PathParam("roomId") String roomId, @PathParam("bool")
String state) throws IOException {
		client.postHeaterTrigger(roomId, "<m2m:cin
xmlns:m2m=\"http://www.onem2m.org/xml/protocols\"><cnf>message</cnf><con>" + state +
"</con></m2m:cin>");
		System.out.println("The heater has been triggered : " + state);
		return ;
    }
}
```

## D. Web services SOAP - Scenarios

To implement the 3 scenarios, we have deployed 3 different SOAP Web Services, one for each scenario.

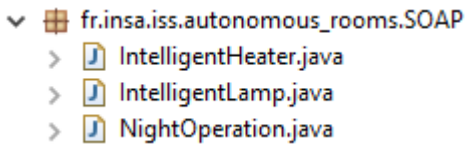



*Figure 5 : One SOAP Web Services per scenario*

Each Web Service has been deployed using the bottom-up approach, then tested with the Web Services Explorer tool in Eclipse.

The IntelligentHeater web service implements the first scenario. It uses a REST client to ask our REST Web Services for the sensors and threshold values. It plays the scenario and posts new values to our REST Web Services (that posts them to OM2M).

The NightOperation web service implements the second scenario and the IntelligentLamp implements the third scenario with the same philosophy.

The description of our Web Services :

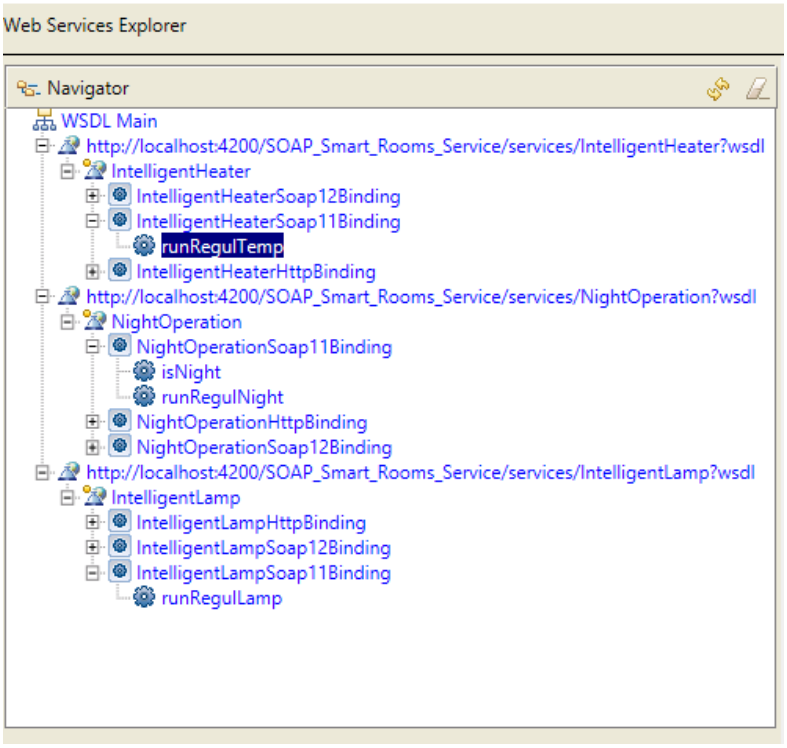



*Figure 6 : Deployed SOAP Web Services*

The example of the IntelligentLamp class is shown next page.

```
public class IntelligentLamp {
private static final String REST_WS_PATH =
"http://localhost:4200/REST_Smart_Rooms_Project/webapi/";

	@SuppressWarnings("resource")
	public static int runRegulLamp(String roomId) {
		// retrieve room illuminance
		Client client = ClientBuilder.newClient();
		Response response = client.target(REST_WS_PATH + roomId + "/illuminance-
sensor").request().get();
		Float illuminance = response.readEntity(Float.class);

		// retrieve illuminance threshold for the lamps
		response = client.target(REST_WS_PATH + roomId + "/illuminance-
sensor/threshold").request().get();
		Float illThreshold = response.readEntity(Float.class);

		// retrieve presence sensor
		response = client.target(REST_WS_PATH + roomId + "/presence-
sensor").request().get();
		Float presence = response.readEntity(Float.class);
		int pres = presence.intValue();

		System.out.println("Illuminance threshold : " + illThreshold);
		System.out.println("Interior illuminance : " + illuminance);
		System.out.println("Presence : " + pres);

		if((illuminance < illThreshold) && (pres == 1)){
			System.out.println("Illuminance regulation : switch on lights");
			response = client.target(REST_WS_PATH + roomId + "/lamp-
actuator/trigger/true").request().post(null);
		} else {
			System.out.println("Illuminance regulation : switch off lights");
			response = client.target(REST_WS_PATH + roomId + "/lamp-
actuator/trigger/false").request().post(null);
		}
		return response.getStatus();
	}
}
```

## E. SOAP Client – Monitor

To use our SOAP Web Services and launch each scenario, we created a monitoring app that is a SOAP client of our SOAP Web Services. When launched, the Monitor runs each scenario (Web Service) one after the other, in a loop with a three-seconds pause.

The Monitor class is shown next page.

```
public class Monitor {
	public static void main(String[] args) throws RemoteException,
InterruptedException {
		// instanciation des stub
		IntelligentHeaterStub heaterWS = new IntelligentHeaterStub();
		NightOperationStub nightWS = new NightOperationStub();
		IntelligentLampStub lampWS = new IntelligentLampStub();

		// instanciation des objets request pour construire les requêtes
		RunRegulTemp request = new RunRegulTemp();
		request.setRoomId("GEI_READING_ROOM");
		RunRegulNight requestNight = new RunRegulNight();
		requestNight.setRoomId("GEI_READING_ROOM");
		RunRegulLamp requestLamp = new RunRegulLamp();
		requestLamp.setRoomId("GEI_READING_ROOM");

		// invocation des méthodes
		while(true){
			RunRegulTempResponse response = heaterWS.runRegulTemp(request);
			System.out.println("Resultat de l'opération runRegulTemp : " +
response.get_return());
			RunRegulNightResponse responseNight =
nightWS.runRegulNight(requestNight);
			System.out.println("Resultat de l'opération runNightOperation : "
+ response.get_return());
			RunRegulLampResponse responseLamp =
lampWS.runRegulLamp(requestLamp);
			System.out.println("Resultat de l'opération runRegulLamp : " +
response.get_return());

			Thread.sleep(3000);
		}
	}
}
```

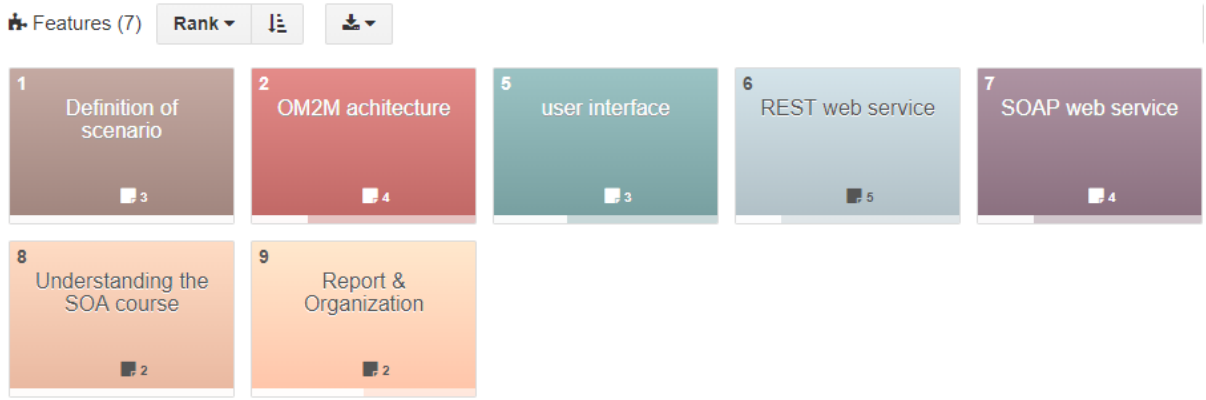
# IV. WORK ORGANIZATION WITH ICESCRUM

To plan our organization during this project, we have used a new tool, a new software: IceScrum. First, we have done the tutorial to understand what is a feature, a sprint, a story and a task.

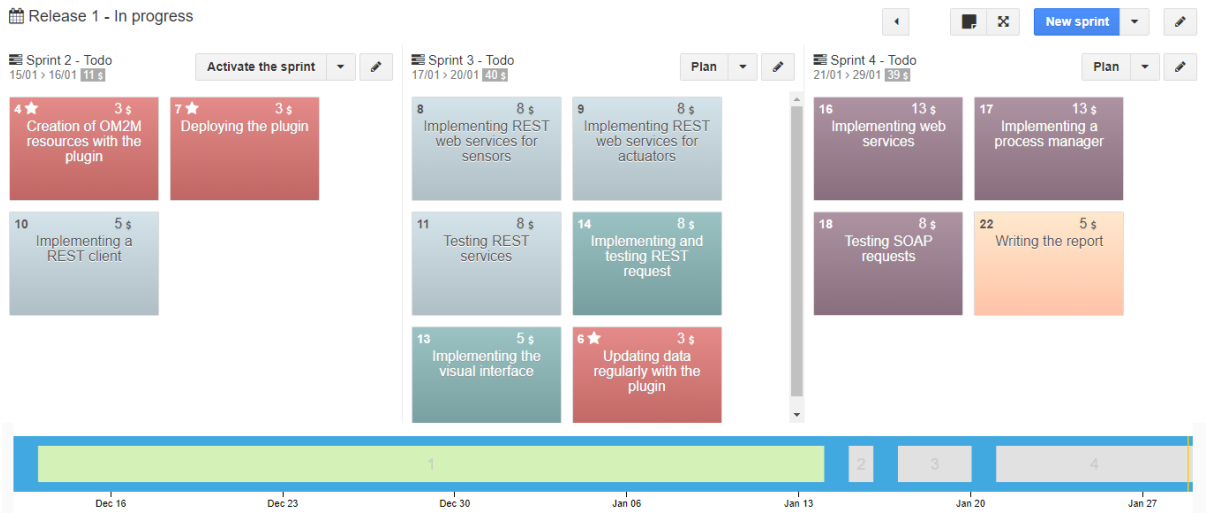
The tasks were separated as follow:

- Elie: developing the web interface for the user
- Sophie: OM2M set up and Web services creation
- Agathe: dealing with the IceScrum organization and OM2M set up
- Jonathan: writing the report and helping Sophie and web services creation

Our iceScrum features were:




And we defined our sprints according to the dedicated TP hours at INSA:




After discussing with our teachers, we realized that our stories should have been more scenario-oriented with the full architecture done for each scenario, and not keeping the SOAP part for the end like we did.

# CONCLUSION

We ended up with a working project that fulfills all the specifications: designing an SOA architecture, implementing services and services calls, implementing a Web Interface and using Java & OM2M. We also decided to implement REST and SOAP Web Services to have some experience in manipulating them, since it was relevant to have both of them.

To conclude, this project was really complete and challenging. We did not have any knowledge in service-oriented architecture before this module, and thanks to this project and the TDs exercises, we now have a better understanding of what is a Web Service, the difference between SOAP and REST and their interest, and how to create SOAP and REST services and clients.

*The project files can be found on github:*

*https://github.com/sophierougeaux/SOA-project-5iss*