MIDDLEWARE FOR THE INTERNET OF THINGS

TP REPORT – 5 ISS INSA TOULOUSE

This report covers all the TPs for Internet of Things, it is individual/by pair and will be integrated into your portfolio. The purpose of this report is to show the skills you acquired and the concepts you understood as a result of the courses and the various practical sessions.

The structure can be modified, as long as all the different concepts are addressed. If additional elements seem relevant for you to detail, you are free to integrate them. The size of this report should be limited to 10 pages maximum excluding annexes. (Do not hesitate to attach diagrams in appendix if it helps you to clarify your speech)

Lastname 1 : Erb

Firstname 1 : Vincent

Lastname 2 : Péfau

Firstname 2 : Marine

Group : 5 ISS – Groupe A1

TP teacher : Tanissia DJEIMAI

1. KNOW HOW TO POSITION THE MAIN STANDARDS OF THE INTERNET OF THINGS

Briefly explain the principle of the oneM2M standard and how it is positioned with respect to other existing standards and technologies.

OneM2M is a standard driven by a mixed consortium, composed by industrials and standardization bodies, and includes today more than 200 members. The purpose of oneM2M is to develop a global standard for the Machine to Machine and the IoT (Internet of Things). OneM2M is a service layer, a software Middleware sitting between processing / communication hardware and IoT applications providing a set of functions commonly needed by IoT applications. One of the particularities of oneM2M is the architecture which is generic and horizontal. That allows a homogeneous vision of the system regardless the application field, and then interoperability with common APIs to share data and information.

2. DEPLOY A STANDARD-COMPLIANT ARCHITECTURE AND IMPLEMENT A NETWORK OF SENSORS

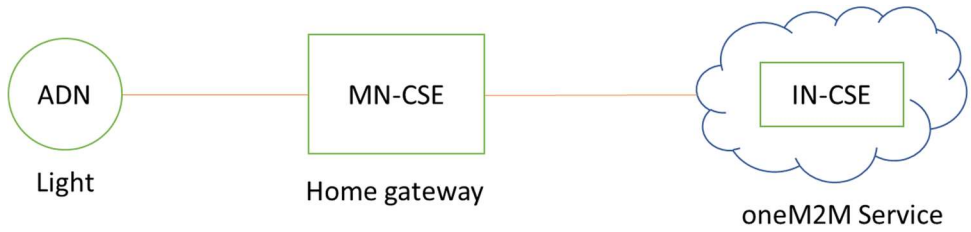
2.1. DEPLOY AND CONFIGURE AN IOT ARCHITECTURE USING OM2M

Explain how you deployed an IoT architecture with OM2M: the types of nodes used, the entities interacting with the middleware, the level at which the objects / sensors interact with the system as well as the different applications interacting with it, and so on. ***(TP 1 + 2 +3)***

OM2M provides an open source service platform for M2M interoperability based on the oneM2M standard. During our Labs, we used two main types of nodes: IN (Infrastructural nodes) and MN (Middle Node). An IN can represent a server and a MN can represent a gateway. In our labs, the MN corresponds to the home gateway and the IN to the OneM2M Service. The IN handles the hierarchy between the different resources and the architecture. We can have several MN for one IN. All the information about the connected devices are handled by the MN. When we want to interact with a device, we register it in the MN and we give all the information about it. When we change the state of the light, the API updates the information of the MN, as there are in constant interaction.

There are also two other types of nodes:
- ASN which have the same characteristics than a MN node, but is a terminal node: we can't register on it
- ADN (Application dedicated node) which register on another node but doesn't host the API

A simple example of architecture with different nodes



2.2. INTERACT WITH OBJECTS USING A REST ARCHITECTURE

*Explain how you could interact with objects and view the data sent by them (including resources generated by IPE Sample, simulated lamps). Explain the different oneM2M resources that you have manipulated (****CSE-BASE, AE, CNT, CIN, SUB, REMOTE CSE****) and how to interact with them (HTTP client + server).* ***(TP 1 + 2)***

During our labs, we used a REST architecture to interact with the object and view the data, because OM2M provides a RESTful API. There are different types of resources:

- CseBase: describes the CSE executed on the node, and is the root for all other resources (type, access point, …)
- AE (Application Entity): represents an applicative resource (device) and stores the information about it
- CNT : a container can contain CINs or other containers. It structures the data.
- CIN : a content instance is an instance of data at a specific point in time.
- Subscription: the idea when you subscribe is to receive a notification when there is an event like the creation, update, or delete of a resource.

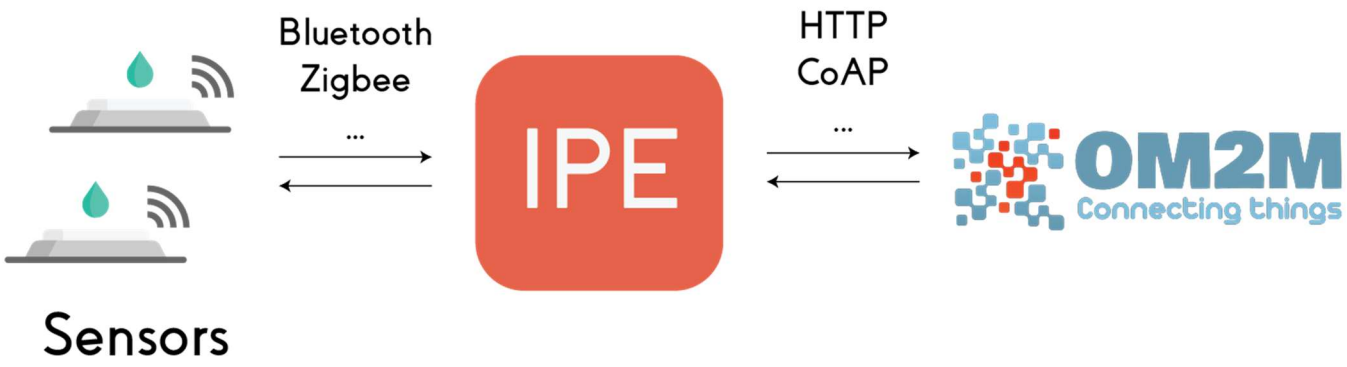
How did we interact with them? To interact with the different resources, we created mainly two types of request: GET and POST. We learned first how to do it with Postman and in a second time with eclipse. As their name suggest it, a GET request allows us to know information about the device (e.g. status) when a POST allows us to create or update a resource. With both methods (Postman or Eclipse), the idea is to define the header containing the information about the request (origin, type, …) and a body (containing the element we wanted to add or change) for the POST requests. To do that, we created a Client class and methods that instantiate a client, the corresponding HTTP method and returns a response object of the needed type.

We also used the PUT method briefly, as it is used to modify the body of an already existing node. In OM2M, it can be used to update the general information of an application entity for example. To update the values from a sensor, it is not correct to update an already existing CIN, you have to POST a new one.

2.3. INTEGRATING A NEW / OTHER TECHNOLOGY IN AN EXISTING IOT ARCHITECTURE

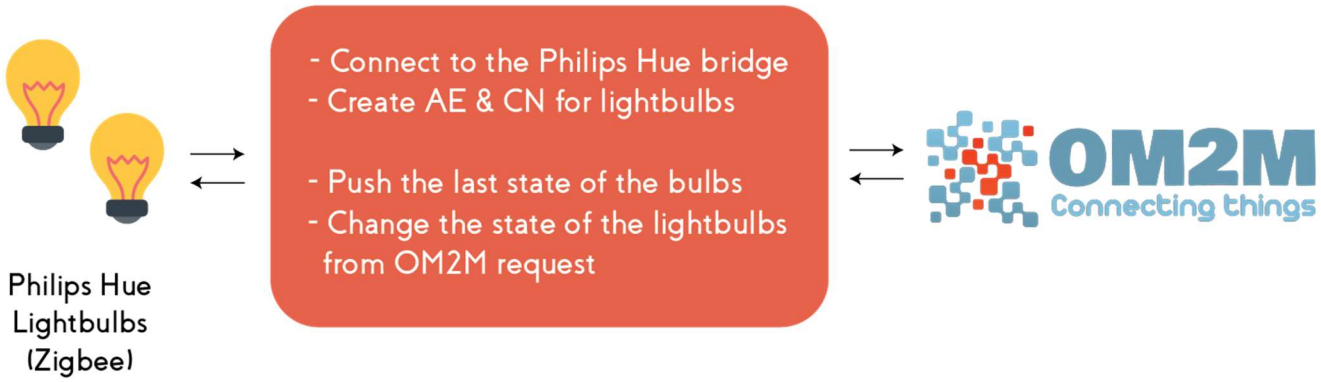
Explain the principle of interworking Proxy Entity of oneM2M as well as the architecture used to integrate in the middleware a new technology not natively compatible with the standard. (Software architecture, oneM2M nodes used, oneM2M resources used to interface.) ***(TP 3)***

An Interworking Proxy Entity, or **IPE**, is used to bridge the gap between a sensor and oneM2M. Sometimes, a sensor manufacturer does not provide an API that is directly compatible with oneM2M, creating Application Entities, Containers or Content Instances. That is where the IPE is useful. Using the sensor's API, it acts as a translator and sends/receives requests that are compatible with the oneM2M standard.



We created our own IPE to control **Philips Hue lightbulbs** on OM2M. We set up a Java server and used the Philips Hue SDK to connect and to use the lightbulbs.

First, we connect to the Philips Hue bridge when starting the server. If we detect a sensor, we send a POST request (REST) to OM2M, creating an Application Entity (AE) and a Container (CON). Then, the server enters its routine where it offers **up-to-bottom or bottom-to-up** services:
- The IPE allows to push the last known state of the lightbulbs to OM2M as Content Instances (CIN)
- The IPE allows to change the state of a lightbulb when receiving a request from OM2M.



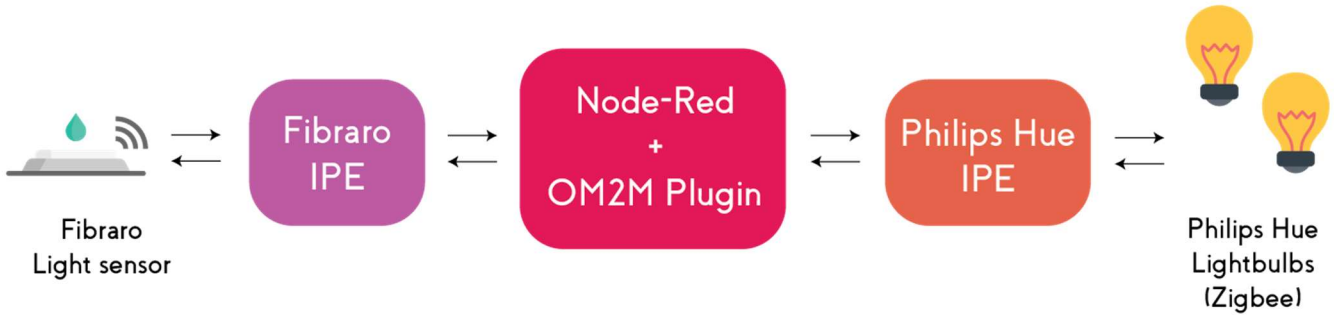
3. DEPLOY A COMPOSITE APPLICATION USING VARIOUS TECHNOLOGIES THANKS TO NODE-RED,

Briefly explain the operating principle of NODE-RED.
Give examples of application(s) you have deployed with NODE-RED (screenshot flow and explanations for instance). Also export your application (export flow > json) and attach file(s) to your report. ***(TP 4)***

Node-red is a graphical tool based on Node.js used to connect devices, APIs and services from the web. It has many applications in IoT because it allows to create a user interface to visualize or use IoT services very fast. A community of users and developers created many plugins to increase the functionalities of node-red, and one of them allows us to use OM2M easily.

We used node-red to build a realistic scenario using the **Philips Hue lightbulbs** used previously and a **Fibraro light sensor.** The idea was that if the luminosity detected by the light sensor dropped below a threshold, we would turn on the lights in the room.

We already created an IPE for Philips Hue, and the Fibraro light sensor came with an already functioning IPE, so all we had to do was use node-red and its OM2M plugin to connect everything.



In Node-red, we used an **inject** element to periodically (every 1s) send a request to the Fibraro Light sensor using the om2m plugin **NamedSensor** (called *Motion here) and* **DataExtractor** elements.

Then, we use a **switch** on the payload of the DataExtractor to check it against a threshold (the threshold was fixed manually in the switch). If it's below the threshold, we turn on the light, else we turn it off. To do so, we use the **NamedActuator** (called *ACT_LIGHT* here) element from the OM2M plugin.