Middleware for Internet of Things

Report – 5ISS INSA Toulouse

GAUTIER Renaud Tamon

5ISS - A2

Standards for Internet of Things (IoT)

Foreseeing an explosion in the field of IoT, most actors are concerned about security, energy consumption, scalability and interoperability issues. It is estimated that there would be around 50 billion devices around 2020 [1], and recent security breach, such as the Mirai botnet that infected several IoT devices (cameras) to conduct massive DDOS attacks, suggest that the IoT environment is still immature.

To face those issues some standards have been proposed, such as OMA-DM, OMA LightweightM2M (LWM2M), MQTT, TR-069, HyperCat, OneM2M, Google Thread and AllJoyn/AllSeen Alliance [2]. Each of them is tackling one or multiple issues of IoT, such as the code footprint, the interoperability or the security. Among them, some standards have gathered several “big names” such as Samsung Electronics and ARM for Google Thread, and Cisco and Microsoft for AllJoyne/AllSeen Alliance.

OneM2M is more particular, since it is combining LWM2M, OMA-DM and TR-069. Institutions such as the CCSA (China Communications Standards Association), the ETSI (European Telecommunications Standards Association) and the TIA (Telecommunications Industry Association, US) are also behind the OneM2M standard. In this report we will focus on the use of OneM2M as a standard describing the way to communicate with several IoT devices.

Actually, there are several standards for IoT communication, each addressing a particular issue of Machine to Machine (M2M) communication: low-energy, speed, bandwidth, security, etc. During their conception, those protocols did not considered the compatibility with other protocols, which resulted in a myriad of protocols that can only understand themselves.

OneM2M tried to face this issue by proposing an horizontal integration of M2M communication, in opposition to the conventional development scheme (vertical integration). We can see OneM2M as standard that describes the way the data is formatted and transmitted between IoT nodes, regardless of the communication technology. The abstraction allows the use of multiple IoT protocols, without the need to make them compatible with each other. They just need to be compatible with OneM2M, and OneM2M will handle the communication between each protocol, as if it was an interpreter for the protocols.

Deploy a standard-compliant architecture and a sensor network

Deploy and configure an IoT architecture using OM2M

OM2M is an implementation of OneM2M made by the LAAS, accessible at the following github address: https://git.eclipse.org/r/om2m/org.eclipse.om2m. It is coded in Java, and can be built using Maven. A bundle of executable files can also be downloaded at the following address: http://wiki.eclipse.org/OM2M/Download.

Below is a typical deployed OM2M structure:

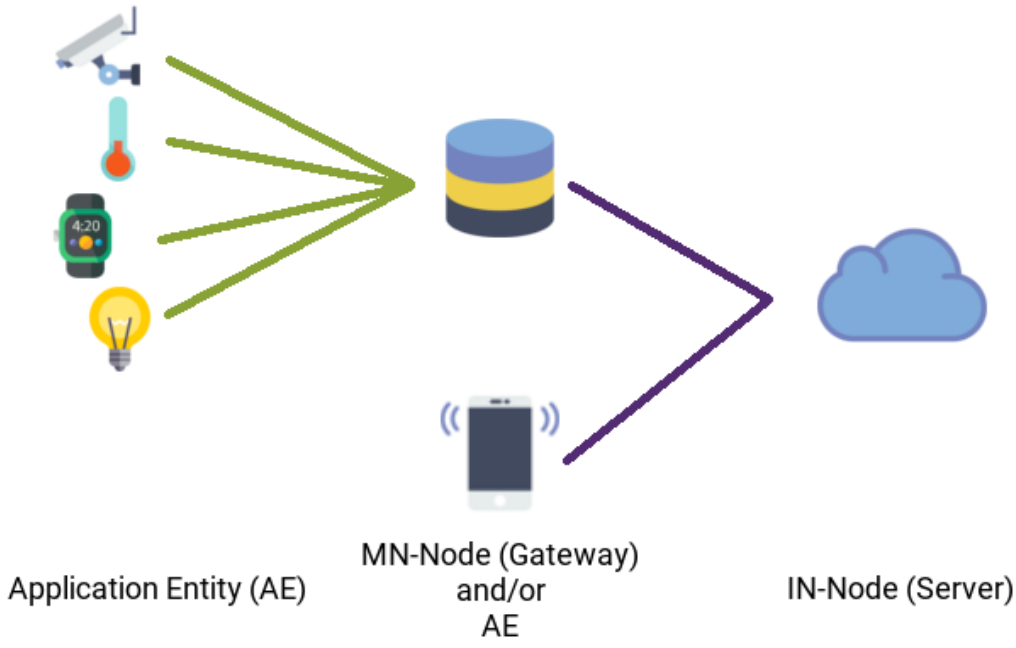

Figure 1: OM2M deployment example

The principal OM2M components are the following: Application Entity (AE), Middle Node (MN, Gateway) and Infrastructure Node (IN, Server). A sensor will communicate using an AE, which is an application that is able to interact with the sensor using the sensor-specific communication protocol. Only the AE is aware of the protocol used to communicate, so that OM2M can make an abstraction of the sensor's management. Those special AE that are bonding the sensors with OM2M are also called proxies. Each AE can be connected to a MN or directly to an IN. Each MN is connected to an IN, which can be considered as the root of the IoT network.

For our work we will use OM2M to interact with Philips's Hue lamps. The Hue lamps are smart lights whose colour and brightness can be set from a remote application. The lamps are connected to a bridge (the Hue bridge) that can communicate with the exterior using the Hue SDK.

We will develop an AE which will integrate the Hue SDK in order to communicate with the bridge. The AE will allow us to send orders to the bridge, and also to retrieve the lights' state in order to push them to OM2M. The AE will communicate with a MN node, in order to access to OM2M as shown below:

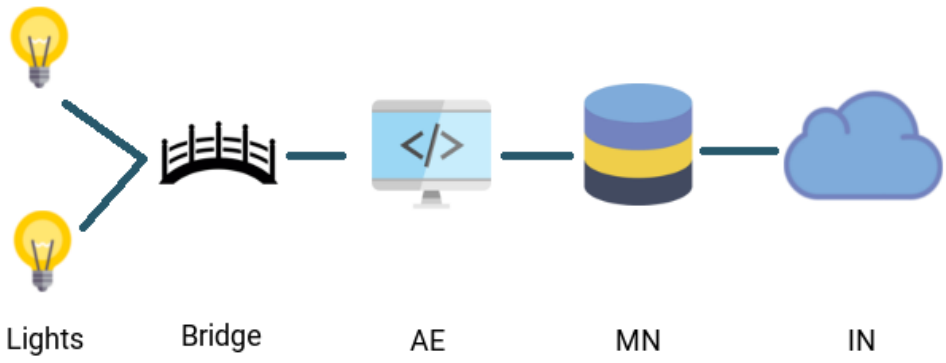

Figure 2: structure of the whole project

A default configuration file is used for the MN and the IN, since the AE is used on a local network and do not require complex IP configuration. The Hue bridge is using a token-based authentication scheme, but the provided skeleton of the AE integrates a save-and-load service, which we will be using to retrieve the token.

Interact with objects using REST architecture

OM2M implements a REST api to retrieve, create, modify and delete different resources. The resources can represent an application, an aspect of the application, or the data handled by the application. All the data is stored in the IN or in the MN node, the IN having access to all the connected MNs. The MN will be used to store all the application-related resources through the url http://localhost:8080/~/mn-cse/mn-name/ (root).

OM2M allows two ways to access a resource: by using the whole path to the resource (root/parent_resource_name/wanted_resource_name) or by using the id of the resource (root/wanted_resource_id). All the requests need to be authenticated with a login and a password.

Three different resource types will be used for the project: AE, Container (CNT) and ContentInstance (CIN). The AE is used to represent an application, which can be a sensor, an actuator or both of them. The AE can contain several CNTs, used to represent a particular aspect of the application, such as the description or the data produced by the AE. Those data will be represented using CINs, which can be considered as small pieces of information.

The AE resource type will be used to represent a Hue lamp: for each Hue lamp there will be one AE. Each AE will have two different CNTs: one for the description, the other for the data. The description CNT is named "DESCRIPTOR", and can contain a CIN describing the application. The data CNT (named "DATA") will contain CIN describing the state of the lamp (on or off). All this structure is automatically generated by IpeServer when we launch the application.

Each time there is a modification in the light state, the application will send a new CIN containing the updated state of the light. A user will be able to access to the latest state of the light through OM2M.

OM2M also allow the user to send orders from OM2M to the application by the mean of subscription. The application will be listening on a predefined port (1400) to which OM2M will send a notification each time there is a change on the subscribed topic. We can apply filters so that the subscribing application receives only the notification or the modified content too. The subscription will appear as SUB in the OM2M resource tree, under the resource we subscribed.

Finally, the user can interact with the lights through OM2M resource tree. The CIN in the DESCRIPTOR resource contains a section that allow the user to send commands to the application, through a http request. The request form is in a http GET request format, with parameter couples “key=value” separated with “&”. Upon receiving the request, we parse the parameter to a hashmap in order to execute the orders with the right arguments.

Upon creating the described resource tree, we mostly used the POST request for creating the desired resource, exiting the application when facing conflicts. We could have used other REST utilities, such as the DELETE for removing the existing application, or the GET to retrieve the existing resource and adding the necessary resources to them. OM2M even allow the use of research queries, which we could have used to search all Hue – related AEs.

Integrate a novel technology in an IoT architecture

Most of the time, sensors are either incapable of complex behaviour as described in OM2M, or are implementing similar functions in a different protocol. In order to make those sensors compatible with OM2M we need to implement an Interworking Proxy Entity. An IPE can be seen as a special AE that is translating one’s communication standard into an OM2M compatible one, as previously seen with the integration of the Hue SDK.

An easy way to integrate a novel technology is to create a specific application for each sensor, that can communicate with a gateway (MN node). The application can on one side initialize the sensor and retrieve its data, while generating the necessary resources on the OM2M resource tree. Below is an example of an OM2M representation of a sensor we used to represent a Philips’s Hue lamp.

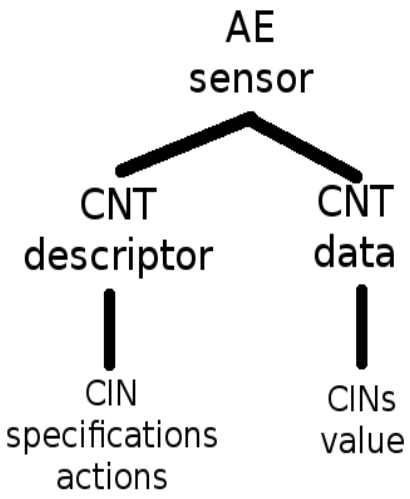

Figure 3: sensor representation

A root AE will represent the sensor, which will have two different containers. The descriptor will contain the description of the sensor (type of the measure, manufacturer, precision, location, etc) and the possible actions (turn on/off, modify sensor-specific values, etc). Each time the sensor get a new value, we put it in the data container.

When handling the sensor, it is better if the AE can have access to the SDK of the sensor, so that it is easier to interact it. In the case there is no SDK and/or it is a very simple sensor without much functionalities, it is the AE's role to implement the needed methods to make sure the integration is seamless.

[1]http://www.ericsson.com/spotlight/transformative-it/blog/heading-towards-50-billion-connections/

[2]http://internet-of-things-innovation.com/insights/the-blog/key-standards-for-m2m-and-iot/#.WF-p4WfpW7s