Middleware for Internet of Things
Practical Work - Report
5ISS-A1
INSA Toulouse

Cécile DUTHOIT, Linn MJELSTAD

January 2017

1 Introduction

This report covers what we did during the IoT sessions. It aims to show what we learned and what skills we got from these sessions.

Summary:

1. Understand the major standards for the Internet of Things
   (a) OM2M: the project we used
   (b) What is oneM2M ?
   (c) What is SmartM2M ?
   (d) How does OM2M implement oneM2M standard ?
2. Deploy an architecture according to a standard and implement a sensors network system
   (a) Deploy and configure an IoT architecture using OM2M
   (b) Interact with objects using a REST architecture
   (c) Integrate a new technology in an IoT architecture

2 Understand the major standards for the Internet of Things

2.1 OM2M: The project we used

The Eclipse OM2M project we used, initiated by LAAS-CNRS, is an open source implementation of oneM2M and SmartM2M standard. OM2M provides a horizontal M2M service platform for developing services independently of the underlying network.

2.2 What is oneM2M?

Created in July 2012 by seven international organizations of standardisation (including European European Telecommunications Standard Institute) with the contribution of more than 200 companies and five industrial associations, oneM2M aims to lead the societies and researchers that are working on IoT to propose and suggest standard specifications.

Its purpose is to develop a standard offering technical specifications to fulfill the need for a common M2M (machine-to-machine) Service Layer that will be compatible with many software and hardware components, in order to be able to connect all the connected objects we want in the Internet of Things all around the world.

The domains it aims to affect are many: telematics and intelligent transportation, health care, utilities, industrial automation, smart homes, etc. The technical specifications oneM2M works on mainly focus on:

- Use cases and requirements for common and compatible Service Layer features
- Detailed Service Layer features
- Protocols, APIs and standard objects based on this architecture (open interfaces protocols)
- Security and privacy protection
- Reachability and discovery protocols
- Interoperability
- Identification and elaboration of standards for relevant devices and applications naming

Figure 1: Global architecture of a oneM2M-following project

2.3 What is SmartM2M?

SmartM2M is a solution providing company that aims to make systems efficient with machine to machine communication solutions and is part of ETSI Committee Support Staff.

2.4 How does OM2M implement oneM2M standard ?

The OM2M project follows oneM2M standard by dividing its architecture as well:

- **M2M Device:** a machine that runs an M2M device program.
  An M2M device can connect directly to the M2M Network or indirectly via a gateway that acts as a network proxy.
- **M2M Area Network:** part of the global network that provides connectivity between M2M devices and between those devices and the M2M gateways.
  It is based on existing technologies such as Zigbee, Phidgets, M-BUS, KNX, etc.
- **M2M Gateway:** a machine that runs an M2M Gateway program.

- **Wide Area Network:** core network that access other networks to enable M2M devices and gateways to communicate with the M2M network.
  It is based on existing technologies such as xDSL, GERAN, UTRAN, eUTRAN, W-LAN, WiMAX, etc.
- **M2M Network:** part of the global network that can be accessed by M2M Network applications (NAs) via an open interface.
- **M2M service capabilities Layer (SCL):** horizontal service layer for M2M applications that enhances M2M interoperability by abstracting the complexity of heterogeneous devices.
- **M2M application a domain-specific software application:** application that interacts with the M2M machines by querying the service layer through a set of open interfaces.

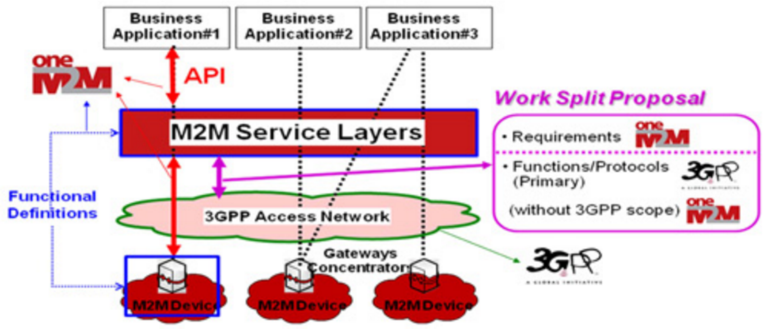

Figure 1: Global architecture of a oneM2M-following project

3 Deploy an architecture according to a standards and implement a sensors network system

3.1 Deploy and configure an IoT architecture using OM2M

We used different tutorials to deploy and configure the project.

- To install and build: https://wiki.eclipse.org/OM2M/one/Clone
- To configure the IN and MN: https://wiki.eclipse.org/OM2M/one/Configuration

We kept the initial configuration parameters because these can be used for a local demo version of OM2M.

By deploying an IoT architecture with OM2M we can access a resource tree that visualizes all the resources that we have created. In order to access this tree, we first have to start a MN and an IN instance locally on the computer. These have been downloaded and configured via eclipse.

- IN is the Infrastructure Node

- MN is the Middle Node

When starting the IN this enables the IN-CSE web interface, that contains the resource tree, which can be accessed via http://localhost:8080/webpage using 'admin' as both login and password. When the MN gets started, it uses the IN specified in its gateway configuration file for authentication. If the IN hasn't been started first, the MN will continue to send authentication requests every 10 seconds. When both instances have been deployed, there will be a MN-CSE accessible within the IN-CSE, which is a sub-resource tree with remote resources. We have kept the configuration files as they are in order to be able to run the project correctly.

The application interacts with the IN, the IN interacts with the MN, the MN interacts with the connected object (sensor, lamp, etc.).

3.2 IPE sample plugin

We tested an IPE sample plugin that came with the project. This is located in the MN-CSE and the monitoring application can be activated via the MN instance by running 'ss ipe', followed by 'start ¡id¿'. This starts the following java interface (figure 2).

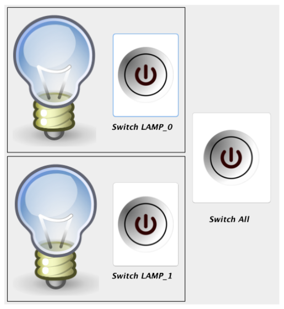

Figure 2: View of the IPE sample plugin graphical interface.

We can observe that for each time that we push one of the buttons and turn on or off a light bulb, this will create a new content instance. A content instance can not be modified after it has been created, and therefore new one have to be made for each change of state. The previous content instances is kept until the one reaches the the limit of the size of the container, after this the oldest one is deleted.

3.3 Interact with objects using a REST architecture

We have used two different methods to communicate with the M2M architecture via REST requests. The first one we used was Postman, where you can easily create requests that can be sent to the web service. We also created our own REST Client based on the HTTPClient API from Apache.

When using REST, there are four different types of request that can be sent:

- POST, to create the resource
- PUT, to update the resource
- GET, to retrieve information about the resource
- DELETE, to delete the resource

There are several other parameters that also have to be specified in a request in addition to the type:

- The URI, either the URI of the "mother-resource" if you want to create a resource, or the URI of the resource itself if you want to update, delete, or retrieve information from the resource
- The headers, containing:
  - The language of the request (xml, json etc)
  - The login and password for the M2M interface
  - The resource type (AE, Container, Content Instance, ACP etc.)
- The body

We used different OM2M resource types during this project:

- **CSE-base:** Common Services Entity Base
  It describes the hosting CSE, and is the root for all other resources within the hosting CSE.
- **AE:** Application Entity Instance
  It stores information about the Application Entity after a successful registration on the hosting CSE.
- **CNT:** Container Instance
  It acts as a mediator for data buffering to enable data exchange between applications and CSEs.
- **CIN:** Content Instance
  It stores content in the previous container instance. Contents are states of the application for instance.
- **SUB:** Subscription
  It stores information related to subscriptions for some resources. It allow subscribers to receive asynchronous notification when an event happens such as the reception of new sensor event or the creation, update, or delete of a resource.
- **remote-CSE:** *It stores information related to M2M CSEs residing on other M2M machines after successful mutual authentication. It enables CSEs interactions using re-targeting operations.*

We manipulated these resources using HTTP requests as explained above. By building a relevant REST request using our HTTPClient, we can create, update, delete and get info from our resources and check the modification by logging in our OM2M server webpage (http://localhost:8080/webpage).

We also manipulated access and modification permissions, which are Access Control Policies (ACP). The ACP manage permissions and permissions holders to limit and protect the access to the resource tree structure. Two types of permissions exist:

- **PV:** Privileges
  They manage the access to the resource itself.
- **PVS:** Self-privileges
  They manage the right to modify the access rules;it defines the highest administrator.

3.4 Integrate a new technology in an IoT architecture

To communicate with IoT devices that are not a part of the OM2M standard, we have used the Interworking Proxy Entity (IPE) technology. The IPE creates the interface between the device and the OM2M standard.

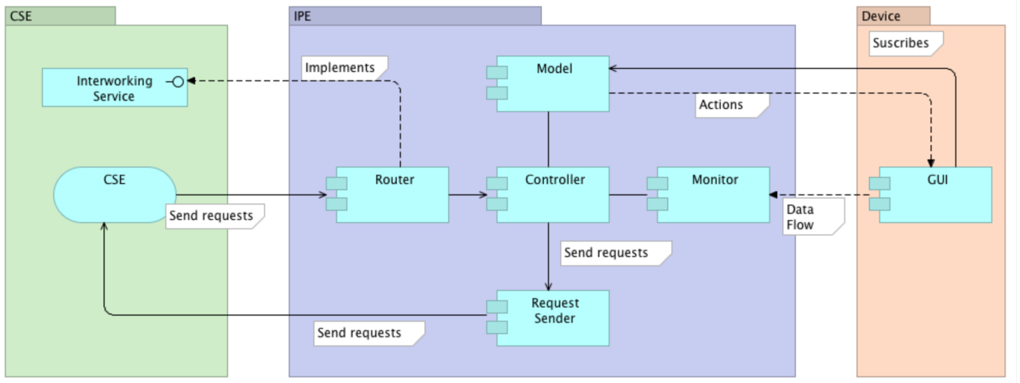

Figure 1:Architecture of an IPE

- The monitor monitors the devices, and reads their values. The values are then pushed onwards to the middleware. This can be done periodically or, as in the case of HUE, it can be done on a callback signifying an update of the cache.
- When a execute command is received, the controller will implement the method.
- The model stores the objects used to represent the devices we are controlling.

We are working with HUE lamps from Philips, that uses Zigbee to communicate with a HUE Bridge and onwards to a router. OM2M also uses the router to connect to the bridge, and this creates the link between the device and the CSE.

For each lamp we created an AE in the resource tree with a descriptor container and a data container. In the descriptor container we also created a content instance. The descriptor container holds information about the resource, and the data container hold the measurements.

When we want to change the color of the lamp, we send a command via the bridge using libraries supplied by Philips.

As we have created the software that links the lamp to the IN-CSE resource tree, we also have to make sure that every new value affected to the lamp, by us or any other user connected to the same object, gets uploaded to the associated AE. Each time that the light value has been updated we will receive a notification of an update of the bridge cache, and we will then have to update the resource on the platform.