MIDDLEWARE FOR THE INTERNET OF THINGS

TP REPORT – 5 ISS INSA TOULOUSE

The purpose of this report is to show the skills that I have acquired and the concepts those I understood as a result of the courses and the various practical sessions.

The report is of **8 pages** and in addition I have enclosed the **4 pages** of Annexure. In the annexure, I have presented the software code snippets to explain how I developed and deployed the additional modules such as REST Client, Mapper (JAXB, Obix). Also, the code snippets in the annexure are used to explain how I used the modules to deploy the OM2M resources and used OM2M as the middleware.

Lastname 1 : SELVARAJ

Firstname 1 : MOHAN PRABHU

A Kind Note- I did the TP's and the report as the only person.

Date
08-Dec-2019

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.

I answer this question in a structured approach as find below

- What is 'Internet of things' ? What are the technologies that make an IoT architecture?
- What are the applications domain in IoT ? What are the main standards in IoT ?
- What are key principles of OneM2M standard ? How OneM2M interworks with other standards and technologies?

Technologies/Protocols in IoT

From the perspective of Web technologies evolution, IoT is a Web 3.0 technology. The convergence of following web technologies creates the 'Internet of things' web,

- Physical web, Real-time web, Semantics web, Social web, Programmable web

From the perspective of **Network layer**, IoT is a wireless network which uses wireless technologies to connect the physical and virtual things and to communicate between them. Some of the wireless technologies,

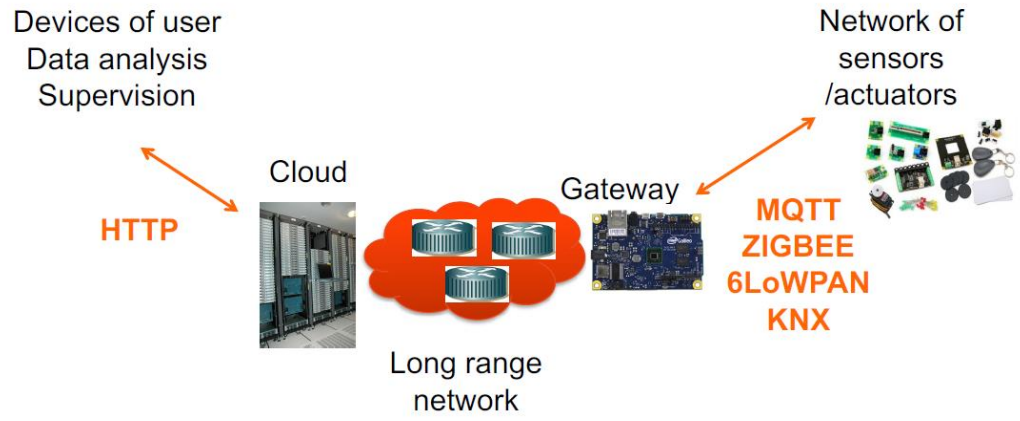
- Short range – NFC, Bluetooth, BLE, Zigbee, Wi-Fi
- Long range Licensed spectrum – GPRS, 3G, 4G, NB-IoT, LTE-M
- Long-range Unlicensed spectrum - LoRA, Sigfox

From the perspective of **application layer**, the IoT applications have the following choices for data transport protocols,

- HTTP, MQTT, CoAP, TR-069, LWM2M

Architecture of IoT

The IoT sensors/actuators are smart electronics devices that generate the measurement data which are transmitted by short range or long range wireless technologies to the gateway. From the gateway, the traditional IP network carries the data to the cloud for storage, analysis and insights generation.

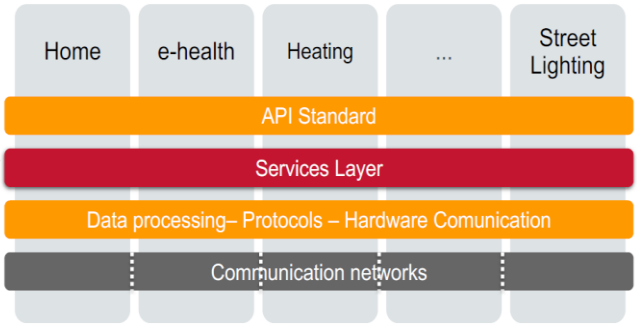


Vertical Application domains in IoT/M2M

Network & systems, Security, Distribution, Transport, Industries, Health/Science, Home/Building, smart cities/towns, Energy, Construction, Agriculture, Wearabales

OneM2M Standard and Key Principles

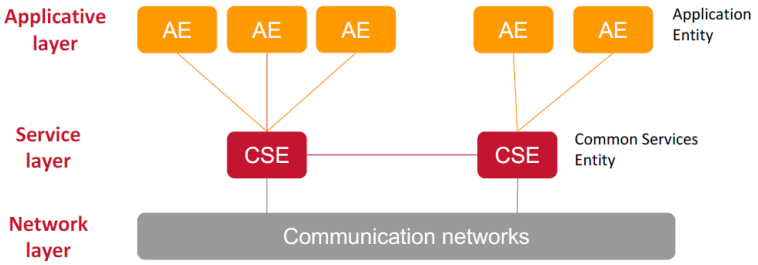
OneM2M is a IoT/M2M standardization agency that has developed a **common M2M Service Layer** which can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with IoT/M2M application servers worldwide.



OneM2M's **common service layer (CSE)** enables to develop IoT/M2M services independent of the underlying network, thereby, facilitates the deployment of vertical applications and heterogeneous devices.

OneM2M Architecture

The Sensors/actuators 'AE' devices in the AE layer can be from any vendor and can be made for any vertical application domain.
The communication network protocols in the Network layer can be any technology.
Still, the OneM2M's CSE layer helps to connect the Sensors/actuators to the IoT/M2M application servers; And, thus the data retreived from sensors/actuators of any vertical application can be transported to the cloud.

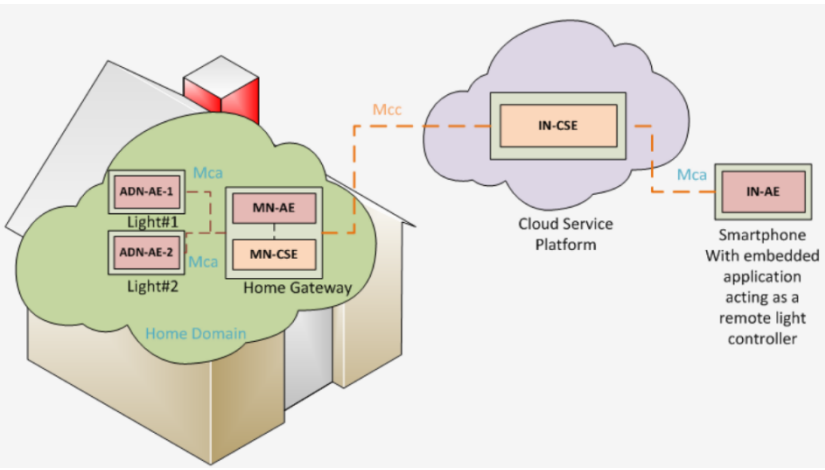


OneM2M Positioning (vs) other standards

OneM2M is a open standard made with the architecture of a horizonatal **middleware** and therefore can support multiple vertical application domains – which is in sharp contrast with other popular standards such as Thread, Homekit, FIWARE. Therefore, OneM2M is not a standard which is specific to some particular business application domains.

OneM2M Vertical Applications

In the Smart-home architecture figure, we can see the light devices (AE) and the smartphone (AE) with remote light controller application getting connected to the cloud using the OneM2M CSE middleware.
The light devices, gateway, smartphone can be made by any vendor. Also, the cloud application server can be from any vendor and can be distributed worldwide.
With OneM2M, they all can be binded together as a smart-home application.



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)***

I followed the following approach/steps as per the coursework from 'activity' and 'TP' materials to deploy the OM2M architecture. I will explain each of the steps and answer this question.

STEP	Exercises	Actions	Outcome
1	P2_en_activity	Configure & launch OM2M	IN-CSE deployed at localhost:8080 And MN-CSE at locathost:8282
2		Launched 'Sample' Lamps	Observed the communication between external Actuator (Lamp) and OM2M
3		Manage the OM2M resource tree	Create 'MY_SENSOR' resource tree using POSTMAN- XML or JSON Create 'Subscription' & receive notifications from 'MY_SENSOR'
4	TP-1	Create 3 new sensors & subscriptions	Using POSTMAN – Electricity/Light/temperature sensors created and subscriptions mechanisms were registered
5	TP-2	Middleware entity - REST Client development	Develop REST Client & XML Mapper – Prepare the OM2M resources as middleware between the monitoring application and the Sensors/actuators
6		Middleware entity - XML Mapper development	

STEP-1 Configure & Launch OM2M

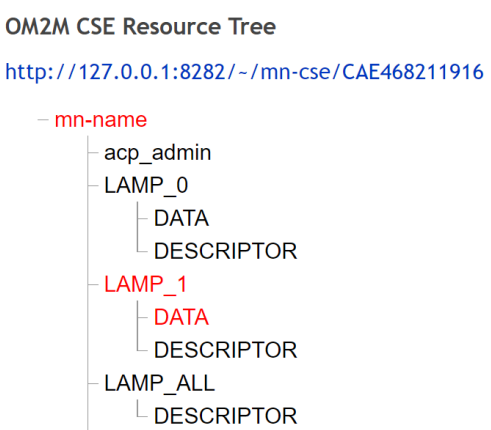
In this 1st step, I downloaded the OM2M software, launched the servers and created the following 2 nodes of OM2M,

1) IN-CSE, the Infrastructure Node with poa= http://127.0.0.1:8080, ty=5 and csi=in-cse
2) MN-CSE, the Middle Node (Gateway) with poa= http://127.0.0.1:8282, ty=5 and csi=mn-cse

STEP-2 Launched IPE Sample Lamps

In this step, I started the IPE Sample service in MN-CSE OM2M server

- Thus, I created 2 Lamps as ADN-AE nodes (ty=2) in the OM2M. The Lamp controller application acts as the IN-AE for the lamps
- Observed from the Switch-ON and Switch-OFF of the lamps, the 'Container Instances' were created within the 'DATA'
- Understood that the 'Container Instances' hold the event/state data of the actuator – LAMP_0 and LAMP_1.



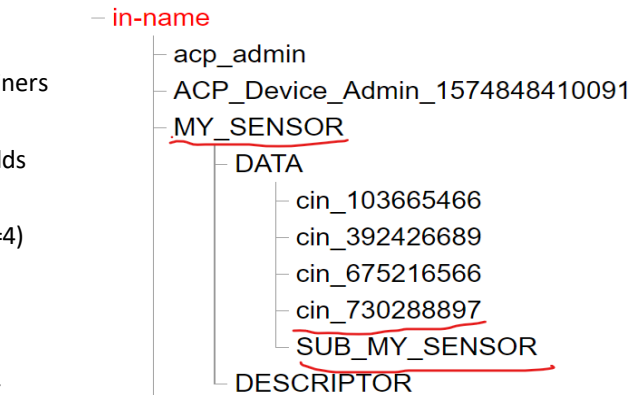
STEP-3 Manage the OM2M Resource Tree

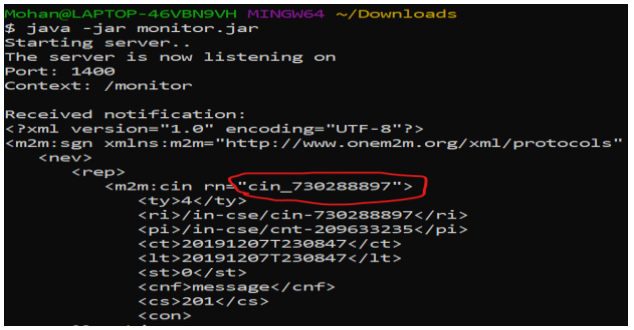
In this 3rd step, I used **POSTMAN** as the REST client to communicate with the OM2M **IN-CSE** and created the other nodes of OM2M: **ADN-AE and IN-AE**.

- I created the ADN-AE: 'MY_SENSOR' (ty=2) and the 2 Containers – 'DATA' (ty=3) and 'DESCRIPTOR' (ty=3)
- Within 'DATA', I created the 'Container Instances' which holds the data values of the 'MY_SENSOR'
- Within 'DESCRIPTOR', created the 'Container Instances' (ty=4) which holds the description about the 'MY_SENSOR'
- Within 'DATA', I created the 'SUB_MY_SENSOR' as a Subscription service (ty=23)
- 'SUB_MY_SENSOR' points to http://localhost:1400/monitor where the monitor application runs
- From POSTMAN, when I created the new CIN (CIN_730288897), the 'SUB_MY_SENSOR' notified the monitor application (IN-AE).
- I understood that, the OM2M middleware notifies the 'IN-AE' about the events/status related to 'ADN_AE' via the subscription services.

OM2M CSE Resource Tree

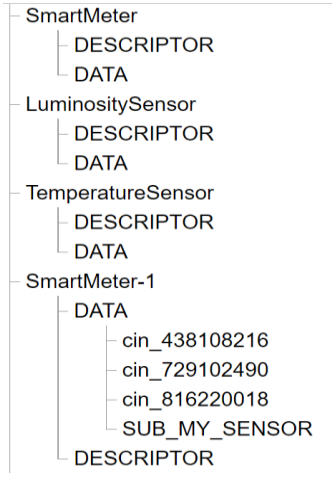
http://127.0.0.1:8080/~/in-cse/cnt-209633235





STEP-4 Create 3 new sensors & subscriptions

With knowledge of the 3rd step, in step-4, I used **POSTMAN** as the REST client to communicate with the OM2M **MN-CSE** and created 3 new **ADN-AE** sensors (electricity, temperature, light). And, I created a new AE 'SmartMeter-1' with subscription service 'SUB_MY_SENSOR', registered subscription services for electricity/temperature/light.



	SmartMeter (Electricity)	Luminosity (Light)	Temperature	SmartMeter-1 (Subscription)
poa	none	None	none	http://localhost:1400/monitor
rr	false	false	false	true
Subscription_ resource	none	none	none	SUB_MY_SENSOR

STEP-5 Middleware entity - REST Client development

From the step-3 & step-4 learnings, in the 5th step, I developed the REST client module. The main idea is to implement the GET/POST features and use them to create/retrieve resources in OM2M middleware platform. The development uses the below packages, files and functions.

Packages & files	fr.insat.om2m.tp2.client; fr.insat.om2m.tp2.test; *Client.java; ClientInterface.java; Response.java; Main.java; create_ae.xml; create_cin.xml; create_cnt.xml*
Annexure-1	Retrieve(); create(); -> Pls refer the implementation & output in Annexure-1.1, 1.2

I used the REST Client implementation, to create an AE resource in MN-CSE and retrieve it.

Things learnt

- Likewise, the REST client **create()** implementation can be used to create 'DATA' and 'DESCRIPTOR' containers, Container Instances (CIN)
- Likewise, the REST client **retrieve()** implementation can be used to retrieve 'DATA' and 'DESCRIPTOR' containers, Container Instances (CIN)

STEP-6 Middleware entity – XML Mapper development

In this step-6, I implemented an mapper module which has the 2 main functions listed below,

- UNMARSHAL function which interprets **XML representation and presents its OM2M resource objects**
- MARSHAL function which represents the **OneM2M resource object in XML format**

The packages/files used are as found below. The functions used & their output are in **Annexure-2**

Packages & files	package fr.insat.om2m.tp2.mapper; fr.insat.om2m.tp2.test; *Mapper.java; MapperInterface.java; MapperTest.java*
Annexure-2	Mapper.marshal(); mapper.unmarshal(); Please refer the output in Annexure-2.1

Things observed

- In MapperTest.java, org.eclipse.om2m.commons.resource framework is used as an example to show how the objects created from this framework can be represented in its XML format
- In Mapper.java, marshaller of type JAXB receives the 'OM2M resource object' and serializes it into a payload of type XML string type. Similarly, unmarshaller of type JAXB receives the 'XML string representation' of the object and convert it into 'OM2M object' representation

Things learnt

- Likewise, the **marshal()** implementation can be used to serialize the 'DATA' and 'DESCRIPTOR' containers, Container Instances (CIN) into XML string type
- Likewise, the **unmarshal()** implementation can be used to convert the XML representation of 'DATA' and 'DESCRIPTOR' containers, Container Instances (CIN) into a complete OM2M object.

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

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)***

I followed the following approach/steps as per the coursework from 'activity' and 'TP' materials to **interact with** the OM2M resources. I will explain each of the steps and answer on how I manipulated them & retreived data.

Steps	Exercises	Actions	Outcome
1	P3_activity	Redirection of MN-CSE query via IN-CSE	Even when MN-CSE is not reachable directly, through, IN-CSE it can be reached
2	TP-1 & TP-2	Use REST Client module to deploy AE, Containers & descriptors	At, http://127.0.0.1:8080/~/in-cse AE, Containers, CIN are deployed
		Use the unmarshal() on deployed AE and containers	From the XML response, decode the OM2M object and print the resource ID
3	TP-2	Using Obix & marshal() in combination to serialize objects and share data between different functions	Encode the CIN string attributes into object using Obix. Use marshal() to convert into string representation
		Obix encode -> marshal() -> create CIN	A CIN is created by using Obix and marshal() ; no XML file is used this time for REST create() body.
4	TP-2	Retrieve CIN -> unmarshal() -> obix Decode to print the sensor values	The CIN object was retrieved and using Obix the attributes are decoded and the sensor values are printed

STEP-1 Redirection of MN-CSE query via IN-CSE

Using the configuration previously launched, I have create 2 CSE:
- An IN-CSE listening in HTTP on port 8080
- An MN-CSE listening in HTTP on port 8282

I have created the CSE-Base from the IN-CSE and the MN-CSE using both the options, that is - Postman and the REST Client that I developed.
In this step, I tried to create CSE-Base at MN-CSE by directing the POST query via IN-CSE.
This is to validate the redirection feature provided by the OM2M system.

Please refer **Annexure-3**. I was able to create the AR **'ae_793930710'** at MN-CSE by directing the 'POST' query to http://127.0.0.1:8080, which is the 'poa' for IN-CSE.

I understood that, this was possible because, MN-CSE is registered with its 'pi' as 'in-cse'. This allows, to reach MN-CSE even through the address of IN-CSE. This feature allows to reach any resource in the OM2M system through an alternate route when it's direct route is not reachable.

STEP-2 REST CLient module deployment

In this step, I will present how I used the REST Client Module that I developed earlier, to automate the OM2M resources creation, which means, creation of AE, DATA & DESCRIPTOR containers, Container instance.

Packages & files	fr.insat.om2m.tp2.client; fr.insat.om2m.tp2.test; *Client.java; ClientInterface.java; Response.java; Main.java; create_ae.xml; create_cnt.xml*
Annexure-4	Automated creation of AE, containers - Pls refer the implementation & output in Annexure-4

As can be find from Annexure-4, the client module first creates the AE resource at the prescribed server. In this case, I prepared the request to create the resource tree at http://127.0.0.1:8080. Followed by, as per the implementation, the DATA container is created automatically.

- ae-916352415 (ty=2) at http://127.0.0.1:8080/~/in-cse
- cnt-911731103 (ty=3) at http://127.0.0.1:8080/~/in-cse/in-name/ae_916352415

The XML string response after the resources creation can be seen to be unmarshal() and the resource ID's were derived and printed.

This automation of resources creation can be extended to create the CIN also. But, before the demonstration of CIN creation, I would present the Obix implementation in step-3.

STEP-3 Obix ENCODING and MARSHAL deployment in OM2M

In this step-3, using Obix & marshal() in combination, I show how I serialized the objects while I created the CIN. I created the CIN attributes object, used Obix to encode it as string and shared the string representation to marshal(). Thus, I was able to share the data between different functions in OM2M system to achieve different objectives.

Please find the packages/files & functions used and the annexure details in the table below.

Packages & files	fr.insat.om2m.tp2.client; fr.insat.om2m.tp2.test; *Client.java; ClientInterface.java; Response.java; Main.java; create_ae.xml; create_cnt.xml*
Annexure-5	obj.add(); ObixEncoder.toString(); mapper.marshal() - Pls refer the implementation & output in Annexure-5

- ae_467145998 (ty=2) at http://127.0.0.1:8080/~/in-cse
- cnt_968844396 (ty=3) at http://127.0.0.1:8080/~/in-cse/in-name/ae_467145998
- cin_551751705 (ty=4) at http://127.0.0.1:8080/~/in-cse/in-name/ ae_467145998/cnt_968844396

STEP-4 Obix DECODING and unmarshal deployment in OM2M

In this step-4, using Obix decoding & unmarshal() in combination, I show how I deserialized the objects and retrieved the CIN. I created the CIN attributes object, used Obix to encode it as string and shared the string representation to marshal(). Thus, I was able to share the data between different functions in OM2M system to achieve different objectives.

In the previous step, I had deployed the implementation to create the CIN. In the automation process, in this step, I have to retrieve the CIN, check the content values in the CIN and print the sensor/actuator values.

Packages & files	fr.insat.om2m.tp2.client; fr.insat.om2m.tp2.test; *Client.java; ClientInterface.java; Response.java; Main.java; create_ae.xml; create_cnt.xml*
Annexure-6	ContentInstance(); ObixDecoder.toString(); mapper.unmarshal(); - Pls refer the implementation & output in Annexure-6

So, I did the following,

1) used the response of CIN create() in unmarshal() to retrieve the CIN object in order to retrieve the resource URL

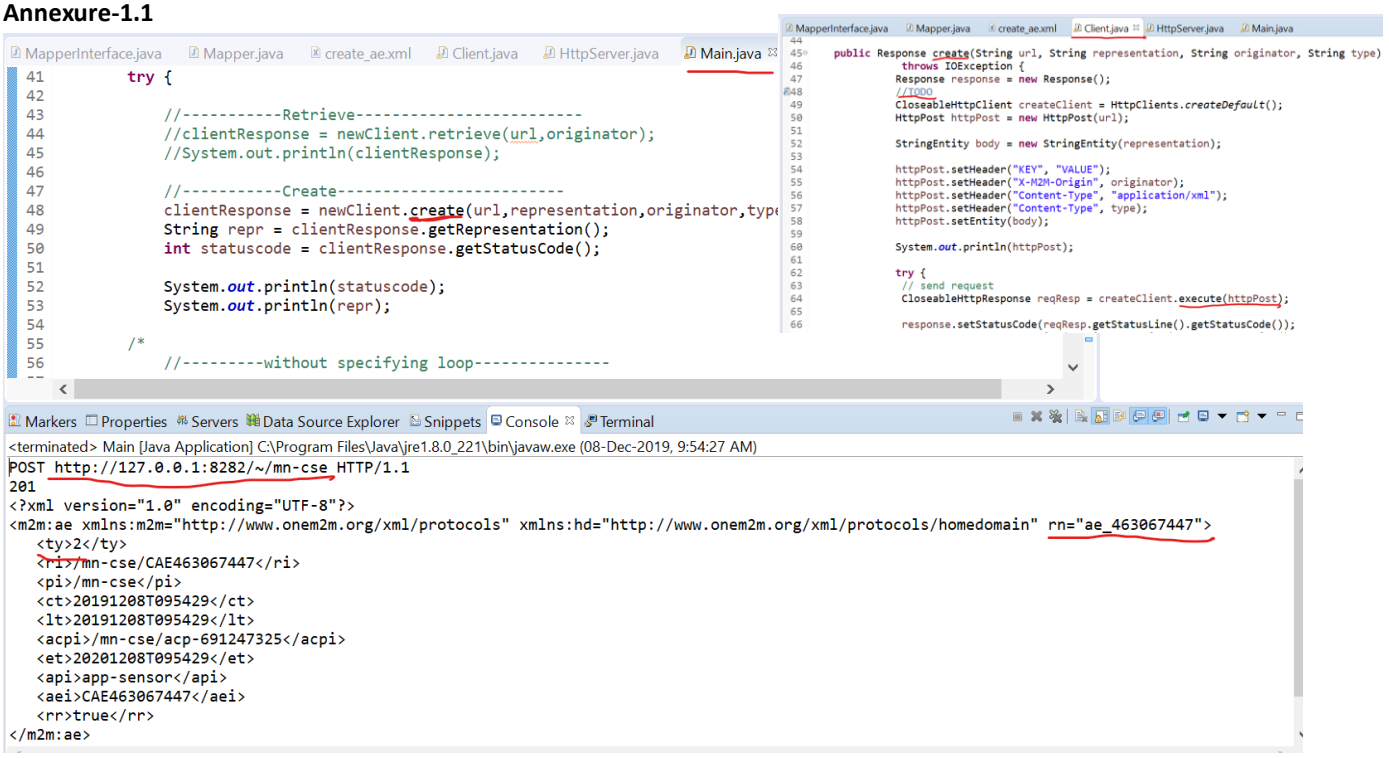
2) used the ObixDecoder.fromString() to retrieve the CIN attributes with the CIN object

3) From the retrieved CIN object, I retrieved the attributes/sensor values and printed them.

Thus, I deployed the OM2M system as a middleware by implementing the REST Client module, Mapper module (JAXB, Obix). And, I automated the resources creation and retrieved the sensor/actuator values from the OM2M system and printed it.

ANNEXURE-1 FOR 2.1/STEP-5 - REST CLIENT DEVELOPMENT

Annexure-1.1

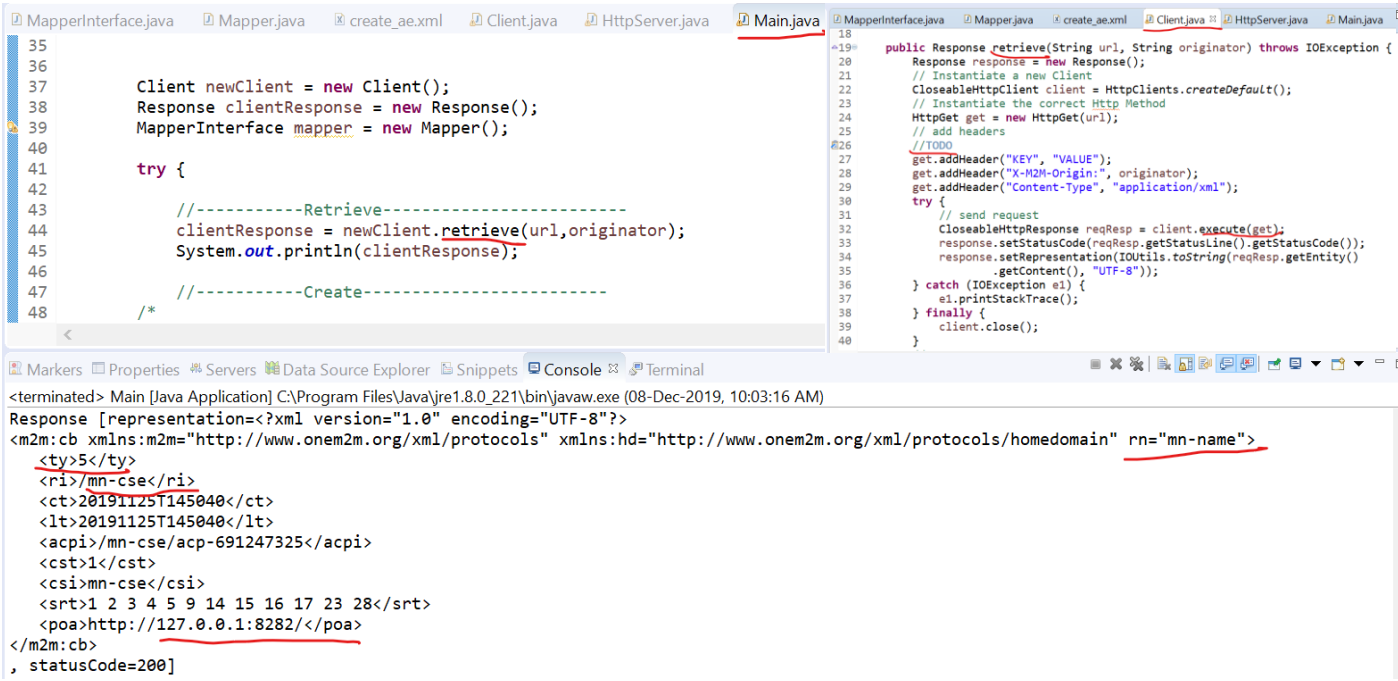


```
MapperInterface.java  Mapper.java  create_ae.xml  Client.java  HttpServer.java  Main.java

41        try {
42
43            //-----------Retrieve-------------------------
44            //clientResponse = newClient.retrieve(url,originator);
45            //System.out.println(clientResponse);
46
47            //-----------Create-------------------------
48            clientResponse = newClient.create(url,representation,originator,type
49            String repr = clientResponse.getRepresentation();
50            int statuscode = clientResponse.getStatusCode();
51
52            System.out.println(statuscode);
53            System.out.println(repr);
54
55        /*
56            //---------without specifying loop---------------
```

```
MapperInterface.java  Mapper.java  create_ae.xml  Client.java  HttpServer.java  Main.java

45    public Response create(String url, String representation, String originator, String type)
46            throws IOException {
47        Response response = new Response();
48        //TODO
49        CloseableHttpClient createClient = HttpClients.createDefault();
50        HttpPost httpPost = new HttpPost(url);
51
52        StringEntity body = new StringEntity(representation);
53
54        httpPost.setHeader("KEY", "VALUE");
55        httpPost.setHeader("X-M2M-Origin", originator);
56        httpPost.setHeader("Content-Type", "application/xml");
57        httpPost.setHeader("Content-Type", type);
58        httpPost.setEntity(body);
59
60        System.out.println(httpPost);
61
62        try {
63         // send request
64         CloseableHttpResponse reqResp = createClient.execute(httpPost);
65
66         response.setStatusCode(reqResp.getStatusLine().getStatusCode());
```

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Console  Terminal

<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (08-Dec-2019, 9:54:27 AM)

POST http://127.0.0.1:8282/~/mn-cse HTTP/1.1
201
<?xml version="1.0" encoding="UTF-8"?>
<m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" xmlns:hd="http://www.onem2m.org/xml/protocols/homedomain" rn="ae_463067447">
   <ty>2</ty>
   <ri>/mn-cse/CAE463067447</ri>
   <pi>/mn-cse</pi>
   <ct>20191208T095429</ct>
   <lt>20191208T095429</lt>
   <acpi>/mn-cse/acp-691247325</acpi>
   <et>20201208T095429</et>
   <api>app-sensor</api>
   <aei>CAE463067447</aei>
   <rr>true</rr>
</m2m:ae>
```

Annexure-1.2



```
MapperInterface.java  Mapper.java  create_ae.xml  Client.java  HttpServer.java  Main.java

35
36
37        Client newClient = new Client();
38        Response clientResponse = new Response();
39        MapperInterface mapper = new Mapper();
40
41        try {
42
43            //-----------Retrieve-------------------------
44            clientResponse = newClient.retrieve(url,originator);
45            System.out.println(clientResponse);
46
47            //-----------Create-------------------------
48        /*
```

```
MapperInterface.java  Mapper.java  create_ae.xml  Client.java  HttpServer.java  Main.java

18
19    public Response retrieve(String url, String originator) throws IOException {
20        Response response = new Response();
21        // Instantiate a new Client
22        CloseableHttpClient client = HttpClients.createDefault();
23        // Instantiate the correct Http Method
24        HttpGet get = new HttpGet(url);
25        // add headers
26        //TODO
27        get.addHeader("KEY", "VALUE");
28        get.addHeader("X-M2M-Origin:", originator);
29        get.addHeader("Content-Type", "application/xml");
30        try {
31            // send request
32            CloseableHttpResponse reqResp = client.execute(get);
33            response.setStatusCode(reqResp.getStatusLine().getStatusCode());
34            response.setRepresentation(IOUtils.toString(reqResp.getEntity()
35                    .getContent(), "UTF-8"));
36        } catch (IOException e1) {
37            e1.printStackTrace();
38        } finally {
39            client.close();
40        }
```

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Console  Terminal

<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (08-Dec-2019, 10:03:16 AM)

Response [representation=<?xml version="1.0" encoding="UTF-8"?>
<m2m:cb xmlns:m2m="http://www.onem2m.org/xml/protocols" xmlns:hd="http://www.onem2m.org/xml/protocols/homedomain" rn="mn-name">
   <ty>5</ty>
   <ri>/mn-cse</ri>
   <ct>20191125T145040</ct>
   <lt>20191125T145040</lt>
   <acpi>/mn-cse/acp-691247325</acpi>
   <cst>1</cst>
   <csi>mn-cse</csi>
   <srt>1 2 3 4 5 9 14 15 16 17 23 28</srt>
   <poa>http://127.0.0.1:8282/</poa>
</m2m:cb>
, statusCode=200]
```

ANNEXURE-2 FOR 2.1/STEP-6 MAPPER DEVELOPMENT

Annexure-2.1

MapperInterface.java | Mapper.java | Client.java | Main.java | MapperTest.java

```
16 public class MapperTest {
17
18     public static void main(String[] args) {
19         MapperInterface mapper = new Mapper();
20
21         // example to test marshal operation
22         AE ae = new AE();
23         ae.setRequestReachability(false);
24         //System.out.println(mapper.marshal(System.out));
25
26         String resp = mapper.marshal(ae);
27         System.out.println(resp);
28
29
30         // TODO test unmarshal
31         // get the XML representation, parse it with unmarshal operation
32         System.out.println(mapper.unmarshal(resp));
33
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console | Terminal

```
<terminated> MapperTest [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (08-Dec-2019,
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols">
    <rr>false</rr>
</m2m:ae>

org.eclipse.om2m.commons.resource.AE@4232c52b
```

ANNEXURE-3

MapperInterface.java | Mapper.java | Client.java | Main.java | MapperTest.java

```
1  package fr.insat.om2m.tp2.test;
2  import java.io.IOException;
16
17 public class Main {
18
19     public static void main(String[] args) {
20
21         // TODO test client
22         // ...
23
24         String url = "http://127.0.0.1:8080/~/mn-cse";
25         String originator = "admin:admin";
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console | Terminal

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (08-Dec-2019, 12:59:15 PM)
POST http://127.0.0.1:8080/~/mn-cse HTTP/1.1
Response [representation=<?xml version="1.0" encoding="UTF-8"?>
<m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" xmlns:hd="http://www.onem2m.or
   <ty>2</ty>
   <ri>/mn-cse/CAE793930710</ri>
   <pi>/mn-cse</pi>
   <ct>20191208T125918</ct>
   <lt>20191208T125918</lt>
   <acpi>/mn-cse/acp-691247325</acpi>
   <et>20201208T125918</et>
   <api>app-sensor</api>
   <aei>CAE793930710</aei>
   <rr>true</rr>
</m2m:ae>
, statusCode=201]
/mn-cse/CAE793930710
CAE793930710
```

127.0.0.1:8282/webpage/welcome/index.html?context=/~&cseId=mn-cse

http://127.0.0.1:8282/~/mn-cse/CAE793930710

- mn-name
  - acp_admin
  - LAMP_0
  - LAMP_1
  - LAMP_ALL
  - SmartMeter
  - LuminositySensor
  - TemperatureSensor
  - SmartMeter-1
  - SmartMeter-2
  - ae_370102626
  - ae_204164331
  - ae_132378006
  - ae_95455611
  - ae_463067447
  - ae_793930710

ANNEXURE-4

Mapper.java | Client.java | Main.java | MapperTest.java | create_ae.xml | create_cnt.xml | create_cin.xml

```
27          String type = "ty=2";
28          String representation = RequestLoader.getRequestFromFile("create_ae.xml");
29
30          String cnt_type = "ty=3";
31          String cnt_representation = RequestLoader.getRequestFromFile("create_cnt.xml");
32
33          String cin_type = "ty=4";
34          String cin_representation = RequestLoader.getRequestFromFile("create_cin.xml");
35
36          Client newClient = new Client();
37          Response clientResponse = new Response();
38          MapperInterface mapper = new Mapper();
39
40          try {
41
42              // (1) Creating AE
43              clientResponse = newClient.create(url,representation,originator,type);
44              //System.out.println(clientResponse);
45              AE ae = (AE) mapper.unmarshal(clientResponse.getRepresentation());
46              System.out.println(ae.getResourceID());
47              System.out.println(ae.getAEID());
48
49
50              // (2) Creating CNT
51              String cnt_url = url + "/in-name/" + ae.getName();
52              clientResponse = newClient.create(cnt_url,cnt_representation,originator,cnt_type);
53              //System.out.println(clientResponse);
54              Container cntr = (Container) mapper.unmarshal(clientResponse.getRepresentation());
55              System.out.println(cntr.getResourceID());
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (08-
POST http://127.0.0.1:8080/~/in-cse HTTP/1.1
/in-cse/CAE916352415
CAE916352415
POST http://127.0.0.1:8080/~/in-cse/in-name/ae_916352415 HTTP/1.1
/in-cse/cnt-911731103
```

ANNEXURE-5

Mapper.java | Client.java | Main.java | MapperTest.java | create_ae.xml | create_cnt.xml | create_cin.xml

```
42              // (1) Creating AE
43              clientResponse = newClient.create(url,representation,originator,type);
44              //System.out.println(clientResponse);
45              AE ae = (AE) mapper.unmarshal(clientResponse.getRepresentation());
46              System.out.println(ae.getResourceID());
47              System.out.println(ae.getAEID());
48
49
50              // (2) Creating CNT
51              String cnt_url = url + "/in-name/" + ae.getName();
52              clientResponse = newClient.create(cnt_url,cnt_representation,originator,cnt_type);
53              //System.out.println(clientResponse);
54              Container cntr = (Container) mapper.unmarshal(clientResponse.getRepresentation());
55              System.out.println(cntr.getResourceID());
56
57
58              // (3) Creating CIN
59              String cin_url = cnt_url + "/" + cntr.getName();
60
61              // --------- creating object & serialize using obix
62              ContentInstance descriptor = new ContentInstance();
63              Obj obj = new Obj();
64              obj.add(new Str("name", "TemperatureSensor"));
65              obj.add(new Str("location", "Home"));
66              obj.add(new Str("unit", "celsius"));
67
68              descriptor.setContent(ObixEncoder.toString(obj));
69              descriptor.setContentInfo("application/xml");
70              System.out.println("Creating CIN");
71              clientResponse = newClient.create(cin_url,mapper.marshal(descriptor),originator,cin_type);
72              System.out.println(clientResponse);
```

```
Markers Properties Servers Data Source Explorer Snippets Console Terminal
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (08-Dec-2019, 6:40:26 PM)
POST http://127.0.0.1:8080/~/in-cse HTTP/1.1
/in-cse/CAE467145998
CAE467145998
POST http://127.0.0.1:8080/~/in-cse/in-name/ae_467145998 HTTP/1.1
/in-cse/cnt-968844396
Creating CIN
POST http://127.0.0.1:8080/~/in-cse/in-name/ae_467145998/cnt_968844396 HTTP/1.1
Response [representation=<?xml version="1.0" encoding="UTF-8"?>
<m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols" xmlns:hd="http://www.onem2m.org/xml/protocols/homedomain" rn="cin_551751705">
   <ty>4</ty>
   <ri>/in-cse/cin-551751705</ri>
   <pi>/in-cse/cnt-968844396</pi>
   <ct>20191208T184029</ct>
   <lt>20191208T184029</lt>
   <st>0</st>
   <cnf>application/xml</cnf>
   <cs>129</cs>
   <con>&lt;obj>
  &lt;str name=&quot;name&quot; val=&quot;TemperatureSensor&quot;/>
  &lt;str name=&quot;location&quot; val=&quot;Home&quot;/>
  &lt;str name=&quot;unit&quot; val=&quot;celsius&quot;/>
&lt;/obj>
</con>
</m2m:cin>
, statusCode=201]
```

ANNEXURE-6

Mapper.java | Client.java | Main.java | MapperTest.java | create_ae.xml | create_cnt.xml | Response.java

```
57              // (3) Creating CIN
58              String cin_url = cnt_url + "/" + cntr.getName();
59              // --------- creating object & serialize using obix
60              ContentInstance descriptor = new ContentInstance();
61              Obj obj = new Obj();
62              obj.add(new Str("name", "TemperatureSensor"));
63              obj.add(new Str("location", "Home"));
64              obj.add(new Str("unit", "celsius"));
65              obj.add(new Str("value", "24"));
66
67              descriptor.setContent(ObixEncoder.toString(obj));
68              descriptor.setContentInfo("application/xml");
69              System.out.println("Creating CIN");
70              clientResponse = newClient.create(cin_url,mapper.marshal(descriptor),originator,cin_type);
71
72              // (4) Retrieve CIN - decode & print desired sensor values
73
74              Client client =  new Client();
75
76              ContentInstance cin_retrieved = (ContentInstance) mapper.unmarshal(clientResponse.getRepreser
77              String retrieve_url = "http://127.0.0.1:8080/~" + cin_retrieved.getResourceID();
78
79              Response res = client.retrieve(retrieve_url, "admin:admin");
80              ContentInstance cin = (ContentInstance) mapper.unmarshal(res.getRepresentation());
81
82              Obj obj_retrieved = ObixDecoder.fromString(cin.getContent());
83              System.out.println("Retrieving CIN Value");
84              System.out.println(cin_retrieved.getResourceID());
85              System.out.println(obj_retrieved.get("location"));
86              System.out.println(obj_retrieved.get("unit"));
87              System.out.println(obj_retrieved.get("value"));
```

```
Markers Properties Servers Data Source Explorer Snippets Console Terminal
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (08-Dec-2019, 7:36:13 PM)
POST http://127.0.0.1:8080/~/in-cse HTTP/1.1
/in-cse/CAE981773468
POST http://127.0.0.1:8080/~/in-cse/in-name/ae_981773468 HTTP/1.1
/in-cse/cnt-468252171
Creating CIN
POST http://127.0.0.1:8080/~/in-cse/in-name/ae_981773468/cnt_468252171 HTTP/1.1
Retrieving CIN Value
/in-cse/cin-863908623
Home
celsius
24
```