INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

INNOVATIVE SMART SYSTEMS

Intergiciel pour l'Internet des Objets

Rapport de TP – 5 ISS INSA Toulouse

Julien Chouvet - Wael Ben Jemaa
Encadrant : **Karima Khadir**
Groupe : 5ISS - B2

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

INTRODUCTION

Aujourd'hui, la troisième génération du web ou brièvement web 3.0 est un mélange de différents web avec des differents caractéristiques particulières, les webs : sociale, programmable, physique, temps réel et web sémantique sont ses principaux affiliés. L'IoT ou plus précisément l'internet des objets se trouve au croisement de ces webs.

Brièvement, l'internet des objets est un ensemble de réseau d'objets physiques ou virtuels qui communique via des réseaux sans fil.

OneM2M est un standard développé par consortium d'organisme internationaux de standardisation et d'industriels qui sert comme un moyen de communication pour l'IoT et qui propose un standard pour plateforme horizontale.

Tout au long de ce rapport, nous allons expliciter les différents principes du standard oneM2M, déployer une architecture IoT en utilisant OM2M, mettre en place un système de réseau de capteurs et développer une application composite entre divers technologies en se basant sur un intergiciel standardisé.

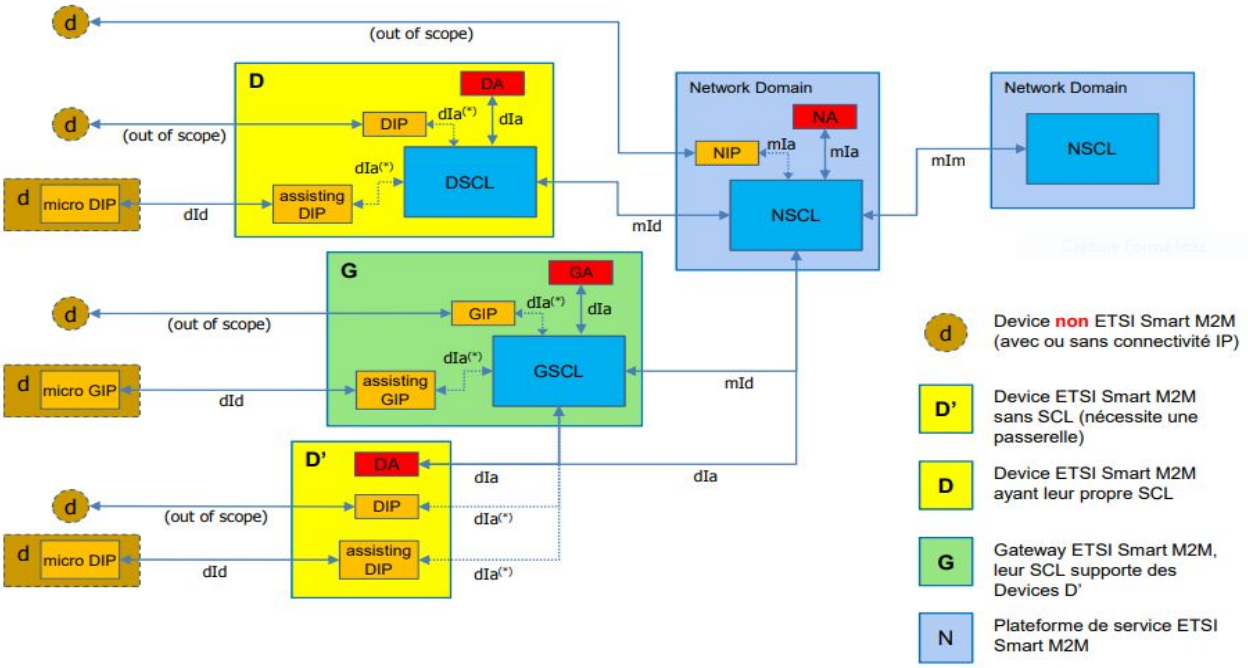
1. SAVOIR POSITIONNER LES STANDARDS PRINCIPAUX DE L'INTERNET DES OBJETS

*Expliquez brièvement le principe du standard **oneM2M** et comment il se positionne vis à vis des autres standards et technologies déjà existantes.*

Comme on a déja précisé dans l'introduction, oneM2M est un standard développé par un consortium composé de plusieurs organismes de standardisation dans le monde dont le but est de travailler ensemble à la proposition d'un standard global pour le machine to machine et l'internet des objets.

OneM2M est un standard qui aborde des spécifications qui couvrent des domaines comme les exigences, l'architecture, les API, les mécanismes de sécurité et les processus d'adaptation à des protocoles répandus comme CoAP, MQTT et HTTP.

L'architecture oneM2M suit les standards ETSI, ces standards sont représentés dans la figure suivantes:



Standards ETSI

Pour plus précisé :

SCL ou Service capabilities layer: est une couche de service horizontale pour les applications M2M afin de faire abstraction de la complexité des dispositifs hétérogènes. On peut distinguer deux types de devices:

- **Legacy devices :** Devices qui n'ont pas la couche SCL et nécessitent une gateway pour communiquer avec le réseau M2M.
- **M2M Device :** Un device qui a une couche SCL (DSCL) qui peut être interrogé par des applications M2M par l'intermédiaire d'une interface ouverte.
- **M2M Gateway :** une machine qui exécute une gateway SCL (GSCL).
- **M2M Network :** fournit un réseau SCL M2M (NSCL) qui peut être interrogé par les applications M2M via une interface ouverte.

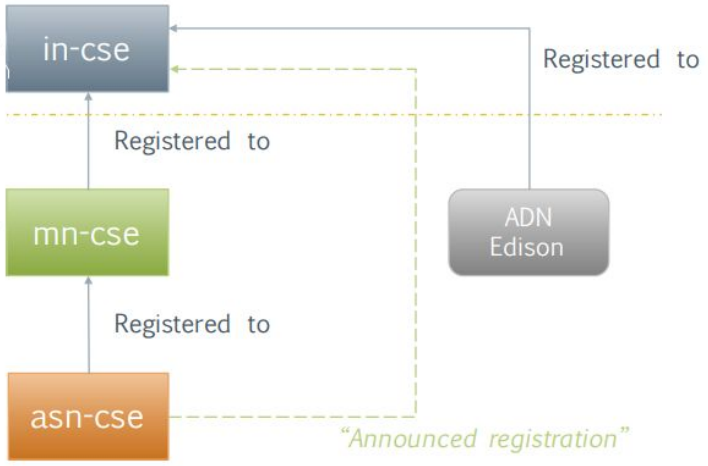
2. Déployer une architecture conforme à un standard et mettre en place un système de réseau de capteurs aux services

2.1. Déployer et configurer une architecture IoT en utilisant OM2M

*Expliquez comment vous avez déployé une **architecture IoT** avec **OM2M** : les types de nœuds utilisés, les entités en interaction avec l'intergiciel, à quel niveau les objets / capteurs interagissent avec le système ainsi que les différentes applications en interaction avec ce dernier, etc. **(TP 1 + 2 +3)***

En OM2M il y a différents types de noeuds utilisés :

- Les noeuds capables de générer les souscriptions à leurs réseau et qui possèdent des commun service entity (cse), on peut distinguer:
  - **Les infrastructure node (IN) :** qui correspondent à des serveurs, qui implémentent toutes les fonctionnalités définies dans les spécifications oneM2M telles que l'enregistrement, la gestion des données, la souscriptions et la notification, etc.
  - **Les middle node (MN) :** qui correspondent à des gateway et qui fournissent à la fois une interface avec les capteurs et actionneurs.
  - **Les Application Service Nodes (ASN) :** qui sont des noeuds qu'on ne peut pas enregistrer, des noeuds qui ne possèdent pas de cse:
  - **Les ADN :** se sont des noeuds finaux, ou plus précisément se sont des noeuds correspondant à des devices qu'on ne peut pas enregistrer.

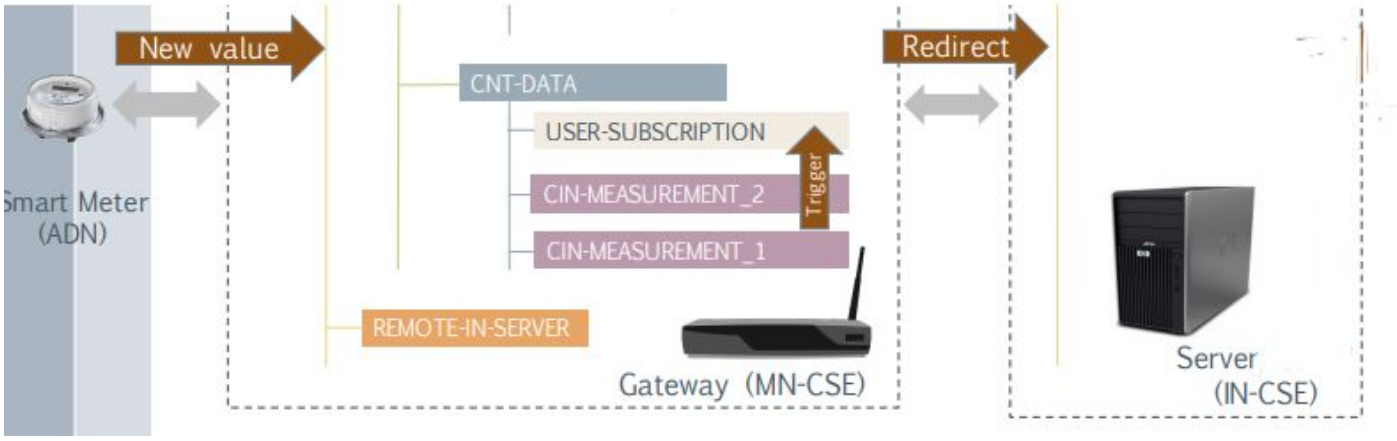


Différents types des noeuds dans une architecture OM2M

OM2M génère une API REST pour la création et le management des ressources , ce qui permet à chaque ressource d'avoir son unique *URI*, ainsi chaque requête contient tous les états d'application nécessaires pour traiter cette requête.

Les objets/capteurs communiquent avec le MN en se basant sur l'API REST, et le *MN* stocke les données au *CIN*.

Les nouveaux états des devices sont placés dans les *CIN*, et du *CNT* on peut envoyer des requêtes aux devices.



Communications au sein de l'architecture déployé sous OM2M

2.2. Interagir avec les objets en utilisant une architecture REST

Expliquez comment vous avez pu interagir avec des objets et visualiser les données envoyées par ces derniers (notamment les ressources générées par IPE Sample, lampes simulées). Expliquer les différentes ***ressources oneM2M*** *que vous avez manipulées (****CSE-BASE, AE, CNT, CIN, SUB, REMOTE CSE****) et la façon d'interagir avec ces dernières (client HTTP + serveur).* ***(TP 1 + 2)***

Afin d'entamer une connection sur la plateforme OM2M il faut tout d'abord lancer les deux CSE (MN et IN), puis avec l'adresse http://localhost:8080/webpage, nous obtenons la page d'accueil de la plateforme OM2M avec le MN-CSE qui est authentifié directement sous l'arbre de IN-CSE.

Vue de noeud in-cse

2.2.1 Les Ressources de base de ONEM2M:

La ressource la plus importante est le CSE (Commun Service Entity), c'est la ressource racine, qui représentera l'arborescence des ressources sur un nœud, cette ressource permet de regrouper différentes informations sur un CSE et de stocker de nouvelles ressources sous ce dernier, c'est-à-dire d'un point de vue hiérarchique, la base CSE sera la ressource mère, alors il y aura des sous-ressources, parmi ces sous ressources il y'a:

- **Application Entity (AE) :** plusieurs AE peuvent être enregistrées sous un CSE, une AE représente la liaison logique entre une application et le CSE, cette ressource a diverses utilisations, en particulier elle permet d'enregistrer une application distante ou locale dans la couche service CSE.
- **Container (CNT) :** ils permettent de structurer les données dans l'arborescence des ressources.
- **Content Instance (CIN) :** sont à proprement parler les ressources qui permettent de stocker des valeurs dans la plateforme.

Everything is connected! La figure suivante expose la hiérarchie des ressources OM2M, chaque ressource conserve un URI de sa sous-ressource ainsi que l'URI de sa ressource parent.

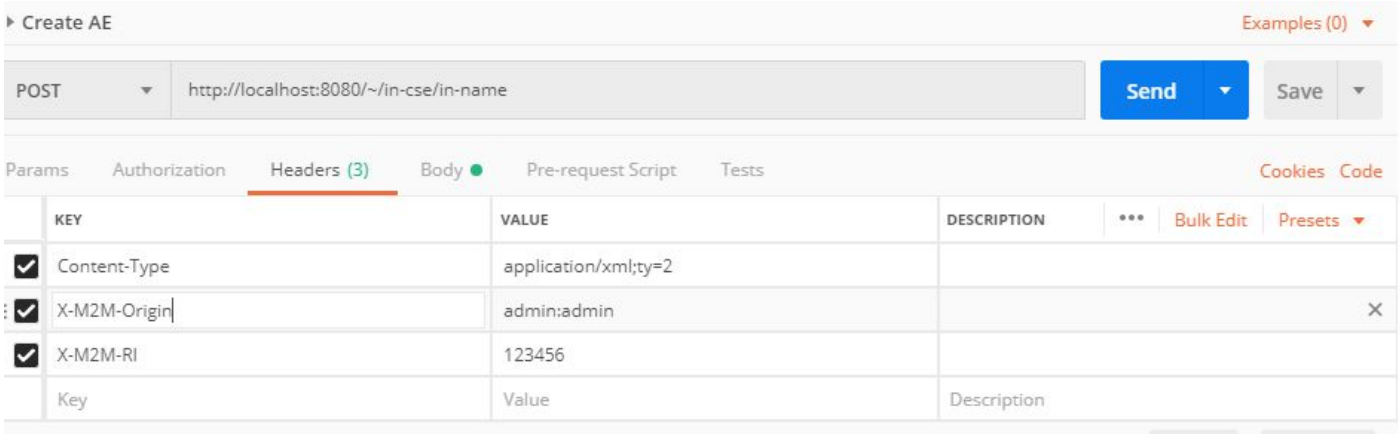
Ressources de base de ONEM2M

2.2.2 Interagir avec les ressources:

Maintenant on va expliciter comment ces ressources peuvent être créer ou récupérer via des requêtes HTTP:

- **GET :** pour récupérer une instance de ressource on doit procéder par une requête GET, nous allons récupérer un *content* stocké sur la plateforme.
- **POST :** Afin de stocker une valeur dans une plateforme oneM2M, il faut créer une nouvelle instance de contenu, donc créer une ressource. Cela passe par une opération POST sur l'URI du parent sous lequel on veut créer une ressource.

Dans un premier lieu on a utilisé le logiciel POSTMAN-REST CLIENT pour interagir avec nos ressources.



Header d'une requête HTTP

Le POST est exécuté sur L'URI du parent sous lequel on veut créer notre ressource. Une fois la requête envoyée et exécutée sur la plateforme, on a sur la figure un exemple de réponse que l'on peut recevoir. Différents attributs sont présents dans cette CIN, comme la *Ressource Name* (RN), le type de ressource (TY) (4 pour CIN), l'identifiant de la ressource (RI), l'identifiant du parent (PI), la date de création de la ressource et la date de la dernière modification (CT & LT).

POSTMAN: requête POST

2.2.3 Création du client REST sous eclipse :

Au cours de deuxième TP on a développé une classe client pour gérer l'arborescence des ressources oneM2M en se basant sur l'API Httpclient. Ensuite on stocke les réponse de la plateforme oM2M dans une classe Response.

Afin de pouvoir implémenter les RESTful méthodes: RETRIEVE, UPDATE, CREATE et DELETE on a implémenté l'interface *ClientInterface* dans la classe Client.

On a utilisé la méthode *getRequestFromFile* qui nous a permis de récupérer la représentation d'un fichier xml afin de pouvoir tester les méthodes implémentés (figure suivante).

```
10
11    public static void main(String[] args) throws IOException {
12        String originator = "admin:admin";
13
14        //Retrieve the attributes of the CSE Base with child resources (query string rcn=5)
15        Response rep = new Response();
16        Client cl = new Client();
17        rep = cl.retrieve("http://127.0.0.1:8181/~/in-cse", originator);
18        System.out.println(rep);
19
20        //Register an "appTest" resource (create an AE)
21        //Get the "representation" parameter for the create method, from the XML file
22        String representation = RequestLoader.getRequestFromFile("create_ae.xml");
23        //create the "appTest" resource
24        rep = cl.create("http://127.0.0.1:8181/~/in-cse", representation, originator, "2");
25        System.out.println(rep);
26
```

Création d'un AE dans le IN

2.2.4 XML Mapping :

JAXB Mapping:

JAXB est un outil qui permet de structurer et de décomposer des ressources(marshalling et unmarshalling). La méthode Marshall permet de structurer les objets ou les classes dans une structure XML, alors que *Unmarshall* fait l'opération inverse.

On a utilisé ces deux méthodes en les implémentant dans la classe *MapperTest* en créant une ressource et en la "convertissant" en XML, puis en retrouvant l'objet à partir du fichier XML.

```
23          // example to test marshal operation
24          AE ae = new AE();
25          ae.setRequestReachability(false);
26          ae.setAppName("test_marshal-apn");
27          ae.setAppID("test-marshal-api");
28
29          String xml_marshal = mapper.marshal(ae);
30          System.out.println(xml_marshal);
31
32          // TODO test unmarshal
33          // get the XML representation, parse it with unmarshal operation
34          Object unmarshal = mapper.unmarshal(xml_marshal);
35          //System.out.println(unmarshal);
```

Implémentation de JAXB Mapping

Obix Mapping:

On a utilisé la bibliothèque Obix afin de générer des représentation Obix/XML, On utilise notre classes Client et Mapper avec l'API Obix afin de créer des ressources sans les identifier par des nom ou id.

OM2M CSE Resource Tree

http://localhost:8181/~/in-cse/CAE167637778

- in-name
  - acp_admin
  - acpae-545993829
  - acpae-672640723
  - acpae-167637778

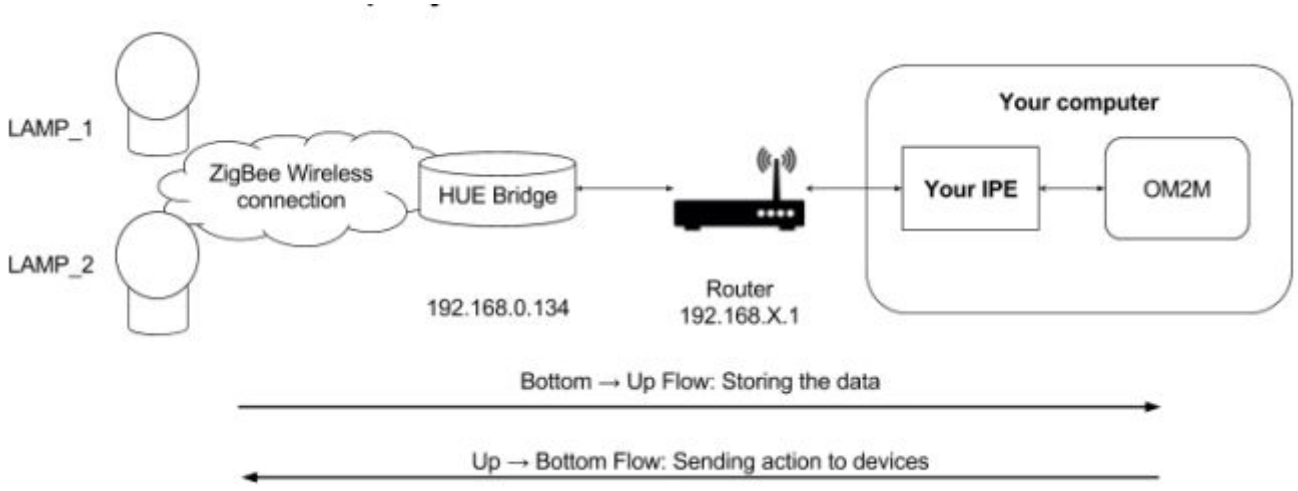
Ressources générées par Obix Mapping

2.3. Intégrer une nouvelle / autre technologie dans une architecture IoT

Expliquez le principe d'Interworking Proxy Entity de oneM2M ainsi que l'architecture utilisée pour intégrer dans l'intergiciel une nouvelle technologie non nativement compatible avec le standard. (Architecture logicielle, nœuds oneM2M utilisés, ressources oneM2M utilisées pour faire l'interface.) ***(TP 3)***

Lors du troisième TP de ce module, le but était d'intégrer une technologie existante et non nativement compatible avec le standard. Pour cela, nous avons utilisé une lampe Philips HUE qui utilise la technologie sans fil *Zigbee* pour communiquer avec un bridge fournissant une *API REST*. Afin, de pouvoir intégrer la gestion de cette lampe dans une interface OM2M, nous devions créer un *Interworking Proxy Entity (IPE)* qui joue le rôle d'interface entre les devices *HUE* et le standard oneM2M.

L'architecture déployée est donc la suivante :



Architecture deployée

Les classes HUE

Le but de notre IPE est de récupérer les informations provenant des lampes afin de les retransmettre sur la plateforme OM2M et également de permettre l'exécution d'opérations sur les lampes à partir du système oneM2M. Pour cela, nous nous appuyons sur un *SDK* Java fourni par Philips et contenant deux packages Package HUE et Package IPE. On notera notamment que la classe *HueUtilè* intégre des méthodes permettant la connexion avec le bridge et l'échange d'informations, ainsi que la classe *HueSdkListener* dont le rôle est d'écouter les évènements intervenant lors de l'utilisation du système et d'agir en conséquence.

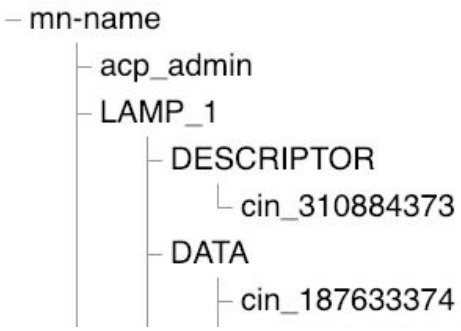
En se basant sur ces classes et en y ajoutant certaines fonctionnalités, nous avons pu mettre en place notre IPE afin d'interagir avec la lampe HUE.

Connection au bridge et création des ressources sur la plateforme *oneM2M*

Comme nous pouvons le pouvoir sur la figure représentant l'architecture mise en place, notre PC et le bridge *HUE* se trouve sur le même réseau, défini par le routeur. La première étape est de se connecter au bridge. Pour se faire, nous devons renseigner l'adresse IP du bridge.

Une fois la connexion établie, la méthode *onBridgeConnected* de la classe *HueSdkListener* va alors créer les ressource représentant les lampes connectées au bridge dans la plateforme oneM2M. Pour cela,

elle fait appel à la méthode *getAllLights* qui détermine l'ensemble des lampes connectées à notre bridge, puis pour chaque lampe trouvée, invoque la méthode *createResources* qui va effectuer une requête *HTTP POST* permettant la création de la ressource sur la plateforme OM2M. Nous obtenons ainsi une AE "Lamp_1" avec deux *containers* "DESCRIPTOR" et "DATA".



Création des ressources correspondantes à la lampe HUE dans OM2M

Récupération des informations de la lampe

Une fois connecté et les ressources créées, le *listener onCacheUpdate* de la classe *HueSdkListener* va mettre à jour les informations de la lampe sur OM2M , à chaque fois que celles-ci seront modifiées (changement de couleur, de luminosité, etc). Pour ce faire, cette méthode va recevoir une notification à chaque fois le cache du bridge est mis à jour. Elle va alors récupérer cette information, puis mettre à jour les informations sur OM2M grâce à la fonction *pushData* qui va effectuer une requête *HTTP* sur la ressource concernée.

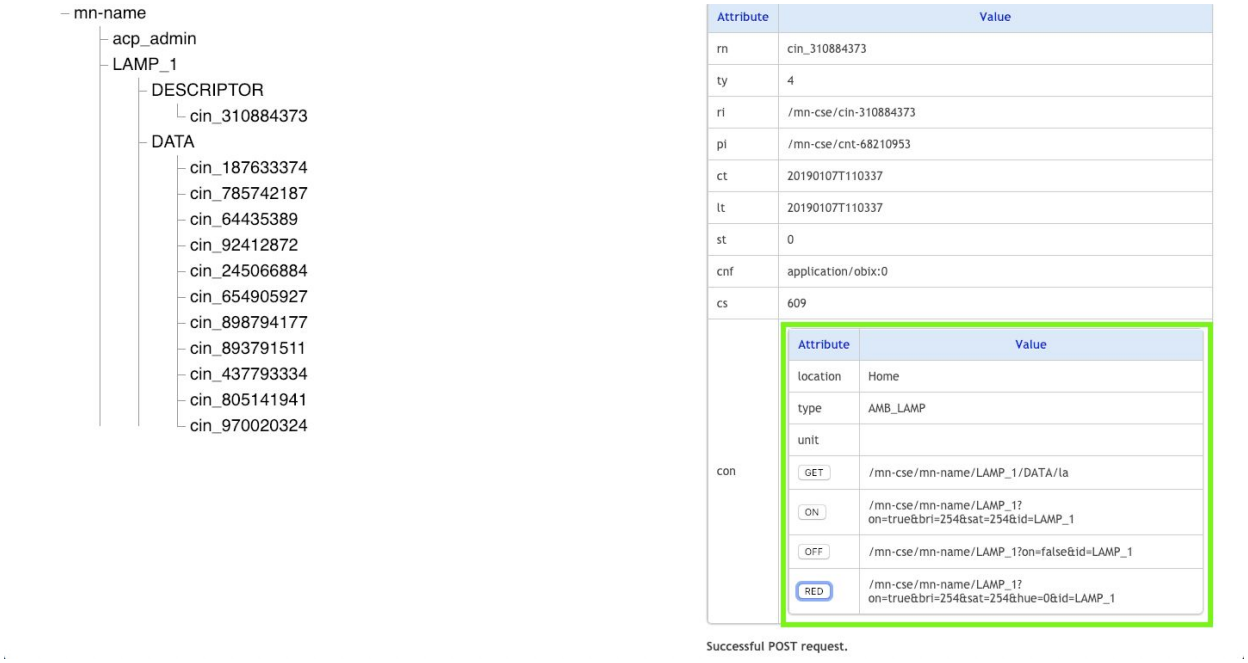
Ainsi, les informations disponibles sur la plateforme sont continuellement mises à jour. Il est alors possible de récupérer des informations sur une lampe grâce à la méthode *getStates* de la classe *HueUtil*. Cette dernière va effectuer une requête *HTTP GET*, à l'adresse correspondante à la ressource de la lampe qu'on lui passe en paramètre, afin de récupérer les informations.

Agir sur la lampe

Dans cette dernière partie, le but est de contrôler la lampe en lui envoyant des commandes permettant, par exemple, de changer la couleur, l'intensité lumineuse, etc.

Dans un premier temps, nous avons utilisé la méthode *updateDevice*, de la classe *HueUtil*, qui prend en paramètre l'ID de la lampe sur laquelle appliquer les modification et les changements à appliquer.

Ensuite, nous avons utilisé le *controller* qui permet de modifier l'état de la lampe directement depuis l'interface *OM2M.*



Modification de l'état de la lampe depuis OM2M

3. Déployer une application composite entre divers technologies grâce à NODE-RED en se basant sur un intergiciel standardisé

Expliquez brièvement le principe de fonctionnement de NODE-RED.
Donnez les exemples d'application(s) que vous avez déployée(s) avec NODE-RED (capture écran du flux et explications). Faites également un export de votre application et joignez le(s) fichier(s) à votre rapport. ***(TP 4)***

Lors du dernier TP, le but était de créer une application de haut niveau permettant de gérer des capteurs et des "actionneurs" de différents types et technologies, afin de mettre en scène un *use-case* concret.

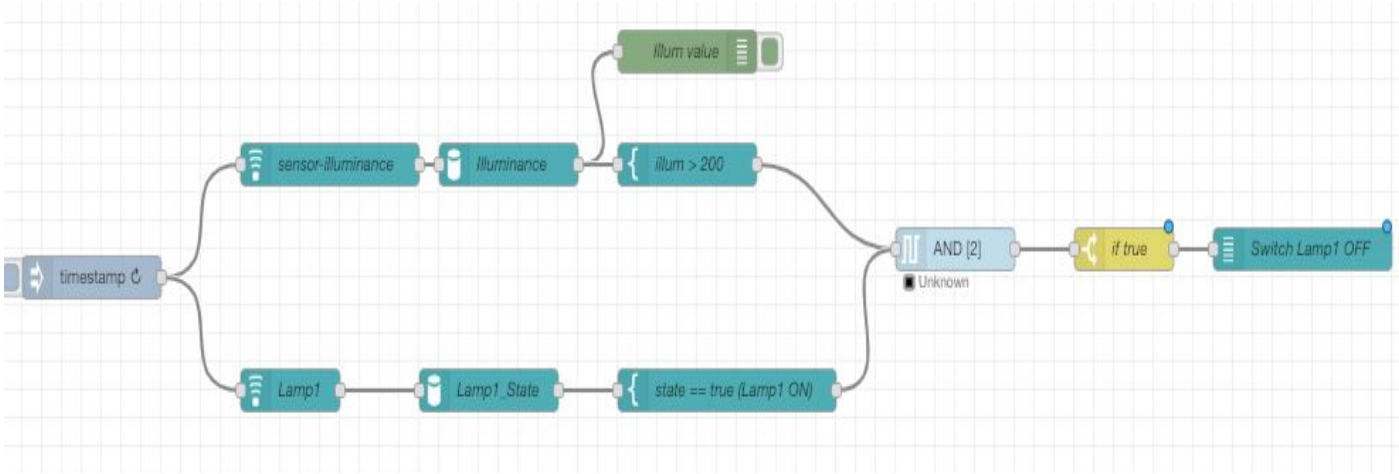
Pour cela, nous avons bien entendu utilisé le standard oneM2M qui permet l'interopérabilité de plusieurs technologies, comme vue précédemment.

Ainsi, pour développer cette application haut niveau, nous pouvions le faire en utilisant le client *HTTP* développé lors des TP précédents, mais cette technique reste complexe et requiert pas mal de temps à mettre en place. C'est pour cela que nous avons utilisé *Node-RED*.

Node-RED est un outil de programmation qui permet de lier des *devices* (capteurs, lampes, etc), des API et des services haut niveau. La programmation se fait simplement, à l'aide de blocs que l'on relie entre eux afin de créer des liens, et donc des fonctionnalités, entre les *devices* et les services.

Après avoir installé *Node-RED* et le framework *IDE-IoT*, permettant l'utilisation de *Node-RED* avec OM2M pour des applications IoT, nous avons pu commencer le développement de notre application. Le but de cette dernière été de créer un *use-case* mettant en jeu un capteur Fibaro, communiquant en *ZWave* avec une *Raspberry Pi* hébergeant le *MN-CSE (mn-zwave)* de *OM2M*, et des lampes virtuelles qu'il est possible de virtualiser en lançant le *bundle "org.eclipse.om2m.ipe.sample_1.1.0.20180313-1100"*.

Ainsi, sur *Node-RED*, nous avons créé le flow suivant :



Flow Node-RED

Dans ce dernier on peut voir que périodiquement (ici *timestamp* réglé à 3sec), on va, d'une part, chercher la valeur d'illuminance renvoyée par le capteur Fibaro et la comparer à une valeur seuil (ici 200), et d'autre part, regarder si la lampe est allumée ou non. Si la valeur d'illuminance est supérieure à 200 et que la lampe est allumée, alors on décide de l'éteindre.

L'export de cette application est disponible en annexe.

CONCLUSION

Ces TP nous auront permis d'étudier en détail le standard oneM2M ainsi que son application sur la plateforme OM2M développée par Eclipse. Nous avons pu déployer une architecture oneM2M et par du code Java, utilisant l'API *REST,* nous sommes parvenus à créer les ressources au sein de OM2M et à interagir avec des *devices* du monde réel, notamment grâce aux *SDK* fournis par les constructeurs. Enfin, nous avons pu voir que le standard oneM2M présente une solution adaptée à l'hétérogénéité des technologies existantes.

ANNEXE

Application Node-RED

[{"id":"f5027137.401be","type":"tab","label":"Flow2","disabled":false,"info":""},{"id":"528db203.57b9bc","type":"NamedSensor","z":"f5027137.401be","name":"sensor-illuminance","platform":"13194bd.ddf6eb4","sensor":"84b00a7.14621f8","container":"ILLUMINANCE","cntInstance":"/la","x":410,"y":280,"wires":[["ef0990be.23374"]]},{"id":"c022b334.6d6f7","type":"inject","z":"f5027137.401be","name":"","topic":"","payload":"","payloadType":"date","repeat":"3","crontab":"","once":false,"onceDelay":0.1,"x":170,"y":360,"wires":[["528db203.57b9bc","3e9f522a.9d4a0e"]]},{"id":"ef0990be.23374","type":"DataExtractor","z":"f5027137.401be","name":"Illuminance","viewtype":"data","viewunid1":"","viewunid2":"","x":590,"y":280,"wires":[["e5d322f2.886f6","e01f6d38.c246"]]},{"id":"3e9f522a.9d4a0e","type":"NamedSensor","z":"f5027137.401be","name":"Lamp1","platform":"45d3f3e7.84bdac","sensor":"b61c38a5.051678","container":"DATA","cntInstance":"/la","x":370,"y":440,"wires":[["e31ae41b.c28058"]]},{"id":"e31ae41b.c28058","type":"DataExtractor","z":"f5027137.401be","name":"Lamp1_State","viewtype":"state","viewunid1":"","viewunid2":"","x":570,"y":440,"wires":[["55151376.75c3ec"]]},{"id":"87d71720.9671a8","type":"NamedActuator","z":"f5027137.401be","platform":"45d3f3e7.84bdac","name":"Switch Lamp1 OFF","actuator":"b61c38a5.051678","command":"op=setOff&lampid=LAMP_1","x":1390,"y":340,"wires":[]},{"id":"e5d322f2.886f6","type":"debug","z":"f5027137.401be","name":"Illum value","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","x":770,"y":200,"wires":[]},{"id":"e01f6d38.c246","type":"SimpleCondition","z":"f5027137.401be","operator":">","value_c":"200","inputType":"msg","input":"","name":"illum > 200","x":770,"y":280,"wires":[["d63c1ea1.cca38"]]},{"id":"55151376.75c3ec","type":"SimpleCondition","z":"f5027137.401be","operator":"=","value_c":"true","inputType":"msg","input":"","name":"state == true (Lamp1 ON)","x":810,"y":440,"wires":[["d63c1ea1.cca38"]]},{"id":"d63c1ea1.cca38","type":"BooleanLogic","z":"f5027137.401be","name":"","operation":"AND","inputCount":"2","topic":"result","x":1040,"y":340,"wires":[["248a040c.f4e0dc"]]},{"id":"248a040c.f4e0dc","type":"switch","z":"f5027137.401be","name":"if true","property":"payload","propertyType":"msg","rules":[{"t":"true"}],"checkall":"true","repair":false,"outputs":1,"x":1210,"y":340,"wires":[["87d71720.9671a8"]]},{"id":"13194bd.ddf6eb4","type":"xN_CSE","z":"","platform":"mn-zawe","URLBase":"http://192.168.1.120:8181/~/mn-zwave/mn-name","user":"admin","password":"admin"},{"id":"84b00a7.14621f8","type":"AE","z":"","appId":"Zw_FIBARO_MOTION_SENSOR_4248273193-2"},{"id":"45d3f3e7.84bdac","type":"xN_CSE","z":"","platform":"lamp_mn","URLBase":"http://192.168.1.102:8080/~/mn-cse/mn-name","user":"admin","password":"admin"},{"id":"b61c38a5.051678","type":"AE","z":"","appId":"LAMP_1"}]