INTERGICIEL POUR L'INTERNET DES OBJETS

RAPPORT DE TP – 5 ISS INSA TOULOUSE

Le termes intergiciel, aussi appelé middleware, désigne un logiciel qui va permettre des échanges entre applications. Ces différents TP ont eu pour but de nous faire découvrir le standard OM2M. C'est un des nouveaux standards d'intergiciel pour l'IOT et qui veut proposer une solution qui marche pour toutes les situations liées à l'IOT.

Durant ces TP nous avons essayé de comprendre le standard oneM2M et de le mettre en place au travers de plusieurs applications.

Nom 1 : Prié

Prénom 1 : Pierre

Nom 2 : Bayle

Prénom 2 : Axel

Groupe : A

Encadrant : N.SEYDOUX

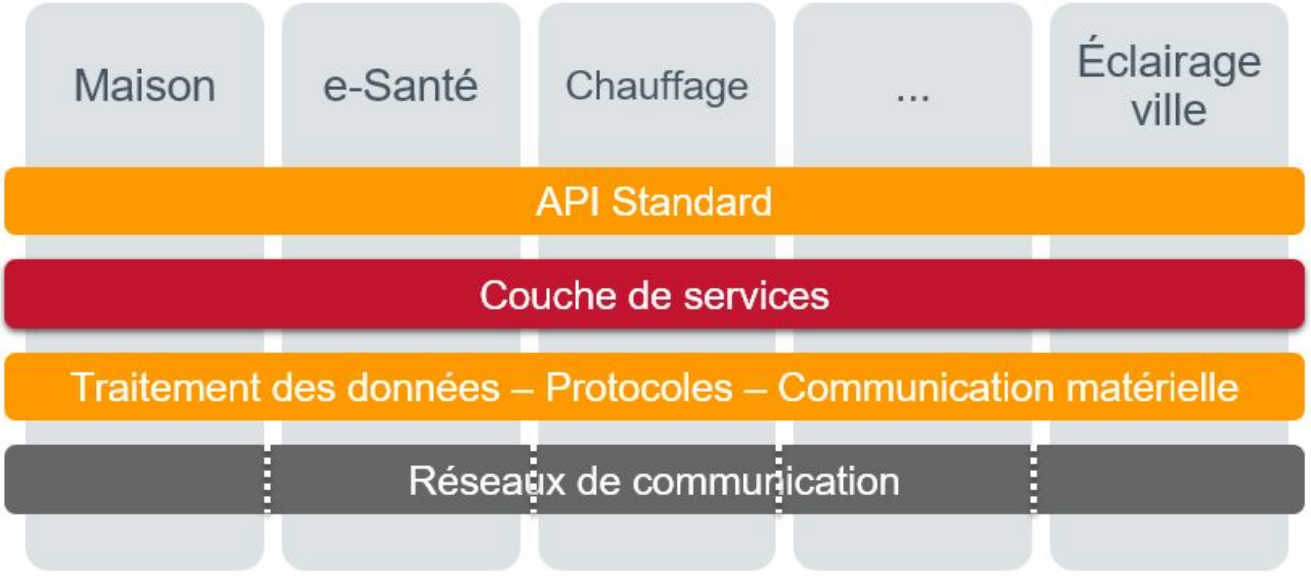
1. Savoir positionner les standards principaux de l'Internet des Objets

Il existe un grand nombre de consortiums et d'alliances industriels dans le domaines des objets connectés. Nous pouvons citer :

- Thread, une alliance industrielle qui propose un standard de communication basée sur IPv6 et 6LOWPAN.
- Homekit, une alliance portée par Apple ayant pour but de l'interconnection des appareils Apple et leurs contrôle par l'assistant personnel Siri.
- Fiware, une alliance européenne qui propose un standard horizontal pour connecter tous types d'équipements.
- OneM2M, un consortium mixte composé d'organismes de standardisation et d'industriels. C'est ce standard que nous avons utilisés au cours des TPs.

L'objectif du standard oneM2M est de créer une interface générique et horizontale qui va permettre de relier tous types d'objets connectés. Là où, à l'heure actuelle, les standards sont propres à un domaine ou une utilisation, oneM2M veut généraliser un processus de communication peut importe les capteurs ou les actionneurs en jeu. Le standard s'efforce également d'avoir un rayonnement mondial de par ses membres (divers organismes de standardisation européens, américains et asiatiques, ainsi que des industriels comme Amazon, Cisco, Huawei, Intel, Interdigital, LG, ou même Orange en France).

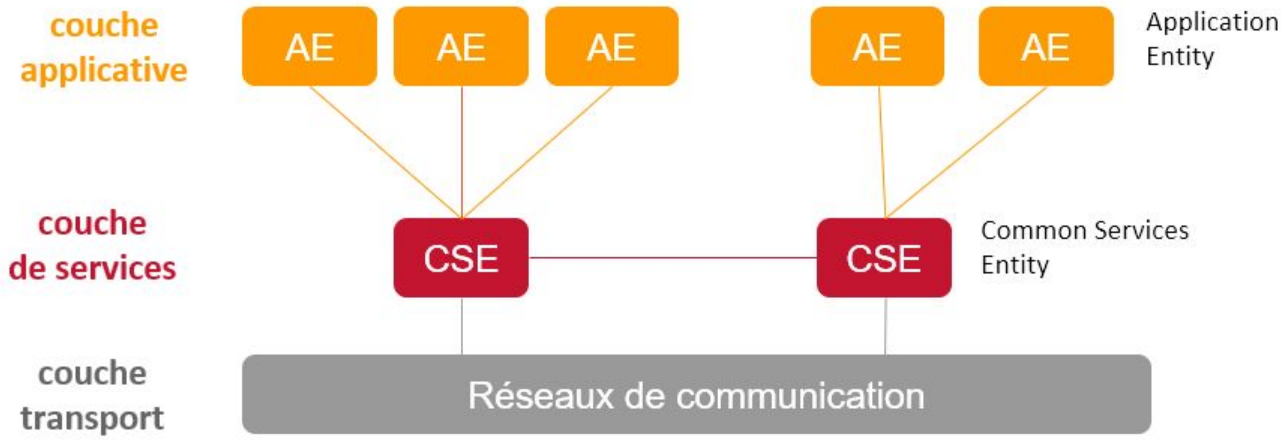
Comme nous l'avons dit, oneM2M se positionne de manière transverse en apportant une vision globale des systèmes à contrôler quels que soient les domaines d'applications. Pour se faire le standard oneM2M est découpé en plusieurs couches comme exposées sur la figure ci-dessous.



Grâce à cela chaque couche peut être développée indépendamment. On n'a pas besoin d'avoir un protocole par types de communications, puisque lors du développement de l'API ou de la couche de service on fait abstraction de la communication.

Le standard apporte une architecture structurée de la manière suivante. Elle se compose d'un ensemble de couches de services, dit Common Services Entity (CSE). C'est à ces CSE que les différentes applications vont être enregistrées sous la forme d'Application Entity (AE). C'est également grâce à elles que les AE vont accéder aux différents services liés au standard. Finalement, le transport des données entre les différentes CSE est réalisé grâce à la couche la plus basse, appelée la couche transport. On note

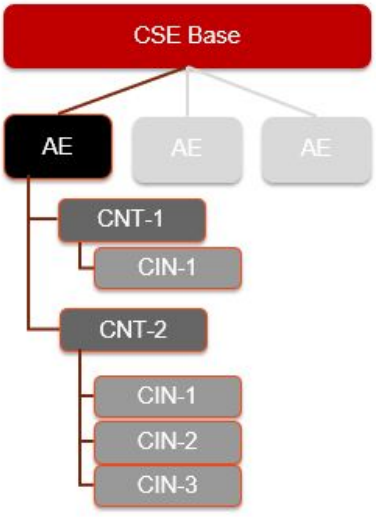
cependant que cette communication entre CSE se fait par abstraction de la couche transport.



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

Au cours de nos TPs, nous avons déployé une architecture IoT reposant sur le standard OM2M. Cette architecture est composée d'un serveur IN-CSE Base (IN pour Infrastructure Node) d'une gateway MN-CSE (MN pour Middle Node) et au cours du dernier TP d'une seconde gateway MN-ZWave-CSE hébergée sur une raspberry. Le serveur IN-CSE et la gateway MN-CSE sont hébergés en local sur nos machines. On note que chacune des gateways sont enregistrées au serveur IN-CSE Base et inversement. Il est donc possible via l'interface web d'OM2M (basée sur AJAX et JS) d'observer les arbres de ressources des différentes gateways et du serveur et de passer facilement de l'un à l'autre.



Comme expliqué en partie 1, il est possible de créer différents types de ressources sous chacun de nos noeuds (IN-CSE, MN-CSE et MN-ZWave-CSE), chacun a un rôle précis, voici les ressources de OM2M :

Ressource	Utilisation(s)
CSE-BASE	Couche de service qui est à la base de l'arborescence d'un projet.
AE	Application Entity, se situe juste sous la couche service et permet de regrouper les données/valeurs reliées à une même application.
CNT	Container, une AE peut avoir plusieurs containers, représentant chacun une information ou une variable.
CIN	Content instance, il se trouve sous le CNT et va porter l'information à proprement parler. C'est dans un CIN qu'on va stocker la valeur de la variable CNT. On peut avoir plusieurs CIN sous un CNT pour étudier par exemple l'évolution temporelle. On note que les CIN ne sont pas prévus pour être mis à jours, vu qu'on crée un nouveau CIN à chaque nouvelle donnée générée/enregistrée.
SUB	C'est un type de CIN qui permet de faire des souscriptions, cela est utile lorsque l'on veut

	automatiser un comportement, sur un changement d'état d'une variable par exemple.
REMOTE - CSE	Représente un CSE distant, le remote CSE est créé quand un CSE distant (par exemple un MN-CSE) est enregistré au CSE local (par exemple le IN-CSE). Des données relatives à ce CSE sont enregistrées comme le point d'accès par exemple.

L'objectif final des 4 TPs est d'arriver, à contrôler automatiquement une lampe HUE Philips, connectée via ZigBee à un routeur, en fonction des données relevées par un capteur "Fibaro", capable notamment de relever la luminosité de la pièce, et connecté via Z-Wave à une raspberry.

Pour se faire, la première étape est de créer une AE pour la lampe HUE Philips à contrôler. Par simplicité, nous avons décidé de créer notre AE_Lampe dans le IN-CSE, de ce fait, pour la suite du projet nous utiliserons uniquement le IN-CSE et le MN-ZWave-CSE. Cependant en toute rigueur, le IN-CSE ne devrait servir que de porte d'entrée aux applications utilisateur et l'AE de la lampe aurait dû être implémenté dans le MN-CSE. Nous avons ajouté deux containers à l'AE_Lampe : un CNT DESCRIPTOR où l'on retrouve l'ensemble des opérations possibles sur la lampe comme récupérer son statut, changer sa couleur ou également éteindre ou allumer la lampe et un CNT DATA regroupant les dernières commandes envoyées à la lampe (pour modifier notamment son état, sa couleur, son intensité..). On retrouve donc dans le CNT DATA, sous la forme de CIN l'historique de toute les commandes envoyées à la lampe.

Par la suite, pour faire l'interface entre OM2M et le bridge où se trouve la lampe HUE, nous avons développé un Interworking Proxy Entity (IPE) spécifique reposant sur la librairie java Philips HUE. Comme expliqué en première partie OM2M étant un standard, l'utilisation d'IPE spécifique est nécessaire pour faire l'interface entre les objets de différentes technologies. L'IPE est, dans notre cas, un servlet HTTP capable d'interroger l'état de la lampe sur le bridge via son ID et de lui envoyer une nouvelle commande à chaque création d'un nouveau CIN dans le CNT DATA.

Finalement, la récupération des données du capteur "Fibaro" a été réalisée par une IPE spécifique déjà développée et se trouvant sur la raspberry. Cette IPE était déjà configurée pour communiquer avec la Gateway MN-ZWave-CSE, également hébergée sur la raspberry. Nous avons uniquement eu à réaliser l'enregistrement de la Gateway à notre IN-CSE Base. A ce stade, les données du capteur "Fibaro" sont donc publiées dans la Gateway MN-ZWave-CSE, cependant nous voulons, en fonction des données relevées, générer un changement de l'état de la lampe HUE, donc une publication d'un nouveau CIN dans le container DATA de l'AE_Lampe. Nous avons donc développé un dernier IPE mais cette fois-ci grâce à node red. Ainsi, à chaque création d'un nouveau CIN par le capteur "Fibaro" dans l'AE correspondant de la Gateway MN-ZWave-CSE notre IPE Node-Red va récupérer la valeur de la luminosité et, en fonction de celle-ci, publier un nouveau CIN dans le container DATA de l'AE_Lampe correspondant soit à un état allumé de la lampe, soit à un état éteint.

2.2. Interagir avec les objets en utilisant une architecture REST

Le découpage des ressources, comme présenté en partie 2.1, est à la base d'une architecture de type REST, en effet, on met à disposition des ressources plutôt que des services. Après c'est à l'application d'utiliser ces ressources pour proposer des services.

OM2M vient avec un base de données et un serveur, pour les utiliser, il faut lancer un noeud IN sur lequel va tourner le serveur et la base de données.

Une fois la base de données déployée, on va pouvoir interagir avec celle-ci via des requêtes HTTP basiques :

- la requête POST, va nous permettre des créer des nouvelles ressources.
- la requête GET, permet de récupérer une ressource et son contenu.
- la requête PUT, permet de mettre à jour une ressource.
- la requête DELETE, permet de supprimer une ressource.

Pour être utilisées, ces différentes requêtes HTTP ont besoins de plusieurs paramètres. Il y a tout d'abord l'URI, qui correspond au chemin d'accès dans l'arborescence à l'entité où la requête sera effectuée. Les requêtes disposent également d'un header. Un header peut comprendre un Orginator (qui correspond à un nom d'utilisateur et un mot de passe,dans notre cas admin:admin) qui permet de s'identifier comme administrateur lors de la requête et accéder aux droits d'écriture et de lecture en conséquence. Il est également possible de préciser un Content-Type, dans le cas des requêtes POST, pour préciser le format de l'application dans le corps de la requête (par exemple JSON ou XML) ainsi que son type (par exemple ty=2 pour une Application Entity, ty=3 pour un Container, ty=4 pour un Content Instance ou ty=23 pour une Subscription). Dans le cas des requêtes PUT, DELETE et GET on utilise le champ Accept du header pour préciser le format de la réponse attendu (la encore JSON ou XML).

Au cours du TP 1 en autonomie, nous avons utilisé le logiciel Postman, un client REST, qui nous a permis de faire nos premières requêtes HTTP sur notre serveur OM2M. Par la suite, au cours du TP 2 nous avons utilisé les librairies de Apache fournit pour créer nos propres méthodes Create(), Retrieve(), Update() et Delete() sous JAVA, correspondant aux requêtes POST, GET, PUT, et DELETE.

Sous Java nos ressources (AE, CNT, CIN...) sont sous la forme d'objet, il est donc nécessaires de les transformer en format de données pouvant être échangés via les requêtes HTTP, c'est à dire en format string XML ou JSON. Pour ce faire nous avons utilisé le mapper JAXB et le mapper oBIX qui permettent d'effectuer ces transformations de données. Avec le mapper JAXB, la méthode "marshall" permet de sérialiser l'objet Java en string XML et la méthode "unmarshall" de transformer le string XML en objet Java. Dans le cas du mapper oBIX, c'est la méthode "ObixEncoder" qui permet de transformer un objet oBIX en string XML et la méthode "ObixDecoder" de transformer le string XML en objet oBIX.

Dans le TP 1, grâce à l'IPE SAMPLE et à l'utilisation d'un serveur de surveillance, nous avons pu voir un premier aperçu du fonctionnement du système de souscription. En effet nous avons pu observer la création de nouveaux CIN, dans l'arbre des ressources sous l'AE de la lampe simulée, sous l'action des boutons d'allumage de la lampe. Nous avons également fait le cas inverse, en souscrivant notre serveur de surveillance à l'AE MY_Sensor que nous avions crée. En créant de nouveaux CIN dans son arborescence, une ressources Notify était transmise au serveur de surveillance.

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

Comme expliquée en partie 2.1, la technologie que nous avons intégrée à notre architecture qui était non nativement compatible avec le standard, est la lampe HUE Philips. La lampe fonctionne grâce à une communication Zigbee avec un routeur. L'AE que nous utilisons pour la lampe ainsi que l'arborescence correspondante avait déjà été réalisée au cours du TP 2. Nous avons commencé par connecter notre machine personnelle (hébergeant le IN-CSE et le MN-CSE) à ce même routeur. En utilisant les librairies Java HUE (HueProperties.java, HueSdkListener.java et HueUtil.java) il nous a été possible de nous connecter

automatiquement au bridge interfaçant la lampe HUE, récupérer la liste des lampes connectées au bridge et contrôler leurs états (ON/OFF, intensité et couleur).

Nous avons alors développé un serveur Interworking Proxy Entity (IPE) pour faire l'interface entre OM2M et le bridge de la lampe HUE Philips. L'IPE est un servlet HTTP qui a était configuré grâce à une Subscription pour écouter toute nouvelle créations de CIN dans le CNT DATA de l'AE Lampe_1. Le serveur IPE reçoit alors un "Notify" dont il va extraire les informations nécessaires pour mettre à jour l'état de la lampe, grâce aux librairies Java HUE, comme nous l'avions fait manuellement en communiquant de lui même avec le bridge.

Finalement, le container DATA contiendra bien l'ensemble des différentes commandes qui ont été envoyées à la lampe par le serveur IPE. Le container DESCRIPTOR contient lui des commandes envoyables, manuellement grâce à l'interface web OM2M par l'utilisateur, à la lampe via le serveur IPE.

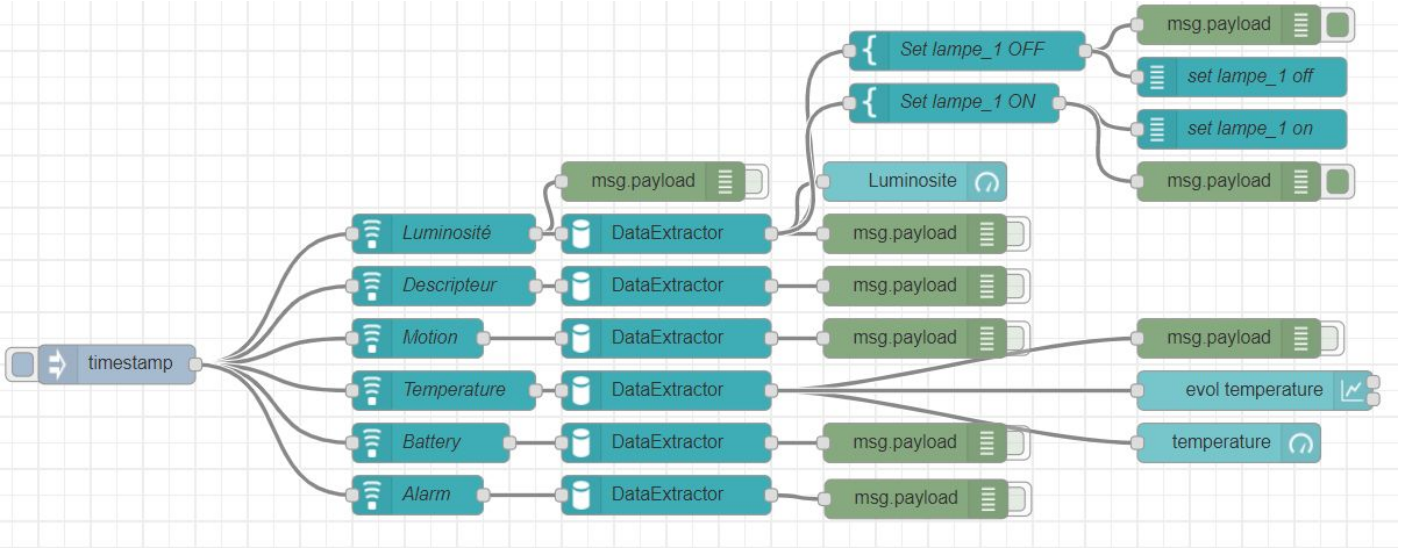
3. Déployer une application composite entre divers technologies grâce à NODE-RED en se basant sur un intergiciel standardisé

Avant de commencer, il faut savoir que Node-Red est une interface de programmation représentant les fonctions sous forme de blocs, l'application fonctionne alors sous forme de flux passant par ces différents blocs, de gauche à droite. Il est alors possible d'enchaîner plusieurs fonctions et donc de traiter notre information. Une interface graphique est également disponible avec Node-Red et permet de présenter les données relevées sous forme de graphiques ou de compteurs par exemple.

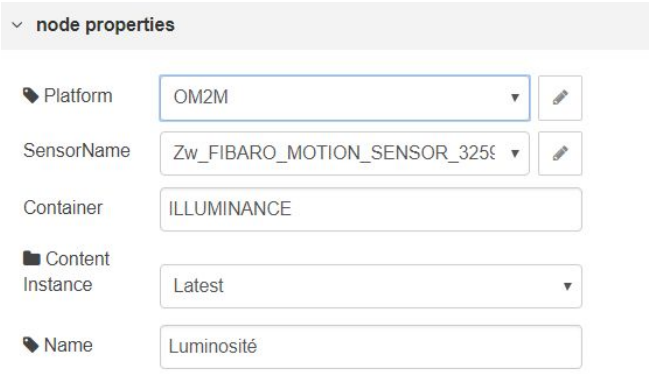
Comme expliqué en partie 2.1, le dernier TP a eu pour but d'implémenter un serveur IPE basé sur Node-Red pour remonter les données mesurées par le capteur "Fibaro" de la gateway MN-ZWave-CSE au IN-CSE et donc à la lampe HUE par l'interface de son serveur IPE.

Pour cela, nous avons vérifié que le capteur "Fibaro" envoyé bien des données sur la gateway MN-ZWave-CSE via son IPE fourni (le tout étant hébergé sur une raspberry). A cause d'un problème graphique de l'interface web, nous avons utilisé Postman pour vérifier l'arborescence de l'AE Fibaro.

Après installations des packages nécessaires, nous avons pu réaliser le flux suivant sur Node-Red :

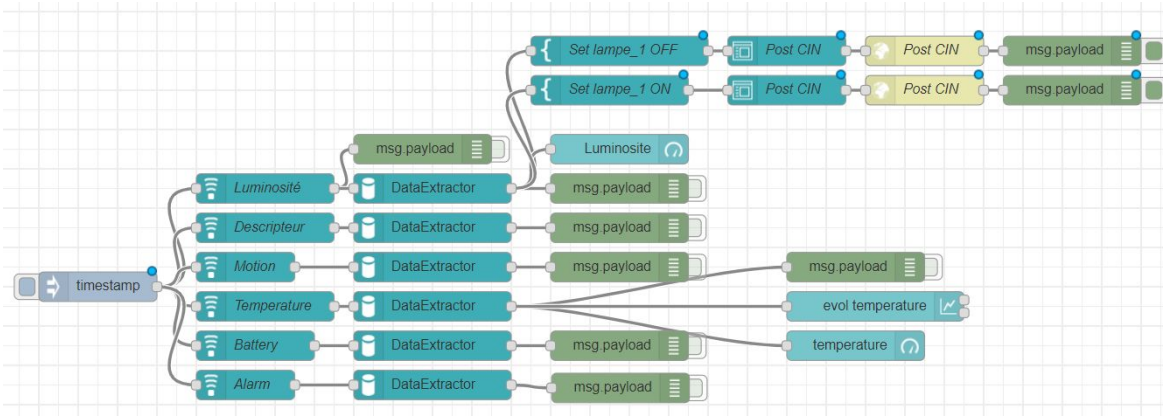


Le blocs timestamp est utilisé pour démarrer le flux uniquement sur demande. Les blocs Luminosité, Descripteur, Motion...sont des blocs de type NamedSensor et permettent de récupérer le dernier CIN du CNT ciblé dans l'arborescence. Leur configuration se fait de la manière suivante (exemple avec Illuminance):



Les blocs DataExtractor permettent eux de ne récupérer que la valeur de l'information stockée dans le CIN. Les blocs Set lampe_1 ON et Set lampe_1 OFF permettent eux de comparer la valeur relevée à une constante et si le test est TRUE d'exécuter l'actuator associé. Ici ,si la luminosité est inférieur à 300, la lampe est allumée:

Nous n'avons pas réussi à faire fonctionner l'actuator correctement, de ce fait nous avons plutôt le bloc ContentInstance avec le bloc httpRequest pour poster des Content Instance dans le CNT DATA de l'AE Lampe_1 et contrôler la lampe HUE :



Conclusion

Ces TPs nous ont permis de mieux appréhender la complexité des architectures pour l'IOT en utilisant le standard open-source OM2M. Nous avons également pu observer l'utilité du standard OM2M qui permet une grand interopérabilité qui nous a permis de développer une applications IOT regroupant des systèmes utilisants différents protocoles de communications comme la lampe HUE et le capteur Fibro (respectivement Zigbee et Z-Wave).

Pour notre part, ces TPs ont été très difficiles à appréhender. En effet, nous avons découvert java ainsi que eclipse, ce qui a grandement ralenti notre progression lors des TPs, l'aide de notre enseignant de TP nous a été d'un grand secours. Finalement, la répartitions des différentes ressources sur plusieurs plateforme (2 pages moodles différentes et un cours OpenClassroom) ne nous a pas facilité la tâche.