RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE ECHAHID HAMMA LAKHDAR - EL OUED

FACULTÉ DES SCIENCES EXACTES

Département D'Informatique

Mémoire de Fin D'étude

Présenté pour l'obtention du Diplôme de

MASTER ACADEMIQUE

Domaine : **Mathématique et Informatique**

Filière : **Informatique**

Spécialité : **Systèmes Distribués et Intelligence Artificielle**

Présenté par :

- **ZABI ABDELHAMID**
- **HAMMI Messaoud**

Thème

Abstraction de la couche Hardware par un serveur web embarqué pour le web des objets

Soutenue le 07-06-2017 Devant le jury:

M.	BALI Mouadh	MAB	Président
M.	BEN ALI Abdelkamel	MAA	Rapporteur
M.	BEGGAS Mounir	MCB	Encadreur

AnnéeUniversitaire:2016-2017

Dédicace

***À** mes chers parents qui sont décédés.*

***À** ma chère épouse.*

***À** mes chers enfants (Ali, Dalal, Yacine et Aicha).*

***À** mes frères et mes sœurs.²*

***À** toute ma famille.*

***À** tous mes amis.*

***À** tous mes collègues.*

Messaoud

Dédicace

Je dédie ce mémoire :

À la mémoire de ma très chère mère. Toi maman qui a tant veillé, sacrifié, soutenu, défié tous les obstacles et les difficultés de la vie pour que je sois un homme vrai et réussi. Toi qui attendais toujours mes réussites avec certitude parce que tu y croyais fortement. Tu vivais pour moi et tu étais la lueur qui illuminait ma vie. Envers toi maman, ma reconnaissance et mon amour resteront éternels.

À la mémoire de mon très cher père. Ce grand homme honnête et valeureux qu'il était, n'a épargné aucun effort pour me mettre sur le bon chemin de la vie. Les nobles valeurs qu'il ma inculquées, ainsi que les précieux conseils qu'il n'a cessé de me les donner m'ont beaucoup servi.

À vous deux mes très chers parents, jamais les mots seuls pourront vous rendre le mérite et la récompense que je dois envers vous. Vos âmes doivent reposer en paix. Et je prie le Puissant Dieu pour qu'il vous accueille dans son vaste paradis.

À ma très chère sœur, qui était plus qu'une sœur. Toi qui étais toujours près de moi. On s'est partagé de grandioses peines dans cette vie. Tu m'as soutenue, m'a encouragé et m'a fait bénéficier de ta propre expérience dans la vie. Toima sœur tu resteras la personne la plus remarquable pour moi après la disparition de notre très chère maman.

À mon très cher frère et sa famille. À qui vont mes sentiments les plus affectueux.

AbdelHamid

Remerciements

Nous tenons à remercier tout d'abord le bon dieu de nous avoir donné le courage et la force pour mener à bien ce travail.

Nos remerciements sont, ensuite, adressés à notre encadreur Docteur Beggas Mounir, qui a su bien nous orienter et nous faire progresser tout au long de ce mémoire.

Nous tenons à remercier aussi, notre Co promoteur Kertiou Smail.

Nos vifs remerciements sont adressés à Monsieur Hammi Brahim pour ses aides et conseils précieux.

Nous remercions aussi, le Docteur: David R. GNIMPIEBA ZANFACK, *pour sa thèse récente de doctorat, qui est était une source présieuse*

Nous voudrons également remercier les membres du jury, pour avoir accepté d'évaluer ce travail, et pour toutes leurs remarques et critiques, ainsi que les enseignants du département d'informatique de l'université d'El oued.

Enfin, nos remerciements à toute personne nous ayant aidé d'une façon ou d'une autre.

Résumé :

Aujourd'hui, les objets connectés sont partout et pour tous. Ils interagissent entre eux et entre des softwares, pour offrir des services améliorant notre quotidien, et cela sans intervention humaine. En effet, l'intervention humaine n'est plus récurrente, car les machines gèrent les objets et toutes leurs ressources et interagissent avec eux en machine to machine.

Les objets physiques sont, à priori, de nature hétérogène, ce qui mène à introduire le Web of Things (WoT), qui standardise les interfaces de communication avec les objets physiques à travers une API Web. Pour intégrer tous les objets physiques dans le WoT, il est nécessaire d'avoir un serveur web embarqué dans chaque objet, ce qui n'est possible pour tous les objets existants. Cela mène à introduire des Gateway pour l'intégration des objets dans le WOT.

L'objectif de notre travail, est de proposer une Gateway sous forme de middleware permettant d'intégrer ces objets dans le web, en se basant sur les technologies disponibles tel que l'architecture REST et le protocole CoAP, dont chaque objet physique sera représenté par une ressource virtuelle web accessible par un URI, et manipulable à travers son interface web de type REST (API Web). Le middleware est un conteneur de composants java, où chaque composant représente un objet physique. Chacun représente un objet physique et permet l'interaction avec cet objet selon l'interface offerte par cet objet. Nous avons validé notre proposition par quelques scenarios.

Mots – clés : Web des objets, le protocole CoAP, l'architecture REST, Middleware.

Abstract:

Today, connected devices are everywhere and for everyone. They interact between themselves and between software, to provide services that improve our daily lives, and this without human intervention. In fact, the human intervention is not recurrent, because the machines handle the devices and all of their resources, and interact with each other in machine-to-machine.

The connected devices are of heterogeneous nature, which leads to introduce the Web of Things (WOT), which standardizes the interfaces for communication with the physical devices. To integrate all of the physical devices in the WOT, it is necessary to have a web server embedded in each device which is not possible for all existing devices. This leads to introduce Gateway for the integration of objects in the WOT.

The aim of our work, is to provide a Gateway as a middleware allowing the integration of these devices in the web, based on the available technologies such as the REST architecture and the protocol CoAP, in which every physical object will be represented by a virtual resource accessible by a URI, and manipulated through its uniform API (REST Web API). The middleware is a container for java components, where each component represents a physical device. and allows interaction with them. We have validated our architecture through a few scenarios.

Key Words: WOT, Connected devices Middleware, REST

ملخص:

ازداد عدد الكائنات الفيزيائية أو بما يعرف بالكائنات المتصلة ،في وقتنا الحالي بشكل كبير، حيث أصبحت تستعمل على نطاق واسع وفي عدة مجالات، وهي تتفاعل فيما بينها بالاضافة لتفاعلها مع البرمجيات، وهذا من أجل توفير خدمات تمكن من تحسين حياتنا اليومية وهذا بصفة آلية دون أي مساعدة أو تدخل من الانسان، حيث أصبحت الآلة هي المسؤولة عن التحكم في مواردها و كائناتها الفيزيائية والتحاور معها أو بما يعرف بالاتصال "آلة مع ألة".

لكن الطبيعة غير المتجانسة لهاته الكائنات، تحتم اللجوء لتكنولوجيات ويب الكائنات من أجل توحيد واجهة الاتصال فيما بينها، وهذا عن طريق واجهات الويب البرمجية API . يتم ادماج كل الكائنات الفيزيائية في الويب، عن طريق ادراج خادوم (سرفر) في كل كائن وهو غير متاح مع كل الكائنات، لذا فاننا سوف نستعمل بوابة GateWay التي يتم من خلالها عملية الادماج في الويب.

إن الهدف من هذا العمل هو اقتراح بوابة برمجية تتمثل في وسيطة Middleware تمكن من ادماج أي نوع من الكائنات الفيزيائية، وذلك باللجوء لتكنولوجيا REST التي توفرها الويب بالاضافة للبروتوكول البرمجي COAP، حيث يتم تمثيل كل كائن فيزيائي بمورد افتراضي معرف ب URI ويتم التحكم فيها عن طريق واجهة برمجية ويب موحدة REST API. هاته الوسيطة Middleware، عبارة عن حاوية مكونات جافا أين كل مكون منها يمثل كائنا فيزيائيا، وهو يلعب دور الوسيط معه عن طريق الواجهة البرمجية التي يوفرها هذا المكون.

وفي الأخير سوف نقوم باختبار العمل (Middleware) عن طريق بعض السيناريوهات.

الكلمات المفتاحية:

ويب الكائنات، واجهات الويب البرمجية، بوابة، الوسيطة.

Table des matières

Introduction

Le bouleversement du monde numérique par l'arrivée massive des objets connectés, fait qu'internet subit une révolution majeure, pleine d'opportunités pour tous les auteurs. Il y va de notre perception au quotidien du monde réel et de notre monde social, dans lequel on vit, on partage et on consomme.

Avant il y avait un dialogue technique unique Machine-to-Machine constamment "drivé" par l'homme sans qui rien ne se passe ou presque. Aujourd'hui, les objets connectés interagissent entre eux pour améliorer notre quotidien, et l'intervention humaine n'est plus récurrente, car les machines gèrent les objets et toutes leurs ressources.

Cela ressemble à défaut, d'une nouvelle révolution industrielle, du moins à une rupture technologique, comme c'était le cas en 2007/2010 pour les smartphones et les tablettes.

Les objets connectés ont envahi notre quotidien. Beaucoup de biens humains sont disponibles via ces connexions. Les enjeux sont énormes, car statiquement d'ici 2020 il y aura entre 30 à 50 millions d'objets connectés. Plusieurs nouvelles problématiques apparaissent en conséquence.

Ces objets physiques sont à priori de nature hétérogène ce qui mène à introduire le Web of Things (WOT) qui standardise les interfaces de communication avec les objets physiques au API Web. Pour intégrer tous les objets physique dans le WOT, il est nécessaire d'avoir un serveur web embarqué dans chaque objet ce qui n'est possible pour tous les objets existants. Cela mène à introduire des Gateway pour l'intégration des objets dans le WOT.

L'objectif de notre travail, est de proposer une architecture permettant d'intégrer ces objets dans le web en se basant sur les technologies disponibles tel que l'architecture REST, et le protocole CoAP, dont chaque objet physique sera représenté par une ressource virtuelle web accessible par un URI et manipulable à travers son interface web de type REST(API Web), cette interface permet d'exposer les services offerts par ces ressources à travers des interfaces Web utilisant les protocoles de communication standards ce qui standardisent la manière de l'invoquer de toute sorte d'objets, en assurant une communication M2M interopérable indépendante de toute plateforme ou langage. D'un autre côté, notre contribution consiste à développer un serveur web embarqué, qui sera capable de stocker et charger à chaud des composants, représentant les objets physiques, grâce à une méthode basée sur le chargement dynamique des composants représentant les objets physiques qui apparaissent et disparaissent dynamiquement dans l'environnement.

Le présent manuscrit comporte cinq parties : la première partie est une introduction où nous proposons une problématique générale et notre contribution, ainsi que le plan et les difficultés que nous avons rencontrées. Dans la deuxième partie nous présentons l'état de l'art sur l'IoT, notamment : la définition de l'IoT, l'architecture M2M, les capteurs, la couche d'abstraction matérielle. La troisième partie sera consacrée au Web des Objets (WoT), dont nous allons mettre l'accent sur l'architecture REST. Tandis que, dans la quatrième partie nous allons aborder les protocoles de communication. Quant à la cinquième partie de notre recherche, elle s'articule en quatre éléments : les travaux connexes aux approches d'intégrations des objets physiques aux WOT, ainsi, il sera consacré à la conception et l'implémentation de notre plateforme, et les scénarios proposés pour la validation. Enfin, nous conclurons ce présent travail par une conclusion générale comprise les résultats obtenus de notre travail.

Nous soulignons que notre travail connait des imperfections, qui sont liées aux contraintes en rapport avec le manque du temps, vu nos engagements socioprofessionnels au quotidien, d'une part, et d'autre part au fait que ce travail s'inscrit dans un domaine récent, où il n'ya pas encore suffisamment de ressources de recherches. Ce qu'il fait que notre travail, n'a pas pu apparaître dans un aspect parfait. Aussi, d'autres facteurs d'ordre expérimental, ont davantage compliqué notre tâche, tel que l'on pas pu réussir facilement à valider nos scénarios. Le manque des données bien définies dans le domaine en est derrière.

Chapitre 1

I. Principes de l'internet des Objets (Internet of Things IoT)

I.1. Introduction

Selon la loi de Moore, la puissance des machines informatiques double chaque deux ans. Un autre effet de cette loi est la diminution du prix des composants informatiques. Ainsi, il devient de plus en plus facile d'ajouter des composants informatiques aux objets de tous les jours et de les relier entre eux**[1]**.

Une manière de relier ces objets entre eux est de les connecter à Internet, ce qui leur permet de communiquer avec des humains et d'autres machines**[2]**.

I.1.1. Communication Machine to Machine(M2M)

M2M est une communication directe entre deux machines sans intervention humaine**[3]**.

Dans le domaine de l'IoT, le M2M consiste à la communication entre les objets communicants (ex. capteur, compteur, etc.) qui sert à capturer un événement (ex. température, mesure sismique, etc.) à travers un réseau de communication mobile, fixe ou hybride. Cette application traduit l'événement capturé en des informations significatives (Voir la figue infra).

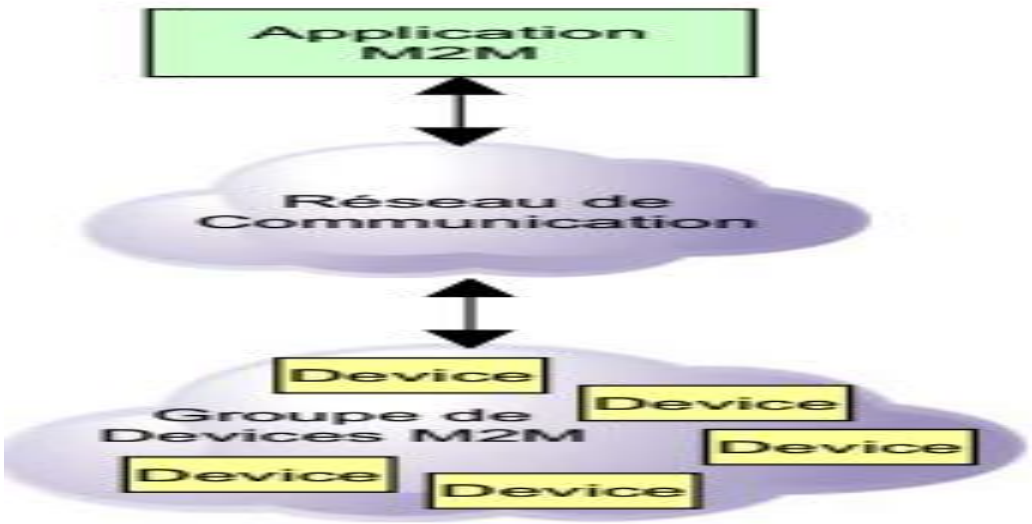


Figure 1: Architecture M2M

[David Boswarthick,Omar Elloumi,Olivier Hersent,

M2M Communications : a Systems Approach »]

Une solution M2M est le résultat d'une interaction continue entre les dispositifs communicants, et les applications à travers les réseaux de communication **[4]**.

I.1.2. Eléments de base de Internet des Objets:

Avant de parler de l'IoT, nous commençons à discuter brièvement les éléments de base des systèmes IoT : les objets communicants, les capteurs et les réseaux des capteurs.

a. Les objets communicants

Plusieurs technologies sont utilisées pour faire communiquer un objet avec l'Internet, parmi lesquelles RFID (Identification par Radio Fréquence), NFC (Near Fields Communication), le protocole de communication ZigBee.

a.1. Les technologies RFID : sont des technologies d'identification automatique. Elles sont utilisées pour identifier automatiquement et électroniquement un objet. Les technologies RFID sont constituées de marqueurs/capteurs, de lecteurs, de logiciels de traitement des informations et sont classées suivant plusieurs paramètres : la fréquence utilisée, la portée, le prix, et la consommation d'énergie**[5]**.

La RFID est constituée d'un couple lecteur/étiquette. Le lecteur envoie une onde radio, l'étiquette envoie à son tour une trame d'identification. Une fois la puce alimentée, l'étiquette et le tag communiquent suivant le protocole TTF (Tag Talk First) ou ITF (Interrogator Talk First). Dans le mode TTF 27 l'étiquette transmet en premier les informations contenues dans la puce à l'interrogateur. En mode ITF, l'interrogateur envoie une requête à l'étiquette, et ce dernier répond par la suite **[5]**.

a.2. La technologie NFC : est le résultat de plusieurs évolutions des microcontrôleurs, des cartes à puce, et des communications à courte portée. NFC est basée sur le même principe que la RFID, c'est-à-dire l'identification par radiofréquence. Elle permet l'échange d'informations à courte distance entre deux objets (un lecteur et une carte), sans contact et fonctionne suivant deux modes, le mode passif et le mode actif. En mode passif, le terminal de l'utilisateur émule une carte à puce et acquiert de l'énergie des radiations du lecteur (téléphone mobile par exemple). En mode actif, le terminal se comporte comme un lecteur d'étiquettes électroniques (code à barres, étiquettes 2D), et possède sa propre source d'énergie (une batterie embarquée par exemple). NFC permet à l'utilisateur d'échanger des informations avec son environnement, notamment dans le domaine des transports, des loisirs, des achats, ou la lecture d'informations sur des panneaux d'affichage. L'utilisation de la NFC facilite la gestion des données de ventes dans la chaîne logistique, la gestion et la validation de tickets de bus dans le transport urbain. Lorsqu'un utilisateur scanne un tag NFC, il peut en outre avoir accès à des informations sur le produit (origine, fabricant, contenu/ingrédients, procédé de fabrication) **[5]**.

a.3 ZigBee : est un protocole de communication sans fil à bas coût qui permet des échanges à courte distance entre les nœuds d'un réseau WPAN (Wireless Personal Area Networks). Ce protocole est basé sur la norme IEEE 802.15.4 qui spécifie les protocoles de communication entre les couches physiques, et liaison de données du modèle OSI, en définissant trois types d'équipements : les FFD (Full Function Devices) qui sont des équipements à fonctionnalité complète, les RFD (Reduced Function Devices) équipements à fonctionnalité réduite, et les coordinateurs de réseau. Les FFD coordonnent l'ensemble du réseau, ce sont des coordinateurs PAN (Personal Area Network), routeur ou dispositif relié à un capteur. Les RFD (Reduced Function Device) sont des équipements à fonctionnalité réduite, conçue pour des applications simples comme l'allumage d'une lampe. Les RFD ne peuvent communiquer qu'avec un FFD. Parmi les avantages que procure ce protocole de communication, nous pouvons citer la faible consommation d'énergie, l'utilisation optimale de la bande passante, et son faible coût de mise en œuvre. Ces avantages permettent d'adopter le protocole Zigbee dans les environnements embarqués et les réseaux industriels, ou le développement de nouveaux produits basés sur ce protocole **[5].**

On a distingué deux types d'objets :

- **Les objets passifs** : ils utilisent généralement un tag (puce RFID, code barre 2D). Ils embarquent une faible capacité de stockage (de l'ordre du kilo-octet) leur permettant d'assurer un rôle d'identification. Ils peuvent parfois, dans le cas d'une puce RFID, d'embarquer un capteur (température, humidité).
- **Les objets actifs** : Ils peuvent être équipés de plusieurs capteurs, d'une plus grande capacité de stockage, être doté d'une capacité de traitement ou encore être en mesure de communiquer sur un réseau[**6**].

b. Les capteurs

Les capteurs sont de petits appareils disposant de capacités de mesures, voire d'actions, sur leur environnement (La température, le taux d'humidité…). Les spécificités innovantes de ces capteurs résident dans leur taille et leur coût réduit. Ces capteurs sont des équipements électroniques soumis à de considérables contraintes, parmi ces contraintes nous citons les plus importantes : l'énergie, la puissance de traitement et les échanges sans fil**[7]**.

L'échange d'informations, voire l'interaction entre différents objets est envisageable, et ce même sans intervention des utilisateurs. L'objet va "capter", mesurer une caractéristique physique de son environnement, éventuellement lui appliquer un traitement informatisé, et fournir le résultat à d'autres utilisateurs (ordinateur par exemple)**[7]**.

c. Les réseaux de capteurs

Les réseaux de capteurs est un ensemble de capteurs autonomes à faible coût, interconnectés par un réseau de communications qui, coopèrent pour acquérir et transmettre des informations. Afin de satisfaire les besoins de communication entre eux, les capteurs sont équipés de dispositifs sans fil pour l'émission et la réception de données. Cela ne suffit, cependant pas, à rendre un ensemble de capteurs accessibles, ou du moins de manière interopérable, transparente et simplifiée. Pour cela, les capteurs doivent aussi s'organiser. Ce qui caractérise un réseau de capteurs, c'est que ses éléments sont de très petits appareils, dotés de capacités de transmission sans fil, autonomes en énergie et dont le positionnement est, le plus souvent, libre**[7]**.

VIPRET Julien**[8],** définit les réseaux capteurs dans son article intitulé : **«** Utilisation du protocole CoAP pour la découverte de ressources dans les réseaux de capteurs **»**, d'après le paragraphe suivant : « Les réseaux de capteurs sont des composants qui possèdent des ressources contraintes. Ils disposent notamment d'une capacité mémoire, d'une puissance, d'une bande passante et d'une réserve énergétique très limitées. Pourtant, son utilisation est de plus en plus fréquente de part son faible coût et de sa facilité de déploiement. Les réseaux de capteurs sont très polyvalents, ils peuvent être utilisés dans le cadre de la surveillance environnementale (glissements de terrain, volcans, etc...) mais aussi dans des infrastructures intelligentes (domotique, gestion de la climatisation, lumières) ».

A cause de ces caractéristiques, les spécialistes prévoient que les réseaux de capteurs vont tenir dans le futur une place dominante dans l'architecture RESTful **[8]**.

I.1.3. Définition de l'IoT

Les spécialistes ont constaté qu'il n'y a pas de définition standardisée de l'Internet des objets, et c'est pour ce là, on trouve plusieurs définitions, par exemple NumaMontmollin **[1]** a défini l'IoT : « Internet des objets est le fait de relier des objets physiques au monde virtuel. Dont, chaque objet est unique et peut communiquer avec d'autres acteurs d'Internet à travers un protocole de communication défini ».

Et parmi les autres définitions, on cite la suivante : « L'Internet des Objets est un réseau de réseaux qui permet, via un système d'identification électronique normalisé et unifié, et des dispositifs mobiles sans fil, d'identifier directement et sans ambiguïté des entités numériques et des objets physiques et ainsi de pouvoir récupérer, stocker, transférer et traiter, sans discontinuité entre les mondes physiques et virtuels, les données s'y rattachant »**[9]**.

Dans ce cas là, on peut définir l'Internet des objets comme un système d'objets physiques qui peuvent être découvert, surveillé, contrôlé ou interagi par des dispositifs électroniques qui

communiquent sur différentes interfaces de réseautage et peuvent éventuellement être connecté à Internet **[1]**.

Et en ce qui concerne l'historique du concept de l'IOT, on trouve qu'il est relativement ancien à l'échelle d'Internet. Il est développé à partir de 1991 par Marc Weiser sous le terme d'informatique ubiquitaire («Ubiquitous computing »).

À l'origine, le terme Internet des objets a été utilisé, pour la première fois en 1999 par Kevin Ashton, pour décrire des objets équipés de puces d'identification par radiofréquence (ou puce *RFID*). Chaque objet, identifié de manière unique et universelle, peut alors être rattaché à un ensemble d'informations le concernant, ces dernières étant lisibles par d'autres machines caractéristiques, état courant et position sont alors autant de métadonnées échangées entre les objets, formant un nouveau réseau qui leur est dédié**[6]**.

Les premiers balbutiements de l'IoT se font au début des années 2000, avec l'étiquetage d'objets grâce à des puces RFID, qui permettent d'identifier les objets des manières uniques et de les tracer. Le premier pas est franchi, grâce à une adresse unique, une URI (Uniforme Ressource Indicator), chaque objet est ainsi répertorié et accessible. Par la suite, de par le développement croissant des technologies, de plus en plus de possibilités de développements sont données. Localisation, possibilité d'embarquer de véritables systèmes d'exploitation et serveurs dans l'objet, véritable interaction avec l'environnement grâce au développement de capteurs et d'actuateurs performants, miniaturisés et de faible coût [1].

I.1.4. La couche d'abstraction matérielle

La couche d'abstraction matérielle (Hardware Abstraction Layer HAL) est un logiciel intermédiaire entre le système d'exploitation et le matériel informatique. Elle comporte une interface de programmation, qui fournit des fonctions génériques, pour la manipulation du matériel en masquant ses détails techniques**[10]**.

a. Modèles d'architectures de l'Internet des Objets

La figure ci-dessous, montre l'architecture de l'IoT, et pour l'illustrer nous précisons le rôle des différents processus présentés sur ce schéma :

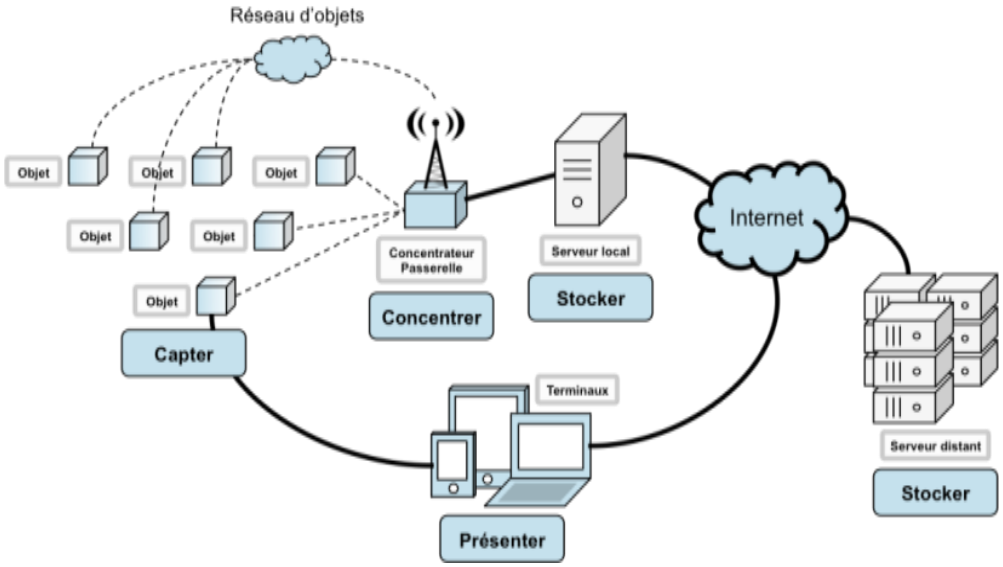


Figure 2: Architecture de l'IoT

[http://blog.octo.com/modeles-architectures-internet-des-objets,Avril 2017]

Capter : désigne l'action de transformer une grandeur physique analogique en un signal numérique.

Concentrer : permet d'interfacer un réseau spécialisé d'objet à un réseau IP standard (ex. WiFi) ou des dispositifs grand public.

Stocker : qualifie le fait d'agréger des données brutes, produites en temps réel, arrivant de façon non prédictible.

Enfin, **présenter :** indique la capacité de restituer les informations de façon compréhensible par l'Homme, tout en lui offrant un moyen d'agir et/ou d'interagir

b. Hétérogénéité de l'Internet des objets

Les objets ne sont pas égaux, car ils ont des caractéristiques variées. Certains objets sont mobiles, et ont une position qui évolue au cours du temps. D'autres peuvent être des objets transportables, tels qu'un téléphone mobile, un livre ou des vêtements, ou d'objets mobiles autonomes dotés de capacités motrices, tels qu'une voiture **[6]**.

Benjamin Billet **[6]** résume l'impact de l'hétérogénéité sur l'Internet des objets, en distinguant deux sortes de l'hétérogénéité :

b.1. Hétérogénéité fonctionnelle : Les objets possèdent des capacités spécifiques (statique ou mobile, alimenté en continu ou par une batterie, ressources matérielles, capteurs et actionneurs disponibles, etc.) et à chacune d'elle correspond des contraintes particulières (connectivité intermittente, durée de vie, tâches réalisables, etc.). Différentes approches et

techniques doivent être considérées pour gérer les objets en fonction de leurs contraintes, et de leurs différences propres.

b.2. Hétérogénéité technique : Les technologies matérielles et logicielles utilisées pour construire les objets sont multiples, et compromettent l'idéal de collaboration autonome entre objets. De plus, le développement d'applications est complexité, nécessitant des développeurs des connaissances spécifiques sur le fonctionnement de chaque objet **[6]**.

c. Limites de l'internet des objets

L'internet des objets peut intégrer des appareils de divers fabricants, dans une seule application ou système. Les projets IoT se concentrent principalement sur la construction de déploiements à petite échelle, fermés et isolés, les dispositifs n'ont pas été conçus pour être facilement accessibles ou reprogrammables. Le couplage sur mesure entre dispositifs et applications dans un cas d'utilisation donné, signifie que le changement à un déploiement existant est complexe et coûteux. Cela limite à la fois, le maintien et l'évolution de l'Internet des objets, parce que des ressources considérables (temps, argent et compétences techniques), sont nécessaires chaque fois qu'une nouvelle fonction est ajoutée. Pour ces limites, le web des objets a été proposé. Le Web des Objets se base sur Internet, mais avec un niveau d'abstraction plus élevé **[6]**.

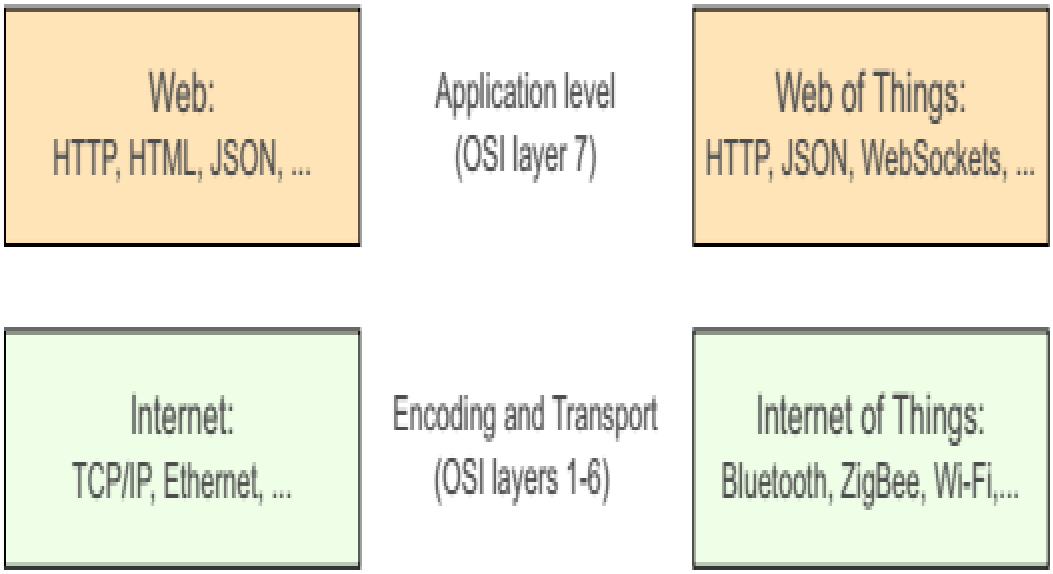


Figure 3: L'Internet et les Web des objets

[Dominique D. Guinard et Vlad M. TRIFA, Building the Web of Things**,]**

Chapitre 2:

II. Principes du Web des Objets (Le Web des Objets WoT)

L'idée du WoT, est d'intégrer des objets au Web, afin de les faire communiquer avec des humains ou d'autres machines. Le Web est un ensemble de technologies permettant d'interroger des ressources, identifiées par des URI, accessibles sur Internet .Ces ressources contiennent des informations textes structurées (HTML, JSON, XML) ou binaires (images, vidéos, etc.). Dans bien des cas, le Web est considéré comme un système d'information, composé de ressources liées entre elles par des hyperliens**[11]**.

Figure 4 : Le web des objets

[Dominique D. Guinard et Vlad M. TRIFA, Building the Web of Things**,]**

Un point important du WoT, est de fournir un protocole de communication commun entre les objets. L'utilisation des technologies Web adaptées aux objets fournit la même facilité d'interaction que pour n'importe quel site Web. Par exemple, l'utilisateur a uniquement besoin d'un navigateur Web, de plus n'importe quel moteur d'indexation, tel que Google, peut y accéder et tous les clients HTTP utilisables. Un moteur de recherche, pourrait tout à fait permettre la recherche d'objets selon différents critères et la création d'applications clients n'est plus compliquée que pour un site classique. L'utilisation d'un protocole différent ou propriétaire ne permettant pas cette facilité d'utilisation, c'est la raison principale d'utiliser le Web des objets **[1]**.

Le Web des objets permet d'intégrer des objets physiques, à priori hétérogénies provenant de plusieurs fabricants dans le Web, en utilisant des protocoles standard.

Mais avant d'aborder le principe du WoT, nous mettons d'abord, l'accent sur l'historique des tentatives d'intégration des objets physiques à l'internet.

II.1. Historique de WoT :

L'idée de lier les objets physiques au web n'est pas une idée récente, plusieurs approches ont été proposées. Les premières approches ont commencé en attachant des codes à barres à des objets pour diriger l'utilisateur vers des pages sur le Web contenant des informations sur ces objets**[12]**.

Ces pages ont d'abord été desservies par des serveurs Web statiques sur des ordinateurs centraux, puis par un système de passerelle, qui a permis aux périphériques de faible puissance de faire partie de réseaux plus larges**[13]**.

Guinard**[14]**, cite que l'idée clé de ces travaux, et de fournir une contrepartie virtuelle des objets physiques sur le Web. Les utilisateurs analysent les URI des pages HTML ont été analysés par des utilisateurs, ces pages contiennent la représentation des objets physiques. Mais Les appareils mobiles et les a dirigé vers la représentation des objets physiques (par exemple, contenant l'état des appareils sur des pages HTML ou des manuels d'utilisation). Avec les progrès de la technologie informatique, de petits serveurs Web pourraient être intégrés dans la plupart des appareils**[15].**

Un certain nombre de projets proposent des protocoles ouverts permettant aux appareils de collaborer de manière inter reliée. Parmi eux, **JINI**, **UPnP**, **DNLA**, etc. JXTA, c'est un ensemble de protocoles ouverts pour permettre aux appareils de collaborer de manière inter reliée.Avec l'avènement de WS-* Web Services, plusieurs travaux ont tenté de déployer les fonctionnalités des objets physiques sur des périphériques embraqués, mais cette solution est

très lourde pour les systèmes embarqués, du à leurs capacités limitées, conduit à un certain nombre de travaux visant à les déployer sur des périphériques intégrés et des réseaux de capteurs. Parmi les systèmes qui ont proposé l'intégration des capteurs avec l'internet,SenseWeb et Pachube, ces deux systèmes offrent une plate-forme permettant le partage des données collectées par les capteurs à l'aide de services Web, pour transmettre les données sur un serveur central**[14]**.

Une des premières tentatives d'un Web des objets composé d'objet intelligents est mentionnée par V. Stirbu**[16]**. Mais ces tentatives, et d'après le commentaire de Guinard**[13],** s'intéressent principalement à la découverte des objets, mais pas sur la façon dont les services exposent leur fonctionnalité.

La concrétisation de l'idée de l'intégration des objets physiques au web a commencé en 2002 avec le projet Cooltown[17].

L'idée du Web en tant que couche d'application pour l'IOT a commencé à apparaître en 2007, dont plusieurs chercheurs ont commencé à travailler sur ces concepts. Parmi eux, Dominique Guinard et Vlad Trifa ont lancé la communauté en ligne Web des objets **[2]** et ont publié le premier manifeste WoT, préconisant l'utilisation de normes Web (REST, sémantique légère, etc.) pour créer la couche d'application de l'IoT. Le manifeste a été publié avec une implémentation sur la plate-forme Sun SPOT. Dans le même temps, Dave Raggett du W3C[1] a commencé à parler d'un web des objets lors de divers événements W3C et IoT. Erik Wilde a publié « Putting Things to REST», un document conceptuel auto-publié sur l'utilisation de REST pour détecter et contrôler des objets physiques.

Les mentions précoces du Web des objets comme terme apparaissent également dans Stirbu et al **[16]**.

À partir de 2007, Trifa, Guinard, Wilde et d'autres chercheurs ont essayé de publier leurs idées et leurs concepts, mais leurs travaux ont été rejetés par la communauté de recherche Wireless Sensor Networks WSN, en raison que les protocoles web utilisés sont très gourmands et nécessitent une optimisation pour les adapter aux contraintes des capteurs**[2]**.

Au début de 2009, un certain nombre de chercheurs WSN tels que David Culler, Jonathan Hui, Adam Dunkels et Yazar Dogan **[18]**ont montré l'efficacité des protocoles évalué l'utilisation de protocoles Internet et Web pour les nœuds capteurs de faible puissance et ont montré la faisabilité de l'approche**[19]**.

[1] organisme de standardisation à but non lucratif, fondé en octobre 1994 chargé de promouvoir la compatibilité des technologies du World Wide Web telles que HTML5, HTML, XHTML, XML, RDF.

A la lumière de ces résultats, Guinard et Trifa **[14]** ont présenté leur mise en œuvre de ces concepts lors d'une conférence, et leurs résultats, cette fois ci sont acceptés par l'organisation World Wide Web en 2009.

Et en 2010, Guinard et Trifa et Wilde **[14]**, ont proposé leur premier modèle de l'architecture Restful pour les objets physiques.

À la suite de cela, Guinard et Trifa ont présenté leur mise en œuvre de bout en bout des concepts et l'ont présenté dans une publication évaluée par les pairs acceptée lors de la conférence World Wide Web en 2009**[14]**.

En s'appuyant sur cette mise en œuvre et, en réunissant les efforts, une architecture RESTful pour les choses a été proposée en 2010**[20]**, par Guinard, Trifa et Wilde **[2]**, qui ont fondé aussi et dans la même, le premier forum des web des objets où les chercheurs pourraient exposer leurs idées et leurs découvertes sur le domaine de web des objets.

En 2011, Dominique Guinard et Vlad Trifa avec Niall Murphy et Andy Hobsbawm, ont fondé EVRYTHNG**[21]**, l'une des premières entreprises de logiciels de cloud, qui exploitent pleinement le Web des objets pour répondre aux besoins de l'industrie **[14]**.

II.2. Intégration des objets physiques dans le web :

Selon Guinard**[22]**, il y a trois **(**03) approches d'intégration des objets physiques dans le web : Intégration directe, intégration par un proxy, et intégration par le Cloud **(voir Figure: 5)**.

a. Intégration directe :

Cette approche consiste à embarquer des serveurs web directement sur les systèmes, ou objets physiques limités en ressources, en sorte que leurs interfaces web API soient directement accessibles depuis le web.

b. Intégration par une passerelle intelligente Smart Gateway :

Dans le cas des objets intelligents limités en ressources, les protocoles web standards ne sont pas adaptés, car trop consommateurs en termes d'énergie, de calcul, de mémoire et de bande passante. De plus, certains objets intelligents ne les supportent pas nativement**[23]**. C'est généralement le cas des réseaux de capteurs sans-fil**[2]**. Dans ce cas, l'intégration du monde physique (objets intelligents) au Web, passe par l'utilisation d'un dispositif matériel appelé passerelle intelligent "Smart Getway", qui joue le rôle d'un proxy entre le réseau interne (les objets physiques), qui ne communiquent pas via IP) et le Web, où le serveur web est embarqué sur lui, et tous les objets physique exposent leur API à travers lui.

c. Intégration par le Cloud :

Une autre approche pour intégration, c'est l'intégration par le Cloud **[11].**

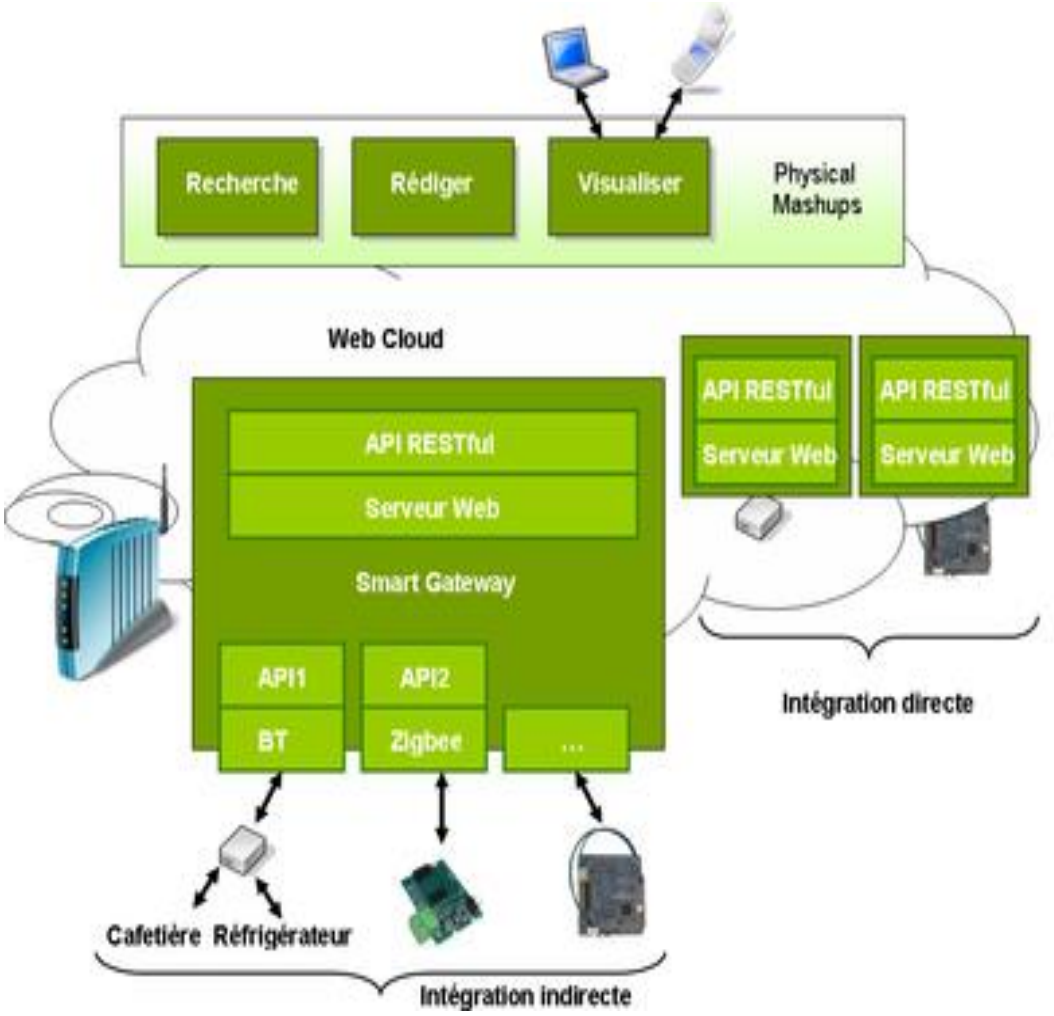


Figure 5 : Les différentes approches d'intégration des objets au Web

[**Dominique Guinard**, A Web of Things Application Architecture -Integrating the Real-World into the Web, These, 2011.]

II.3. Architecture pour le web des objets

Guinard **[2]** a proposé une architecture pour le web des objets, cette architecture est une Pile de 04 couches basée sur la pile de l'IOT *Networked Things* (**voir Figure : 6).** Ces couches sont :

a. Couche d'accessibilité

C'est la couche la plus basse du modèle, elle permet aux objets physiques d'accéder à Internet, en leur garantissant d'exposer leurs services via des API Web programmables. C'est la couche centrale du WoT.

b. Couche de recherche *Findability Layer*

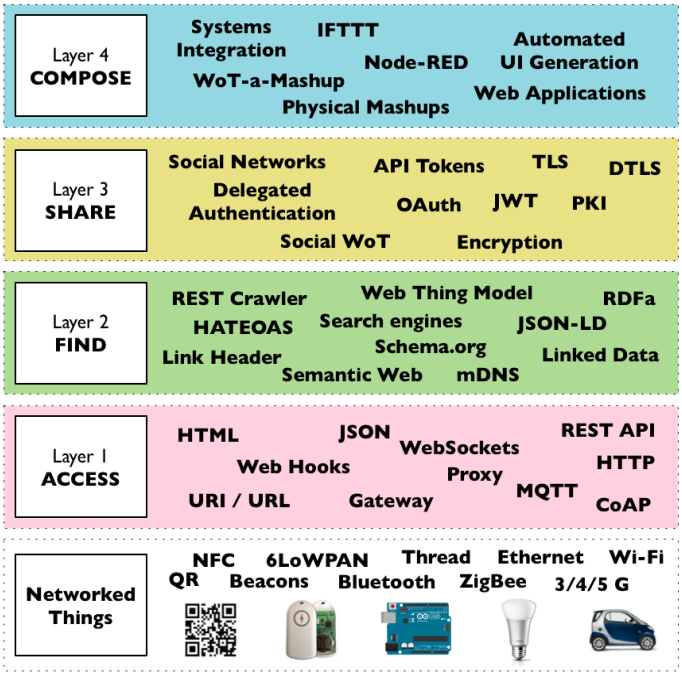
Elle permet d'intégrer les technologies Web sémantiques telles que les micro-données JSON-LD et HTML5 dans les 'API des objets physiques. Elle définit un modèle et une syntaxe de données que les applications et les objets physique doivent les respecter, cette couche l'interopérabilité.

c. Couche de partage *Sharing Layer*

Cette couche est responsable de partager les données générées par les objets physiques, comme les mesures de température, humidité …ect, de manière efficace et sécurisée sur le web, grâce aux mécanismes d'authentification Web.

d. Couche de composition *Composition Layer*

Cette couche permet de composer les services WOT pour l'intégrer dans des applications web, appelée applications web composite ou ***Mashup***.



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

Figure 6 Architecture des web des objets WOT

[Dominique D. Guinard et Vlad M. TRIFA, Building the Web of Things, with exemples in node JS and RASPBERRYPI]

II.3.1. Raspberry PI

D'une part les objets connectés interagissent avec la réalité physique comme ce sont des objets, d'autre part, comme ils sont connectés à Internet ou au Web, ils font également partie de la réalité virtuelle. La nécessité de simplifier au maximum les montages électriques et

électronique ça conduit vers le choix d'un microcontrôleur, qui est le Raspberry Pi2, car il a un énorme potentiel dans le monde d'Internet of Things, où les machines vont directement interagir et contrôler d'autres machines, sans intervention humaine.

C'est un petit ordinateur de la taille d'un Smartphone, vendu à faible prix. Tout l'intérêt du Raspberry PI est de pouvoir installer une distribution de GNU LinuxDebian légèrement modifiée, ce qui permet l'exécution de tous les outils et langages de programmation disponibles sous distribution Linux (Debian). C'est à dire Java, Python, Git, etc **[1]**.

Le Raspberry Pi est un ordinateur entièrement fonctionnel, un système sur puce, qui fonctionne sur un système d'exploitation Linux spécialement conçu pour lui, appelé Rasbian, qui est l'OS officiel de Raspberry Pi, où d'autres systèmes d'exploitation tiers, tels que Firefox OS, Android, RISC OS, Ubuntu Mate…etc., peuvent être installés sur Pi, même la version Windows 10 est également disponible pour Pi. Tant que un ordinateur, il a la mémoire, le processeur, les ports USB, la sortie audio, le pilote graphique pour la sortie HDMI et, comme il s'exécute sur Linux, la plupart des applications logicielles linux peuvent y être installées **[1]**.

II.3.2. Architecture REST

a. Introduction aux web services

Les Web Services sont des composants web basés sur le protocole web standard http, ils exécutent des tâches précises, et qui respectent un format spécifique, tel que XML. Cela permet aux applications de faire appel à des fonctionnalités à distance, en simplifiant ainsi l'échange de données. Les Web Services permettent aux applications de dialoguer à travers le réseau, indépendamment de leur plate-forme d'exécution et de leur langage d'implémentation [**1]**.

b. Définition de REST

REST (Representational State Transfer) ou RESTful est un style d'architecture pour les systèmes hypermédia distribués, créé par Roy Field en 2000 utilisant les spécifications originelles du protocole http, et permettant de construire des applications (Web, Intranet, Web Service). Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière **[7]**.

REST permet d'implémenter des systèmes fiables, flexibles et interopérables. Pourtant, les protocoles qui implémentent cette architecture ne sont pas toujours adaptés au monde des systèmes embarqués, HTTP, par exemple, a des entêtes codés en ASCII, ce qui augmente beaucoup l'*overhead* du protocole quand chaque requête retourne un nombre faible de données (ex. dans les réseaux de capteurs sans fil, où chaque message est souvent une seule

mesure.

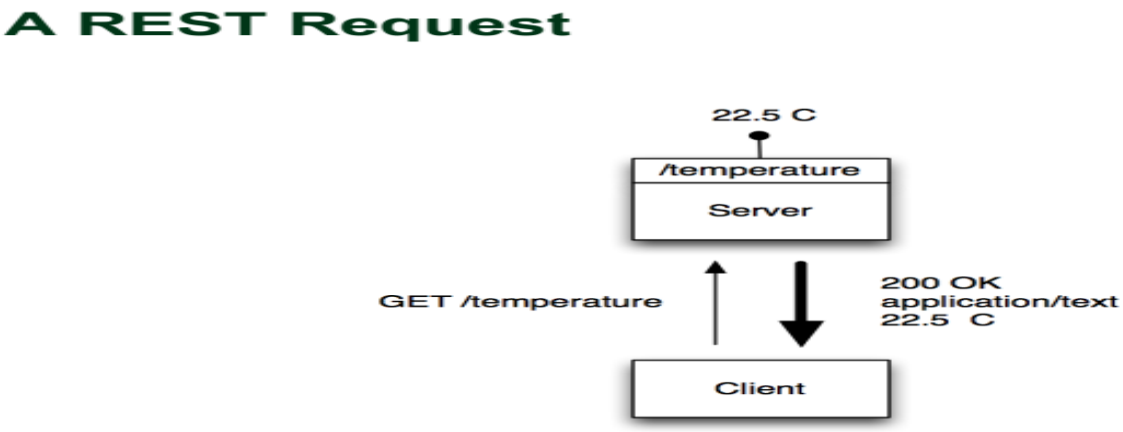


Figure 7 : Requête REST

Cette figure montre un exemple Requête réponse REST, un client envoie une requête GET vers la ressource température sur le serveur, ce dernier renvoie la température sous forme texte.

D'après VIPRET Julien **[8]**, Les réseaux de capteurs vont tenir dans le futur une place dominante dans l'architecture RESTful. Et il a présenté un schéma montrant le rôle important joué par les réseaux capteur. Il a bien expliqué ce rôle dans un schéma sur la figure ci-dessous, présentant l'environnement RESTful. Les réseaux de capteurs vont pouvoir interagir avec le reste du web ou même imaginer que plusieurs réseaux de capteurs interagissent ensemble par l'intermédiaire du nuage. Dans cette vision, les réseaux de capteurs sont complètement intégrés à l'environnement REST.

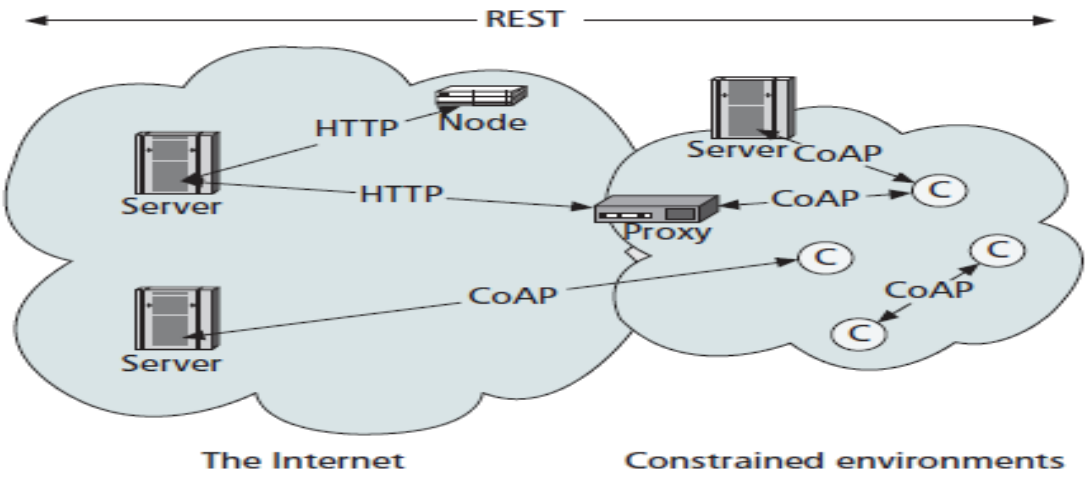


Figure 8 : Environnement RESTful contraint

[**VIPRET Julien**, Rapport d'IRL, Utilisation du protocole CoAP pour la découverte de ressources dans les réseaux de capteurs]

c. Principe de REST

REST propose deux règles de base : l'identification d'une ressource à l'aide d'un URI, et la définition d'une interface uniforme par l'utilisation du protocole HTTP**[24]**.

Le modèle REST peut même être considéré comme la base de l'architecture originale du Web, il est bâti sur cinq principes :

c.1. Client-serveur

Il y a une stricte séparation entre le client et le serveur. Le serveur stocke les données et le client accède ou modifie ces données à travers une interface. Ce qui signifie que le développement du client est indépendant du développement du serveur et que le code du client et du serveur est plus simple.

c.2. Sans état

Chaque requête effectuée vers le serveur est indépendante des autres requêtes passées ou futures. Cela implique que chaque requête doit contenir toutes les informations nécessaires. C'est au client de conserver ces informations.

c.3. Mise en cache

Toutes les ressources peuvent être mises en cache du côté du client. Ainsi, chaque ressource doit indiquer, implicitement ou explicitement, si la ressource peut être mise en cache et pour quelle période. Cela permet de réduire potentiellement le nombre de requêtes entre le client et le serveur.

c.4. Interface uniforme

Ce principe est décomposé en quatre concepts :

- Chaque ressource est identifiée par un nom unique (URI)
- Chaque ressource peut être manipulée par différentes actions
- Le type de contenu de la ressource est nommé explicitement
- Les ressources sont du contenu hypermédia reliées entre elles par des liens.

c.5. Un système hiérarchisé

Chaque ressource peut être divisée en sous-ressources. L'accès à une ressource ne donne pas accès à tout son contenu, mais uniquement à une partie, le reste étant accessible au moyen de liens. De même que le client n'a pas de moyen de savoir sur quel serveur la ressource est localisée. Il peut tout à fait interroger un serveur qui interrogera lui-même un autre serveur et qui renverra la réponse au premier serveur, qui renverra à son tour la réponse au client **[1]**.

Les contraintes précédentes peuvent établir la caractérisation d'un système respectant le modèle REST. Comme évoqué, ces services se mettent facilement en correspondance avec les opérations de HTTP.

Enfin, REST présente l'intérêt d'ouvrir l'accès à des données de façon plus structurées et donc plus simple.

Chapitre 3:

III.Protocoles de communication

III.1. HTTP

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet, son architecture « request/response » le but du protocole HTTP est de permettre un transfert de fichiers (essentiellement au format HTML) localisés grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web.

III.2. MQTT

MQTT est un protocole ouvert, simple, léger et facile à mettre en œuvre[25]. Ce protocole utilise une architecture « publish/subscribe » et qui a les caractéristiques suivantes

- ✓ Particulièrement adapté pour utiliser une très faible bande passante,
- ✓ Idéal pour l'utilisation sur les réseaux sans fils,
- ✓ Faible consommateur en énergie,
- ✓ Très rapide, il permet un temps de réponse supérieur aux autres standards du web actuel,
- ✓ Permet une forte fiabilité si nécessaire.
- ✓ Nécessite peu de ressources processeurs et de mémoires **[16]**.

Figure 9 : Architecture MQTT "Publish/Subscribe"

[**Ph. Truillet** , MQTT : MQ Telemtry/Transport Un protocole pour l'IoT, Novembre 2016.]

Ce schéma explique le principe de « publish/subscribe » :

1. Un client Control Center souscrit sur un topic (sujet) - concernant un capteur d'un autre client RaspberryPi - auprès d'un serveur MQTT appelé Broker.
2. Le client RaspberryPi publie ses capteurs sur le serveur Broker sous forme de topics.

3. Le Broker envoie un message de notification vers le Control Center sur le topic, sur lequel a souscris.

III.3. CoAP

CoAP est un protocole applicatif défini comme un sous ensemble du protocole http, développé par le groupe de travail CoRE, dont le but est d'étendre le paradigme service web à l'Internet des objets et aux applications M2M.**[7] [26]**.

Son architecture est composée de deux couches :

- **Couche message :** assure les séquencements des échanges de bout au bout.
- **Couche Request Response :** cette couche, utilise des méthodes et codes réponses pour les interactions **[17](voir Figure infra).**

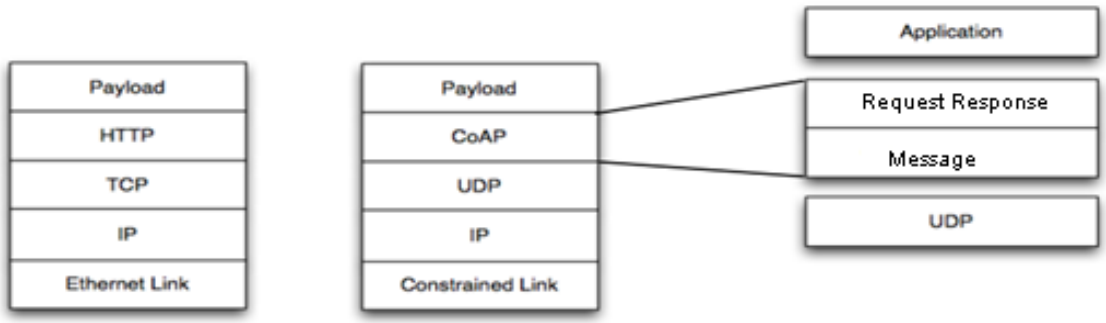


Figure 10 : Architecture de CoAP

Pour implémenter l'architecture REST, CoAP reprend plusieurs concepts d'HTTP. En effet, il y a une correspondance directe entre les méthodes des deux protocoles, ce qui simplifie la conception d'un proxy pour connecter un réseau CoAP à l'Internet "usuel". Pourtant, plusieurs optimisations ont été faites pour le rendre plus adapté aux systèmes embarqués. Premièrement, les échanges se font par des messages asynchrones, avec des mécanismes de fiabilité optionnels ; les protocoles de transport par datagramme (notamment UDP, qui est utilisé par défaut) sont donc privilégiés. En outre, les entêtes sont assez compressés, pour réduire la complexité du décodage et les besoins en bande passante. Finalement, plusieurs fonctionnalités supplémentaires, dont des capacités de proxy et cache simplifiées, de découverte et observation de ressources augmente l'intérêt du protocole pour les réseaux contraints **[9]**.

III.3.1. Les caractéristiques du protocole

CoAP est un protocole conçu pour être utilisé dans un environnement contraint, nous allons voir les caractéristiques qui font de lui le protocole adapté pour ce milieu.

Les principales caractéristiques de CoAP sont les suivantes :

1- Un en-tête concise : Le protocole a un en-tête de base de seulement 4Ko, et une taille totale de 10 à 20 Ko selon les requêtes (en ajoutant les options et les données) **[7]**.

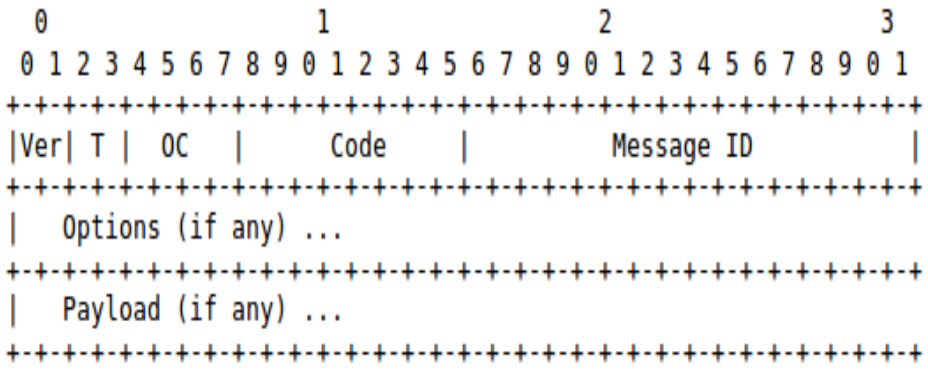


Figure 11 :En-tête du protocole CoAP

Cette figure représente l'entête d'un message CoAP dont ses attributs :

Ver : la version de message CoAP

T :type du message :Confirmable : Le message requiert une réponse ou acquittement

Non-confirmable Le message ne requiert aucune réponse ou acquittement ;

Acknowledgment : Le message confirme la réception d'un message Confirmable. Reset : Initialiser le message dans le cas où le message n'a pu être traité.

Code : le type de message, requête, réponse ou erreur.

Message ID : Identificateur utilisés pour détecter la duplication de messages

Payload (if any) : Ou charge utile, c'est le contenu du message.

Gestion des méthodes : CoAP met à disposition les requêtes GET, PUT, POST et DELETE. Ces méthodes vont pouvoir être utilisées par le client pour accéder aux données. Leur utilisation est similaire à celle de HTTP.

Les URIs : Le protocole supporte les Uniform Resource Identifier, ce qui va permettre de spécifier la cible grâce à un nom d'hôte, un port, un chemin et un paramètre de requête, voici un exemple d'URI CoAP :

Coap: //example.net/.well-known/core?n=Light

Le type de contenu : Le protocole permet de transporter des données d'usages différents, en mettant dans l'entête le type de données de Payload **[26].**

2- La découverte de ressources :

La découverte directe des ressources n'est pas pratique en raison des nœuds dormants, des réseaux dispersés ou des réseaux, où le trafic multicast est inefficace. Ces problèmes peuvent être résolus par l'emploi d'une entité appelée Resource Directory (RD), qui héberge la description des ressources détenues sur d'autres serveurs **[8]**. Afin que les ressources soient découvertes, le serveur CoAP doit implémenter deux interfaces : Lookup et Registration.

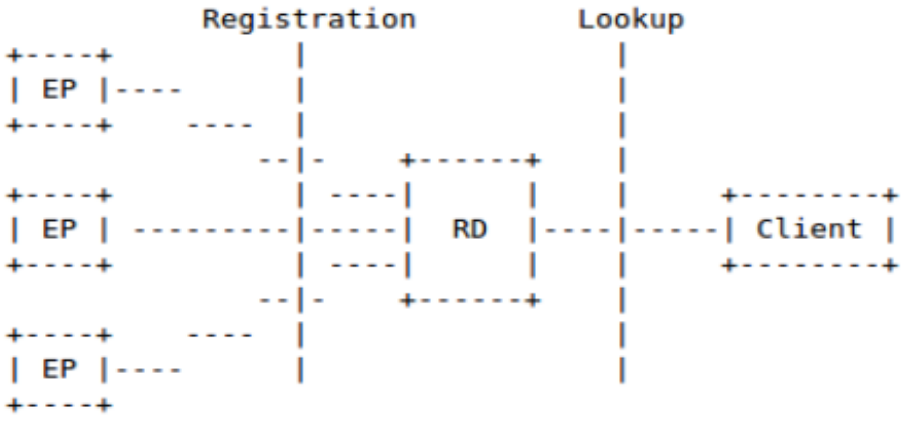


Figure 12 : Architecture d'un Ressource Directory

[**VIPRET Julien**, Rapport d'IRL, Utilisation du protocole CoAP pour la découverte de ressources dans les réseaux de capteurs]

Les EPs, qui sont des serveurs contenant les ressources, envoient des requêtes d'enregistrement de leurs ressources vers l'annuaire RD, à travers l'interface Registration. Le client communique avec l'annuaire à travers son interface Lookup pour chercher la description des ressources.

Afin d'être découvert, tout serveur CoAP doit posséder un chemin commençant par .**wellknow/.** Ainsi, un client va pouvoir envoyer une requête GET à un serveur avec une URI .**wellknow/core** et il recevra, en retour, une liste des services que le serveur propose, comme par exemple, un capteur de température, de luminosité, etc...

Un client va pouvoir cibler sa recherche de ressources selon des critères passés en paramètre de la requête GET. Il va pouvoir spécifier quel type de ressources il recherche.

/.well-known/core ?rt=Light

Les serveurs vont répondre aux requêtes dont le Payload sous Link Format [Z shl Link Format]. Typiquement, il va ressembler à ceci :

<$uri-path> ; ct=$ct ; rt="$rt

- ✓ **$uri-path** désigne l'uri que le client doit fournir pour une requête ultérieure.
- ✓ **$ct** désigne les content-types que le serveur connait.
- ✓ **$rt** désigne le nom de la ressource **[8].**

Nous allons voir une stratégie de découverte de ressources. Cette stratégie est destinée aux réseaux de capteurs composés d'un nombre conséquent de nœuds. Ces nœuds peuvent ne pas être les mêmes. Par exemple, certains nœuds peuvent avoir une autonomie plus importante que d'autres. Nous allons utiliser une nouvelle entité appelée Ressource Directory (RD) qui va nous permettre de stocker les informations liées aux capteurs. Voici les principales caractéristiques de cette découverte de ressources :

- **Utilisation du réseau** : cette stratégie ne surcharge pas le réseau de capteurs de trafic inutile ce qui va réduire le nombre de collisions, et donc le nombre d'erreurs.
- **Type de routage** : le routage multicast n'est pas utilisé. Il surchargerait le trafic.
- **Nœuds dormants** : dans cette stratégie, les nœuds dormants vont pouvoir être découverts avec un système de cache.

3- Découverte du RD

Le rôle principal du RD va être de garder en mémoire l'état du réseau. Pour cela, les nœuds vont lui envoyer leurs caractéristiques (capteurs etc...) via des requêtes POST. La première étape est de découvrir l'adresse du RD pour que les nœuds puissent lui envoyer des requêtes.

Il existe différentes stratégies pour découvrir l'adresse du RD, par exemple, les nœuds peuvent avoir l'adresse du RD en statique au démarrage. Une deuxième méthode consiste à envoyer une requête anycast ou multicast, avec l'utilisation du paramètre Ressource Type (rt).

RequêteGET coap://[ff02::1]/.well-known/core?rt=core-rd

Réponse 2.05 Content </rd> ; rt="core-rd

L'adresse du RD étant connue, la deuxième étape va être d'enregistrer les ressources d'un serveur CoAP au RD. Le serveur CoAP va utiliser la méthode POST, la liste des ressources du serveur sera placée dans la charge utile dans un format adapté. Le serveur va disposer de plusieurs paramètres pour définir ses ressources :

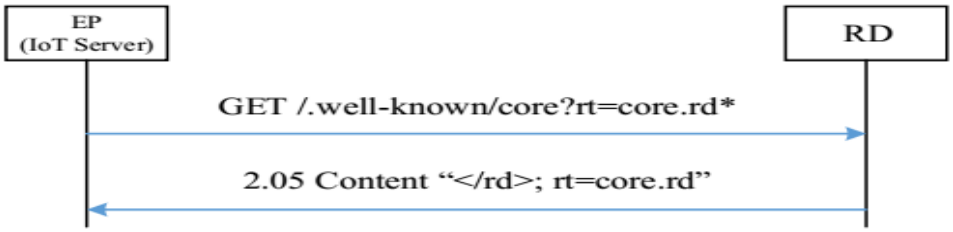


Figure13 : Chercher un Resource Directory

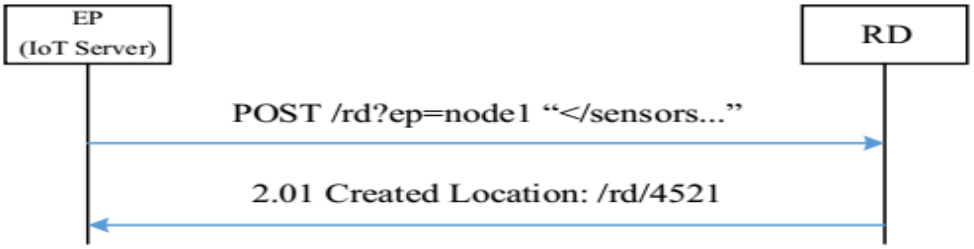


Figure 14 : Enregistrement d'une ressource dans un Resource Directory

- **Le temps de vie (lt)** : ce paramètre va permettre de définir le temps pendant lequel les informations envoyées sont valides. On peut imaginer que lorsque le temps de vie est dépassé, le RD peut supprimer les informations de sa base.
- **L'Etag** : est une clé envoyée par le serveur au RD qui est une sorte de *savepoint*. Ceci va permettre au RD de revalider les informations dans le futur, en envoyer une requête au serveur avec cet Etag. Si le Etag serveur est différent que celui envoyé par le RD, alors, il faut mettre à jour les données.

4- Le proxy et la mise en cache

Le principe de Proxy et la mise en cache est expliqué par la figure ci-dessous.

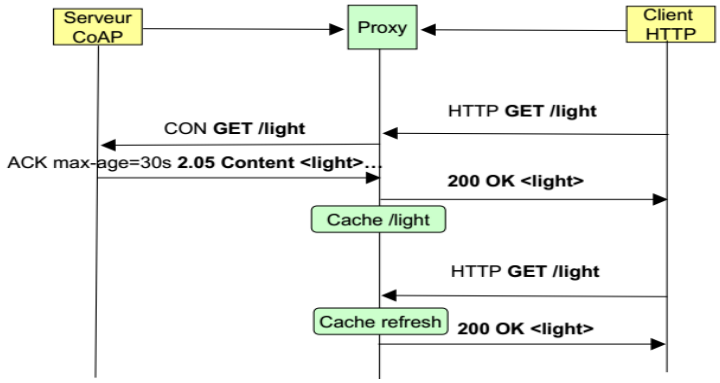


Figure 15 : Interfonctionnement entre CoAP et http

[www.efort.com:COAP (Constrained Application Protocol) : Protocole d'Application pour l'Internet des ObjetsEFORT]

Cette figure représente le principe du Proxy et la Cache du CoAP :

Un client envoie des requêtes HTTP vers un serveur Proxy, qui mappe ces requêtes vers serveur CoAP.

Le serveur CoAP renvoie des réponses vers le Proxy, ce dernier est doté d'une fonction de Cache permettant de stocker ces réponses pour une utilisation ultérieur, afin de minimiser de temps de réponses. Ensuite les réponses sont mappées et envoyées vers le client HTTP.

III.3.2. Fonctionnement du protocole CoAP

Le protocole CoAP utilise le protocole UDP au niveau de la couche transport, il supporte l'IPv6 pour la couche réseau et utilise le protocole de communication 802.15.4 au niveau de la couche liaison, si l'on considère un réseau de capteurs. Concernant le modèle de message, un message CoAP possède un unique identifiant Message ID de 16 bits. Cela permet de l'identifier et de retransmettre le message en cas d'erreurs. Le message peut être de 4 types :

- Le message Confirmable **CON** : Le client attend alors une réponse d'acquittement du serveur de destination. Si au terme d'un temps *T*, le client n'a pas eu de réponse, il considère que le paquet s'est perdu, et retransmet le message dans un intervalle de temps aléatoire, pour contrôler la congestion du réseau.
- Le message Non-Confirmable **NON** : Le client envoie le message sans attente d'acquittement.
- Le message Acquittement **ACK** : permet d'acquitter le client, et donc, de lui assurer la bonne réception du message.
- Le message Reset **RST** : permet au serveur de rejeter la requête envoyée par le client.

III.3.3.Exemple de communication CoAP

Pour mieux comprendre le fonctionnement de CoAP, nous allons voir un exemple de communication entre un client et un serveur.

a) Etape 1 : Le client envoie une requête GET avec un id=123, de type confirmable CON et avec une uri-query = */light*.

b) Etape 2 : Le serveur répond par un 2.00 OK avec le même id que le client, de type acquittement ACK et avec, comme charge utile, la réponse à la requête (non visible ici).

c) Etape 3 : Le client envoie une requête GET avec un id=124, de type confirmable CON, et avec une uri-query = */humidity***.**

d) Etape 4 : Le client retransmet la requête, car le serveur n'a pas répondu dans les délais. Il y a eu un timeout.

e) **Etape 5** : Le serveur répond par un 2.00 OK avec le même id que le client, de type acquittement ACK, et avec comme charge utile la réponse à la requête (non visible ici**).**

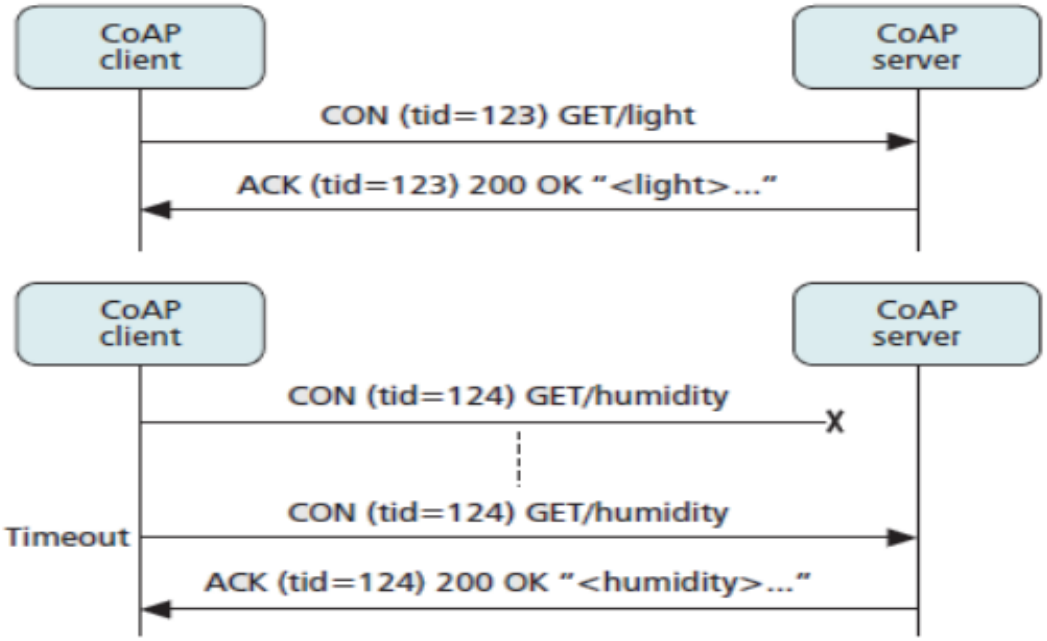


Figure 16 : Exemple de communication client / serveur CoAP

[www.efort.com:COAP (Constrained Application Protocol) : Protocole d'Application pour l'Internet des ObjetsEFORT**]**

III.3.4. Approches alternatives à la découverte

Le RD n'est pas la seule approche du problème de découverte pour les réseaux IoT. Alternativement, il existe d'autres méthodes telles que le service de noms de domaine découverte du service (DNS-SD), qui permet la recherche d'un service donné Via DNS, il existe également Universal Plug and Play (UPnP), qui permet la découverte de périphériques à la maison Réseaux[**27**]**.**

III.3.5. Observation des ressources dans CoAP

Le protocole est basé sur le modèle de conception d'observateur bien connu[**27**], dans ce modèle de conception, les composants appelés "observateurs" s'inscrire à un fournisseur spécifique et connu appelé «sujet», qu'ils ont sont intéressés à être notifiés chaque fois que le sujet subit un changement d'état. Le sujet est chargé d'administrer son liste des observateurs enregistrés. Si plusieurs sujets sont intéressants à un observateur, l'observateur doit s'inscrire séparément pour tous[**27**].

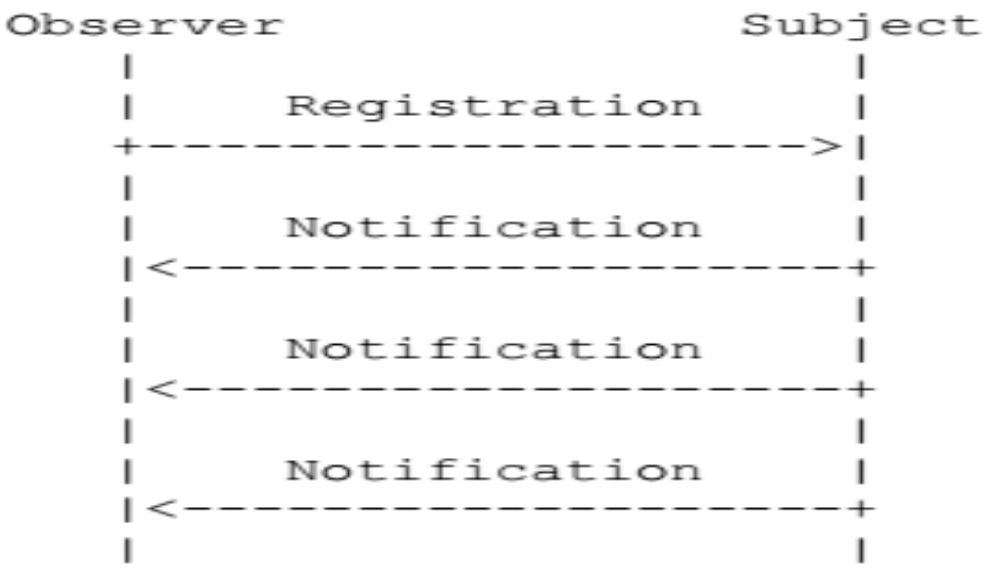


Figure 17 : Le modèle de conception d'observateur

Le modèle de conception de l'observateur est réalisé dans CoAP comme suit :

- **Objet** : Dans le contexte de CoAP, le sujet est une ressource dans l'espace de noms d'un serveur CoAP. L'état de la ressource peut changer au fil du temps, allant des mises à jour peu fréquentes à l'état continu transformations.
- **Observateur** : Un observateur est un client de CoAP, qui est intéressé à avoir une représentation actuelle de la ressource à un moment donné.
- **Inscription** : un client enregistre son intérêt pour une ressource par lancer une requête GET étendue sur le serveur. En plus de renvoyer une représentation de la ressource cible, cette demande fait en sorte que le serveur ajoute le client à la liste des observateurs de la ressource.
- **Notification** : chaque fois que l'état d'une ressource change, le serveur notifie chaque client dans la liste des observateurs de la ressource. Chaque notification est une réponse CoAP supplémentaire envoyée par le serveur en réponse à la requête GET étendue unique, et **La figure 18** ci-dessous montre un exemple d'un client CoAP s'inscrivant s'intéresse à une ressource et reçoit trois notifications.

La première avec l'état actuel lors de l'inscription, puis deux lors des modifications à l'état des ressources. La demande d'inscription et les notifications sont identifiées comme telles par la présence de l'Observateur Option définie dans ce document. Dans les notifications, l'Observateur l'option fournit en outre un numéro de séquence pour la réorganisation

détection. Toutes les notifications portent le jeton spécifié par le Client, de sorte que le client peut facilement les corréler à la demande.

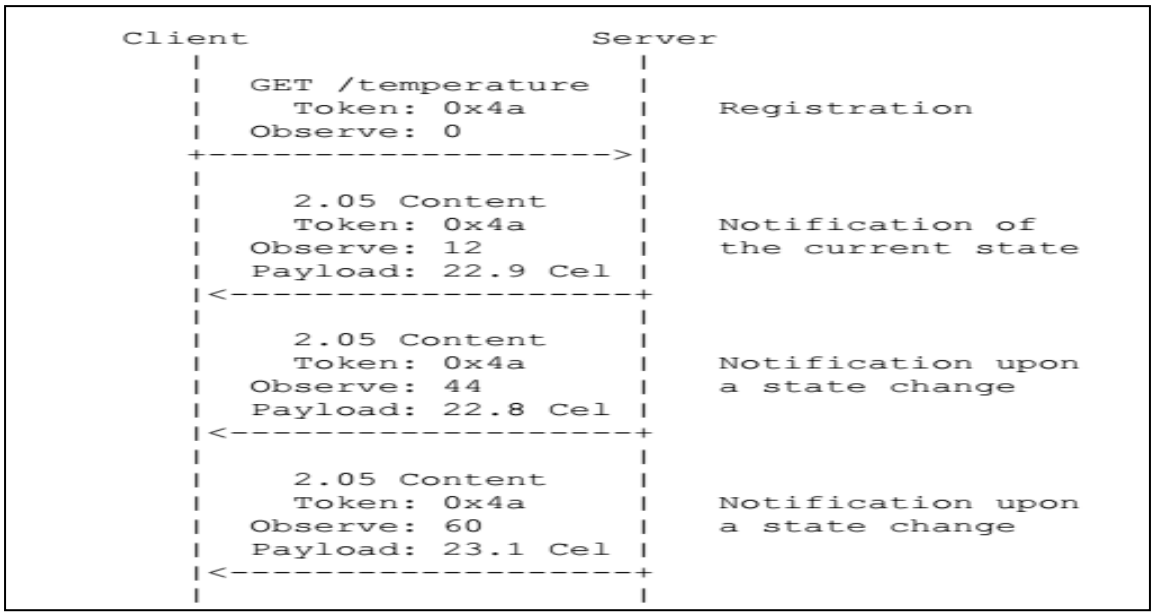


Figure 18 :Observation d'une ressource dans CoAP

Un client reste sur la liste des observateurs tant que le serveur peut déterminer l'intérêt continu du client pour la ressource. Le serveur peut envoyer une notification dans un message CoAP confirmable demander un accusé de réception auprès du client. Lorsque le client Dés enregistre, rejette une notification ou la transmission d'un la notification éteint après plusieurs tentatives de transmission, le client n'est plus intéressé par la ressource et est supprimé par le serveur de la liste des observateurs **[27]**.

III.3.6. Interopérabilité et représentation des informations

Afin d'assurer l'interopérabilité, il est nécessaire d'avoir des normes standards de fait, deux formes d'échange de données, qui sont : XML et JSON[**28**].

XML :

La syntaxe d'XML contient deux éléments principaux permettant de structurer les informations : les balises, et les attributs : une balise peut contenir un ou plusieurs attributs dont chacun possède une valeur, puis on trouve le contenu, et enfin la balise fermante : **<balise attribut_1="valeur_1" attribut_2="valeur_2"> Contenu de la balise </balise>** XML n'est en réalité pas un langage, mais un métalangage. Il définit un certain nombre de règles, mais ne définit pas de vocabulaire (les mots-clés). Tout langage XML est accompagné de la définition de son vocabulaire, définition qui peut prendre plusieurs formes, comme une DTD ou un schéma XML.

XML est très utile pour la conception de documents structurés de taille importante, et offre aux documents une forte valeur sémantique. Par ailleurs, XML gère les espaces de noms, permettant d'utiliser plusieurs vocabulaires différents dans un même document.

Si XML est très adapté à la représentation de documents structurés, il semble l'être moins pour la représentation de petites quantités d'information, ce qui nous amène à considérer une représentation plus adaptée : JSON [**28]**.

JSON :

JSON (prononcer « Jason »), acronyme pour JavaScript Object Notation, propose une alternative agréable à XML pour la transmission de données. Apparu vers le début des années 2000, il a mis quelques années à se démocratiser, mais est aujourd'hui assez répandu et utilisé comme substitut à XML en AJAX.

Syntaxe de JSON

JSON reprend la syntaxe du langage JavaScript, et se base sur deux structures de données :

- **L'Array** : un liste ordonnée de valeurs, entre crochets et séparées par des virgules : [Valeur_1, Valeur_2, ..., Valeur_n].
- **L'Object** : un tableau associatif ou hashtable, associant des clés (uniques) à des valeurs :

```
{  "Clé_1" : Valeur_1,
  "Clé_2" : Valeur_2,
 ...
 "Clé_n" : Valeur_n}
```

Une valeur peut être un objet, un array, une chaine de caractères, un nombre flottant, ou encore les valeurs ture, false et null.

Points forts

Nous venons de voir que la grammaire de JSON est très simple, puisque 5 règles suffisent à la décrire. Certaines seraient éclatées en plusieurs s'il fallait écrire une grammaire BNF lisible (la description des chaînes ou nombres ne se ferait peut-être pas en une seule règle), mais même dans ce cas, elles resteraient en nombre raisonnable. Même si ces cinq règles devaient se séparer en vingt, lors de l'implémentation d'un analyseur syntaxique, on serait encore loin derrière les 89 utilisées pour décrire la grammaire d'XML. Le parsage de données représentées en JSON apparaît donc comme étant plus léger, plus simple que pour du XML,

ce qui est intéressant en particulier sur des systèmes à ressources limitées, comme les systèmes embarqués.

Chapitre 4:

IV.Conception et Implémentation

Dans ce chapitre nous allons proposer et développer une plateforme, nous permettant de charger les composants dynamiquement, puis nous allons tester cette plateforme par des scénarios afin d'évaluer les fonctionnalités offertes.

IV.1. Conception

IV.1.1. Introduction

Notre contribution sera d'implémenter la couche d'accès ***Access Layer*** du modèle WOT **(voir Figure : 19)**, dont l'objectif est d'intégrer des objets physiques communicants à travers le Raspberry qui joue le rôle d'une passerelle ***Smart Gateway***, et en utilisant le CoAP comme un protocole applicatif. Cette implémentation est un middleware, qui joue le rôle d'un conteneur de composants, où chaque composant représente une abstraction software d'un objet physique. Ces composants, sont des composants écrits en java et chargés dynamiquement par le middleware.

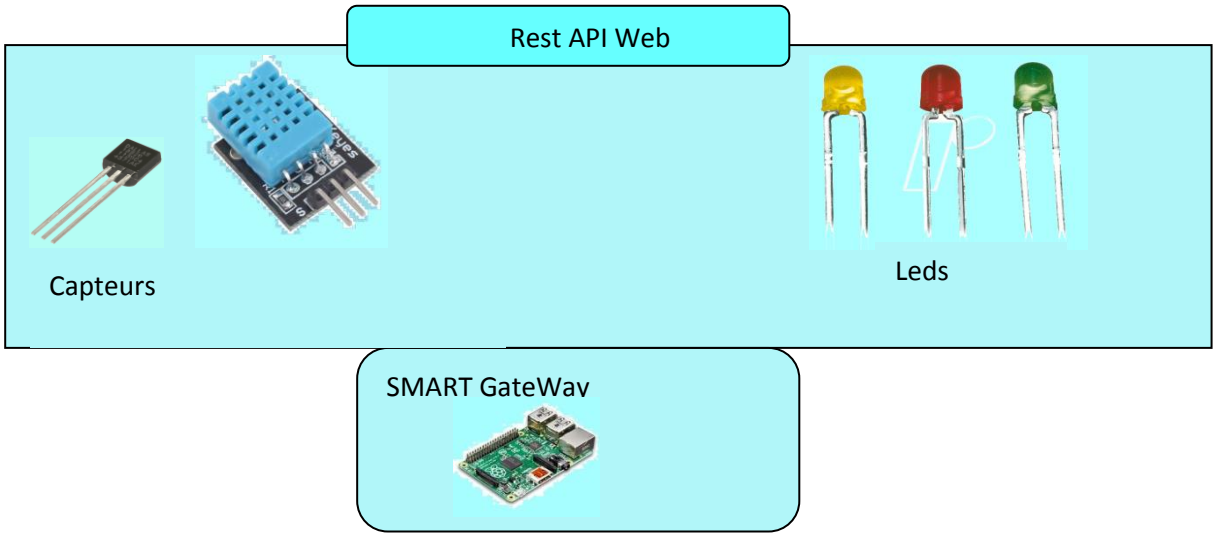


Figure 19 :Implémentation de la couche d'accès

Access Layer du modèle WOT

IV.1.2. Architecture :

Dans cette partie nous décrivons notre plateforme que nous allons développer. Pour intégrer les objets physiques dans le web, nous avons opté à une architecture de Middleware en trois (03) couches :

Web communication interface : Elle gère les interfaces web et s'occupe de la communication.

Components Container : Elle sert à contenir les composants chargés et mis en place, elle gère le cycle de vie des composants. En plus, elle assure la communication avec les API natives des objets physiques.

Natives APIs : Cette couche assure l'interface avec les objets physiques, elle sert à envoyer les requêtes provenant des composants vers les objets physiques, elle sert aussi à communiquer avec les objets physiques en offrant un ensemble de fonctions dédiées au matériel.

En plus, cette architecture est composée de quatre (04) modules :

Starter : Son rôle est de démarrer la plate forme.

Stopper : Ce module permet de stopper la plateforme.

Loader : Ce module sert à charger les composants des objets physiques.

Uploader : Il permet de décharger un composant. (**VoirFigure : 20**)

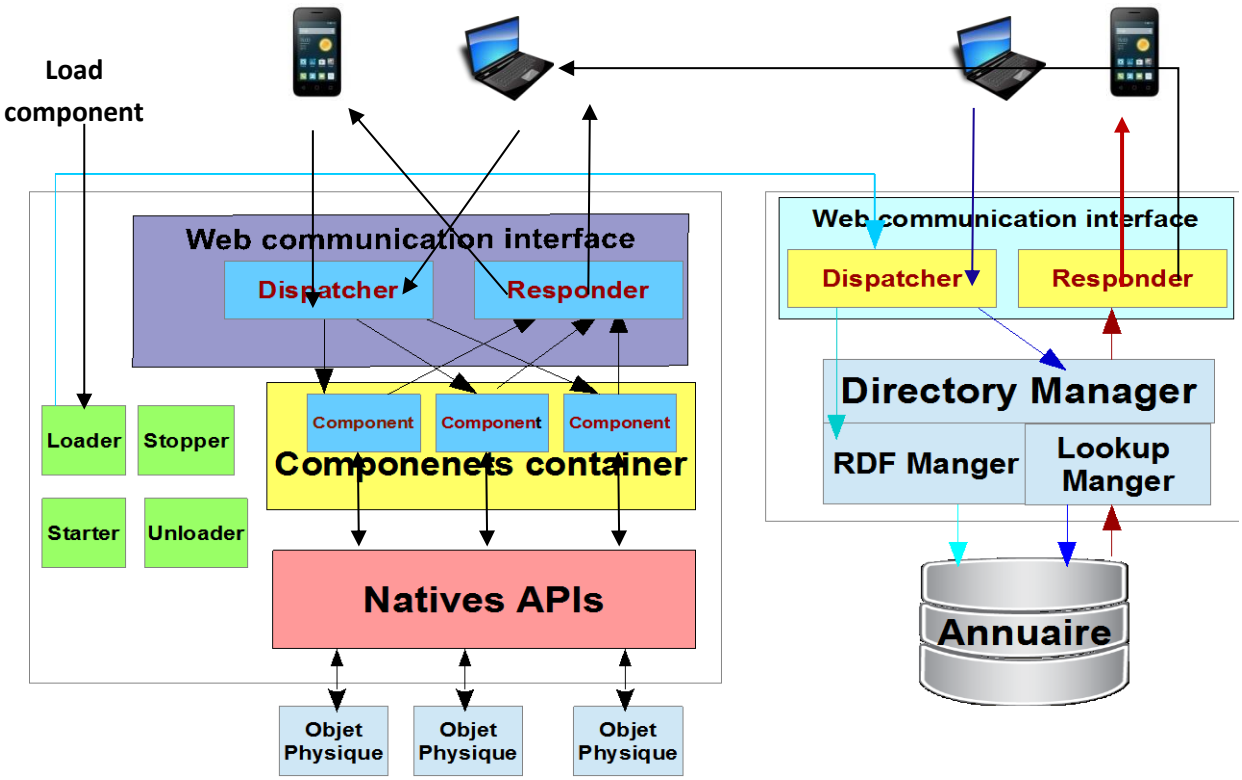


Figure 20 : Architecture de la plateforme proposée

Cette plateforme contient également une 2e partie qui assure la gestion de l'annuaire. Cette partie est formée de deux couches :

Couche Web Communication interface : Elle s'occupe de la communication avec le middleware et, aussi, avec les clients au moment de la découverte des ressources.

Directory Manager: Cette couche a deux fonctions : elle permet l'enregistrement des descriptions des composants dans l'annuaire d'un côté assuré par le RDF MANAGER, et de

l'autre côté, elle permet de chercher la description des ressource dans l'annuaire à travers le Lookup Manger.

Notre plateforme est composée de deux serveurs CoAP.

Description de la plateforme :

- **Partie 1 :**

La couche Web Communication Interface : est un serveur Coap qui intercepte les requêtes par son Dispatcher, ce dernier génère des appels vers les composants appropriés.

Les composants sont des composants précompilés écrit en java, permettant la représentation des objets intelligents. Ils offrent les interfaces permettant au Middleware de communiquer avec les objets physiques de l'environnement à travers les API natives. Ces composants sont chargés et déchargés dynamiquement, et cela selon l'apparition des objets physiques dans l'environnement.

On trouve aussi, dans cette couche, le respondre, qui reformule les réponses des objets physiques générées par les composants.

Les composants sont hébergés dans la couche components container, un conteneur qui gère le cycle de vie de ces composants, il offre aussi une interface uniforme pour l'accès aux objets physiques en masquant leurs détails techniques

Notre plateforme offre une interface avec un jeu de commandes pour charger et décharger les composants, et pour démarrer et stopper la plateforme, implémenté par les modules ci cités.

Pour mieux décrire les composants , et en conséquence, les objets physiques, nous associons à chaque composant une description sémantique suivant une ontologie du domaine de l'WOT définie par le W3C

- **Parti2 :**

Cette partie de la plateforme, se charge de la manipulation de l'annuaire des objets physiques , elle contient la couche Web Communication Interface qui est un serveur Coap, ce dernier reçois par son Dispatcher, les requêtes d'enregistrement des descriptions ,provenant du middleware au moments du chargement des composants par le loader, ainsi les requêtes envoyées par les clients, au moment de la découverte des objets physiques. Pour cela, le serveur Coap doit implémenter une interface assurant les fonctionnalités de la découverte et l'enregistrement. Le responder de cette couche, reformule les réponses aux requêtes de découverte.

Vu les contraintes de la passerelle intelligent ***Smart Gateway***, taille mémoire et capacité du traitement limitées par rapport aux pcs, et pour faciliter le processus de découverte des objets, nous avons proposé de séparer les deux serveurs Coap, celui du middleware et celui de l'annuaire, afin de rendre l'architecture extensible, en partageant l'annuaire entre plusieurs Raspberrys, ce qui rend l'ajout de nouveaux équipements raspberrys plus facile**.(Voir Figure : 20)**

IV.2. Implémentation :

IV.2.1. Choix du langage et motivation :

Les objets physiques apparaissent et disparaissent fréquemment dans l'environnement, ce qui nécessite un langage qui permet d'ajouter et enlever dynamiquement les composants, manipulant les objets physiques selon leur apparition et disparation dans l'environnement.

C'est pour cette raison que nous avons opté à un langage semi interprété basé sur la machine virtuelle, cette dernière, offre la possibilité de charger et décharger dynamiquement les classes représentant les composants à manipuler, plusieurs langages offrent cette possibilité, parmi ces langages, nous avons choisi JAVA , c'est le langage avec lequel nous avons nous familiarisé à programmer.

Les classes à charger sont des classes précompilées en Bytecode, qui est un code intermédiaire indépendant de la machine physique, et de la plateforme sur lesquels seront exécutées, ce code sera chargé et interprété par la machine virtuelle.

Avant de continuer l'explication de notre implémentation, un petit rappel sur la machine virtuelle , et son principe de fonctionnement.

La machine virtuelle java et l'environnement d'exécution java JRE :

Le JRE est une plateforme logicielle composée de :

- ✓ Un ensemble d'APIs exploitables par les programmeurs, une machine virtuelle qui joue le rôle d'un émulateur de machine physique, dans laquelle les classes seront chargées par Le Classloader.
- ✓ Classloader : est un module de JRE qui permet de chercher l'emplacement des classes pour les charger dans la machine virtuelle. Java offre la possibilité aux utilisateurs pour définir leur chargeur pour répondre à leur besoin.

IV.2.2. Choix du Californium et Motivation :

C'est l'implémentation de CoAP en java développée par Daniel Pauli, et Dominique Im Obersteg ETH Zurich en suisse**[29]**, dont l'idée est d'offrir un framework open source, permettant aux programmeur java, de développer leur code pour interagir avec les points

terminaux EndPoints de CoAP, en masquant les détails techniques de CoAP, comme la transmission du message.

Notre choix pour californium, réside dans sa souplesse d'utilisation, avec un peu de code de programme, on arrive à développer une application…

Il offre un ensemble de packages de classes et interfaces. Son architecture hiérachisée est représentée par le diagramme ci-dessous.

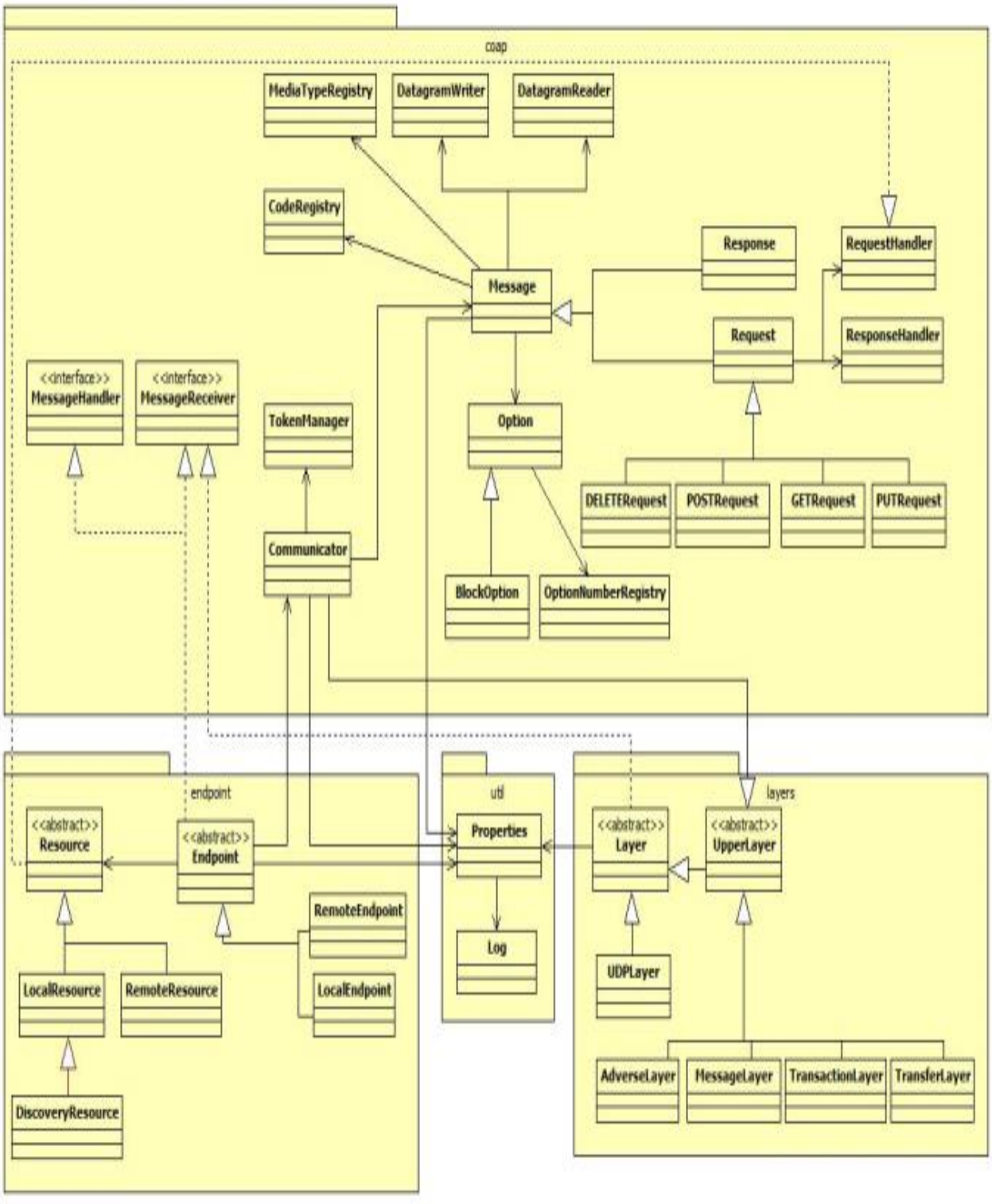


Figure 21 : Architecture Hiérarchisée de Californium

[Daniel Pauli and Dominique Im Obersteg, Californium, Spring 2010.**]**

Pour développer notre plateforme, nous avons utilisé un ensemble de classes de Framework Californium : CoapServer, CoapResource, CoapRequest, CoapResponse et Request CoapExchange.

Les deux figures **(Figure : 22,23)** représentent les diagrammes de séquence d'une communication Coap Requête Réponse en Californium.

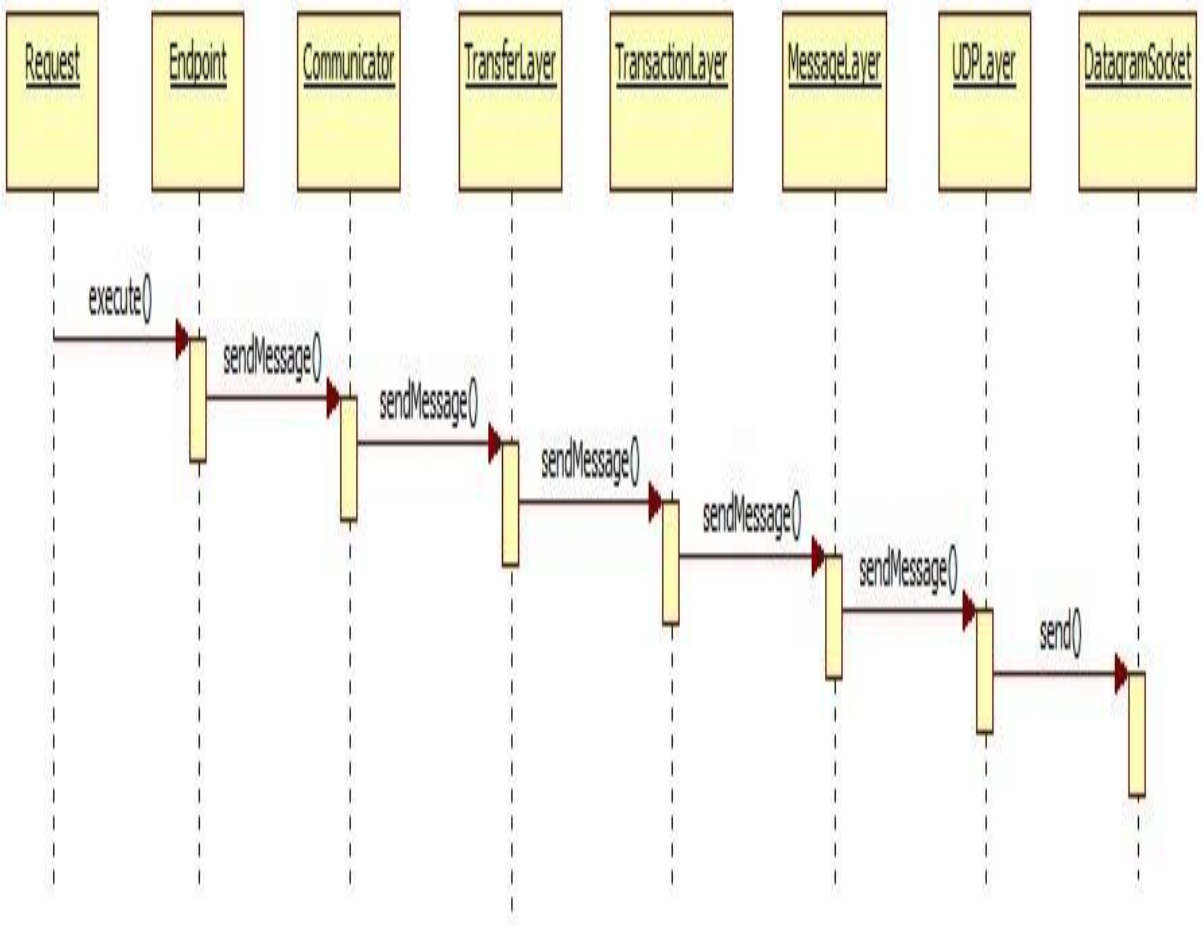


Figure 22 : Diagramme de séquence de l'envoi d'une requête en Californium

[Daniel Pauli and Dominique Im Obersteg, Californium, Spring 2010.**]**

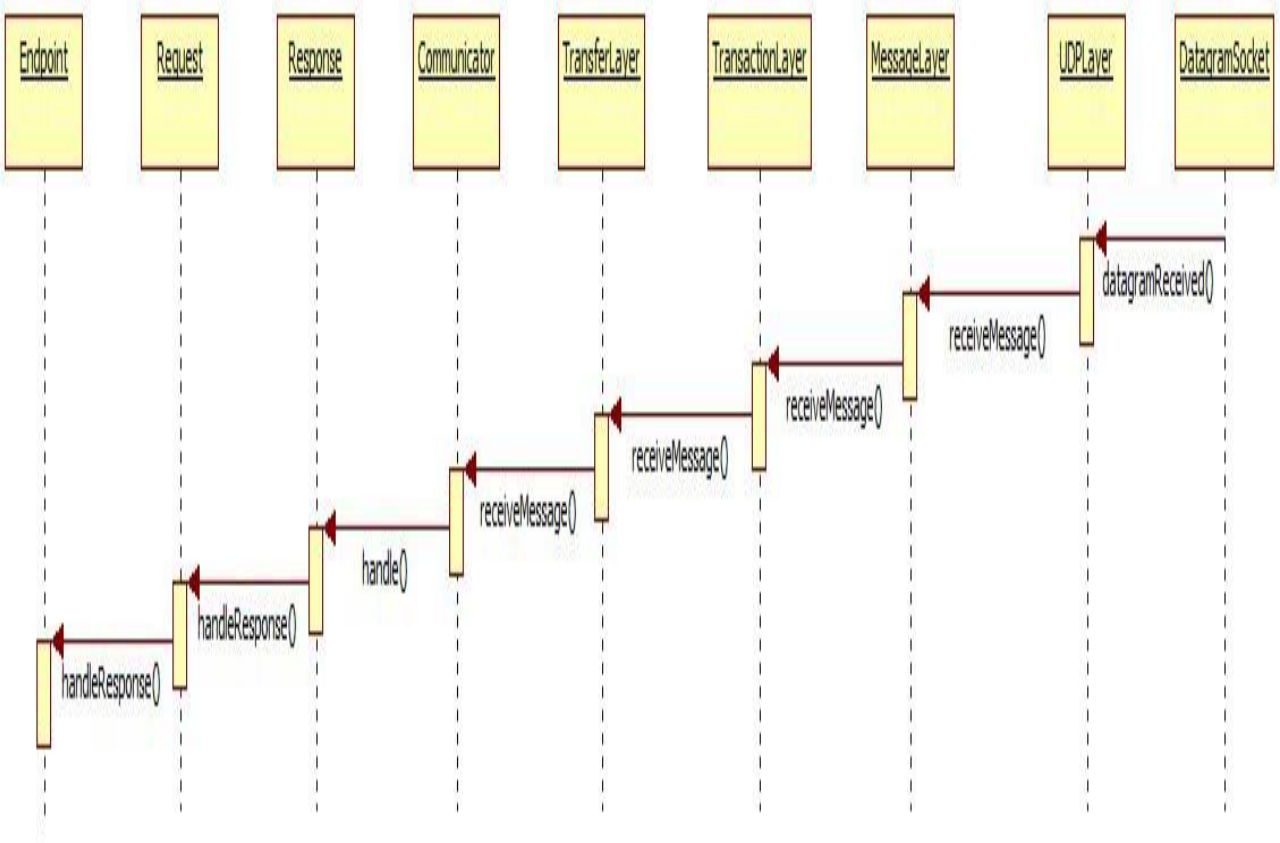


Figure 23 : Diagramme de séquence d'une réponse en Californium

[Daniel Pauli and Dominique Im Obersteg, Californium, Spring 2010.**]**

IV.2.3. MONGODB :

C'est la bibliothèque offrant une API (Gestionnaire de base de données) NOSQL DBMONGO, ce type de base de données sera détaillé dans les prochaines sections de l'implémentation du serveur annuaire.

IV.2.4. Pi4J :

Pi4J **[30]**est un projet open source fournissant une interface de programmation API pour Java, permettant aux programmeurs d'interagir avec les capacités d'E / S de bas niveau, sur la plate-forme Raspberry Pi. C'est une couche d'abstraction du matériel où le changement du matériel n'influe pas sur le code du programme.

IV.2.5. Développement de la plateforme :

Pour implémenter notre plateforme, nous avons choisis le langage java version de JDK 1.8, avec l'IDE blueJ et Eclipse mars. Notre plateforme est composée de deux parties :

La première partie :

Une couche Web Communication Interface implémentée par un serveur Coap, en utilisant la classe CoapServer.

La première partie est implémentée par la classe CoapServer du californium :

Import org.eclipse.californium.core.CoapServer;

Public class ComponentsServer {

Public static void main (String [] args) {

Code server=new CoapServer () ; //* permet de créer le serveur CoAP implémentant la couche Web Communication Web du middleware

}

}

Start : permet de démarrer le serveur

```
staticprivatevoidStarter () {/* Module Starter du démarrage du serveur de
composant en conséquence, demurrage du middleware
Up=true;
server.start();
}
Stop: Permet de stopper le serveur
staticprivatevoid Stopper(){/* Module de qui est chargé de stopper le serveur
CoAP et conséquence , arrêter le middleware
server.stop();
Up=false;
}
Unloade:Décharger un composant Charger
staticprivatevoid Unloader(Object composant){//*Module de déchargement des
composants
System.out.println("Veuiller Tapper le nom du composant à décharger:");

}
}
```

Implémenté par l'instruction :Server. Stop

Load : Cette commande permet d'implémenter le module Loader de la plate-forme, pour ce faire on définit un chargeur Loader une classe héritant de la classe abstraite de Java UrlclassLoader.

Code du module :

Exemple :

Pour implémenter les composants à charger, nous avons définit une classe abstraite IOTResource enxtends de la classe abstraite CoapResource. Celui qui veut ajouter un nouveau objet physique à la plate forme, il n a qu'à définir une classe héritant de la classe IOTResource et implémente ses méthodes abstraites en effectuant un Override de ses méthodes par le code

communicant avec l'objet physique de la bibliothèque Pi4j, ce code sert à implémenter la couche nativeAPIs.

Pour implémenter les composants à charger, un fichier INI d'initialisation utilisé, qui est chargé à chaque démarrage du serveur, ce fichier d'initialisation contient des paramètres utilisés par le serveur comme l'URI du serveur annuaire qui sera utilisé par le serveur des composants au moment de l'enregistrement, ou au moment de la découverte des ressources.

La figure montre le diagramme de classes de notre middleware.

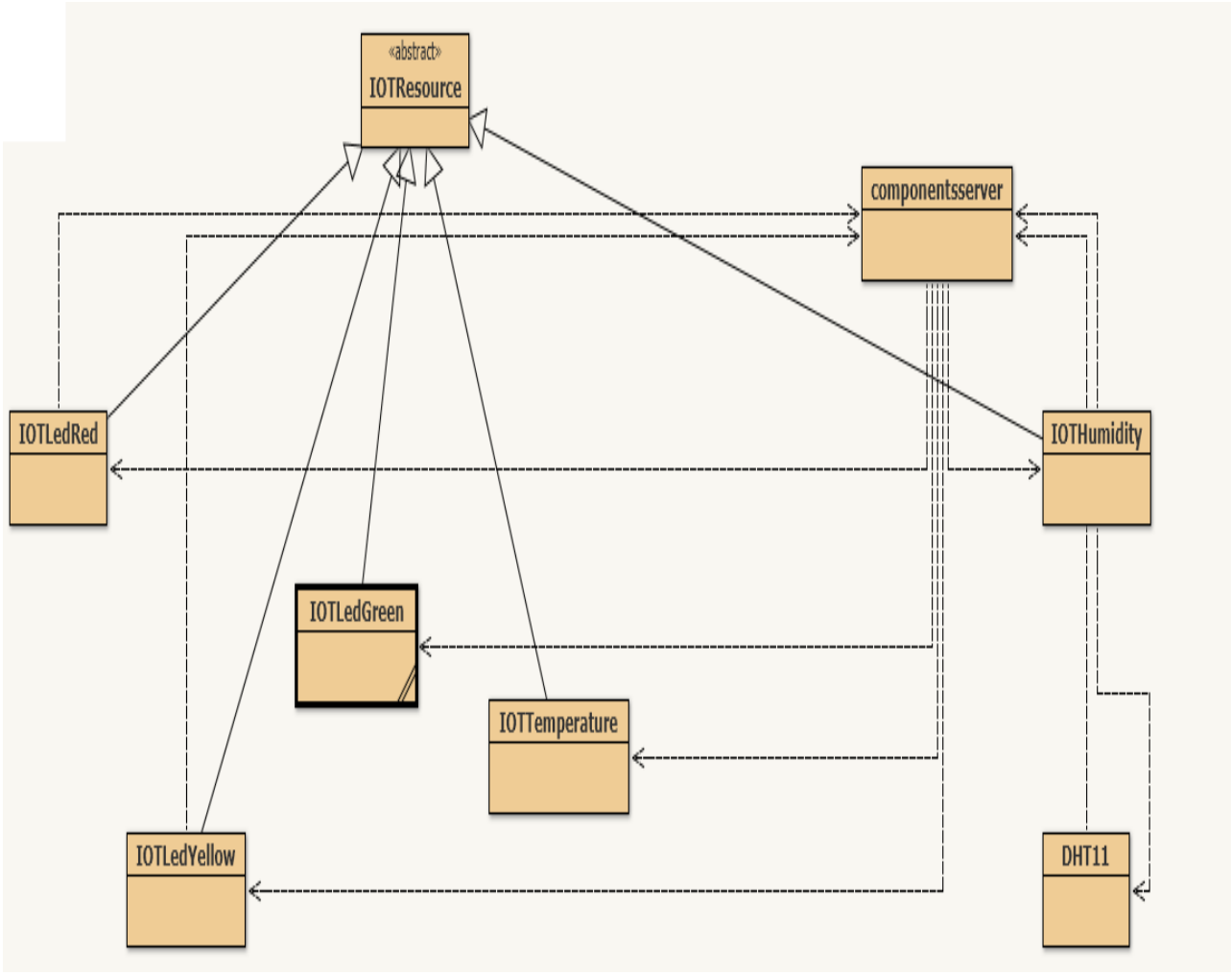


Figure 24: Diagramme de classes du Middleware

Deuxième partie :

La couche Web communication Interface côté annuaire, est implémentée par la Classe CoapServer similaire à son homologue du middleware. La couche native APIs Data BaseAccess :

Cette couche est implémentée par l'API MANGO de JAVA, cette interface est une implémentation du SGBD MONGO pour JAVA, qui offre un ensemble de classes permettant de communiquer avec la base de données de l'annuaire. L'Annuaire de ressources est stocké sur une base de données NOSQL.

Nous avons choisi ce type de base de donnée, pour stocker les descriptions des objets physiques, de ce fait que les structures de ces descriptions ne sont pas uniformes (n'ayant pas tousse la même structure), en plus le NOSQL offre la possibilité de représenter les données sous forme de collections (Base de données dans le modèle relationnel), et de documents(Tables dans le modèle relationnel) de format BSON, qui est très convenable aux descriptions des ressources, ce qui facilite le parcours de l'annuaire pour découvrir les ressources, en minimisant le temps de réponse aux requêtes, qui est considéré une exigence pour les environnements contraints.

Notre annuaire de ressource, est une base de données (Collection) MONGO, formée de documents contenant la description des ressources, ainsi leur modèle ontologique.

Pour le modèle ontologique, nous avons opté au modèle proposé par W3C, ce dernier a proposé un modèle ontologique standard, pour décrire les objets physiques dans le WOT, sous 03 formats : RDF, JSON LD et un graphe, en plus elle a proposé aussi, un modèle de mapping en RAML **(2)** qui n'est pas le sujet de notre travail.

Architecture de l'annuaire :

L'annuaire sert à stocker les descriptions des objets physiques, dans notre cas, c'est les deux capteurs avec les trois Leds. Les descriptions sont des documents JSON stockés dans l'annuaire. Parmi les éléments descriptifs des composants :

- **id:** Identifiant du composant de l'objet physique.
- **rt:** Type de l'objet physique.
- **if:** interface.
- **host:** le nom de la machine sur le quel est chargé le composant
- **port:** Le port du serveur embarqué

Et d'autres paramètres qui sont utiles pour décrire l'objet physique.

On peut trouver, aussi, le modèle ontologique.

Exemple d'une description JSON du capteur température :

```
{
    "comment": "sensed or actuated temperature",
```

(2)C'est un langage de description d'une API RESTful , il pemet la conception et la documentation d'une API REST.

```
"id":"IOTTemperature"
    "rt": "TEMPERATURE",
"host":"192.168.0.1",
"port":"5683",
    "types": {
        "scale": [
            "C",
            "F",
"K"
        ]
    },
    "properties": {
        "temperature": "number",
        "units": {
            "union": [
                {
                    "type": "scale",
                    "writeable": false
                },
            ],
            "optional": true
        },
        "range": {
                [
                 "min":"-55",
                 "max":"125"
            ],
            "type": "number",
}
    }
}
```

Afin de permettre aux clients de découvrir les ressources, nous avons utilisé la classe LOOKUP héritée de la classe CoapResource, et nous avons implémenté ses méthodes handleGET et handlePOST, cette classe joue le rôle d'interface pour la découverte.

Chaque application ou client désirant un objet, il doit communiquer avec cette interface à travers sa méthode ses méthodes.

Chapitre 5:

V. Validation:

Avant de commencer, nous devons définir les capteurs que nous allons utiliser dans notre montage.

V.1. Capteurs :

Nous avons choisi deux capteurs numériques :

a. Capteur de température D18B20 :

Est un capteur numérique -55°C à +125°C qui peut mesurer une plage de température de 55°C à +125°C

b. Capteur d'humidité dht11 :

Est un capteur numérique utilisé pour mesurer deux grandeurs physiques : la température et l'humidité **(voir Figure : 24)**, il est composé de 03 pins.

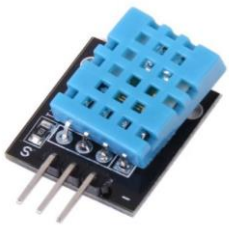


Figure 25: Capteur Humidité DHT 11

VCC : Alimentation

Data. Collecte les données sous forme de deux octets un pour la température et l'autre pour l'humidit**é.**

GND : Commun

V.2. Liste du matériel nécessaire :

Pour notre montage, nous aurons besoin du :

- Raspberry pi B 2
- Plaque d'essai pour monter les capteurs utilisés.
- Des Fils de connection male/femele pour le câblage.
- Deux capteurs numériques :
  - ✓ 01 Capteur de temperature D18B20
  - ✓ 01 Capteur d'humidité dh11

- 03 Leds de couleurs différentes
- Des résistances. **(voir Figure : 26)**

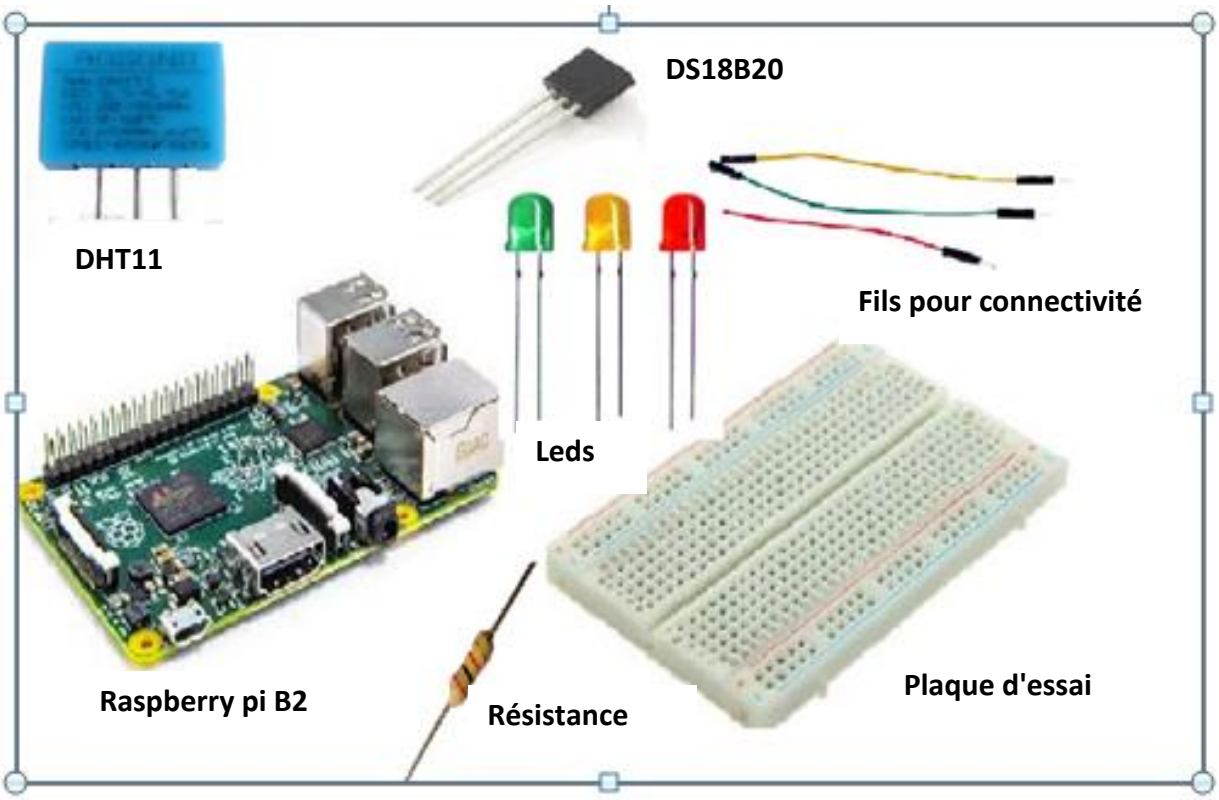


Figure 26 : Liste du matériel utilisé

Le **Raspberry Pi [31]**est un nano-ordinateur monocarte à processeur ARM conçu par le créateur de jeux vidéo David Braben, dans le cadre de sa fondation Raspberry Pi 2.

Il est un ordinateur, qui a la taille d'une carte de crédit, est destiné à encourager, il permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni nu (carte mère seule, sans boîtier, alimentation, clavier, souris ni écran) dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération. Ce Raspberry qui jouera le rôle de passerelle intelligent qui hébergera le serveur web.

De multiples versions ont été développées, mais nous allons décrire le Modèle B 2, qui est le modèle choisi pour implémenter notre plateforme **[1]**. **(Voir Figure : 26)**

Caractéristiques techniques : (Voir Figure : 27)

1. A 900MHz quad-core ARM Cortex-A7 CPU
2. 1GB de RAM.

3. 4 USB ports

4. 40 GPIO pins

5. Full HDMI port

6. Ethernet port

7.Camera interface (CSI)

8. Display interface (DSI)

9. Micro SD card slot

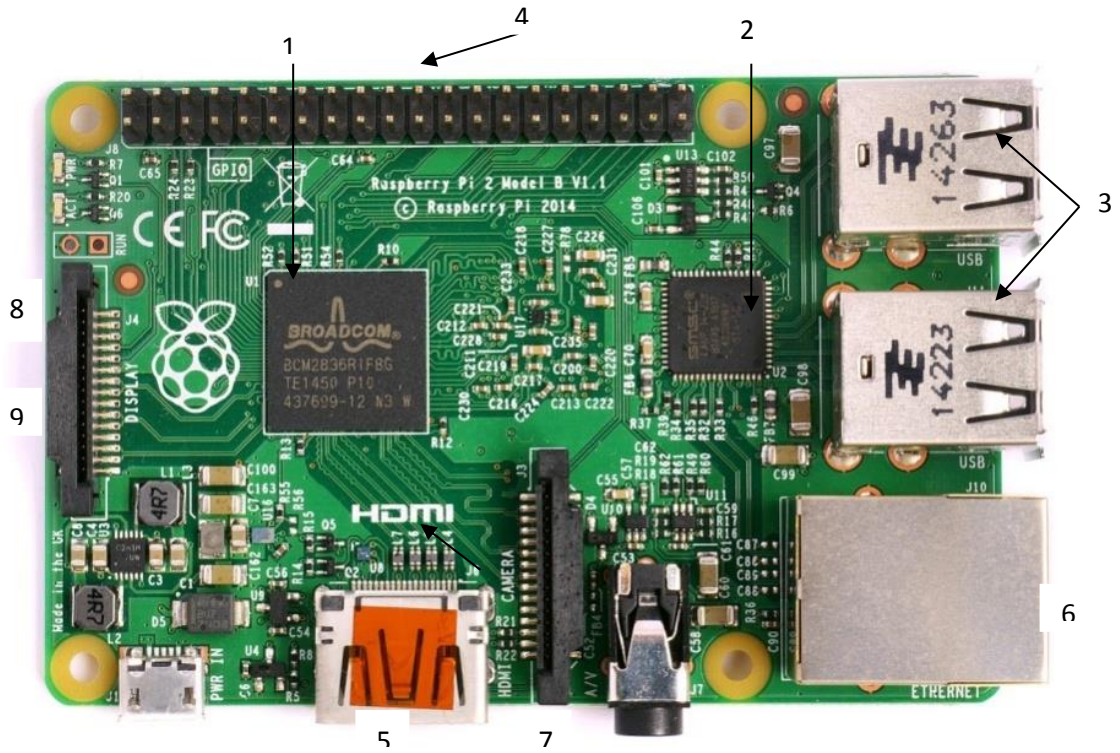


Figure 27 : La carte Raspberry Modèle B 2

Une caractéristique puissante de Raspberry Pi est la rangée de broches GPIO, ce sont des entrée / sortie à usage général le long du bord supérieur de la carte, elle permet de connecter des objets. Ils opèrent comme une interface physique entre le Pi et le monde extérieur.

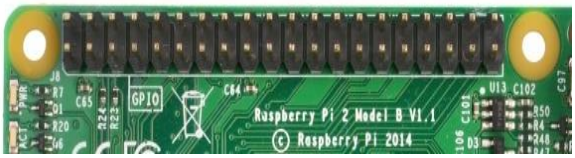


Figure 28 : GPIO Raspberry Pi 2 Modèle B

Le modèle B 2 utilisé dans notre travail, est formé de 40 broches, dont 26 sont des broches GPIO programmables (rattachables) aux objets physiques, tandis que les autres sont des broches d'alimentation ou de masse (plus deux broches ID EEPROM que vous ne devriez pas jouer à moins que vous ne connaissiez vos choses !) (**Voir Figure : 28**)

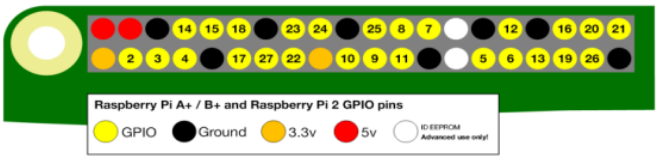


Figure 29 : Schéma de GPIO Raspberry Pi 2 Modèle B

Son principe de fonctionnement est simple : Ces broches ont deux états, On ou Off (Hight ou Low), elles sont utilisées comme des entrées ou sorties, selon le type d'objets physiques capteur ou actuateur, et selon leur mode de programmation, qui sera expliqué au moment du montage.

V.3. Montage :

Monter les 02 capteurs, les 03 Leds ainsi les résistances, sur la plaque d'essai et les relier au raspberry selon les schémas des figures **29,30**.

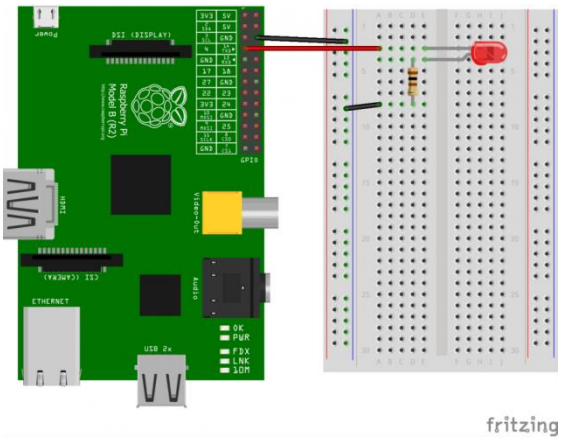


Figure 30 : Exemple du montage Led sur Raspberry

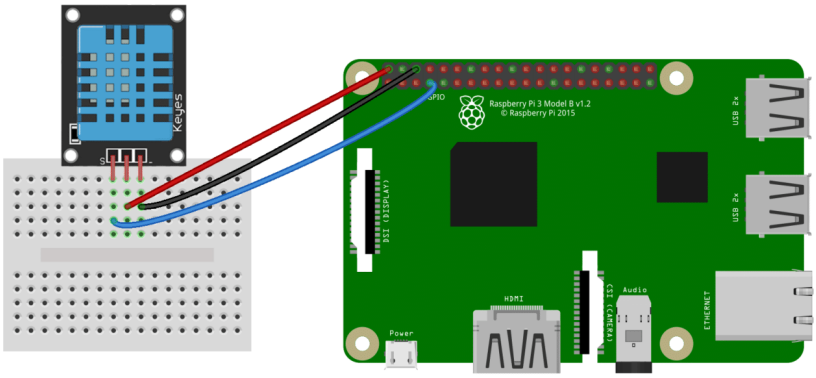


Figure 31 : Schéma du montage du capteur DHT11 sur Raspberry

Maintenant Le montage est fini et prêt à être utilisé, et ne reste qu'à charger les classes représentant ces ressources sur le serveur CoAP de Raspberry.

A fin de lancer le chargement des composants, à l'invite du serveur Coap sur raspberry, on tape la commande Load. Cette commande charge la classe et son instance sur le serveur des composants, ainsi il lance un processus d'enregistrement des descriptions sur le serveur d'annuaire. Le chargement des classes se fait à partir d'un fichier JAR [(3)]

V.4. Scénarios :

Dans le 1 er scénario, nous allons tester la première fonctionnalité de notre plateforme, le chargement dynamique des composants par le middleware.

Avant de commencer ce scénario, nous devons expliquer la structure des classes de composants à charger.

Pour implémenter les composants : nous avons définis pour chaque composant une classe héritée de la classe IOTResource, cette dernière est une classe abstraite héritée de la classe abstraite CoapResource du Package Coap de californium, (**voir diagramme de classes des composants Figure V.13).** La classe IOTResource est une classe abstraite que nous avons proposé pour implémenter les classes de composants, cette classe abstraite hérite des méthodes abstraites de la classe CoapResource :handleGET, handlePOST, handlePUT et handleDELETE, et dans chaque classe des composants, nous avons redéfini ses méthodes abstraites handle, en précédant chaque méthode handle par l'annotation @Override. Les classes des composants utilisées, sont :

IOTTemperature : Pour le capteur de température

IOTHumidity : Pour le capteur d'humidity.

IOTLedRed: Pour la led rouge.

---

(3) JAR(Java Runtime): est un fichier ZIP utilisé pour distribuer un ensemble de classes Java.

IOTLedYellow : Pour la Led jaune.

IOTLedGreen : Pour la Led verte.

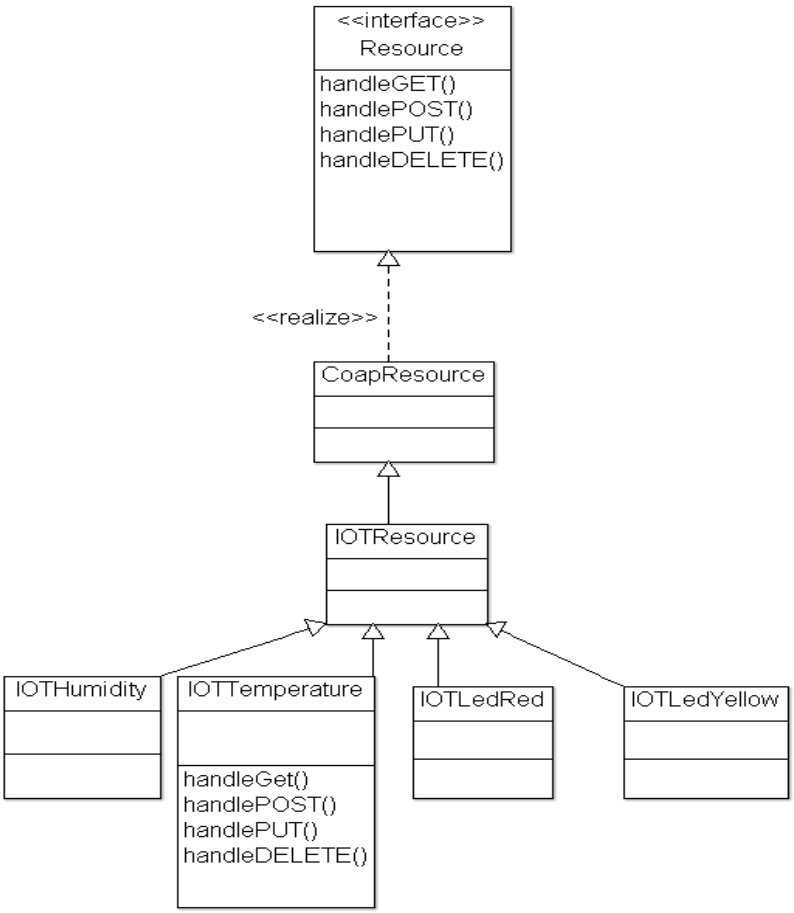


Figure 32 : Diagramme de classes de composants

Les détails de chacune de ces classes :

Classe**IOTTemperature :**
Cette Classe implémente uniquement la méthode handleGET

```
package PackageIOT;

publicclassIOTTemperatureextends IOTResource{
public IOTTemperature() {//*Constructeur de la classe
this("Temperature");
}
public IOTTemperature(String name) {//*Constructeur de la classe

super(name);
}
@Override
publicvoid handleGET(CoapExchange exchange){
```

```
}
}
```

Classe **IOTHumidity:**
Cette Classe implémente uniquement la méthode handleGET

```
package PackageIOT;

publicclassIOTHumidityextends IOTResource{
publicIOTHumidity () {
this("IOTHumidity ");
}
public IOT IOTHumidity (String name) {
super(name);
}
@Override
publicvoid handleGET(CoapExchange exchange){

}
}
```

Classe **IOTLedYellow:**
Cette Classe implémente uniquement la méthode handlePOST

```
package PackageIOT;

publicclassIOTLedYellowextends IOTResource{

public IOTLedYellow () {  /*Constructeur de la classe
this("LedYellow");
}
public IOTLedYellow (String name) { /*Constructeur de la classe
super(name);
}
@Override
publicvoid POST(CoapExchange exchange){
GpioController gpio1=GpioFactory.getInstance();
GpioPinDigitalOutput pin1=gpio1.provisionDigitalOutputPin(RaspPin.GPIO_24)
if (exchange.getRequestText().equals("ON"))
   {pin1.high();
    System.out.println("LED YELLOW is ON");
}else
     { pin1.low();
      System.out.println("LED YELLOW is OFF");
     }
}
}
```

Classe **IOTLedRed**:
Cette Classe implémente uniquement la méthode handlePOST

```
package PackageIOT;

publicclassIOTLedRedextends IOTResource{
public IOTLedYellow (String name) {/*Constructeur de la classe
super(name);
}
@Override
publicvoid POST(CoapExchange exchange){
```

```
GpioController gpio2=GpioFactory.getInstance();
GpioPinDigitalOutput pin2=gpio2.provisionDigitalOutputPin(RaspPin.GPIO_07)
if (exchange.getRequestText().equals("ON"))
   {pin2.high();
    System.out.println("LED RED is ON");
}else
     { pin2.low();
      System.out.println("LED RED is OFF");
     }
}
}
```

Ces classes sont développées séparément de middleware. Nous supposons que le middleware est en cours d'exécution, et que ces capteurs sont ajoutés à l'environnement, et qu'on doit leur charger les composants qui les manipulent.

Ensuite ces classes seront compilées par le compilateur java, les résultats de la compilation, sont des bytecodes précompilés, prêts à être chargés par le middleware de la plateforme.

Pour tester et évaluer le chargement dynamique de ces classes de composants, représentant les objets physiques, on devra d'abord, démarrer la plateforme, puis charger à chaud les composants un par un, ce chargement se fait sans arrêter le fonctionnement de la plateforme, qui est une fonctionnalité objectif à évaluer par le scénario.

Pour se faire, nous devons démarrer la plateforme ainsi :

Etape 1 :

Sur le raspberry , démarrer le serveur de composants offrant une interface web CoAP qui, est la première couche du middleware. Ensuite on démarre le serveur par la commande start vue précédemment, le serveur démarre sur le port : 5683 le port choisi **(voir Figure:33)**

```
Mar 05, 2017 1:59:36 AM
org.eclipse.californium.core.network.config.NetworkConfig
createStandardWithFile
INFO: Create standard properties with file Californium.properties
Mar 05, 2017 1:59:36 AM org.eclipse.californium.core.CoapServer
start
INFO: Starting server
Mar 05, 2017 1:59:36 AM
org.eclipse.californium.core.network.CoAPEndpoint start
INFO: Staring Endpoint bound to 0.0.0.0/0.0.0.0:5683
```

Figure 33 : Démarrage du serveur du composant du middleware

Etape 2 :

Sur un pc dédié pour héberger l'annuaire , on démarrer le serveur de l'annuaire CoAP qui est la première couche de la 2eme partie de la plateforme **(voir Figure : 34)** . Le serveur démarre sur le port 5680.

```
mai 21, 2017 11:57:47 AM
org.eclipse.californium.core.network.config.NetworkConfig
createStandardWithFile
INFOS: Create standard properties with file Californium.properties
mai 21, 2017 11:57:47 AM org.eclipse.californium.core.CoapServer start
INFOS: Starting server
mai 21, 2017 11:57:47 AM org.eclipse.californium.core.network.CoAPEndpoint
start
INFOS: Starting Endpoint bound to 0.0.0.0/0.0.0.0:5680
```

Figure 34 : Démarrage du serveur de l'annuaire

Etape 3 :

Sur le pc, nous démarrons le serveur MONGODB .A la ligne de commande du système du pc , on tape la commande mongod –dbpath, suivi du répertoire de la base de données dans notre cas c'est mongodb\ mongodb\db, donc la commande complète c'est mongod –dbpath c: \mongodb\mongodb\db

Maintenant le serveur MONGODB est démarré , il est en écoute sur le port 27017, le port par défaut. **(**Voir Figure **: 35)**

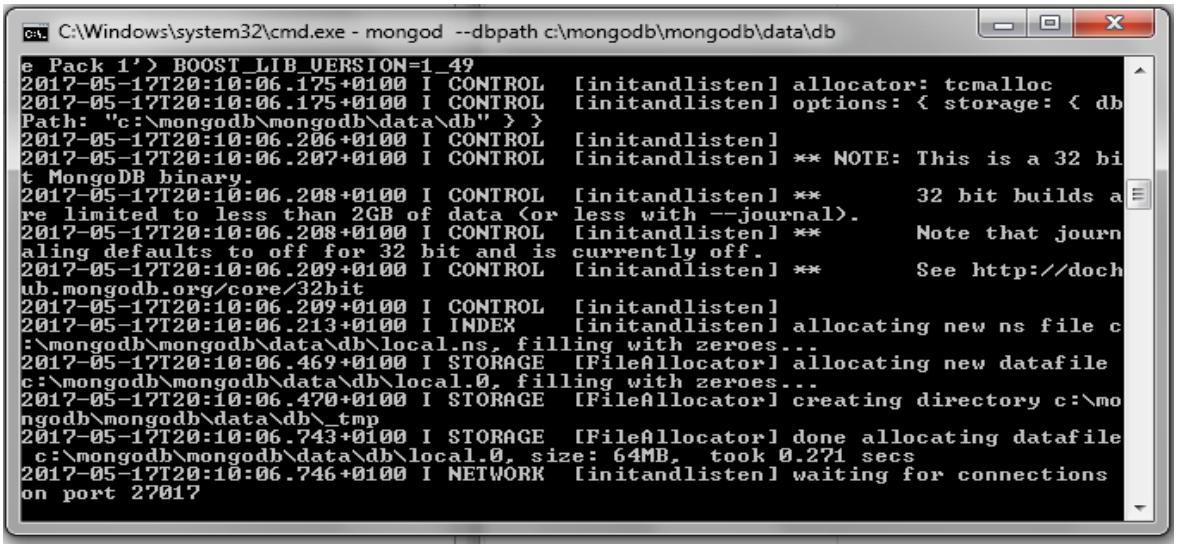


Figure 35 : Démarrage du serveur MongoDB

Les trois serveurs sont démarrés et prêts, nous pouvons commencer notre validation pour évaluer notre plateforme.

Nous commençons notre test du chargement dynamiquement

Charger les différentes classes précompilées, par le module Loader du middleware, pour ce faire, à l'invite du commande du Gateway Raspberry on tape : Load suivi du nom de la classe du composant et sa description.

Syntaxe : Load nom_classe, Description

Exemple : Load IOTHumidity, TEMPERATURE

La description contient uniquement, le type de l'objet physique. Pour cela nous avons proposés la nomenclature suivante pour les d'objets physiques utilisés dans notre validation :

Objet Physique	**Nom du composant**	**Type**
Capteur DHT11	IOTHumidity	TEMPERATURE
Led Rouge	IOTLedRed	LEDRED
Led Jaune	IOTLedYellow	LEDYELLOW

Tableau Nomenclature des objets physiques

```
mai 22, 2017 10:30:36 AM org.eclipse.californium.core.CoapServer start
INFOS: Starting server
mai 22, 2017 10:30:36 AM org.eclipse.californium.core.CoapServer start
INFOS: No endpoints have been defined for server, setting up default
endpoint at port 5683
mai 22, 2017 10:30:36 AM org.eclipse.californium.core.network.CoAPEndpoint
start
INFOS: Starting Endpoint bound to 0.0.0.0/0.0.0.0:5683
Load IOTTemperature
```

Figure 36 :Lancement du chargement des composants par le module Loader

Un message s'affiche indiquant que la classe est chargée avec succès par le middleware

```
mai 22, 2017 10:30:36 AM org.eclipse.californium.core.CoapServer start
INFOS: Starting server
mai 22, 2017 10:30:36 AM org.eclipse.californium.core.CoapServer start
INFOS: No endpoints have been defined for server, setting up default
endpoint at port 5683
mai 22, 2017 10:30:36 AM org.eclipse.californium.core.network.CoAPEndpoint
start
INFOS: Starting Endpoint bound to 0.0.0.0/0.0.0.0:5683
Load IOTTemperature

Le composant IOTTemperature est chargé avec succès par le middleware
```

Figure 37 : Résultat du chargement des composants par le module Loader

Scénario 2:

Dans le scénario précédent, nous avons testé la 1ere fonctionnalité de notre plateforme, le chargement dynamique, maintenant et dans ce scénario, nous allons évaluer la deuxième fonctionnalité offerte par le middleware, qui est la communication avec les objets physiques, à travers les composants chargés par le middleware, sans avoir recours aux API natives de ces objets physiques. Ces composants offrent une interface API REST masquant les détails de la fonctionnalité des objets physiques.

Ce scénario sera validé à travers deux types de clients : un client Desktop et un client navigateur.

Les objets physiques choisis : le capteur de température et deux Leds, une Led rouge et une Led jaune.

Le scénario proposé est le suivant : Un client lit la température ambiante, et si la valeur mesurée atteint un seuil critique de 40°, la Led rouge s'allume, sinon la Led jaune qui va s'allumer.

Déroulement du scénario :

Tout d'abord le client doit chercher les descriptions des composants représentant le capteur de température, Led rouge et la Led Jaune, ces descriptions concernent l'interface API Rest des composants IOTHumidity, IOTLedRed et IOTLedYellow. Ensuite le client commence à communiquer avec les objets physiques à travers l'interface API des composants chargés dynamiquement par le middleware, comme on a vu dans le scénario précédent.

Le scénario se déroule comme suit :

1 èrement avec un client Desktop :

Phase 1: Phase de découverte:

Côté client:

C'est une classe ClientTemperature

```
publicclass ClientTemperature {
publicstaticvoid main(String[] args) {
CoapClientcoapclient=newCoapClient("coap://127.0.0.1:5680/LOOKUP?rt='TEMPERATURE'");//
Request request=new Request(Code.GET);//
coapclient.advanced(request);//
}

}
```

Description

1. Création d'un objet coapclient de la classe CoapClient qui va communiquer avec l'interface LOOKUP du serveur annuaire dont 127.0.0.1:5680 signifie que le serveur est sur la hote locale sur le port 5680.

Création d'une requête GET par l'objet Request de la classe Request

Appel de la méthode Advanced de l'objet coapclient ayant comme paramètre la requête request.

2. Ce code permet d'envoyer une requête coap vers l'interface API LOOKUP résidée sur le serveur annuaire.

Côté serveur :

3. La couche coap server (Web communication interface) Côté annuaire, intercepte le message Request CoAP envoyé par le client.
4. Ce message sera analysé par le dispatcher de cette couche pour déterminer l'interface API et sa méthode à invoquer. Ensuite le dispatcher génère un appel vers la méthode handleGET de l'interface LOOKUP. Cette méthode handleGET implémente Directory Manager dont le rôle est d'exploiter l'annuaire afin de chercher la description du composant IOTHumidity représentant le capteur de température.

Dans notre cas, la description trouvée concerne uniquement un seul composant IOTHumidity, dans le cas général, ce sont des descriptions concernant un ensemble de composants.

5. La description est renvoyée au Directory Manager pour la reformuler en Link Format.
6. Ensuite cette réponse Link Format est envoyée à la couche CoapServer.
7. La description Link Format sera transmise par le Responder sous forme d'une réponse CoAP **(voir Figure : 38).**

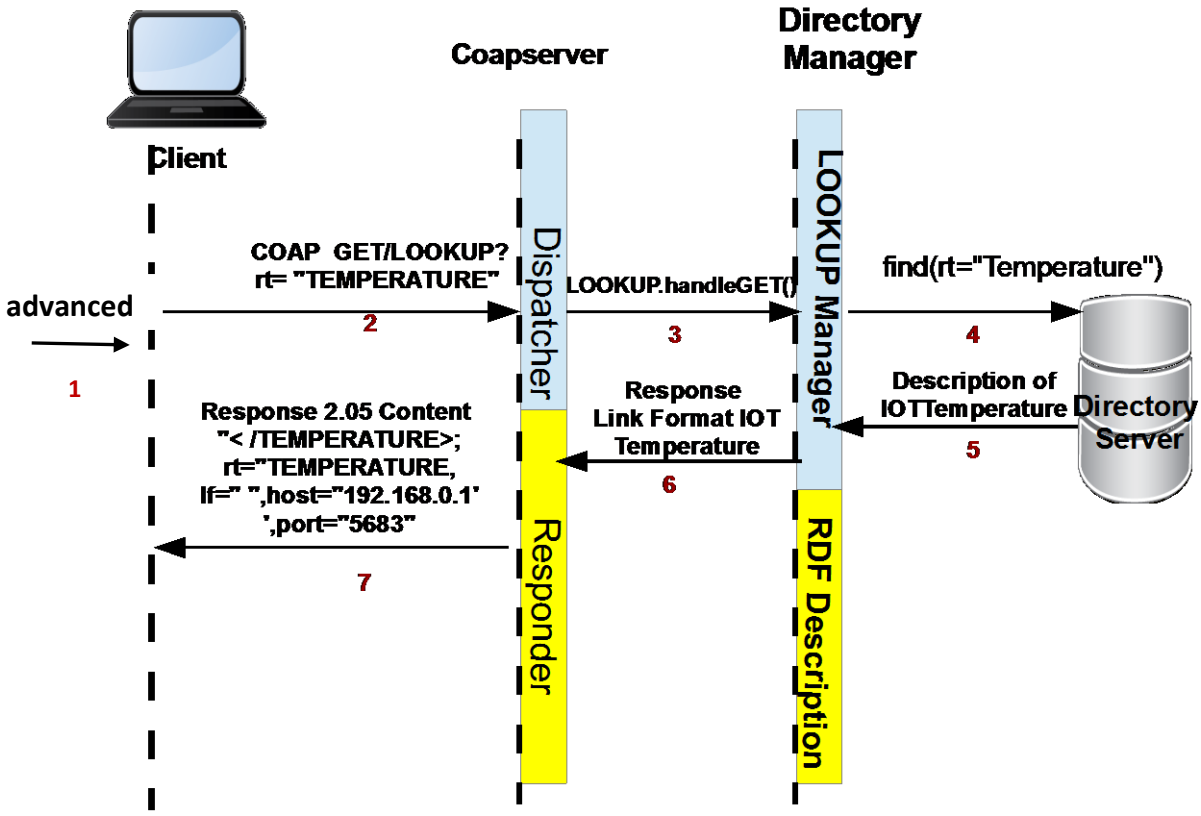


Figure 38 : Digramme de séquence de découverte du scénario du capteur Température

Maintenant, et de la même manière que la découverte du composant IOTHumidity, nous découvrons le composantIOTLedRed et le composantIOTLedYellow, en changeant uniquement la valeur de l'attribut rt de la requête par "IOTLedRed","IOTLedYellow» ; A titre illustratif, nous expliquons que le principe de découverte de la led rouge, pour la led jaune se fait de la même manière que celle de led rouge. **(Voir Figure : 39).**

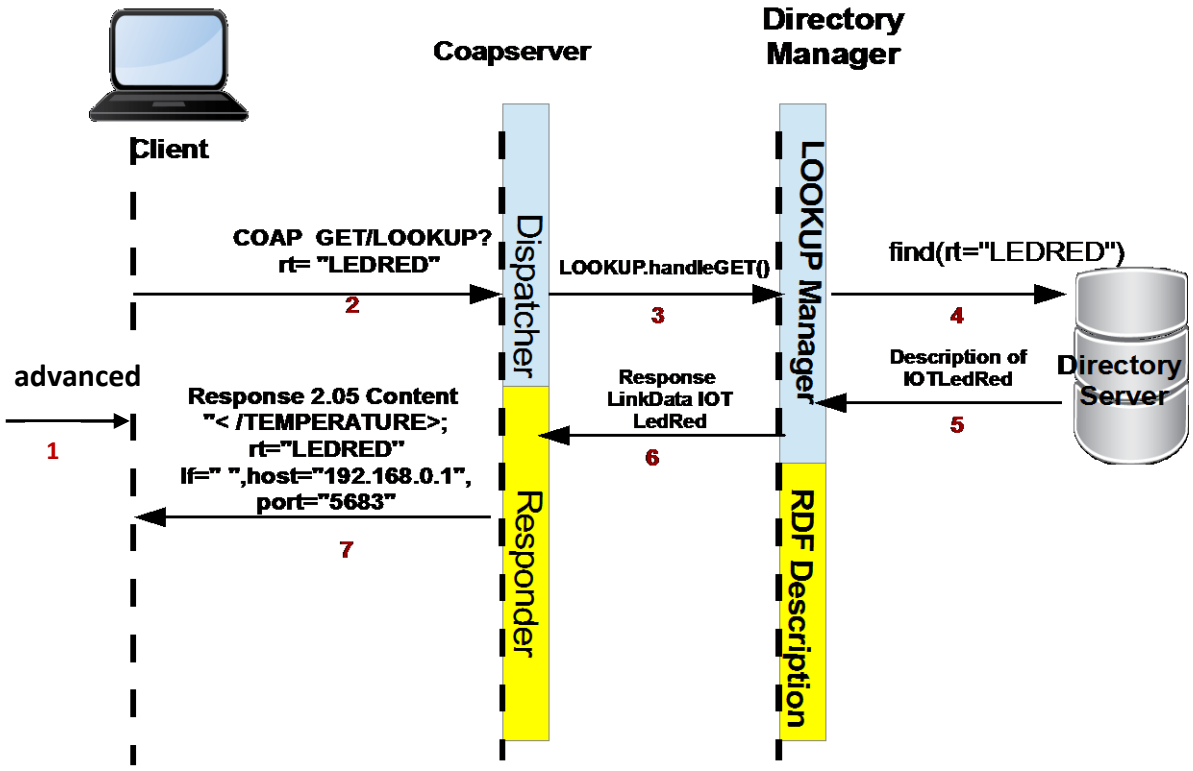


Figure 39 : Digramme de séquence de découverte du scénario de Led rouge

Phase 2: Phase de communication avec les objets physiques

Pour expliquer le principe de ce scénario, on revient sur le code du client du 1er scénario, dans le code nous ajoutons la partie qui implémente notre scénario, le code sera comme suit:

```
publicclass ClientTemperature {
publicstaticvoid main(String[] args) {
CoapClient
coapclient=newCoapClient("coap://127.0.0.1:5680/LOOKUP?rt='TEMPERAT
URE'");
Request request=new Request(Code.GET);//
coapclient.advanced(request);//
/**Partie 2 pour communiquer avec IOTHumidity
CoapClient
coapclient=newCoapClient("coap://192.168.0.1:5683/IOTHumidity);
Request request=new Request(Code.GET);//
coapclient.advanced(request);//(1)
/**Partie 3 pour communiquer avec IOTLedRed
CoapClient
coapclient=newCoapClient("coap://192.168.0.1:5680/IOTLedRed);
Request request=new Request(Code.GET);//
coapclient.advanced(request);// (10)(voir Figure: 40)

}

}
```

Description :

1 La partie 2 du code permet la création d'un objet coapclient de la classe CoapClient, qui va communiquer avec l'interface IOTHumidity, ce client envoi une requête Coap vers le composant IOTTHumidity sur le middleware en invoquant sa méthode GET : CoAP GET 192.168.1.10 :5683/IOTHumidity, où GET le type de la requête, 192.168.1.10 nom du host, 5683 le port du serveur CoAP et IOTHumidity est l'interface API Rest du composant, représentant le capteur Température.

En suite, nous créons une requête GET par l'objet request de la classe Request

Puis nous avons invoqué la méthode advanced de l'objet coapclient ayant comme paramètre la requête Request comme on a vu dans le scénario précédent.

2 Le dispatcher du middleware intercepte et analyse la requête CoAP provenant du client.

3 Le dispatcher fait un appel à la méthode handleGET du composant IOTTemperature

4 Le composant IOTHumidity, à son tour transmet l'appel vers l'API Dédiée au matériel, cetteAPI s'occupera de la communication avec l'objet physique.

8 En réponse ; L'API renvoie la valeur détectée par le capteur vers le composant IOTHumidity, cette valeur sera transmise vers le Responder de la couche coapserver du middleware.

9 Le responder, transforme cette valeur de température en format JSON puis l'encapsuler sur une réponse CoAP.

Pour mieux comprendre le fonctionnement des composants du middleware, nous allons détailler le code du composant IOTLedRed prochainement.

Passons maintenant à la troisième partie du code coapclient

11 Cette fois ci le client génère une requête CoAP de type POST ayant comme Payload (champs de données) l'état ON de Led sous format JSON.

12 Le dispatcher analyse la requête POST et parse son Payload, puis il invoque la méthode handlePOST de composant, du fait que l'interface API de Led implémente uniquement la méthode handlePOST, qui se charge de modifier l'état de l'objet physique.

13 La couche la plus basse de notre middleware, natives API, transforme le message provenant du composant IOTLedYellowen commandes sur le GPIO.

Prenons la classe du composant IOTLedYellow, pour expliquer les principes de l'interaction avec les objets physiques ,à travers les composants du middleware

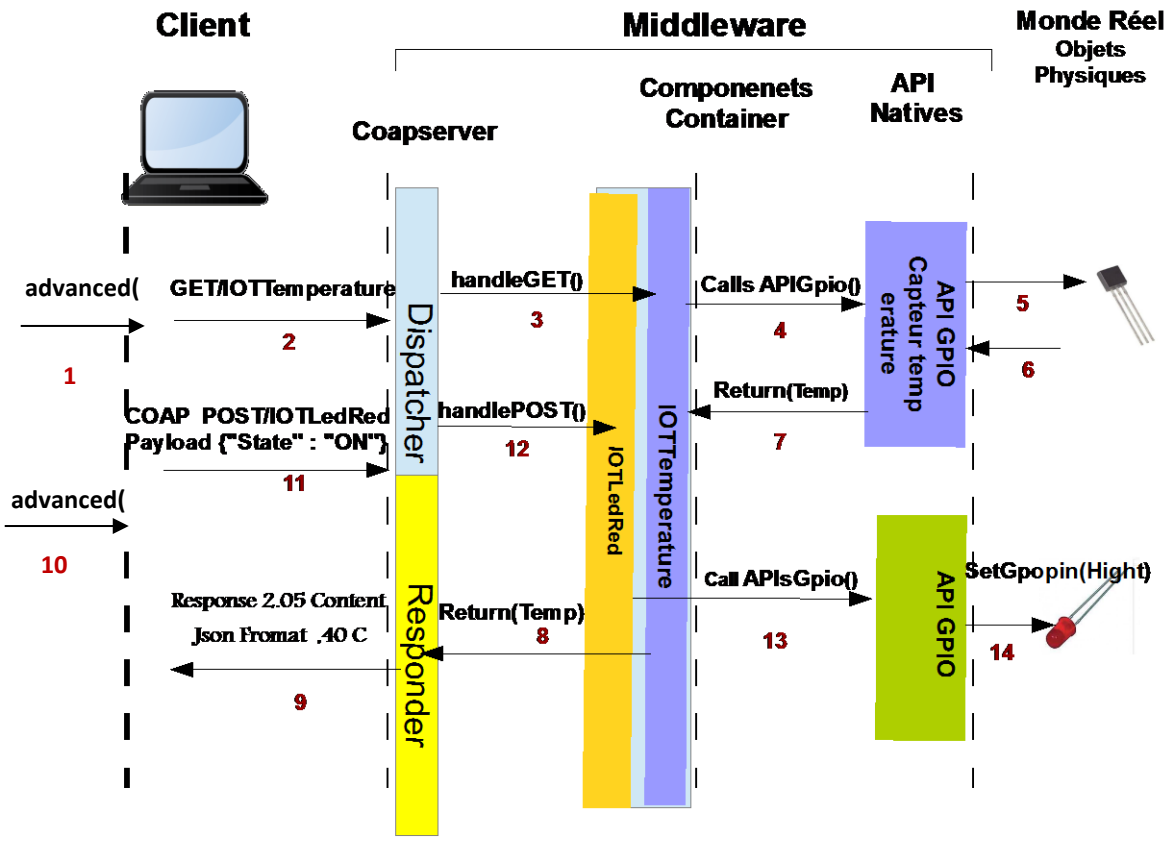


Figure 40 : Digramme de séquence de communication avec les objets physiques

Scénario 2

```
package PackageIOT;
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;

publicclass IOTLedYelllowextends IOTResource{

        public LedRed(Stringname) {
                super(name);
                // TODO Auto-generated constructor stub
        }
@Override//* publicvoid handlePOST(CoapExchange exchange){
final GpioController gpio = GpioFactory.getInstance();/* 1

        final GpioPinDigitalOutput

       pin=gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01);/*

pin.low();

}
}
```

Ce code Permet d'éteindre une led connectée sur le GPIO numéro 1 du Raspbbery la ligne 1 permet de créer un objet GpioController qui permet de manipuler les GPIO du raspberry.

La méthode provisionDigitalOutputPin permet de créer un objet pin (broche) du gpio et le configure comme une sortie, dans notre cas c'est la broche numéro 1

Pin.low permet d'éteindre l'objet physique connecté sur le pin 01.

Validation avec un navigateur WEB:

Nous allons reprendre le même scénario, mais cette fois ci à travers un navigateur web. Pour cela nous allons utiliser le client ***Copper CU*** , proposé par Californium, c'est un agent ajouté au navigateur Firefox sous forme de module plugin. Son principe de fonctionnement sera expliqué au moment du déroulement du scénario.

COPPER CU offre une interface graphique **(voir Figure :41)** formée de :

1. Barre d'url
2. Ressources (Interfaces) offertes par le serveur
3. Zone de Payload contient la réponse d'une requête GET, ou valeurs d'arguments à envoyer avec une requête POST
4. Barre des méthodes à invoquer

Dans la barre d'url, on tape coap://127.0.0.1:5680/LOOKUP?rt="TEMPERATURE", où 127.0.0.0.1 :5680 indique le HOST : port du serveur de l'annuaire, LOOKUP ? Rt="TEMPERATURE" indique l'Uri, Puis on choisit le bouton GET de la barre de méthodes. La liste des composants répondant à l'URI LOOKUP?rt="TEMPERATURE", sera envoyée par le serveur de l'annuaire. Voir la zone de **Payload** de la figure**41**.

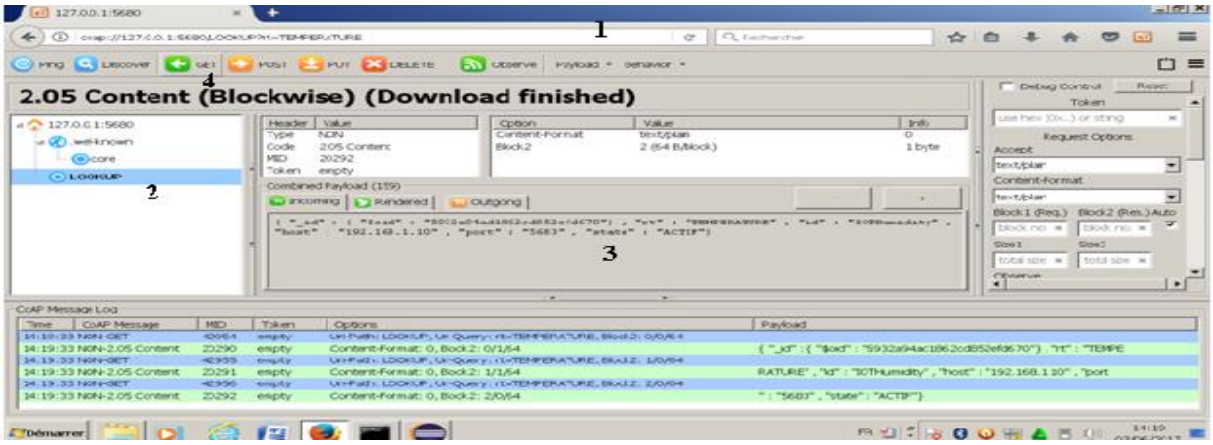


Figure 41: Requête CoAP pour la découverte du composant TEMPERATURE

Nous exécutons une nouvelle instance de Copper CU sur Firefox, avec l'URL :

coap : //192.168.1.10 :5683/IOTHumidity, ou 192.168.1.10:5683 est le HOST : port du serveur de composants, et IOTHumidity est l'URI du composant du capteur de température.

Ensuite on choisit la méthode GET. Dans la zone de Payload, on reçoit la valeur de température envoyée par le composant IOTHumidity **(voir Figure : 42).**

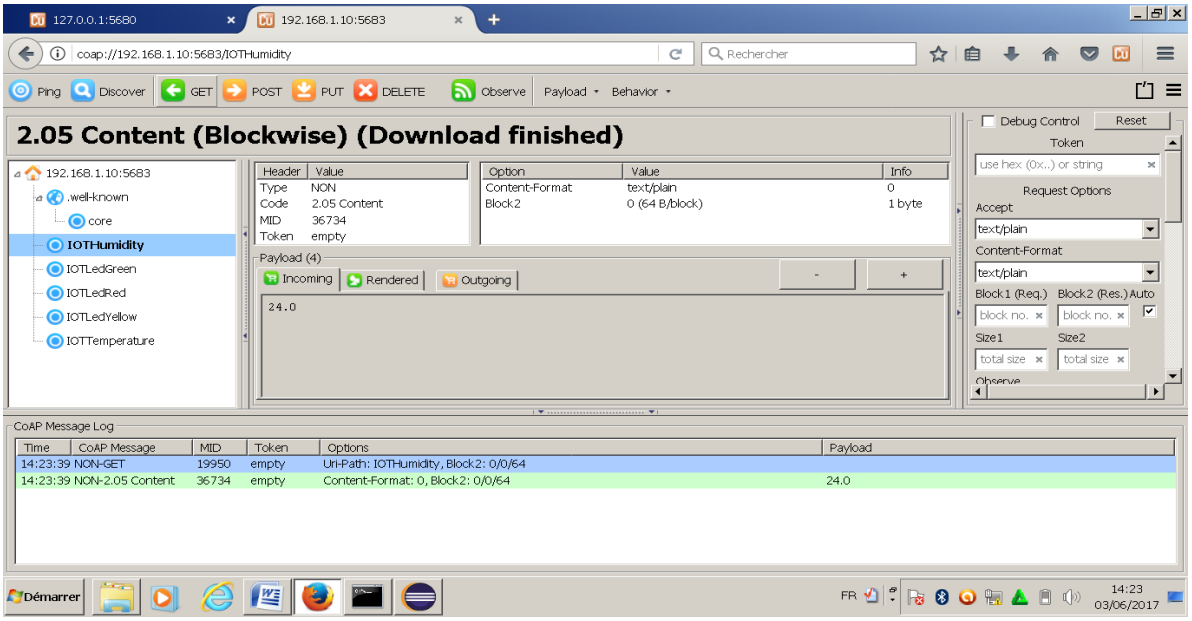


Figure 42:Requête CoAP pour récupérer la valeur de TEMPERATURE

A la fin nous allons envoyer une requête POST vers le composant IOTLedYellow, avec la valeur ON dans le Payload **(Voir Figure : 43).**

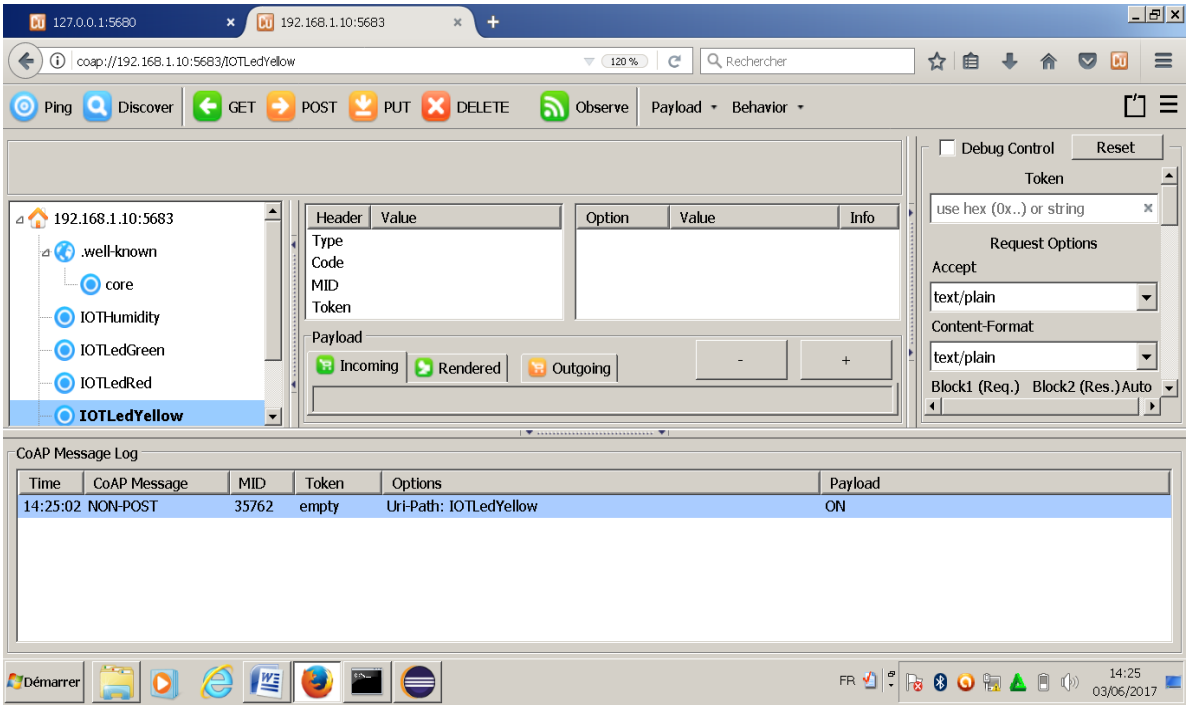


Figure 43: Requête CoAP pour allumer la led jaune.

V.5. Conclusion :

Dans ce chapitre, nous avons mis l'accent sur les travaux antérieurs sur l'intégration des objets physiques.

Puis nous avons expliqué l'architecture de notre plateforme pour l'intégration des objets physiques au web. Cette architecture est un middleware qui offre une abstraction pour les objets physiques à travers des composants, ces derniers sont chargés et déchargés dynamiquement.

Ensuite, nous avons expliqué les outils et la manière de l'implémentation de cette plateforme, notamment le Framework californium qui nous a offert un ensemble de classes ayant facilité notre développement et le chargement dynamique de JAVA qui était une solution puissante pour le chargement dynamique de composants.

A la fin de ce chapitre, nous avons évalué notre plateforme par des scénarios, et on a vu comment notre plateforme est capable de charger à chaud les composants représentant les objets physiques sans altérer le fonctionnement du middleware.

Notre middleware offre une interface uniforme REST permettant aux clients et applications d'interagir avec les objets physiques du monde réel notamment ceux n'ayant pas d'interface pour se connecter à l'internet. Notre middleware est vu comme une boite noire masquant toute le processus de l'interaction avec les objets physiques.

Conclusion générale :

Les objets physiques sont de nature hétérogène ce qui mène à introduire le WoT. L'intégration des objets dans le Web permet aux objets hétérogènes de fournir une interface de communication Web standard généralement REST selon le protocole CoAP. Pour intégrer tous les objets physiques dans le WoT, il est nécessaire d'avoir un serveur web embarqué dans chaque objet ce qui n'est pas possible pour tous les objets existants. Cela mène à introduire des Gateway pour l'intégration des objets dans le WOT.

Notre travail sert à proposer un Gateway WoT pour l'intégration des objets physiques sous forme de middleware. Cela permet d'intégrer ces objets dans le web en se basant sur les technologies disponibles, tel que l'architecture REST, et le protocole CoAP. Chaque objet physique sera représenté par une ressource virtuelle web accessible par un URI et manipulable à travers son interface web de type REST (API Web). Le middleware est un conteneur de composants Java où chaque composant représente un objet physique et permet l'interaction avec cet objet selon une interface standard. Après chargement des composants dans le middleware, les objets deviendront accessibles à travers les API Web REST selon le protocole CoAP. Notre proposition a été validée par quelques scenarios, montrant comment installer des objets physiques au Raspberry ,et créer et charger dynamiquement les composants java qui les représentent, et comment interagir avec ces objets à travers les API REST CoAP.

Table des figures

Liste des abréviations

API	Application Programming Interface
CoAP	Constrained Application Protocol
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet Of Thing
JAR	Java archive
JRE	*Java Runtime Environment*
JSON	JavaScript Object Notation
M2M	Machine To Machine
MQTT	MQ Telemetry Transport
NFC	Near Fields Communication
OSI	Open Systems Interconnection
RAML	*API Modeling Language*
RD	Resources Directory
RDF	Resource Description Framework
REST	Representational State Transfer
RFID	Radio Fréquence Identification
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WOT	Web Of Things
XML	Extensible Markup Language

Références

[1]Numa de Montmollin, Le Web des Objets, de la théorie à la pratique Implémentation de plusieurs cas d'utilisation, utilisation et création d'outils pour développer des applications RESTful du Web des Objets, Groupe Génie Logiciel Département *d'Informatique*, Université de Fribourg (Suisse), Août 2014.

[2] Dominique Guinard, A Web of Things Application Architecture -Integrating the Real-World into the Web, These, 2011.

[3]Amélie GYRARD, Concevoir des applications Internet des Objets Sémantiques inter-Domaine, T H È S E, TELECOM ParisTech, école de l'Institut Télécom - membre de ParisTech, 24 Avril 2015.

[4] David Boswarthick,Omar Elloumi,Olivier Hersent, M2M Communications : a Systems Approach »

[5] David R. GNIMPIEBA ZANFACK, Modélisation des flux logistiques Vers une Plateforme d'interopérabilité des objets logistiques, L'Université de Picardie Jules Verne Soutenue le 15 Mai 2017.

[6]Benjamin Billet, Système de gestion de flux pour l'Internet des objets intelligents, Université de Versailles-Saint Quentin en Yvelines, 2015.

[7] Sylvain Cherrier, Architecture et protocoles applicatifs pour la chorégraphie de services dans l'Internet des objets, Thèse de doctorat, université Paris Est, soutenue le 25 Novembre, 2013.

[8] VIPRET Julien, Rapport d'IRL, Utilisation du protocole CoAP pour la découverte de ressources dans les réseaux de capteurs, Grenoble INP Ensimag, Mai 2012.

[9] Pierre-Jean BENGHOZI, Sylvain BUREAU, Françoise MASSIT-FOLLEA, "L'internet des objets : Quelles enjeux pour l'Europe", Éditions de la Maison des sciences de l'homme, 2009.

[10] Anthony J. Massa, Embedded Software Development with Ecos, Prentice Hall Professional, 2003.

[11] Dominique D. Guinard et Vlad M. TRIFA, Building the Web of Things, with exemples in node JS and RASPBERRYPI**,** by Manning Publications Co. All rights reserved, USA, 2016 .

[12] Roy Want, Kenneth P Fishkin, Anuj Gujar, and Beverly L Harrison. Bridging physical and virtual worlds with electronic tags. In Proc. of the SIGCHI conference on Human

factors in computing systems (CHI '99), pages 370{377, Pittsburgh, Pennsylvania, United States, 1999. ACM.

[13] P.Schramm, E.Naroska, P.Resch, J.Platte, H.Linde, G.Stromberg, and T. Sturm. A service gateway for networked sensor systems. IEEE Pervasive Computing, 3(1):66{74, March 2004.

[14] Dominique Guinard, Architecting a Mashable Open World Wide Web of Things, Institute for Pervasive Computing, ETH Zurich, Switzerland.

[15] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "The Web of Things: interconnecting devices with high usability and performance," in 6th International Conference on Embedded Software and Systems (ICESS'09), Hangzhou, Zhejiang, China, May 2009.

[16] V. Stirbu, «Towards a RESTful Plug and Plug Experience in the Web of Things», in: IEEE. International Conference on Semantic Computing, Aug. 2008, pp. 512 – 517.

[**17**] IEEE Wireless Communications magazine, Feb. **2002**, pp. 30-38. HPL Tech.

[18] Yazar, Dogan; Dunkels, Adam (2009). "Efficient application integration in IP-based Sensor networks": 43. doi:10.1145/1810279.1810289

[19] Hui, Jonathan W.; Culler, David E. (2008). "IP is dead, long live IP for wireless Sensor networks": 15. Doi:10.1145/1460412.1460415.

[20]Guinard, Dominique; Trifa, Vlad; Wilde, Erik (2010). "A resource oriented architecture for the Web of Things": 1–8

[21]https://evrythng.com/, date consultation, mai 2017.

[22] Dominique D. Guinard, A Web of Things. Application Architecture- Intergrating the Real – World into the Web, Thèse, ETH., Zurich, 2011.

[23] Vidhya Gholkar, An introduction to IoT protocols,

[24] Stefan Jucker, Securing the Constrained Application Protocolby, autumn 2012

[25] Ph. Truillet MQTT : MQ Telemtry/Transport Un protocole pour l'IoT, Novembre 2016.

[26] C. Bormann, A. P. Castellani ET Z. Shelby, « CoAP: An Application Protocol for Billions of Tiny Internet 113 Nodes », IEEE Internet Computing, vol. 16, n° 2, 1er mars 2012, p. 62–67.

[**27**] **Z. Shelby,** "Embedded web services. IEEE Wireless Communications", 17(6), December 2010.

[28] Yoann Ricordel, João Prado, Accès structurés et Interopérabilité Cedric Honnet Robotique et Systèmes Embarqués ,Telecom ParisTech, 24 mars 2011.

[29] Daniel Pauli and Dominique Im Obersteg, Californium, Spring 2010.

[30] www.Pi4j.com

[31]www.raspberry.com