Étude comparative de cadriciels libres pour le développement d'applications IoT

par

Zeineb BABA CHEIKH

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE, CONCENTRATION GÉNIE LOGICIEL
M. Sc. A.

MONTRÉAL, LE 11 JUIN 2020

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

Mme Ghizlane EL BOUSSAIDI, directeur de mémoire
Département de génie logiciel et des TI, École de technologie supérieure

M. Julien GASCON-SAMSON, co-directeur
Département de génie logiciel et des TI, École de technologie supérieure

M. Francis BORDELEAU, président du jury
Département de génie logiciel et des TI, École de technologie supérieure

M. Sègla KPODJEDO, membre du jury
Département de génie logiciel et des TI, École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 05 JUIN 2020

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à exprimer mes sincères remerciements à tous ceux qui ont contribué à la réalisation de ce mémoire.

Je voudrais en premier lieu remercier les membres du jury d'avoir accepté de juger mon modeste travail.

Je remercie très chaleureusement mon encadrante Mme Ghizlane EL-BOUSSAIDI pour sa disponibilité et son écoute, la qualité d'encadrement et ses conseils précieux. Je remercie également mon co-directeur M. Julien GASCON-SAMSON, pour l'encadrement et les conseils.

Je désire aussi remercier l'organisme MITACS pour m'avoir offert une telle opportunité de poursuivre les études supérieures au Canada, dans une école prestigieuse, et avec une meilleure qualité d'enseignement, telle que l'École de Technologies Supérieures. Je remercie ainsi le cadre administratif et enseignant de l'ÉTS. J'adresse aussi mes sincères mots de gratitude à tous les professeurs qui ont contribué à ma formation tant en Tunisie qu'au Canada.

Je voudrais aussi remercier ma famille : mon père Jalel, ma mère Sabeh, ma soeur Maryem, son époux Achref et leur petit bout de choux attendu. Merci également à ma chère Fatma de m'avoir tendu la main dans les moments les plus difficiles.

Finalement, j'aimerais remercier mon fiancé Mohamed pour tout ce qu'il fait pour moi.

Étude comparative de cadriciels libres pour le développement d'applications IoT

Zeineb BABA CHEIKH

RÉSUMÉ

Le marché de l'Internet des objets (IoT) se développe rapidement avec un nombre croissant d'appareils connectés. Cela a conduit de nombreuses entreprises à se concentrer sur le développement et la recherche de solutions IoT. Le développement de l'IoT a ses propres défis. En effet, les solutions IoT typiques sont composées d'objets, de protocoles et de logiciels hétérogènes. Pour relever ces défis, de nombreux frameworks sont disponibles pour aider les développeurs à créer des applications IoT. Certains de ces frameworks sont open source et pourraient être d'un grand intérêt pour les petites et moyennes entreprises souhaitant créer des solutions IoT à moindre coût. Dans ce mémoire, nous présentons les résultats d'une étude expérimentale sur des frameworks de développement IoT open source. En particulier, nous avons utilisé ces frameworks pour déployer un échantillon de trois applications IoT et nous les analysons par rapport à un ensemble minimal d'exigences fonctionnelles et non fonctionnelles en IoT. Nous nous concentrons dans notre étude sur le développement IoT pour Raspberry PI, car c'est une plate-forme très populaire et très peu coûteuse.

Mots-clés: internet des objets, framework de développement IoT open source, applications IoT, raspberry Pi

Comparative study of open-source IoT development frameworks

Zeineb BABA CHEIKH

ABSTRACT

The Internet of Things (IoT) market is growing fast with an in-creasing number of connected devices. This led many software companies to shift their focus to develop and provide IoT solutions. IoT development has its own challenges as typical IoT solutions are composed of heterogeneous devices, protocols and software. To cope with these challenges, many frameworks are available to help developers to build IoT applications. Some of these frameworks are open source and might be of great interest for small and medium-sized companies wishing to build IoT solutions at a lower cost. In this thesis, we present the results of our study of open source IoT development frameworks. In particular, we used these frameworks to implement a sample of three IoT applications and we analyze them against a minimal set of IoT requirements.We focus in our study on the IoT development for Raspberry PI as it is a very low-cost and popular platform.

Keywords: internet of things, open source IoT development frameworks, IoT applications, raspberry Pi

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

6LoWPAN	IPv6 Low power Wireless Personal Area Networks
API	interface de programmation applicative
APNS	Apple Push Notification service
AwS	Amazon Web Services
CISCO	Computer Information System COmpany
CoAP	Constrained Application Protocol
COTS	Commercial off-the-shelf
CRM	Customer Relationship Management
DSaaS	Data Storage as a Service
DSL	Domain Specific Language
DSML	Domain Specific Modeling Language
EMF	Eclipse Modeling Framework
GCM	Google Cloud Messaging
GPIO	General Purpose Input/Output
HART	Highway Addressable Remote Transducer
HTTP	Hypertext Transfer Protocol
IaaS	Interface as a Service
IBM	International Business Machines
ICN	Information Centric Networking
IdO	Internet des Objets
IEC	International Electrotechnical Commission
IHM	Intefrave Homme Machine
IoE	Internet of Everything

IoNT	Internet of Nano Things
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Organization for Standardization (Organisation internationale de normalisation)
ISO-RA	International Organization for Standardization-Reference Architecture
JSON	JavaScript Object Notation
LED	Lampe Électroluminescente
LPDDR2	Low Power Double Data Rate 2
LTE	Long-Term Evolution
LTE-A	Long-Term Evolution -Advanced
M2M	Machine to Machine
MPNS	Microsoft Push Notification Services
MQTT	Message Queuing Telemetry Transport
NFC	Near Field Communication
OC	Optical Code
OSGI	Open Services Gateway initiative
OSI	Open Systems Interconnexion
P2M	Person to Machine
P2P	Person to Person
PaaS	Platform as a Service
Pub/Sub	Publish/Suscribe
QRC	Quick Response Code

REST	Representational state transfer
RFID	Radio-frequency identification
SaaS	Software as a Service
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random Access Memory
SEP 2.0	Smart Energy Profile 2.0
SMS	Short Message Service
SOA	Service Oriented Architecture
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TI	Technologie de l'Information
TLS	Transport Layer Security
TSaaS	TSimulus As A Service
UDP	User Datagram Protocol
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
WAN	Wide Area Network
WiFi	Wireless Fidelity
WSN	Wireless sensor network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

INTRODUCTION

Contexte et problématique

L'Internet des objets (IoT) est un réseau qui interconnecte un nombre énorme de dispositifs physiques (ISO, 2018). Ces objets appartiennent généralement à l'une des deux catégories : les capteurs, qui collectent les données du monde physique, et les actuateurs, qui agissent selon la situation afin de modifier l'état du monde physique. Le paysage de l'IoT s'est développé à un rythme considérable au cours des dernières années. Selon des études et rapports industriels récents, 20.6 milliards d'objets sont désormais connectés selon (Ericsson, 2018) et 5 quintillions d'octets de données sont produits quotidiennement par les objets de l'IoT selon (Stack, 2018). Cette tendance a poussé les entreprises informatiques à s'orienter vers l'IoT afin de trouver les solutions appropriées aux enjeux en IoT.

En effet, et contrairement aux applications logicielles traditionnelles qui reposent sur une infrastructure traditionnelle (par exemple, des ordinateurs, des serveurs, le cloud,etc.), l'IoT repose sur l'interconnexion d'objets et services hétérogènes. À titre d'exemple, les extrémités d'une application IoT (par exemple, les capteurs) sont généralement constituées de microcontrôleurs simples, tandis que les plates-formes matérielles en IoT (Raspberry Pi, Beaglebone,etc.) sont constituées de processeurs multicore, plusieurs gigaoctets de mémoire et peuvent exécuter des systèmes d'exploitation complets. Parallèlement, l'écosystème IoT présente une haute hétérogénéité des applications ; les applications en IoT proviennent de plusieurs domaines et émanent de différentes industries. Cette panoplie exige qu'on ait les outils et les pratiques nécessaires afin de gérer efficacement la diversité logicielle et matérielle des objets et des domaines d'application en IoT.

Dans cette perspective, plusieurs cadriciels (i.e. frameworks) IoT ont été proposés pour faciliter diverses tâches reliées à la mise en oeuvre des applications IoT. Certains frameworks permettent

le développement et le déploiement des applications IoT, d'autres offrent la possibilité de gérer les applications localement ou à distance, d'autres encore permettent d'analyser les données IoT pour en tirer des décisions d'affaire. Plusieurs de ces frameworks sont open source et pourraient être d'un grand intérêt pour les petites et moyennes entreprises souhaitant investir en IoT à moindre coût. En effet, au cours des dernières décennies, les technologies reliées à l'IoT changent et évoluent très rapidement. Ceci constitue un défi pour les petites et les moyennes entreprises désirant suivre cette tendance et visant à offrir des solutions innovatrices et à moindre coût à leurs clients. Dans ce contexte, les frameworks IoT open source peuvent aider ces entreprises.

Cependant, les frameworks IoT existants ciblent différents domaines et ont différents objectifs offrant ainsi un support différent pour le développement des applications IoT. En effet, pour la plupart, les frameworks IoT englobent des technologies qui permettent la gestion et l'automatisation des objets connectés dans les applications IoT. Un framework IoT connecte fondamentalement un objet aux autres objets disponibles. Les frameworks diffèrent selon la façon dont ils utilisent le cloud, les options et protocles de connectivité qu'ils supportent, les mécanismes de sécurité qu'ils offrent et leur capacité de traitement des données. Du point de vue de développeurs logiciels, ces frameworks fournissent un ensemble de fonctionnalités prêtes à l'emploi qui accélèrent considérablement le développement d'applications pour les objets connectés et qui prennent en charge l'évolutivité et la compatibilité entre les différents objets. Ces fonctionalités peuvent varier d'un framework à un autre, et elles peuvent inclure, par exemple, la description des interfaces des objets connectés, leur comportement et leurs caractéristiques physiques, ou la génération automatique du code à déployer. Il peut être donc difficile pour les développeurs d'identifier le framework le plus approprié pour concevoir et implémenter une application IoT donnée. Dans ce contexte, il est nécessaire de fournir aux développeurs un moyen leur permettant d'évaluer un framework et de le sélectionner en fonction du support qu'il peut leur fournir afin de développer leurs applications IoT.

Objectifs

L'objectif général de notre projet de recherche est d'évaluer et de comparer des frameworks IoT. En particulier, nous nous concentrons sur l'évaluation du support fourni par les frameworks open source pour développer et déployer des applications IoT.

À partir de notre objectif général, nous avons identifié deux sous-objectifs :

- évaluer le support qu'offrent les frameworks open source IoT pour le développement de différentes catégories d'applications IoT : notre objectif ici est d'analyser la capacité des frameworks étudiés à couvrir plusieurs domaines d'applications nécessitant différents types d'objets connectés ;
- évaluer le support qu'offrent des frameworks open source IoT pour satisfaire aux exigences fonctionnelles et non fonctionnelles des applications IoT : notre objectif ici est d'analyser de façon systématique les frameworks étudiés en fonction d'un ensemble de critères ; ces derniers correspondent à un ensemble minimal d'exigences identifiées dans la littérature et des standards comme étant nécessaires pour la mise en œuvre des applications IoT.

Pour atteindre nos objectifs, nous avons réalisé une étude comparative d'un ensemble de frameworks open source IoT. Notre étude combine deux types d'analyses : 1) des analyses basées sur nos expérimentations avec ces frameworks pour implémenter différentes applications, et 2) des analyses qualitatives basées, entre autres, sur la documentation disponible de chacun des frameworks.

Méthodologie de recherche

Pour réaliser notre projet de recherche, nous avons suivi la méthodologie illustrée dans la figure 0.1

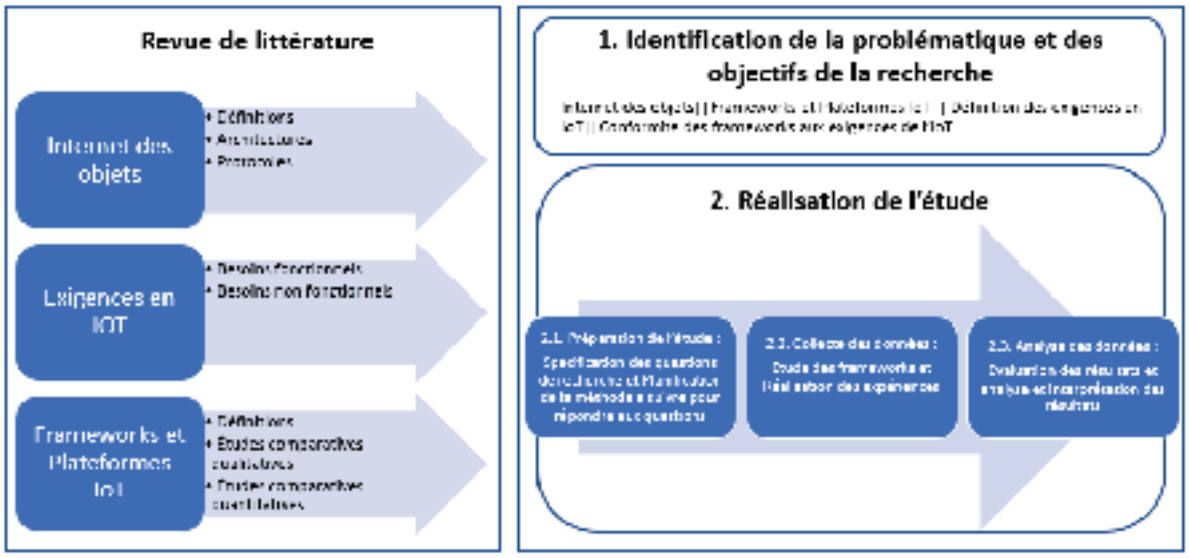


Figure 0.1 Phases de la méthodologie.

Nous avons réalisé une revue de littérature qui nous a permis de nous familiariser avec les concepts de base reliés à l'IoT et en particulier à la mise en oeuvre des applications IoT. Notre revue de littérature avait aussi pour objectif d'identifier et comprendre les exigences des applications IoT et de faire un survol des travaux existants qui se sont intéressés à l'étude des frameworks IoT. Comme montré par la figure 0.1, l'étape de revue de littérature a duré tout au long de notre projet de recherche.

Dans l'étape 1 de notre méthodologie, nous avons défini la problématique et précisé nos objectifs en fonction des résultats de notre revue de littérature et en tenant compte des limites identifiés dans l'état de l'art. Dans l'étape 2 de notre méthodologie, nous avons procédé à la réalisation de l'étude comparative des frameworks IoT en vu de réaliser nos objectifs de recherche. Conformément aux directives de (Wohlin et al., 2012), notre étude se compose de 3 étapes, numérotées 2.1, 2.2 et 2.3 dans la figure 0.1. L'étape 2.1 consiste à spécifier les objectifs de l'étude en formulant les questions auxquelles l'étude doit répondre. Cette étape définit aussi les expérimentations à réaliser pour répondre à ses questions ; elle inclut l'identification des frameworks IoT à étudier. L'étape 2.2 consiste à collecter des données en analysant les frameworks IoT sélectionnés

et en réalisant des expérimentations avec ces frameworks. Nos expérimentations impliquent l'implémentation de différentes applications IoT. L'étape 2.3 consiste à analyser les données collectées et interpréter les résultats par rapport aux questions de recherche posées.

Organisation du mémoire

Ce mémoire est organisé comme suit : le premier chapitre présente un ensemble de définitions de base ainsi qu'un état de l'art sur les travaux reliés à l'évaluation et au survol des plates-formes et frameworks IoT. Dans le deuxième chapitre, nous détaillons la première étape de notre étude en présentant la problématique et les questions de recherches auxquels nous répondons au cours de cette étude ainsi que les choix que nous avons menés au cours de notre recherche. La collecte de données à travers les expériences que nous avons faites est présentée dans le troisième chapitre. Dans le quatrième chapitre, nous analysons les données collectées et nous interprétons et discutons les résultats obtenus. Le cinquième chapitre synthétise les résultats et présente les limitations de notre recherche. Finalement, nous présentons une conclusion générale ainsi que les perspectives de ce travail dans un dernier chapitre.

CHAPITRE 1

REVUE DE LITTÉRATURE

Dans ce chapitre, nous présentons des définitions de concepts de l'IoT et nous faisons un survol des travaux reliés à notre projet de recherche. En particulier, nous commençons par définir les concepts Internet des objets, "framework" et "plate-forme" en IoT. Nous présentons ensuite d'autres concepts et aspects reliés tels que l'architecture des applications IoT et les technologies les supportant. Nous présentons aussi une revue des différents travaux effectués à titre d'étude et/ou évaluation des différents frameworks et plates-formes IoT existantes.

1.1 Définitions des concepts clés

1.1.1 Qu'est-ce que l'IoT

Le terme IoT, de Internet of Things, ou du français, IdO (Internet des Objets) désigne théoriquement l'ensemble des objets intelligents, connectés sur Internet. Toutefois, dans la littérature, plusieurs définitions ont été proposées. Atzori, Iera & Morabito (2010) définissent l'IoT comme étant l'interconnexion des différents objets pour la réalisation d'une valeur ajoutée aux utilisateurs finaux. Matta, Pant & Arora (2017) définissent l'IoT comme un réseau global composé d'un ensemble d'objets connectés, ou intelligents : objets, capteurs, actuateurs, et des composants de stockage. Miraz, Ali, Excell & Picking (2015) définissent l'IoT comme étant l'ensemble des objets électroniques et électriques de différentes tailles et capacités, qui sont connectés à Internet via différentes technologies telles que ZigBee, RFID, WSNs et des technologies basées sur la localisation (par exemple, GPS). Rahman *et al.* (2016) définissent l'IoT comme étant un système composé d'objets à faibles ressources, faible mémoire, puissance de traitement, énergie et accessibilité. Ces objets sont connectés via des réseaux IP restreints, c'est-à-dire avec un débit binaire bas, des limites de cycle de service, une perte de paquets élevée, des liaisons asymétriques, des primitives de communication de groupe restreint, mais réunis avec des réseaux

rapides et des services internet réguliers. Ces services Internet stockent et traitent normalement de grandes quantités de données.

L'IoT offre des possibilités infinies pour des scénarios innovants, caractérisés par le fait qu'il s'agit d'applications distribuées consistant en des services de collaboration s'exécutant sur des appareils distincts. Il est difficile de développer de telles applications en termes de configuration, de programmation et de cycles de vie des entités impliquées. ISO (2018) présente l'IoT comme étant un des secteurs les plus dynamiques et excitants en technologie de l'information. Il consiste à faire connecter des entités physiques avec le monde technologique à travers le réseau internet. Les fondamentaux de l'IoT sont les appareils électroniques qui interagissent avec le monde physique. Les capteurs collectent les variables du monde physique et les actuateurs agissent. Les capteurs et les actuateurs peuvent se présenter sous plusieurs formes, telles que les thermomètres, accéléromètres, caméra vidéo, microphones, transformateurs, chauffages ou des équipements industriels pour le contrôle des procédures et de la fabrication.

Le groupe RFID définit l'IoT comme étant un réseau mondial d'objets interconnectés et adressables de manière unique en se basant sur des protocoles de communication standard. D'après le pôle de recherche européenne (Sundmaeker, Guillemin, Friess & Woelfflé, 2010), les objets sont les principaux acteurs en IoT. Ils sont des participants actifs dans les processus commerciaux, informatiques et sociaux où ils sont capables d'interagir et de communiquer entre eux et avec l'environnement pour mesurer les variables, tout en réagissant de manière autonome aux événements réels/physiques avec ou sans interaction humaine. Enfin, Bélissent et al. (2010) définissent l'IoT comme étant un environnement intelligent qui utilise les informations et les technologies de communication pour rendre l'infrastructure des secteurs de l'éducation, la santé, les services publics, le transport, l'immobilier et tout autre secteur, modernes, efficaces et interactifs. Gubbi, Buyya, Marusic & Palaniswami (2013) proposent, quant à eux, une définition centrée sur l'utilisateur de l'IoT. Selon eux, une application IoT est l'interconnexion des capteurs et des actuateurs offrant la possibilité de partager l'information sur différentes plates-formes en suivant un framework donné. Ceci inclut la mesure des données physiques, l'analyse des données, la représentation des données sur le cloud.

Dans notre étude, nous adoptons la définition la plus commune de l'IoT qu'on considère comme étant l'ensemble des objets intelligents et interconnectés à travers un réseau.

1.1.2 Quels sont les composants d'une application IoT ?

Selon (Matta *et al.*, 2017), l'IoT se compose de : objets, capteurs, actuateurs, et des composants de stockage :

- objets : ce sont les objets équipés en électronique, ayant une capacité de calcul et une interface de communication. On peut découvrir, gérer, interagir avec et contrôler ces objets via internet. Le terme objet peut englober les appareils, les êtres humains, ou toute autre entité à laquelle on peut associer des attributs ;
- capteurs : c'est une interface essentielle pour l'implémentation en IoT, considérée comme le "*frontend*" de l'environnement IoT. La collecte de données des capteurs est considérée comme l'évènement stimulant toutes les activités en IoT. La nature des capteurs utilisés dépend intrinsèquement de la nature de l'application elle-même. Certains capteurs sont spécifiques à un domaine d'application bien déterminé, alors que d'autres sont adaptés à différentes applications IoT. Les capteurs sont classifiés essentiellement selon la taille des données, la fréquence d'échantillonnage, le nombre d'appareils connectés et les types de signaux associés au capteur. On peut aussi identifier les capteurs comme capteur de proximité, capteur tactile, capteur d'humidité, capteur de température, capteur de pression, accéléromètre et gyroscope ;
- actuateurs : les données collectées par les capteurs sont analysées pour prendre des décisions et déclencher les actions appropriées. Cette étape est assurée généralement par les actuateurs. L'actuateur est donc un objet qui transforme les signaux électriques qu'il a reçu des capteurs (ceux-là transforment au paravent l'énergie fonctionnelle (informations/actions) en signaux électriques) en une énergie fonctionnelle (par exemple une énergie mécanique, une énergie dynamique). L'ensemble des capteurs-actuateurs forme la colonne vertébrale des applications IoT. Il est à noter que contrairement aux capteurs, le nombre d'actuateurs offets sur le marché ne répond toujours pas à la demande (Saad, 2016). Ceci peut être une bonne piste d'amélioration aux chercheurs voulant travailler sur cet aspect. Les actuateurs sont classés

en fonction de la source d'énergie dont ils ont besoin et qu'ils utilisent pour provoquer le mouvement. Les actuateurs IoT sont également identifiés comme étant pneumatiques, hydrauliques, électriques et thermiques ;

- les composants de stockage : étant un ensemble constitué d'un grand nombre d'objets, d'applications et de services qui communiquent entre eux, la quantité d'information manipulée en IoT est énorme. Selon le type d'application, l'information peut devoir être stockée. Le stockage des données peut se faire à l'interne (dans un fichier ou dans une base de données), ou sur le cloud.

Rahman *et al.* (2016) définissent les éléments d'un système IoT de façon plus granulaire. Ces éléments sont présentés dans le tableau 1.1.

Tableau 1.1 Éléments d'un système IoT, selon (Rahman *et al.*, 2016)

Élément	Rôle/Description
Capteur	objet à faibles ressources ayant des capacités de capture de données
Actuateur	objet à faibles ressources ayant des capacités d'actionneur
Identificateur	objet qui offre l'identification automatique à des objets qui y sont attachés
Commutateur	objet qui achemine les données aux destinataires, il opère principalement sur la deuxième couche du modèle OSI
Passerelle	objet qui permet de connecter deux réseaux de différents types (par exemple, différents protocoles de communication)
Périphérique de stockage	objet pour stocker l'information
Périphérique des utilisateurs	objets permettant aux utilisateurs d'exécuter les différentes applications (cellulaire, ordinateur,etc.)
périphériques embarqués	objets ayant une capacité de calcul élevée
Ferme	Collection de serveurs informatiques avec une capacité de stockage et d'exécution importante située sur Internet

1.1.3 Qu'est-ce qu'un framework en IoT ?

Selon (Derhamy, Eliasson, Delsing & Priller, 2015), le terme Framework en IoT désigne l'ensemble de principes directeurs, de protocoles et standards permettant l'implémentation

des applications IoT. Les frameworks ont pour principal but d'améliorer le développement des applications IoT et ce, en assurant une implémentation rapide, l'interopérabilité, la maintenabilité, la sécurité et la flexibilité technologique. Rahman *et al.* (2016) définissent un framework comme étant une "boîte à outils" pour développer des applications IoT selon un certain style ou méthode. Un framework IoT adopte un ou plusieurs protocole(s) IoT, et, fournit des APIs et des librairies qui implémentent des services en dessus des protocoles applicatifs. Il peut aussi fournir des outils de développement, test et déploiement. Dans le cadre de notre travail et dans tout le reste de ce mémoire, nous adoptons la définition de (Rahman *et al.*, 2016) pour le concept de framework IoT ; i.e. un ensemble d'outils et d'APIs permettant de développer une application IoT.

1.1.4 Qu'est-ce qu'une plate-forme en IoT ?

Dans la littérature sur l'IoT, il n'est pas toujours évident de différencier les notions de frameworks et de plate-formes. Hejazi, Rajab, Cinkler & Lengyel (2018) définissent une plate-forme IoT comme étant une interface qui pourrait coordonner divers aspects essentiels qui conduisent à la réalisation des objectifs IoT. Cela inclut la définition de la méthode de communication des points finaux du système au réseau, et la façon dont les données sont collectées et leur lieu de stockage. Nous adoptons une définition similaire dans le reste de ce document, et on considère donc une plate-forme comme un outil qui permet à travers une interface de gestion, d'ajouter, supprimer, ou modifier les objets et de les interconnecter entre eux ou à travers le réseau.

1.2 Domaines d'utilisation de l'IoT

Actuellement, quasiment tous les secteurs utilisent l'IoT comme technologie de pointe et en tirent profil pour faciliter les tâches quotidiennes qu'ils mènent. Plusieurs articles survolent les principaux secteurs où l'IoT est le plus utilisé. Parmi les secteurs, nous listons ((Oracle, 2020), (Gubbi *et al.*, 2013)) :

- transport et logistique : depuis l'apparition de l'IoT, les véhicules sont devenus de plus en plus "intelligents". On parle de voiture autonome, assistance propilote et gestion de la congestion

routière. Les voitures sont actuellement munies d'un très grand nombre de capteurs qui leur permettent d'assister le conducteur et d'éviter plusieurs accidents ;

- santé : le secteur de santé a connu une révolution technologique depuis l'apparition de l'IoT, permettant ainsi, aussi bien aux docteurs qu'aux patients, de recevoir des informations sur l'état de santé des malades, parfois en temps réel, et de sauver des vies, des fois sans même l'intervention du personnel médical (par exemple, injection automatique de l'insuline, dès la détection de la chute du taux de la substance chez un sujet diabétique) ;
- environnement intelligent : grâce à l'IoT, toutes les données de l'environnement demeurent disponibles en quasi-temps réel. Les données météorologiques et les données de localisation en sont des exemples. Ces données permettent d'offrir des services adaptés à des utilisateurs ;
- domotique : Ceci englobe principalement 4 applications : 1) L'automatisation des appareils d'électroménager, 2) La sécurité de la maison (par exemple les systèmes d'alarme, les caméras de surveillance), 3) la gestion des accès (ouverture et fermeture automatique des portes et des fenêtres) et 4) la gestion énergétique (chauffage/climatisation automatique). Cela permet d'assurer un certain confort aux habitants et de réduire les risques pour les personnes à mobilité réduite ;
- personnel et social : l'IoT a révolutionné la vie de tous les individus. On parle de montres intelligentes, habits intelligents, des smartphones ou téléphones connectés. Selon planetscope (Planetscope, 2018), il se vend environ 130 millions de téléphones intelligents par mois dans le monde soit 1,56 milliard par an.

1.3 Exigences et caractéristiques des applications IoT

Certains travaux abordent les caractéristiques des applications IoT et des exigences qui en découlent. Dans cette section, nous présentons une synthèse de ces exigences en terme de besoins fonctionnels et non fonctionnels. Nous discutons aussi des caractéristiques d'une application IoT telles que introduites par le standard ISO de l'IoT (ISO, 2018) et de quelques défis en relation avec ces caractéristiques.

1.3.1 Besoins fonctionnels

Les besoins fonctionnels en IoT sont l'expression des exigences opérationnelles qu'une application IoT doit satisfaire (ISO, 2018). Plusieurs travaux discutent des besoins fonctionnels. Le tableau 1.2 fait une synthèse des besoins les plus discutés (Rahman *et al.*, 2016).

Tableau 1.2 Fonctionnalités d'un système IoT, selon (Rahman *et al.*, 2016)

Fonction	Description
Capture	contrôle et connecte le capteur. Elle inclut le pré-traitement des données et la configuration du matériel via une commande au niveau de l'application.
Action	contrôle et connecte l'actuateur. Cela inclut la configuration du matériel via une commande au niveau de l'application.
Profil	expose des services selon le profil des objets, incluant des informations statiques (localisation, identité) et dynamiques (capteurs et actuateurs).
Gestion des objets	gère la configuration et le cycle de vie du système.
Contrôle	détermine le comportement de l'objet (comportement de l'actuateur en fonction de l'information reçue de la part du capteur), et traite les requêtes des utilisateurs pour modifier l'état/comportement du système
Application	la combinaison des différentes fonctionnalités des objets afin de réaliser une application complète répondant aux besoins des utilisateurs finaux
API	fournit des interfaces de programmation des applications pour des fonctions existantes
Découverte	la fonctionnalité de découverte de nouveaux objets
Stockage	collecte les informations et les stocke
Analyse verticale	effectue des analyses sur les données d'un seul système
Analyse horizontale	effectue des analyses sur les données de différents systèmes
Traduction	traduit les données entre deux protocoles et achemine les données entre deux réseaux différents

1.3.2 Besoins non fonctionnels

Les besoins non fonctionnels en IoT incluent autant des attributs de qualité (e.g. performance, disponibilité, fiabilité, etc.) que des exigences en termes de respect de la vie privée et stockage des données. Comme dans les applications traditionnelles, les exigences non fonctionnelles ont une importance majeure dans les applications IoT. En effet, les applications IoT reposent majoritairement sur l'aspect "temps réel" et les données traitées en IoT peuvent être réparties

sur plusieurs objets. Ce dernier aspect présente des défis pour la protection des données et leur confidentialité. Le tableau 1.3 fait une synthèse des besoins non fonctionels les plus discutés dans la littérature (Rahman *et al.*, 2016).

Tableau 1.3 Besoins non fonctionnels en IoT selon (Rahman *et al.*, 2016)

Besoin non fonctionnel	Description
Latence	le temps écoulé pour que les données passent de la source vers la destination
Point unique d'échec	existence potentielle d'un point unique d'échec
Confidentialité	le contrôle des données dans les domaines de gestion
Sécurité	garantir une meilleure disponibilité, intégrité des données et du système, autorisation et confidentialité des données
Complexité de décision	la complexité des décisions qu'une fonction de contrôle élabore
Dépendance en temps	le besoin qu'un système soit à l'état actif à un certain temps
Évolutivité	la capacité de servir un nombre évolutif d'applications
Interopérabilité	la capacité de faire communiquer un objet avec d'autres différents objets
Simplicité	la facilité de développement et déploiement des applications IoT aux utilisateurs

1.3.3 Caractéristiques et exigences selon le standard ISO-RA

Parmi les initiatives visant la standardisation de l'IoT, ISO (International Organization for Standardization) a proposé un standard pour l'IoT (ISO, 2018). Dans ce document, ISO propose une architecture de référence pour l'IoT (discutée à la section 1.5) et présente les meilleures pratiques de l'industrie en IoT. Ces pratiques sont présentées sous forme de caractéristiques auxquelles une application IoT doit se conformer. ISO divise les caractéristiques en trois catégories majeures :

- caractéristiques de fiabilité du système IoT : c'est les critères qui définissent le degré de fiabilité du système. En d'autres termes, jusqu'à quel point les parties prenantes peuvent être sûres que leurs applications peuvent faire face aux erreurs humaines et erreurs de calcul des variables d'environnement. Parmi ces caractéristiques, il y a la disponibilité, la

résilience, la protection des données et la sécurité. Le tableau 1.4 décrit sommairement ces caractéristiques ;

- caractéristiques de l'architecture du système IoT : critères relatifs à l'infrastructure de l'IoT (par exemple, architecture du système, protocoles réseau). Ces caractéristiques incluent, par exemple, la composabilité (capacité de combiner différents objets), la gestion de l'hétérogénéité, l'intégration des systèmes légataires, la modularité et l'identification unique. Le tableau II-1 en annexe décrit sommairement ces caractéristiques ;
- caractéristiques fonctionnelles du système IoT : critères reliés aux fonctions supportées par une application IoT. Ces caractéristiques incluent, par exemple, la précision des données, la configuration automatique du système IoT, la sensibilité au contexte et la découvrabilité des services. Ces caractéristiques sont décrits sommairement dans le Tableau III-1 en annexe.

Tableau 1.4 Caractéristiques de fiabilité du système IoT

Critère	Description
Disponibilité	Capacité d'un système à être accessible et utilisable à la demande par une entité autorisée.
Confidentialité	Propriété que les informations ne soient pas rendues disponibles ou divulguées à des individus, entités ou processus non autorisés.
Intégrité	Propriété de précision et de complétude des données du système
Fiabilité	Cohérence du comportement et des résultats attendus.
Protection des données d'identité personnelle	Concept chevauchant au concept de confidentialité, c'est un règlement judiciaire requis dans les différentes applications IoT impliquant des données personnelles. La protection des informations personnelles est une exigence générale, régie par une série de principes décrits dans la norme ISO / IEC 29100
Résilience	Capacité d'un système ou d'un de ses composants à continuer son activité en présence d'une panne ou une défaillance
Sécurité	État dans lequel le risque sur la sécurité physique des individus et la sécurité du matériel est limité à un niveau acceptable. Le risque est la combinaison de la probabilité de survenue d'un préjudice et de la gravité de ce préjudice.

1.3.4 Défis reliés aux applications IoT

La satisfaction de certaines exigences des applications IoT reste un défi pour les industries. Certains des défis les plus importants sont ((Matta *et al.*, 2017), (Miraz *et al.*, 2015)) :

- plusieurs exigences de l'IoT nécessitent des standards et celà à différents niveaux ; par exemple, la sécurité en IoT, la découvrabilité, etc ;
- l'alimentation énergitique des capteurs est un enjeu en IoT. Cela représente un défi dans le cas où les capteurs sont distants, non accessibles et contrôlés à distance. Certaines technologies peuvent être exploitées pour aider dans ce contexte, notamment les générateurs thermiques et les cellules solaires ;
- l'architecture d'un tel réseau, aussi distribué et hétérogène constitue aussi un enjeu très important en IoT. L'évolutivité, l'interopérabilité, la gestion des objets et services sont des aspects importants à considérer dans le développement de ces architectures ;
- en ce qui concerne le traitement des données, les principaux problèmes sont le partage, la propagation, la collaboration, l'efficacité du traitement et la présentation des données ;
- la sécurité et la confidentialité des données, tant du point de vue des fournisseurs que des utilisateurs, constituent aussi des défis majeurs ;
- l'adoption de l'IoT constitue un enjeu sociétal, car il est parfois difficile de convaincre les utilisateurs de changer leurs habitudes. Plusieurs utilisateurs n'ont pas confiance en ce type de système par souci de sécurité et de protection de la vie privée et des données personnelles, notamment avec le taux alarmant des brèches de données. D'un autre côté, les applications doivent avoir des interfaces conviviales (*user-friendly*), pour inciter les gens à les utiliser.

1.4 Technologies supportant les applications IoT

L'IoT doit son progrès à plusieurs technologies existantes, notamment l'internet, le cloud et le Big-Data :

- Internet : Malgré la divergence des définitions de l'IoT, Internet reste le point commun de ces définitions. Internet forme donc une plate-forme d'implantation de l'IoT. Cependant, Internet sous sa forme traditionnelle permet d'interconnecter les objets, sans donner la capacité de capturer l'information. Une couche par dessus Internet *"Sensing Layer"* a permis d'avoir l'IoT, sous sa forme actuelle ;
- Cloud : L'utilisation du cloud dans l'IoT a permis d'intégrer l'IoT dans tous les secteurs. Ensemble, ils sont devenus omniprésents dans notre quotidien. Plusieurs plates-formes et frameworks Cloud open source et payants existent pour intégrer des services IoT, pour n'en nommer que quelques-uns, on cite : Arkessa, OpenIoT, Nimbits, LittleBits, OnePlatform, SmartThings, SensorCloud, Xively. Ces plates-formes assurent les services basés sur le cloud en IoT ;
- Big-Data : Le nombre énorme d'objets interconnectés à travers l'IoT contribue à génerer des données massives (Big-Data) qui doivent être traitées, analysées et stockées. Certaines technologies et services tels que Hadoop, SciDB et TSaaaS sont disponibles et prennent en charge l'analyse des *Big Data*. Elles sont également adaptées aux exigences en traitement de données de l'IoT.

Les applications IoT sont de nature hétérogène et, conséquemment, utilisent et intègrent diverses technologies et protocoles de communication réseau. Dans ce qui suit, nous faisons un survol rapide de ces technologies de communication suivi d'une présentation sommaire des protocoles utilisés en IoT.

1.4.1 Technologies de communication

Il existe plusieurs technologies de communication ciblant différents types d'équipements et offrant différentes portées et capacités. Les technologies suivantes figurent parmi les plus connues et/ou utilisées (Matta *et al.*, 2017) :

- Near-Field Communication (NFC) et Radio frequency identification (RFID) : La technologie RFID a paru au tout début des années 2000s, son successeur, NFC l'a dominé. La technologie

RFID est le processus d'identifier les objets par un identifiant unique, en utilisant des ondes radio. Elle est basée sur la présence d'un tag RFID, un lecteur de tags RFID et une antenne. Le signal est transmis du lecteur au tag via les antennes. Le tag répond par la génération et la diffusion de son identifiant unique. Les tags qui produisent leur propre alimentation sont nommés tags actifs, les autres reçoivent l'alimentation du lecteur et sont nommés tags passifs. NFC a été introduit pour assurer la sécurisation des échanges. Contrairement au cas de RFID, en NFC, les tags, les lecteurs et l'antenne font tous partie d'un même appareil. Un appareil NFC est donc capable d'assurer par lui-même les communications *Peer to peer* ;

- Optical Codes (OC) et Quick Response Codes (QRC) : Les codes optiques tels que l'indique leur nom, sont généralement des codes de nature optique, lisibles par machine et utilisés pour représenter les données de longueur variée. Ces codes contiennent des informations valides et requises sur l'objet auquel ils sont associés. Ce code peut être à une dimension, ou à deux dimensions s'il est représenté par une figure bidimensionnelle. L'application la plus connue de ce type de code est le code à barres qu'on trouve sur les produits qu'on achète. Le code QRC est un code bidimensionnel dont la lecture se fait à l'aide d'un dispositif de lecture (la caméra d'un téléphone portable par exemple). Le type d'encodage utilisé peut être numérique, alphanumérique, binaire, ou *kanji*. Ce type de technologie doit son utilisation en IoT au fait qu'il est facile à utiliser, permet une lecture rapide, et permet l'identification, le suivi et la traçabilité de l'objet ;
- ZigBee : C'est une technologie de communication qui offre une faible portée, un faible taux de transmission et une alternative moins coûteuse pour la mise en œuvre des réseaux IoT. Basée sur le protocole IEEE 802.15.4, ZigBee opère sur la couche physique et MAC du réseau. L'intervalle de transmission en ZigBee est entre 10 à 100 mètres. Sa force réside dans sa faible consommation d'énergie. ZigBee est donc très répandue dans les domaines qui exigent un faible taux de transmission de données avec une durée optimale des batteries ;
- 6LOwPAN (IPv6 over Low power Wireless Personal Area Network) : C'est un ensemble de standards Internet qui permettent l'utilisation de l'IPv6 sur un réseau sans fil à faible puissance. L'utilisation de l'IPv6 procure un large spectre d'adresses IP, vu le nombre important d'objets connectés sur Internet, qui dépassent le nombre d'adresses fournies en

IPv4. 6LOwPAN vient remédier aux différents inconvénients des différents autres protocoles notamment pour la sécurité, puisqu'il supporte l'authentification IPsec, le cryptage, et des services web qui supportent des mécanismes de sécurisation de la couche transport, ainsi que les sockets sécurisés ;

- LTE-Advanced (LTE- A) : Faisant partie du standard LTE (Long Term Evaluation), LTE-Advanced assure une connectivité mobile à grande vitesse. Cette technologie est conçue principalement pour le réseau 4G. Sa plus connue application est la connexion véhicule à véhicule, dans le domaine des voitures connectées ;
- Z-Wave : Cette technologie est utilisée dans les maison intelligentes, spécialement dans le contrôle à distance des lumières, des fenêtres, les piscines, les garages, et les systèmes de sécurité des maisons. Le système de communication peut être géré par combinaison de l'Internet et de la passerelle Z-wave ;
- WSN (Wireless Sensor Network) : Guadane, Bchimi, Samet & Affes (2017) présentent les réseaux de capteurs sans fils, ou WSN (de l'anglais, Wireless Sensor Network) comme une technologie de pointe utilisée en IoT. Un réseau de capteurs est un réseau *ad hoc* dont la plupart des nœuds sont des micro capteurs capables de récolter et de transmettre des variables d'environnement d'une manière autonome. La position de ces nœuds n'est pas obligatoirement déterminée. Ils peuvent être aléatoirement dispersés dans une zone géographique, nommée « champ de captage » correspondant au terrain d'intérêt pour le phénomène capté. On entend dire par réseau ad hoc, l'ensemble d'appareils auto configurables étant libres de se déplacer. C'est donc un réseau "décentralisé".

1.4.2 Protocoles de communication IoT

Les protocoles de communications sont très importants puisqu'ils définissent les règles et les procédures de communication des couches physique et liaison de données du modèle OSI. En IoT, certains protocoles commencent à dominer ; c'est notamment les cas de COAP (Constrained Application Protocol) et MQTT (Message Queue Telemetry Transport). Ces deux protocoles présentent des similarités et différences (Hejazi *et al.*, 2018) :

- CoAP : CoAP est un protocole de communication conçu pour les objets IoT, inspiré du protocole HTTP (Hypertext Transfer Protocol) et utilisant une communication de bout en bout. Étant conçu pour des objets à ressources restreintes, CoAP est léger et génère le moins de trafic possible. CoAP ne prend donc pas en charge le protocole TCP/IP, mais plutôt le protocole UDP car ce dernier utilise moins de bande passante et ne prend pas en charge l'acquittement de réception (ACK) comme c'est le cas de TCP/IP. Il utilise moins de ressources et implémente plus de fonctionnalités que HTTP telles que des fonctionnalités d'observation, d'exécution et de découverte, ainsi que des fonctions de lecture et d'écriture. La fonctionnalité de découverte incluse dans CoAP permet de rechercher des objets situés et connectés dans l'environnement ;
- MQTT : MQTT est un protocole de messagerie implémenté sur TCP/IP, il s'agit d'un protocole léger adoptant la méthode Pub/Sub (Publier et Souscrire). Il utilise le serveur de courtier de messages (*MQTT broker*) au milieu des communications entre les objets. Ce n'est donc pas un protocole machine à machine. Il se compose de 3 éléments : *Subscriber*, *Publieur* et un *Broker*. Les clients souscrivent et publient au Broker à travers des *topics*. Tel qu'illustré dans la figure 1.1, si un client (Temperature sensor dans la figure) publie des valeurs de la température en lui attachant le topic *"Temperature"*, les deux autres clients recevront les données s'ils sont souscrits au Broker avec le même topic, soit, *"temperature"*. De point de vue sécurité, MQTT supporte SSL et TLS (Secure Sockets Layer et Transport Layer Security).

Le choix d'un protocole ou un autre dépend de la nature de l'application IoT visée, Dans le cas d'un réseau WAN (Wide Area Network), MQTT est approprié vu le concept du Broker qui réside au milieu de la communication entre les appareils. Il est donc utile pour une bande passante limitée ou sur différents sites distants. Par exemple, les services Amazon et Azure utilisent le protocole MQTT. Pour les services Web, CoAP est mieux adapté du fait de sa compatibilité avec HTTP. Il utilise UDP (User Datagram Protocol) qui prend en charge la multi-diffusion et la diffusion. Il est utilisé lorsque les objets doivent transmettre et recevoir des données à grande vitesse.

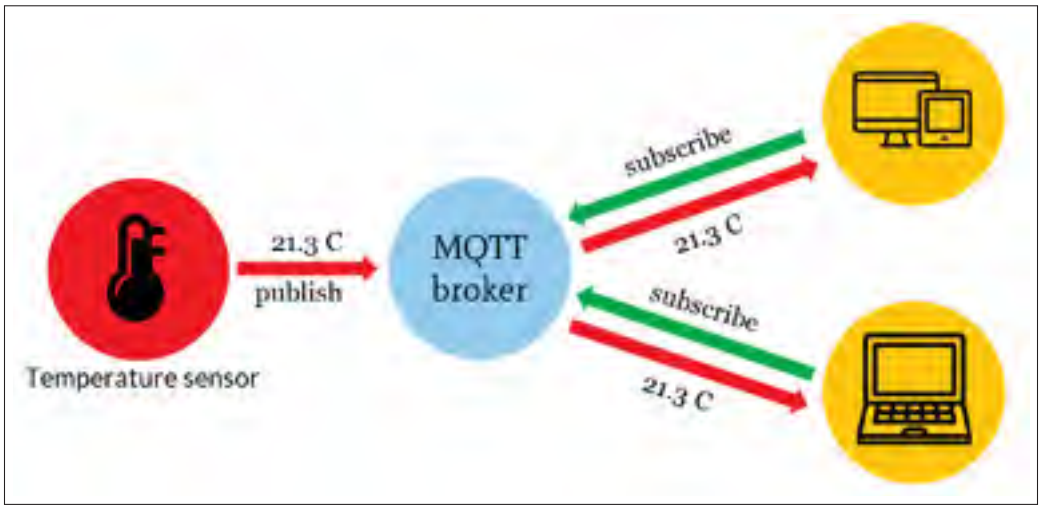


Figure 1.1 Illustration du protocole MQTT, adapté de (Hermanudin *et al.*, 2019).

1.5 Architectures des applications IoT

De façon générale, une architecture définit les différents éléments d'un système informatique et l'interaction entre ces éléments. En IoT, plusieurs architectures ont été proposées. Dans cette section, nous présentons de façon sommaire quelques architecture décrites dans la littérature. Ensuite nous présentons l'architecture de référence proposée par l'ISO.

1.5.1 Survol rapide des architectures décrites dans la littérature

Vashi *et al.* (2017) propose une architecture inspirée de l'architecture orientée service. Cette architecture est organisée en 5 couches, tel qu'illustré sur la figure 1.2 :

- couche perception : Cette couche correspond à la couche physique du modèle OSI. Elle englobe les différents types de capteurs (par exemple, RFID, infrarouge, ZigBee, QR code). Cette couche gère généralement tous les appareils, à savoir, leur identification, la collecte des informations mesurées (par exemple, température, humidité, mouvement, pression). Les informations collectées seront transmises par la suite vers la couche réseau ;
- couche Réseau : Cette couche joue un rôle capital dans la sécurité et la confidentialité de l'information à travers le transfert sécurisé des données sensibles des capteurs aux unités

de contrôle. Ce transfert se fait via des réseaux, tels que 3G, 4G, UMTS, Wi-Fi, WiMAX, RFID, infrarouge, satellite ;

- couche MiddleWare : Partant du fait que IoT est un réseau qui englobe différents objets interconnectés qui communiquent entre eux, cette couche permet la gestion des services générés par les objets. De plus, cette couche permet le stockage des informations reçues de la couche inférieure dans des bases de données ainsi que le traitement et le calcul des informations, afin de générer une décision aux actuateurs ;
- couche Application : Cette couche est responsable de la gestion des applications en fonction du type de données reçues de la couche MiddleWare. Les applications IoT couvrent plusieurs secteurs, tels que la santé, les voitures, les maisons et le transport ;
- couche Business : les fonctions de cette couche couvrent l'intégralité de la gestion des applications et des services IoT. Son principal rôle est analytique, tel que générer des rapports ou des graphes, en fonction de la quantité de données précises reçues de la couche inférieure et d'un processus d'analyse de données efficace. Cette information sera utile aux gestionnaires pour prendre des décisions sur les stratégies commerciales et les feuilles de route à adopter.

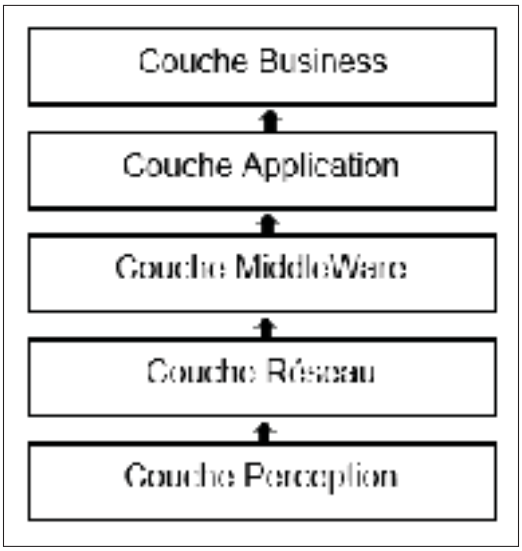


Figure 1.2 Architecture en couches, adaptée de (Vashi *et al.*, 2017).

Muccini & Moghaddam (2018) font un survol des travaux portant sur les architectures IoT et les styles architecturaux en IoT. D'après cette étude, on remarque la popularité des architectures en couches en IoT, avec un nombre de couches allant de trois jusqu'à six couches, tel que présenté dans la figure 1.3 :

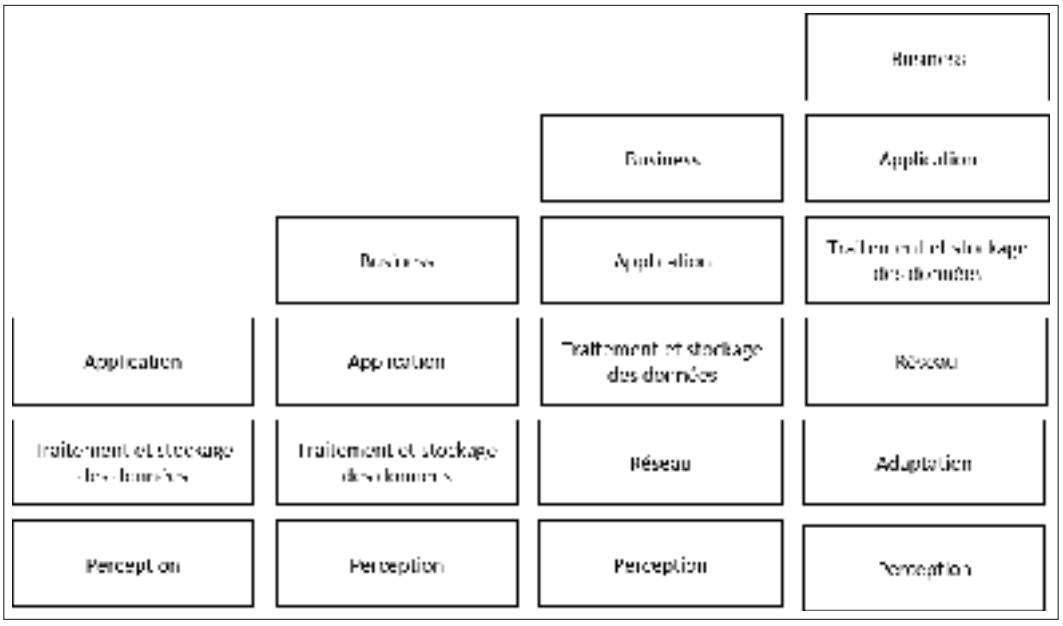


Figure 1.3 Architectures en couches en IoT, adaptée de (Muccini & Moghaddam, 2018).

- perception : représente l'ensemble des capteurs et objets physiques permettant de collecter l'information du monde physique et de l'acheminer vers le monde informatique virtuel ;
- traitement et stockage de données : c'est la couche responsable de l'analyse, le traitement et le stockage des données collectées par les capteurs. Plusieurs techniques sont utilisées, telles que le *Cloud Computing* et le *ubiquitous computing* ;
- application : cette couche est responsable de fournir les différents services demandés par les utilisateurs (les requêtes des utilisateurs) ;
- Business : afin de gérer le système IoT entier, cette couche conçoit des feuilles de route en utilisant les modèles d'affaires relatifs aux domaines d'application ;

- réseau : représente les technologies et les protocoles de transmission des données permettant le transfert de données de la couche perception aux autres couches pour le traitement ;
- adaptation : cette couche se situe entre la couche perception et la couche réseau. Elle permet l'interopérabilité entre les différents objets en IoT.

En plus des architectures en couches, plusieurs autres architectures sont exploitées en IoT :

- architecture orientée service (SOA) : l'architecture SOA est un modèle d'interaction basée sur des services très peu couplés. Un service est fourni par un fournisseur et consommé par un client. Cette interaction se fait à travers un courtier de services qui décrit l'emplacement du service et garantit sa disponibilité. Un consommateur ou client de service demande au courtier de service de localiser un service et de déterminer comment communiquer avec ce service ;
- architecture basée sur le cloud : dans cette architecture, plusieurs fonctionnalités sont déléguées au Cloud, telles que l'analyse des données. Cela permet de traiter et de stocker une plus grande quantité d'information ;
- architecture Fog Computing : c'est un concept relativement récent en IoT. Cette architecture se caractérise par ses différents services pour fournir un système IoT et consiste à amener des services Cloud virtualisés aux extrémités afin de contrôler les appareils IoT ;
- microservices : un microservice est une application qui peut être déployée, mise à l'échelle et testée indépendamment et qui a une seule responsabilité. L'approche de l'architecture microservices consiste donc à utiliser l'architecture SOA et la virtualisation logicielle afin de pallier aux limites du SOA, telles que l'évolutivité ;
- RESTful : une API est RESTful, quand à elle, respecte le principe d'architecture REST. La particularité de cette architecture est que le serveur et le client communiquent sans que le client connaisse la structure et le contenu des informations stockées sur le serveur ;
- Publish/Subscribe : dans le style architectural Pub/Sub, le Publisher publie un message en y incluant un Topic particulier, et le Subsriber le reçoit à condition qu'il ait le même topic.

Généralement, ce processus est assuré grâce à un Broker, qui lui joue le rôle d'un serveur entre le Publisher et le Subscriber (MQTT en est un exemple) ;

- Information Centric Networking (ICN) : ICN fait des données une base de communication des objets. Elle correspond donc au modèle d'application des systèmes IoT et fournit un paradigme de communication efficace et intelligent pour l'IoT.

1.5.2 L'architecture de référence ISO

ISO (2018) propose une architecture de référence en IoT, nommée ISO-RA. Cette architecture est un guide pratique aux architectes voulant développer des applications IoT, afin de les aider à mieux comprendre les systèmes IoT et les différentes parties prenantes (i.e. fabricants des appareils, utilisateurs, développeurs des applications, etc.). ISO présente en premier lieu un modèle conceptuel du système IoT et un ensemble de caractéristiques qu'une application IoT doit respecter. Ces éléments sont utilisés pour définir un modèle de référence de l'IoT et des vues architecturales du système IoT incluant une vue fonctionnelle, une vue de déploiement, une vue communication et une vue d'utilisation.

Le modèle conceptuel (Figure 1.4) est composé d'entités et de domaines. Une entité est tout objet ayant une existence indépendante et distincte, tels que les personnes, les organisations et les objets. Ces entités peuvent être classées en 4 groupes :

- entité physique (Physical Entity) : c'est une entité discrète, observable et identifiable ;
- utilisateur (IoT-user) : c'est une entité pouvant être un humain, ou un non-humain ;
- système TI (ou entité digitale) (Entity) : correspond aux éléments de calcul et d'information, incluant les applications, les services, les entités virtuelles, les dépôts de données, les objets IoT et les passerelles IoT ;
- réseau de communication (Network) : c'est une entité importante de l'IoT permettant aux autres entités de se connecter et d'interagir. Chaque entité voulant se connecter à travers le réseau doit disposer d'un identifiant unique (un ensemble d'attributs qui servent pour

identifiant de l'entité dans un contexte donné). Une entité peut avoir plusieurs identifiants selon le contexte, mais doit au moins en avoir un.

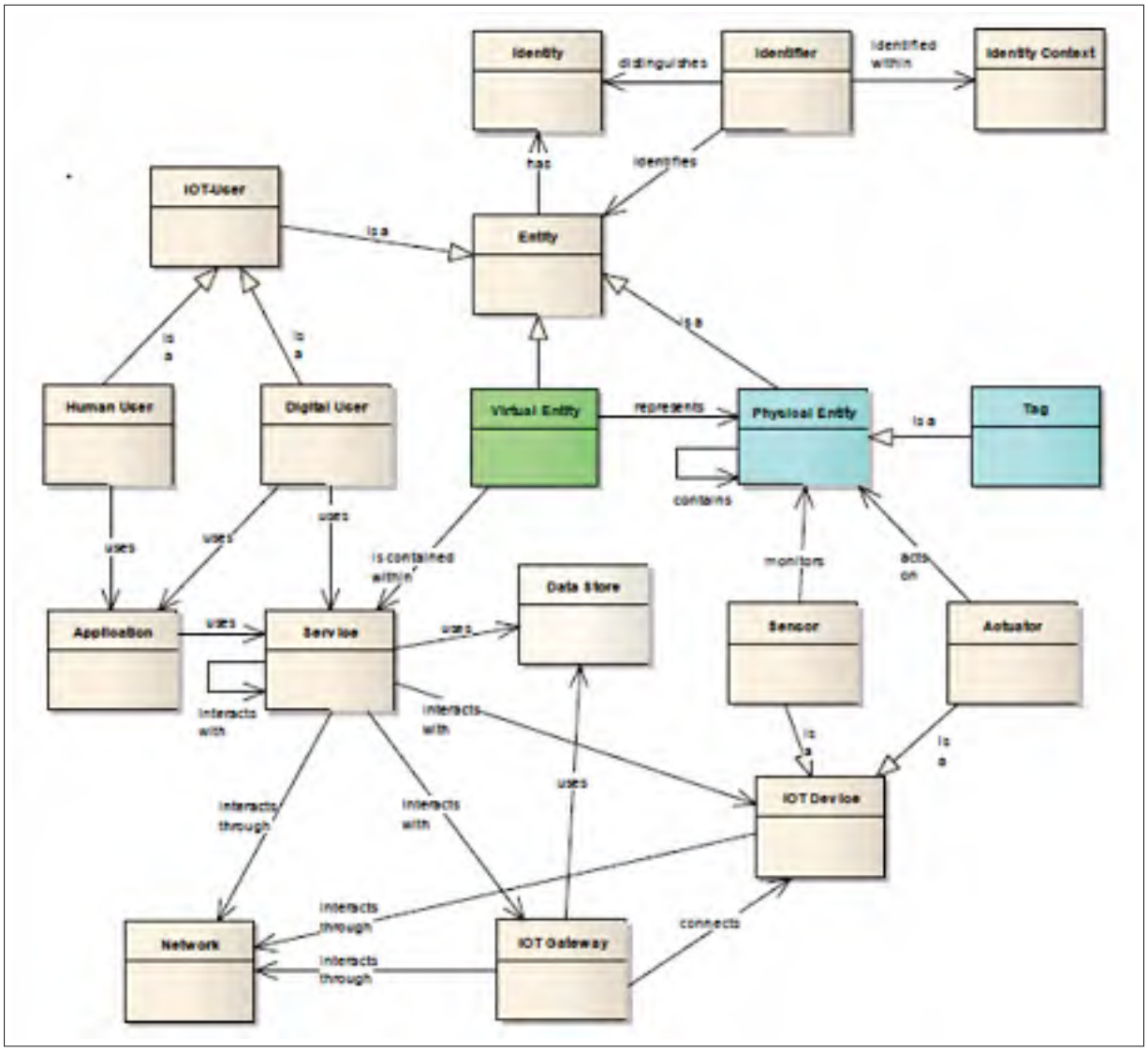


Figure 1.4 Modèle conceptuel de l'IoT, extrait de (ISO, 2018).

La figure 1.5 montre le modèle de référence basé sur les entités du modèle conceptuel. Dans ce modèle de référence, une entité physique (Physical Entity) est physiquement connectée aux objets IoT (IoT Devices) : (capteurs /actuateurs), lesquels sont connectés physiquement à des passerelles IoT (IoT Gateways) et fictivement à un ensemble de sous-systèmes composé de sous-système de gestion et opération (Operation & management Sub-system), sous-système de service et

application (Applicatio & Service Sub-system) et sous-système de communication et accès (Access & Communication Sub-system) . Ces sous-systèmes sont connectés aux utilisateurs IoT (Humains, Objets/IHM). Chacun des Utilisateurs IoT (IoT-Users), les sous-systèmes, les passerelles IoT (IoT Gateways) et les objets IoT (IoT Devices) sont connectés physiquement au réseau. Les passerelles IoT (IoT Gateways) sont couramment utilisées dans les systèmes IoT. Ils forment une connexion entre le ou les réseaux de proximité locaux et le réseau d'accès étendu. Les passerelles IoT peuvent contenir d'autres entités et fournir un plus large éventail de fonctionnalités. Une passerelle IoT contient souvent un agent de gestion fournissant des fonctionnalités de gestion à distance. La passerelle IoT peut contenir un magasin de données, stockant les données des objets IoT associés.

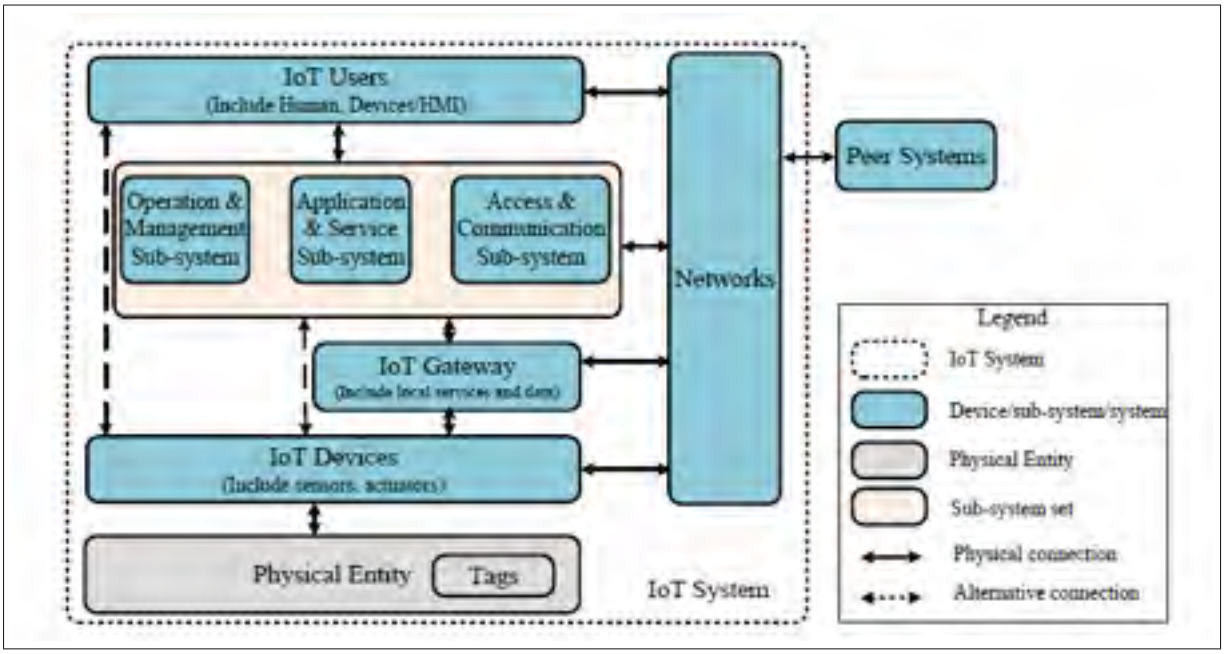


Figure 1.5 Modèle de référence basé sur les entités, extrait de (ISO, 2018).

Les différentes entités IoT appartiennent généralement à un domaine donné. La figure 1.6, montre le modèle de référence basé sur les domaines. Cette représentation a pour but d'aider le concepteur des applications à focaliser sur les différentes tâches devant être réalisées. En d'autres termes, les domaines sont utilisés pour classifier les fonctions par responsabilités. Ces

fonctionnalités vont par la suite être déployées dans différents sous-systèmes. Les domaines

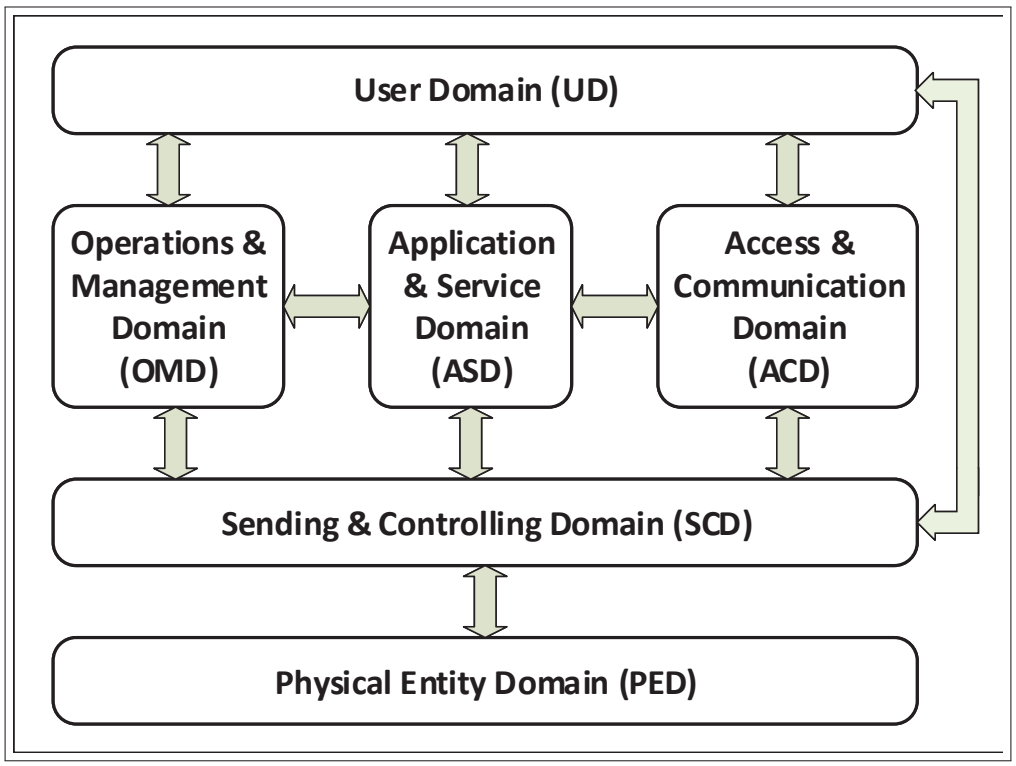


Figure 1.6 Modèle de référence basé sur les domaines, extrait de (ISO, 2018).

identifiés sont :

- domaine utilisateur (User Domain) : l'acteur principal dans ce domaine est l'utilisateur. Les utilisateurs humains interagissent avec les services à travers les utilisateurs virtuels (tels que les ordinateurs, les tablettes, les cellulaires, les panneaux de contrôle) et ce, à travers des interfaces bien définies ;
- domaine de gestion et opération (Operation & Management Domain) : c'est une collection de fonctionnalités responsables du bon fonctionnement de l'application IoT (performance, gestion, optimisation) ;
- domaine de service et application (Application & Service Domain) : les acteurs de ce domaine interagissent avec les différents autres domaines pour réaliser les requêtes des utilisateurs,

obtiennent des informations du domaine de mesure et de contrôle, et communiquent avec les systèmes externes via le domaine de communication et accès ;

- domaine de communication et accès (Access & Communication Domain) : ce domaine offre les mécanismes permettant aux entités externes de se connecter aux ressources du système IoT ;
- domaine de capture (mesure) et de contrôle (Sending & Controlling Domain) : les acteurs sont les capteurs, les actuateurs et les appareils complexes en IoT. Les capteurs surveillent l'état des entités physiques, et les actuateurs agissent. C'est donc un pont reliant le monde physique (le matériel) au monde technologique ;
- domaine des entités physiques (Physical Entity Domain) : c'est le premier responsable de tâches/fonctions telles que la gestion, les mesures, le contrôle.

La figure 1.7 illustre la correspondance entre les deux modèles de références (les entités et les domaines).

1.6 Revue des travaux portant sur les frameworks et plates-formes IoT

Dans cette section, nous présentons les travaux qui se sont penchés sur les frameworks ou les plate-formes IoT. Nous avons divisé ces travaux en deux catégories :

- les travaux qui font un survol des frameworks et plates-formes et les évaluent selon certains critères (section 1.6.1) ;
- les travaux qui font des études comparatives des frameworks et/ou plates-formes (section 1.6.2).

1.6.1 Travaux faisant un survol des frameworks et plates-formes en IoT

Derhamy *et al.* (2015) présentent une revue de 17 frameworks et plates-formes IoT et les évaluent selon différents critères dont : l'approche architecturale, le support de l'industrie, les protocoles et les standards et leur interopérabilité, la sécurité, les exigences matérielles, la gouvernance et

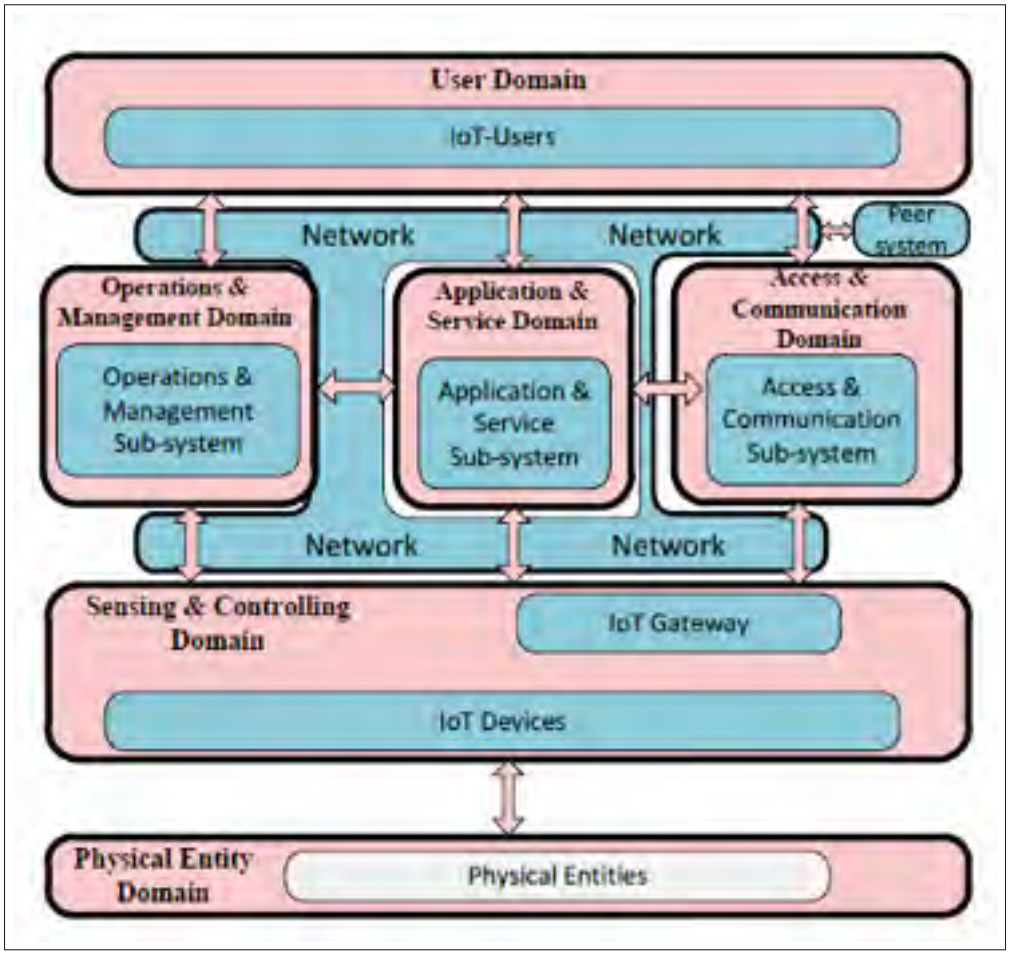


Figure 1.7 Relation entre le modèle de référence basé sur les domaines et le modèle de référence basé sur les entités, extrait de (ISO, 2018).

le support du développement rapide des applications. La notion de framework ici fait référence plutôt à des principes, standards et protocoles. Les auteurs distinguent quatre catégories de frameworks : des frameworks visant l'automatisation des maisons, des frameworks visant le IoT industriel tel que Arrowhead, des frameworks visant le contrôle et la gestion des objets connectés tel que OMA-LWN2M, et les autres frameworks visant d'autres objectifs ou aspects de l'IoT.

L'automatisation des maisons est un domaine clé dans le développement des applications IoT. L'article présente 4 exemples de frameworks appartenant à cette catégorie : 1) IPSO Alliance, qui est un framework IoT développé par IPSO et OMA SpecWorks et qui vise à développer

des objets intelligents, tout en mettant l'accent sur l'identité et la vie privée ; 2) IoTivity, qui est un framework open source développé par Intel et Samsung et qui permet une connectivité transparente de bout en bout pour répondre aux besoins émergents de l'IoT ; 3) AllJoyn, développé par Qualcomm et Sony, est un framework open source collaboratif qui permet aux appareils de communiquer avec d'autres appareils autour d'eux ; et 4) Thread, développé par ARM et Google spécifiquement pour prendre en charge l'IoT en intégrant de nombreuses fonctionnalités qui n'étaient pas disponibles dans les normes précédentes.

Pour ce qui est des plate-formes IoT, Derhamy *et al.* (2015) présentent trois catégories de plates-formes :

- Plates-formes basées sur le cloud : Ces plates-formes permettent de gérer les capteurs suivant une approche basée sur le Cloud. Des exemples de plate-formes dans cette catégorie sont :
  - Cumulocity : c'est une plate-forme IoT de gestion des appareils et des applications qui connecte et gère efficacement les appareils et peut les contrôler à distance ;
  - ThingWorx : c'est une plate-forme pour le développement et le déploiement rapide d'appareils intelligents et connectés. L'ensemble d'outils de développement IoT intégrés prend en charge la connectivité, l'analyse, la production et d'autres aspects du développement IoT ;
  - Xively : c'est une Plate-forme qui permet de créer, gérer et tirer des décisions d'affaire des produits connectés. Il fournit également une API basée sur le cloud avec un SDK qui simplifie le processus de développement.
- Plates-formes de bout en bout : Ce type de plates-formes manipule des objets disposant d'une capacité de calcul leur permettant d'exécuter les opérations du système IoT Des exemples de plate-formes dans cette catégorie sont :
  - IzoT : c'est une famille de puces, de piles, d'interfaces et de logiciels de gestion compatibles avec le protocole IP qui permet le développement d'objets pour l'IoT industriel ;
  - ThingSquare : ThingSquare est une plate-forme commerciale qui permet aux entreprises de déployer rapidement des solutions IoT réussies, du proof-of-concept à la production.

- Plates-formes pour l'intégration du Cloud et des passerelles : Ce type de plates-formes supportent l'intégration sur le cloud, et proposent également des solutions pour l'intégration des "Gateways" IoT. Cependant, ces plates-formes n'offrent pas tous les moyens permettant le développement des applications sur les noeuds finaux. Des exemples de plate-formes dans cette catégorie sont :
  - Intel : en partenariat avec Wind River et McAfee, Intel a proposé plusieurs frameworks IoT pour le développement des applications IoT ;
  - Microsoft : il s'agit de trois familles de produits faisant partie de la solution de Microsoft pour le IoT, Microsoft Azure Cloud, Microsoft StreamInsight et Microsoft Windows Embedded. Chaque produit supporte plusieurs plates-formes ;
  - IBM : il s'agit de plusieurs produits faisant partie de la solution d'IBM pour le IoT tels que IBM WebSphere MQ, IBM BlueMix, IBM MessageSight.

Le tableau 1.5 liste les frameworks, les plate-formes et les protocoles par portée (Cloud global, bout en bout ou cloud local) alors que le tableau 1.6 donne une vue des frameworks et plate-formes par protocole supporté.

Tableau 1.5 Listes des frameworks et plates-formes IoT par espace d'application

Catégories	Frameworks, plates-formes et protocoles IoT
Cloud global	Cumulocity, Xively, ThingWorx, IBM, Microsoft, Intel, LWM2M
Bout en bout	IPSO, Thread, Thingsquare, IzoT, SEP 2.0, AllJoyn, IoTivity
Cloud local	Arrowhead

Le tableau 1.7 présente une vue des frameworks, plate-formes et protocoles par couche. Ce tableau montre que la plupart des frameworks appartiennent à la couche application. Des frameworks tels que Iotivity et AllJoyn supportent une implémentation sur plusieurs couches assurant ainsi une fonctionnalité réduite (limitée) aux objets à capacités limitées. Ceci n'étant pas une règle générale, Cumulocity et ThingWorx ne supportent pas les objets à capacités limitées et se basent plutôt sur des agents intermédiaires ou des passerelles pour intégrer les objets à faibles ressources. En ce qui concerne l'aspect sécurité, IzoT, Iotivity et AllJoyn exigent la prise

Tableau 1.6 Catégorisation des frameworks et plates-formes par protocole supporté

	MQTT	XMPP	CoAP	REST	other
IPSO Alliance			✓	✓	
LWM2M			✓		
Arrowhead	✓	✓	✓	✓	✓
SEP 2.0					✓
AXCIOMA	✓	✓	✓	✓	✓
Thread Group					✓
AllJoyn					✓
ThingSquare			✓	✓	
IzoT				✓	✓
Thing Worx	✓	✓	✓	✓	✓
Xively	✓	✓	✓	✓	✓
Cumulocity				✓	
IBM	✓				✓
Microsoft					✓

Tableau 1.7 Catégorisation des frameworks et plates-formes par couche

Couche	Framework, plate-forme et protocoles IoT
Application	Arrowhead, IPSO Alliance, Xively, Cumulocity, ThingWorx, Smart Energy Profile 2.0 Microsoft, IBM, ThingSquare, Industrial Internet Consortium, AllJoyn, IoTivity, IzoT
Messagerie	Web-Sockets, XMPP, MQTT, CoAP, HART, Thread, AllJoyn
Transport	TCP, UDP, WirelessHART, SMS
Réseau	IP, ZigBee

en charge d'un matériel cryptographique. De son côté, IBM supporte le protocole MQTT avec un serveur dédié utilisé comme un Broker de MQTT.

De façon générale, Derhamy *et al.* (2015) estiment que IBM et Microsoft offrent une meilleure adaptabilité aux besoins des utilisateurs. ThingWorx, Cumulocity et Xively ont démontré leur force dans le développement rapide des applications IoT. Thread, IoTivity et AllJoyn s'adressent principalement aux clients utilisant des objets commerciaux, facilitant ainsi leur déploiement. Quant à Arrowhead, il doit sa force à la re-configurabilité, à travers l'usage des services et systèmes d'orchestration.

D'un point de vue de la gestion et la gouvernance des applications, les plates-formes cloud, telles que Cumulocity, ThingWorx et Xively offrent une excellente gouvernance des applications et une bonne gestion des objets. AllJoyn, IzoT et ThingSquare offrent une bonne gestion des objets, mais ne prennent pas totalement en charge la gouvernance des applications. IBM et Microsoft ont tous les deux une gouvernance et une gestion matures des applications Cloud. Microsoft possède une bonne gestion grâce à sa famille de systèmes d'exploitation ainsi qu'à l'exécution .net intégrée. Arrowhead fournit une gouvernance de configuration et d'autorisation des applications via ses services principaux. Le principal objectif de LWM2M est la gouvernance et la gestion des appareils, à grande échelle pour les opérateurs cellulaires. Cela semble indiquer que les performances seront bonnes pour les réseaux d'objets à grande échelle.

1.6.2 Études comparatives des frameworks et plates-formes en IoT

Très peu de travaux dans le domaine de l'IoT présentent une étude comparative détaillée selon des critères bien établis. Certains travaux comme (Hejazi *et al.*, 2018) se sont plutôt concentrés sur la proposition d'un cadre ou un ensemble de critères pour aider dans le choix d'une plate-forme. Dans cette section, nous discutons uniquement les quelques travaux pertinents qui ont réalisé une comparaison selon les critères proposés.

1.6.2.1 Étude comparative des plates-formes Cloud

Ganguly (2016) propose des critères pour la sélection des plates-formes cloud IoT. Puis, il utilise ces critères pour comparer quelques plates-formes Cloud IoT. Ces critères sont décrits dans le tableau 1.8.

L'auteur analyse les différentes plate-formes par rapport au critère "offres techniques" pour lequel il définit plusieurs sous-critères, dont les suivants :

- domaines pris en charge ou domaines d'utilisation : une plate-forme IoT peut être destinée à un domaine d'application précis, ou peut être généralisée ;

Tableau 1.8 Critères de sélection des plates-formes

Critère	Description
Offres techniques	les fonctionnalités et les aspects non fonctionnels
Stratégie	Vision, temps de mise sur le marché plus rapide, satisfaction client, partenariats
Présence sur le marché	Chiffre d'affaires, clientèle, présence géographique
Conformité et recommandations	Prix gagné, reconnu par une entreprise d'étude de marché tel que Gartner / Forrester, ISO / IEC 27001/27002 : sécurité 2013 certifiée ou similaire, conforme au TR-69 ou similaire, Brevets

- modèles de licence ou de facturation pris en charge : les plates-formes peuvent être open source ou payantes. Les plates-formes payantes peuvent supporter plusieurs formes de paiement. L'offre à payer dépend généralement de la quantité de messages pouvant être envoyée ou la quantité de données supportée ;
- protocoles d'application pris en charge : chaque plate-forme prend en charge un ou plusieurs protocoles d'application. Bien que REST est le plus largement utilisé, le *Websocket* est le meilleur pour la communication en temps réel. MQTT, CoAP et XMPP sont aussi très utilisés. La plate-forme peut offrir une prise en charge d'autres protocoles via l'adaptateur Web ;
- matériel supporté et kits de développement logiciel : la plate-forme peut proposer des accélérateurs pour un développement plus rapide des applications. Les fonctionnalités de gestion telles que la mise à jour du code et les diagnostics sont généralement fournies dans le SDK. La disponibilité de kits de développement logiciel dans plusieurs langages de programmation est nécessaire pour prendre en charge une multitude de plates-formes matérielles telles que Raspberry Pi, TI, Broadcom, Arduino, Atmel, Intel et plein d'autres ;
- formats de sérialisation pris en charge : en IoT, les ressources sont limitées (par exemple, des capteurs avec quelques ko de RAM). Les plates-formes IoT doivent donc prendre en charge des formats de sérialisation avancés pour réduire le volume de trafic. Actuellement, les formats de sérialisation les plus couramment utilisés sont JSON et XML ;
- gestion des objets : un des aspects importants est la capacité de définir un modèle de données d'un objet ; Cet objet peut être simple, par exemple un appareil électroménager,

ou complexe, par exemple une maison ou une ville. L'objet peut exposer une liste de fonctions et de paramètres, permettre d'appeler une fonction en passant un argument et de définir ou d'obtenir la valeur d'un paramètre tel que la température. Les objets peuvent être pré-approvisionnés ou approvisionnés à la demande.

- analytique : une plate-forme peut prendre en charge les analyses en temps réel et hors-ligne sur des données de séries chronologiques, des événements réels, des journaux d'intégrité et des statistiques collectées. Le stockage de données volumineuses est maintenant disponible sur presque toutes les plates-formes ;
- évolutivité : l'évolutivité est l'un des aspects non fonctionnels les plus demandés. La plate-forme devrait être très évolutive, comme le nombre d'objets connectés et le nombre d'utilisateurs sont importants. La plate-forme doit permettres aux développeurs d'applications de configurer la mise à l'échelle automatique ;
- surveillance : la surveillance de l'infrastructure et la visualisation des données de performance du système sont nécessaires. La plate-forme doit enregistrer également la progression des événements entre les services. Des régulateurs doivent mettre fin aux abus des APIs de la plate-forme en définissant des limites de débit par intervalle de temps fixe. Une plate-forme avancée permettrait de configurer ces outils de contrôle ;
- sécurité : la sécurité de la couche de transport est censée être présente par défaut. La sécurité au niveau des messages et le cryptage des données au repos doivent être pris en charge ;
- exigences non fonctionnelles : d'autres exigences non fonctionnelles importantes incluent la disponibilité, la performance, la sauvegarde et la restauration des données et la possibilité de les relier ;
- forum ou communauté : la plate-forme peut avoir des forums gratuits ou payants ou des communautés de développeurs. L'activité du forum et le nombre d'utilisateurs sont deux paramètres importants à vérifier, surtout si le forum est gratuit.

Ganguly (2016) utilise sa liste de critères pour comparer 10 plate-formes, à savoir, Kaa, DeviceHive, Particle, TheThings, Aeris, Ayla Networks, Bluemix, Xively, Things Works et Temboo. Le choix de ces plates-formes s'est basé sur les facteurs suivants :

- répond aux exigences de 60% et plus (environ 12 sur 20 des critères techniques) ;
- au moins une entrée satisfaisante sous le critère "Conformité et Recommandations", en particulier pour les plates-formes payantes ;
- les plates-formes en phase bêta, par exemple AWS, sont ignorées. En outre, quelques plates-formes de géants du logiciel sont également ignorées.

Les résultats de cette étude montrent que les plate-formes sont très différentes, et que plusieurs de ces plate-formes ne satisfont pas plusieurs des critères considérés. L'auteur souligne donc un manque de maturité des plate-formes étudiées.

1.6.2.2 Étude comparative des frameworks IoT basée sur des aspects architecturaux

Rahman *et al.* (2016) ont développé une taxonomie qui classifie les frameworks selon leurs architectures. Pour ce faire, ils ont étudié un ensemble de frameworks et comparé leurs architectures. Les similarités entre les différentes architectures sont retenues comme étant des éléments de la taxonomie. Ensuite, les différences des architectures sont étudiées et, si l'aspect est décisif dans le choix de l'architecture, alors celui-ci est ajouté dans la taxonomie. Ce processus a permis d'identifier les aspects architecturaux qu'un framework IoT doit exhiber. Ensuite, ils ont appliqué cette taxonomie sur 3 frameworks IoT, à savoir, Works de Nest, la plate-forme ARM mbed IoT Device, et AllJoyn. Les auteurs justifient le choix de l'aspect architectural par le fait qu'il influe considérablement les exigences non fonctionnelles telles que la latence, la confidentialité, la sécurité, l'évolutivité et l'interopérabilité.

La taxonomie proposée se compose de 4 critères à savoir, le déploiement du contrôle, le déploiement de la gestion des objets, le déploiement du stockage et le protocole d'application utilisé par le framework. Les 3 premiers aspects concernent le déploiement, qui consiste à mapper entre les fonctionnalités d'un système IoT et les différents composants (éléments) du système. Les auteurs distinguent trois classes de déploiement :

- déploiement sur le Cloud : dans cette classe, les tâches sont assignées au Cloud ;

- déploiement Fog : dans cette classe, les tâches sont assignées à un objet Fog (une passerelle ou un objet embarqué) ;
- déploiement Mist : dans cette classe, les tâches sont assignées aux noeuds finaux (objets connectés).

Cette taxonomie a été utilisée pour étudier et classifier les 3 frameworks choisis. Les résultats de cette étude montrent que le choix de la classe de déploiement affecte la latence du système, la sécurité, le SPOF (Single Point of Failure), le degré de complexité des décisions possibles et le degré de complexité supporté. De façon générale, l'utilisation du cloud ajoute de la latence puisque les données sont acheminées via Internet. Cependant, l'utilisation du cloud permet de supporter une complexité décisionnelle assez élevée puisque le cloud permet d'effectuer des calculs complexes. Le déploiement Fog et Cloud introduisent le SPOF, puisque le contôle est plus centralisé que le déploiement Mist. Aussi, le déploiement Cloud diminue la sécurité, puisque les fonctions de contrôle ou de gestion sont déléguées au Cloud, qui est plus vulnérable. Cloud et Fog supportent la mise à l'échelle, puisqu'ils reposent sur l'architecture Pub/Sub, contrairement à Mist, qui repose sur la communication directe entre les noeuds finaux (Rahman *et al.*, 2016).

Finalement, les résultats de cette étude valident l'utilité de la taxonomie dans le choix du framework approprié selon le type d'application à développer.

1.6.2.3 Cadre d'évaluation des plates-formes IoT

(Salami & Yari, 2018) définissent une plate-forme comme un middleware et une infrastructure permettant aux utilisateurs de se connecter aux objets intelligents. Dans ce contexe, un middleware est un mécanisme qui associe différents composants de systèmes IoT et offre une communication fluide entre les objets et les composants système. C'est une interface qui facilite l'interaction entre l'internet et les objets ; un objet peut être physique (matériel) ou une application (logiciel). Le middleware permet de pallier à cette hétérogénéité. Selon (Salami & Yari, 2018), une

plate-forme est organisée en 8 modules, tel que présenté dans la figure 1.8. Le tableau 1.9 décrit les fonctionnalités des principaux modules d'une plate-forme.

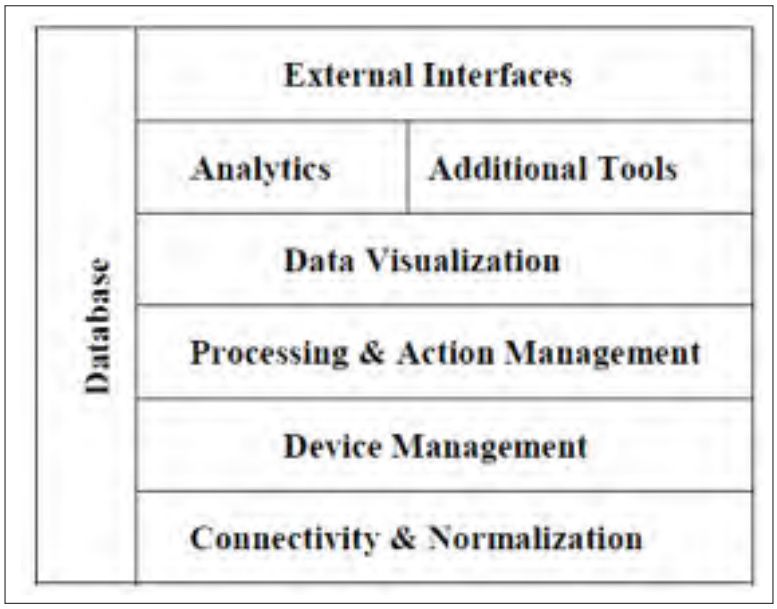


Figure 1.8 Les modules d'une plate-forme IoT, extrait de (Salami & Yari, 2018).

Tableau 1.9 Description des rôles des modules principaux d'une plate-forme IoT

Module	Rôle
Couche de connectivité (Connectivity & Normalization)	Adapter les différents protocoles et réseaux à une seule interface logicielle
Gestion des objets (Device management)	S'assurer que les objets connectés sont fonctionnels et que les applications et les logiciels sont mis à jour
Gestion des actions et des traitements (Processing & Action Management)	Les données mesurées par le module de connectivité sont stockées dans un reposoir de données et par la suite, traitées dans la couche de gestion des actions et des traitements
Visualisation des données (Data Visualization)	Le module de visualisation des données est responsable de la représentation graphique en temps réel des données capturées
Analytique (Analytics)	Plusieurs applications IoT ont besoin d'analyser les données collectées pour supporter une gestion optimale

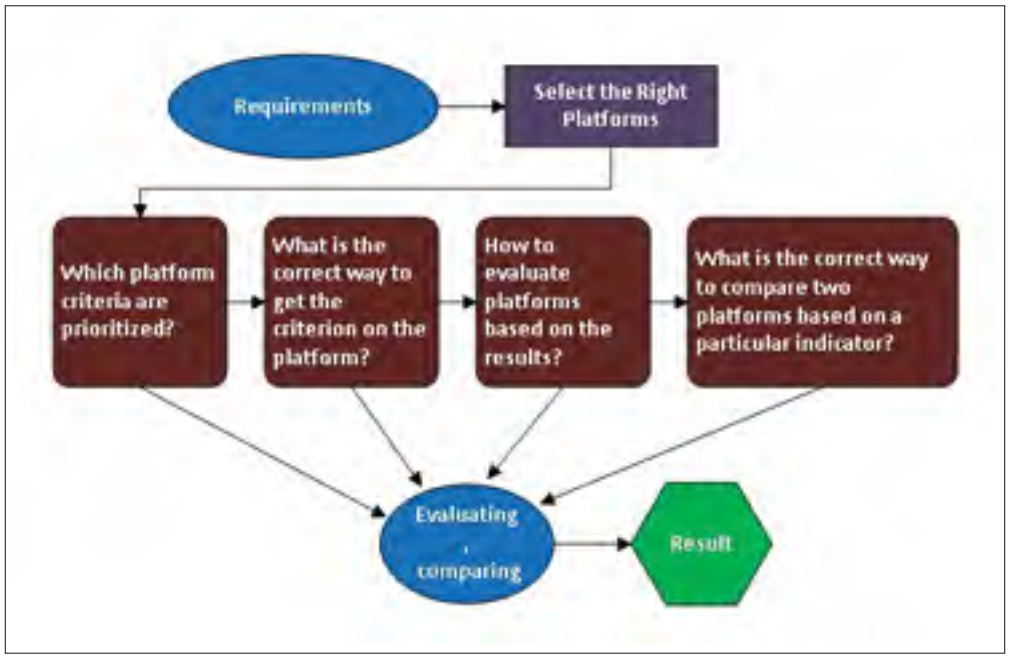


Figure 1.9 Processus d'évaluation des plates-formes IoT, extrait de (Salami & Yari, 2018).

Pour comparer des plate-formes, Salami & Yari (2018) ont proposé un cadre de comparaison. Ce cadre de comparaison comprend un processus à suivre pour évaluer chaque framework. Ce processus est décrit à la figure 1.9.

Ce processus se base sur un ensemble de critères organisés en 5 classes de critères (telque montré par la figure 1.10) :

- sécurité : comprend, entre autres, l'authentification, la confidentialité et la protection des données ;
- utilisations : comprend, entre autres, l'accesibilité, l'utilisabilité et la correction ;
- données : comprend, entre autres, le stockage, le traitement et la visualtion des données ;
- communication : comprend les interfaces, les APIs et le support pour l'hétérogénéité ;
- plate-forme : comprend, entre autres, la testabilité, la mis eà l'écelle, la protabilité et la flexibilité.

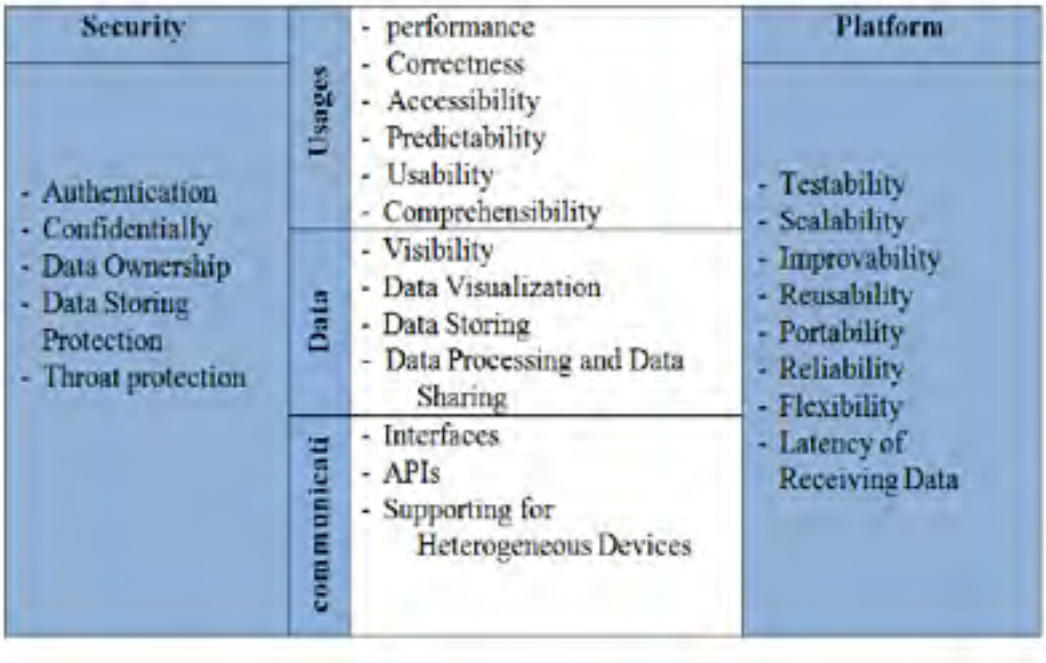


Figure 1.10 Critères de comparaison des plates-formes IoT, extrait de (Salami & Yari, 2018).

(Salami & Yari, 2018) ont utilisé leur cadre pour évaluer trois plates-formes IoT : ThingSpeak, Xively et AWS IoT. En particulier, ils se sont concentrés sur la fiabilité, la mise à l'échelle et la latence de ces plate-formes. Les résultats ont démontré que la plate-forme AWS IoT détient le meilleur score en terme de mise à l'échelle, de fiabilité et de latence.

1.7 Conclusion

L'évaluation des frameworks IoT est très importante, notamment vu le nombre élevé de frameworks et le rôle important qu'ils jouent dans le processus de développement des applications IoT. Cependant, nous avons trouvé très peu de travaux qui évaluent et comparent des frameworks en IoT. Parmi ces travaux, quelques uns se sont plus concentrés sur l'identification de critères d'évaluation que la comparaison elle-même (e.g. (Hejazi *et al.*, 2018) et (Salami & Yari, 2018)). Certains travaux ((Ganguly, 2016), (Hejazi *et al.*, 2018) et (Salami & Yari, 2018)) se sont intéressés aux exigences non fonctionnelles de façon générale alors que d'autres ((Rahman *et al.*,

2016) et (Muccini & Moghaddam, 2018)) proposent de comparer les frameworks d'un point de vue architectural et soulignent l'aspect réseau (i.e. protocoles de communication supportés).

Bien que certains travaux ((Ganguly, 2016), (Hejazi *et al.*, 2018) et (Salami & Yari, 2018)) survolent les exigences fonctionnelles en IoT, et en utilisent quelques-uns pour l'évaluation et la comparaison des frameworks, aucun des travaux proposés ne traite le sujet d'un point de vue développement logiciel. En d'autres termes, aucun travail n'a évalué les frameworks avec la perspective de voir comment ils supportent le développement d'une application IoT concrète. En fait, les travaux existants réalisent très peu (ou pas du tout) d'expérimentations avec les frameworks étudiés.

Ces observations ont motivé notre travail de recherche lequel consiste à étudier et évaluer à travers des expérimentations des frameworks IoT supportant le développement logiciel, et cela à la lumière d'un ensemble minimal d'exigences des applications IoT. Contrairement aux travaux existants, notre étude comprend donc deux volets : 1) une évaluation théorique basée sur l'ensemble des informations disponibles pour le framework analysé ; et 2) une évaluation expérimentale où nous implémentons un ensemble d'applications IoT en utilisant chacun des frameworks étudié et en déployant ces applications sur une plate-forme matérielle populaire (i.e. une Raspberry Pi).

CHAPITRE 2

DÉFINITION DE L'ÉTUDE EXPÉRIMENTALE

Dans ce chapitre, nous présentons l'étape de définition de notre étude expérimentale. En particulier, nous présentons les questions de recherche visées par notre étude et le processus expérimental que nous avons suivi pour y répondre. Nous présentons également les différents choix que nous avons faits reliés à notre montage expérimental, à savoir : les frameworks IoT évalués, la plate-forme matérielle de déploiment, les catégories d'applications IoT que nous avons implémentées dans notre expérimentaion, ainsi que les critères de comparaison que nous avons utilisés pour comparer les frameworks.

2.1 Questions de recherche

Notre étude a pour objectif d'étudier le support fourni par les frameworks open source pour développer des applications IoT. Particulièrement, cette étude vise à répondre aux questions de recherche suivantes :

- (QR1) : Jusqu'à quel degré un framework open source supporte le développement des applications IoT ?
- (QR2) : Jusqu'à quel degré un framework open source supporte un ensemble minimal d'exigences fonctionnelles et non fonctionnelles des applications IoT ?

Pour répondre à la question (QR1), nous avons sélectionné un ensemble de framworks open source, lesquels nous avons utilisés pour développer des applications IoT. Afin de cibler un large spectre d'applications IoT, nous avons étudié les différentes catégories d'applications IoT discutées dans la littérature. Au vue des résultats de cette étude, nous avons adopté une catégorisation de ces applications en 3 catégories, selon leur complexité et leur portée. Nous avons par la suite choisi d'implémenter et de déployer une application de chaque catégorie en utilisant chacun des frameworks séléctionnés.

En ce qui concerne la deuxième question de recherche (QR2), nous avons analysé les frameworks sélectionnés selon un ensemble de critères correspondant à un ensemble minimal d'exigences des applications IoT. Pour identifier cet ensemble de critères, nous avons procédé en 2 étapes : tout d'abord, nous avons analysé les travaux qui survolent les applications IoT pour construire une liste initiale de critères. Ensuite, nous avons complété cette liste en analysant le standard ISO-RA qui propose une liste d'exigences et une architecture de référence pour l'IoT.

2.2 Choix des frameworks

Pour notre étude, nous avons sélectionné un échantillon de cinq framework de développement IoT open source, à savoir Eclipse Vorto (Vorto, 2016), ThingML (F. Fleurey & community, 2016), OpenHab (OpenHAB & the openHAB Foundation e.V., 2019), Eclipse Kura (Kura, 2019) et Node-Red (OpenJSFoundation, 2019).

À l'exception de Node-Red, ces frameworks sont tous basés sur l'environnement de développement Eclipse. Au départ, nous avons concentré notre étude sur les frameworks qui faisaient partie du projet Eclipse IoT, à savoir Eclipse Vorto, OpenHab et Eclipse Kura. Pour diversifier les frameworks étudiés, nous avons également étudié ThingML qui est un plugin Eclipse et Node-Red qui est un outil de programmation très populaire qui facilite l'interconnexion des objets matériels. Chacun de ces frameworks est décrit plus en détail dans le troisième chapitre.

Nous avons utilisé la version v0.10.0 M6 de Vorto, la version v1.0.0.4 de ThingML, la version v0.19.5 de Node-Red, la version 2.4 d'OpenHab et la version 4.0.0 d'Eclipse Kura.

2.3 Choix des applications IoT à implémenter

Pour expérimenter sur un échantillon représentatif d'applications IoT, nous nous appuyons sur la classification de base des applications IoT existantes qui a été introduite dans (Zhang, 2011). Ainsi, en fonction de leur complexité, nous distinguons trois catégories d'applications IoT :

- Catégorie 1 - Applications IoT pour l'identification/localisation des objets : Ces applications visent à localiser des objets à travers des technologies d'identification et de "tracking" (e.g. tags RFID). L'objectif est de gérer efficacement les objets identifiés et tracés ; les objets pouvant être des équipements, des produits ou autres. Un exemple typique de cette catégorie est une application de gestion de l'inventaire des produits d'un magasin.
- Catégorie 2 - Applications pour le suivi d'état des objets et les variables d'environnement en temps réel : Ces applications permettent de surveiller l'état de certains objets. Des exemples typiques de telles applications sont les systèmes de surveillance météorologique et les systèmes de surveillance de congestion routière. Ces applications utilisent divers capteurs (par exemple, thermomètre,caméra, GPS,etc.) pour capturer l'état des appareils surveillés.
- Catégorie 3 - Applications pour le contrôle et la gestion des systèmes : Dans cette catégorie, une application mesure l'état de certains objets et réagit en fonction de ces mesures. Ainsi, ces applications utilisent plusieurs capteurs pour capturer l'état des objets, mais utilisent également des actuateurs pour transformer les décisions prises en actions physiques (par exemple, mettre le chauffage à ON, déplacer un piston,etc.). Un exemple simple de ces applications est un système de chauffage intelligent.

Nous avons choisi d'implémenter une application de chaque catégorie en utilisant chacun des frameworks étudiés. Le tableau 2.1 liste les exemples des applications que nous avons choisi par catégorie.

Tableau 2.1 Les applications IoT sélectionnées par catégorie

Catégorie	Application déployée
Identification des objets	Système de gestion d'inventaire
Suivi en temps-réel des états des objets	Système de surveillance météorologique
Contrôle des états des objets	Système de chauffage intelligent

2.4 Choix de la plate-forme matérielle

Pour notre étude, nous avons utilisé la plate-forme Raspberry Pi (Raspberry-Pi-foundation, 2019) qui est un appareil à faible coût. La Raspberry Pi a été communément utilisée pour apprendre la

programmation, la conception et la construction de diverses applications. Principalement, la Raspberry Pi a été utilisée dans la domotique, la signalisation routière automatisée, la surveillance et l'agriculture intelligente. La Raspberry Pi peut être considérée comme un ordinateur embarqué qui exécute un système d'exploitation et fournit un ensemble de Pins pour communiquer avec d'autres appareils. Ces Pins sont appelés GPIO (General Purpose Input/Output). Les PINs GPIO sont le moyen de contrôler et d'interagir avec les appareils IoT.

Plusieurs modèles de Raspberry Pi ont été publiés par la fondation Raspberry Pi (Raspberry-Pi-foundation, 2020). Nous avons utilisé la Raspberry Pi 3 modèle B + pour nos expériences. Le modèle B + possède un processeur quad-core 64 bits fonctionnant à 1,4 GHz. Il possède également 1 Go de mémoire SDRAM LPDDR2, 40 PINs GPIO et 4 ports USB 2.0. (voir annexe 1 pour plus de détails).

Pour implémenter nos trois exemples d'applications IoT, nous avons utilisé des appareils supplémentaires. Pour le système de gestion d'inventaire, nous avons utilisé des tags RFID et un lecteur de tags RFID pour identifier et suivre les objets. Le lecteur de tags a été branché sur la Raspberry Pi via un port USB (voir figure 2.1.a). Pour le système de surveillance météorologique, nous avons utilisé un capteur de température et d'humidité de base. Plus précisément, nous avons utilisé le capteur DHT11 qui est un capteur populaire, à coût réduit et qui peut être facilement connecté à la Raspberry Pi (voir figure 2.1.b). La fréquence d'échantillonnage du DHT11 est de 1 Hz, c'est-à-dire que le capteur effectue une lecture toutes les secondes. Pour le système de chauffage intelligent (voir figure 2.1.c), nous avons utilisé le capteur DHT11 pour collecter la température, et une LED pour simuler le chauffage ; c'est-à-dire que la LED est allumée ou éteinte en fonction de la différence entre la température ambiante (détectée par le capteur) et la température cible.

2.5 Caractéristiques des applications IoT

Afin de bâtir notre liste des exigences minimales, nous avons tout d'abord passé en revue les différents travaux qui survolent des applications IoT. Nous avons par la suite raffiné notre liste

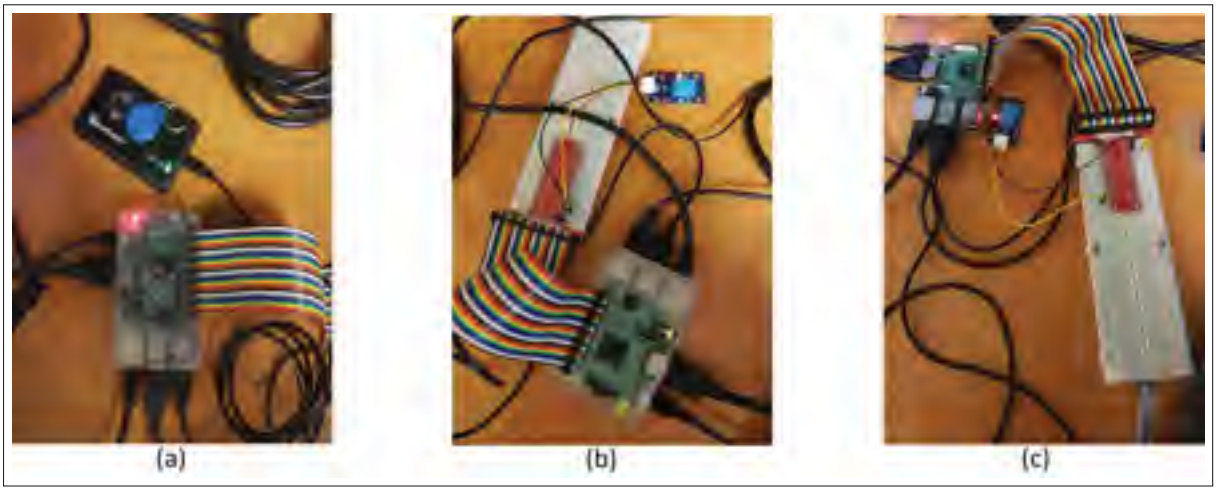


Figure 2.1 Montage expérimental

en nous basant sur la liste des critères présentée dans le standard ISO-RA IoT. Le travail de synthèse que nous avons fait a consisté à :

- ajouter des critères relatifs à la plate-forme matérielle choisie (Raspberry Pi) ;
- ajouter des critères relatifs à la fluidité du framework (les fonctionnalités qu'il peut offrir aux utilisateurs/développeurs afin de rendre leurs tâches plus faciles et automatisées) ;
- regrouper les critères reliés en un critère plus général ;
- détailler les critères génériques en sous-critères, plus ciblés et détaillés ;
- éliminer tous les critères qui ont trait aux législations, lois, ou aspects juridiques ;
- éliminer les critères relatifs aux réseaux (par exemple, protocoles de communication, qualité de service) vu que c'est hors de portée de notre étude. En effet, tout au long du projet, nous avons adopté MQTT comme protocole de communication et nous avons ainsi éliminé le facteur réseau de notre évaluation.

La liste finale des exigences que nous avons adoptées est la suivante :

- spécification de l'interface de l'objet : la capacité du framework à supporter le développeur dans la définition des entrées / sorties des objets ;

- spécification du comportement de l'objet : la capacité du framework à supporter le développeur dans la modélisation du comportement des objets ;
- spécification des propriétés de l'objet : la capacité du framework à supporter le développeur dans la définition des caractéristiques matérielles de l'objet (par exemple, la mémoire) ;
- sécurité : le support fourni par le framework au développeur pour créer des applications sécurisées. En particulier, nous focalisons l'étude sur les problèmes de sécurité communs, à savoir l'authentification, le cryptage et l'intégrité des données ;
- gestion de l'hétérogénéité : la capacité de déployer des systèmes d'appareils hétérogènes connectés ;
- tolérance aux fautes : il s'agit de la capacité du framework à : 1) prendre en charge la spécification des erreurs, et 2) permettre leur détection lors de l'exécution de l'application ;
- intégration des composants COTS : la capacité du framework à prendre en charge l'intégration des composants disponibles sur les reposoirs publics (par exemple Eclipse Marketplace) ;
- découvrabilité : la capacité du framework à faciliter l'ajout de nouveaux appareils ou services à une application IoT existante ; c'est-à-dire permettre l'identification et la description d'un nouvel objet de manière à ce qu'il puisse être découvert par les appareils existants ;
- génération d'un code source complet : la possibilité de prendre en charge la génération d'un code complet et prêt à être exécuté ;
- stockage des données : la capacité du framework à prendre en charge le stockage de données (par exemple, les valeurs capturées, les fichiers logs) ;
- visualisation : la capacité du framework à prendre en charge la visualisation des objets connectés, c'est-à-dire l'affichage de l'état des objets ou des données échangées entre les objets ;
- simulation : la capacité du framework à simuler le fonctionnement de l'application IoT afin d'évaluer l'application par rapport à ses exigences ;

Comme nous avons choisi d'utiliser la Raspberry Pi comme plate-forme matérielle, nous avons défini deux critères supplémentaires liés au déploiement et à l'exécution du code source généré par le framework sur la Raspberry Pi :

- facilité de déploiement sur la Raspberry Pi : la capacité de déployer facilement le code développé à l'aide d'un framework sur la Raspberry Pi ;
- exécution sur la Raspberry Pi : la capacité d'exécuter le code sur la Raspberry.

CHAPITRE 3

RÉALISATION DES EXPÉRIMENTATIONS ET COLLECTE DE DONNÉES

Dans ce chapitre, nous décrivons les expérimentations que nous avons faites avec les frameworks IoT ciblés par notre étude. Pour chaque framework, nous commençons par donner une description générale de son fonctionnement et, ensuite, nous décrivons comment nous avons implémenté chacune des trois applications IoT en utilisant le framework.

3.1 Eclipse Vorto

3.1.1 Présentation générale du framework

Vorto (Vorto, 2016) est un projet open source faisant partie de l'ensemble des projets d'Eclipse IoT. Il permet : 1) la description des objets grâce à un DSL (Domains Specific Language) inspiré des langages de programmation orientés objet tels que JAVA, 2) le partage de la description des objets dans un reposoir Vorto, et 3) la génération d'un squelette de code. Vorto facilite aussi l'intégration de ces objets (à travers le code généré) dans des plates-formes IoT assurant l'interopérabilité entre les objets et les plates-formes IoT quelques soient les technologies utilisées.

Eclipse Vorto offre un environnement composé de 3 éléments essentiels, tel qu'illustré par la partie grise de la figure 3.1 :

1. Un ensemble d'outils Vorto, composé principalement d'un DSL et d'un ensemble de générateurs de code :
   - DSL de Vorto : Spécialement conçu pour être utilisable même par des non-développeurs, le DSL de Vorto est facile à lire et à comprendre. Il est utilisé pour créer des modèles abstraits décrivant les différents objets d'une application et leurs fonctionnalités ;
   - des générateurs de code : basés sur le DSL et les modèles décrivant les objets, les générateurs de code fournissent un moyen sophistiqué, mais simple de créer du code

source pour une intégration des appareils IoT définis avec une plate-forme de solution IoT.

2. Un méta Information Model : C'est le méta-modèle permettant de décrire les objets d'une application en termes de propriétés, fonctions et relations entre objets. Il se base sur des concepts tels que les *InformationModel*, les *FunctionBlock* et les *Datatypes* décrits plus en détails ci-après.

3. Un reposoir de modèles : Le reposoir permet de stocker, gérer et partager les modèles décrivant les objets ; i.e. *InformationModels*.

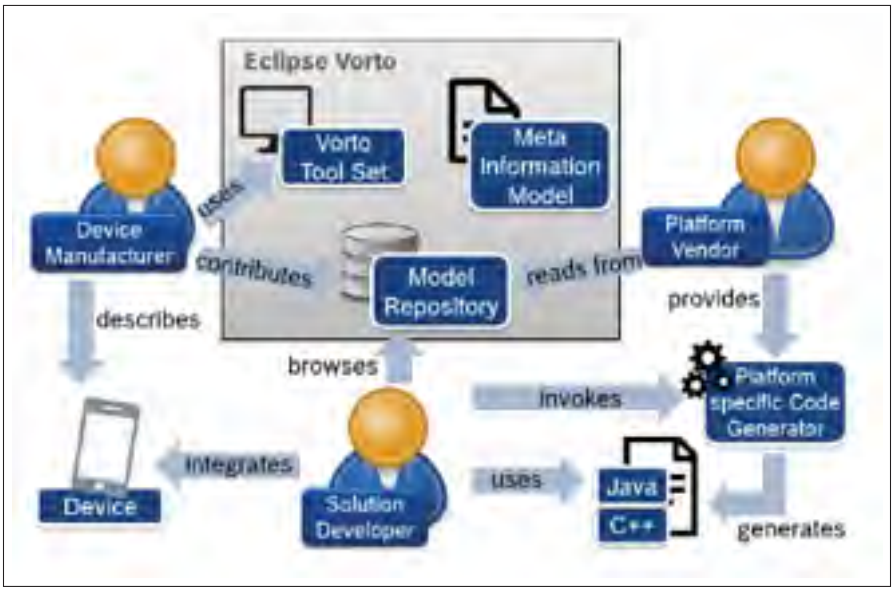


Figure 3.1 Les différents éléments d'Eclipse Vorto et leurs interactions avec l'écosystème IoT, extrait de (Wagner *et al.*, 2016).

Définir un objet dans Vorto revient donc à créer une instance du *Meta Information Model*, d'où son nom Information Model (ou *InfoModel*). Ce processus est supporté par *Vorto Tool Set*. Le modèle, une fois décrit et partagé, peut être téléchargé sur le *Model Repository* et réutilisé par l'ensemble de la communauté Vorto (le reposoir est public et accessible). Ces étapes constituent l'ensemble des activités de modélisation d'un objet en Vorto. 3 intervenants peuvent éventuellement interagir avec l'environnement de Vorto (illustrés par des bonhommes dans la figures 3.1) :

- le *Device Manufacturer* peut définir ses objets en suivant les mêmes étapes ;
- le *Platform Vendor* peut créer lui-même ses générateurs de code lui permettant d'intégrer les différents objets dans sa plate-forme ;
- le *Solution Développer* peut utiliser les différentes descriptions d'objets ainsi que les générateurs de code pour concevoir et implémenter sa propre application IoT.

Les éléments composant le DSL de Vorto sont illustrés dans la figure 3.2 ((Laverman, Grewe, Weinmann, Wagner & Schildt, 2016) et (Wagner *et al.*, 2016)).

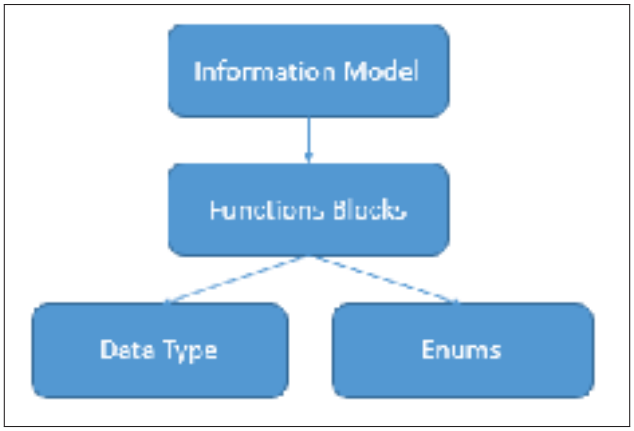


Figure 3.2 Les composants du DSL de Vorto.

L'InformationModel décrit les objets comme étant un ensemble de FonctionBlocks. Ces derniers représentent une vue abstraite des fonctionnalités de l'objet. Une FunctionBlock peut inclure les éléments suivants :

- *Statut* : Le statut actuel de l'objet ;
- *Configuration* : les propriétés d'un objet pouvant être paramétrées ;
- *Fault* : les éventuelles erreurs et défauts du système ;
- *Event* : les événements publiés par l'objet pour informer de son état actuel ;
- *Opérations* : les fonctions qu'un objet supporte pour changer son état ou ajouter une information.

Les variables utilisées dans les *FunctionBlocks* peuvent être des données d'un certain type (i.e. *DataType*) ou des énumérations (i.e. *Enums*).

Prenons le cas d'une caméra de surveillance, la caméra elle-même sera décrite dans Vorto par un Information model dont les *FonctionBlocks* sont, par exemple, la détection de mouvement, la prise de photos et l'enregistrement vidéo. Pour le cas de la *FonctionBlock* "détection de mouvement", les éléments définis comprennent un *Status*, une *Configuration*, des erreurs potentielle définies dans *Fault* et des *Operations*. Dans le cas du *status*, les variables définies sont :

- *Lastdetected* : enregistre le dernier instant où un mouvement a été détecté ;
- *Present* : se met à True si un mouvement est en train d'avoir lieu et est détecté à l'instant même ;
- *PowerConsumption* : Valeur représentant l'énergie consommée.

Concernant les *Configurations*, les variables sont :

- *SensorLocationx* : Localisation du capteur sur l'axe des x ;
- *SensorLocationy* : Localisation du capteur sur l'axe des y.

Les éventuelles erreurs pouvant avoir lieu sont définies dans Fault comme suit :

- *StateError* : utilise les deux valeurs *PowerConsumption* et *Present* pour détecter les éventuelles anomalies du système (*PowerConsumption* à 0 et *Present* à True donc il y a une erreur !) ;
- *Statechange* : utilise la variable *Present* pour décider si un événement est en train d'avoir lieu (si la variable change d'état).

Finalement, une méthode *SendNotification* est définie dans *Operations* qui permet d'envoyer une notification aux utilisateurs ou aux autres objets du système lorsqu'un mouvement est détecté.

Il est à noter qu'Eclipse Vorto ne génère pas un code complet. Il permet simplement la génération d'un squelette de code, et ce, grâce à des générateurs de code conçus pour des plates-formes spécifiques tels que Bosch IoT Suite, Eclipse Ditto et Eclipse Hono. Les générateurs de code

Eclipse Hono et Bosch IoT Suite, génèrent du code Java, Python ou Arduino alors que Eclipse Ditto génère des fichiers Json.

3.1.2 Implémentation du système de gestion d'inventaire

Pour le déploiement de la première catégorie avec Vorto, nous avons défini le lecteur de tags comme étant un *InfoModel* (*RFIDreader*, présenté en haut de la figure 3.3), et sa capacité de lire les identifiants des tags comme étant un *FunctionBlock* (*"Reading"*, présenté en bas de la figure 3.3). Dans le *FunctionBlock Reading*, nous avons défini une variable booléenne ON, qui se met à vrai dès que le tag s'approche pour permettre au lecteur de tags de lire son identifiant. La méthode *writeIDifON()* affiche l'identifiant du tag (variable ID).

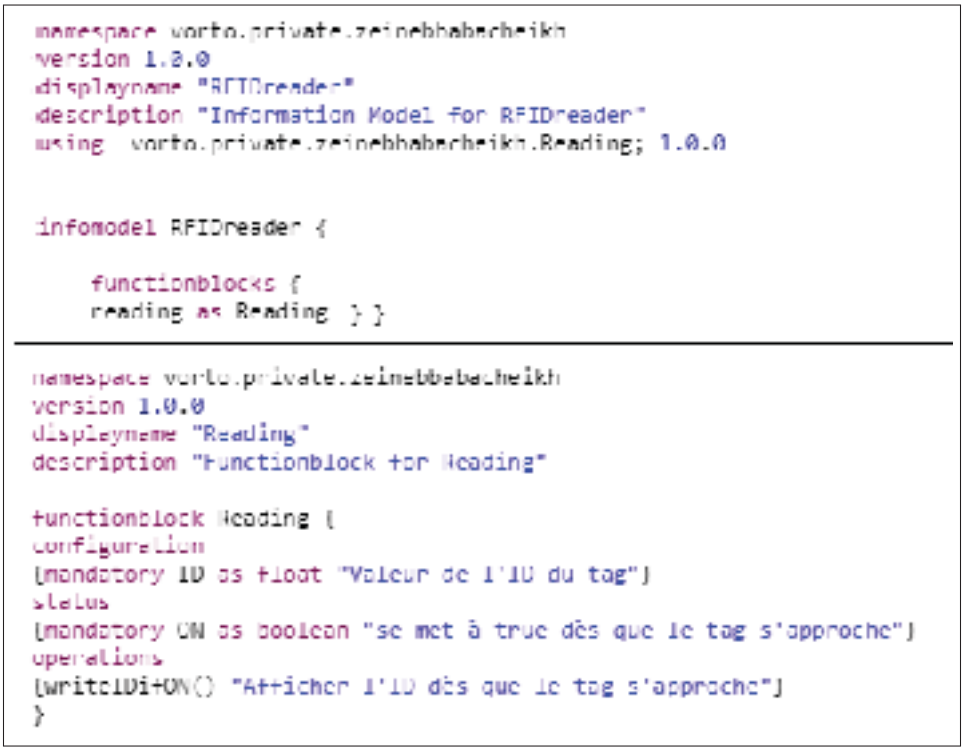


Figure 3.3 InformationModel et FunctionBlock relatifs au lecteur de tags RFID

Ensuite, nous avons généré le code Java de l'application en utilisant l'un des générateurs de code fournis par Vorto. Le code généré par Vorto est un squelette de code ; qui doit être complété manuellement avant d'être déployé et exécuté sur la Raspberry Pi.

3.1.3 Implémentation du système de surveillance météorologique

Comme mentionné au chapitre 2, nous avons utilisé le capteur DHT11 pour implémenter le système de surveillance météorologique. Avec Vorto, nous avons décrit le capteur DHT11 en utilisant un *InfoModel* affiché en haut de la figure 3.4 et sa capacité à mesurer la température comme un *FunctionBlock* affiché en bas de la figure 3.4.

```
namespace vorto.private.zeinebbabacheikh
version 1.0.0
displayname "SensorTemperature"
description "Information Model for SensorTemperature"
using vorto.private.zeinebbabacheikh.Sensor; 1.0.0

infomodel SensorTemperature {

    functionblocks {
    sensor as Sensor } }
```

```
namespace vorto.private.zeinebbabacheikh
version 1.0.0
displayname "Sensing"
description "FunctionBlock for Sensing"

functionblock Sensing {
status{
mandatory sensorValue as float with [readable : true, writable : false]
"Last or current  Measured value from the sensor"
optional minMeasuredValue as float with {readable : true, writable : false}
"the minimum value Measured by the sensor since power ON or reset"
optional maxMeasuredValue as float with {readable : true, writable : false}
"the maximum value Measured by the sensor since power ON or reset"
optional minRangeValue as float with [readable : true, writable : false]
"the minimum value that can be measured by the sensor"
optional maxRangeValue as float with {readable : true, writable : false}
"the maximum value that can be measured by the sensor"
optional sensorUnits as string with {readable : true, writable : false}
"Measurement Units definition, e.g. 'Cel' for Celcius"
}

operations{
resetMinandMaxMeasuredValues() "reset the Min and Max Measured Values"]
}
```

Figure 3.4 L'infomodel et functionBlock du capteur de température DHT11

Le *FunctionBlock* du DHT11 définit un ensemble de propriétés du capteur sous *Status* ; par exemple. la propriété *sensorValue* décrivant la valeur actuelle capturée par le capteur, les propriétés *minMeasuredValue* et *maxMeasuredValue* correspondant, respectivement, aux valeurs capturées la plus basse et la plus élevée, les propriétés *minRangeValue* et *maxRangeValue* définissant l'intervalle des températures du DHT11, et la propriété *sensorUnit* décrivant l'unité de la valeur de température. Une opération *resetMinandMaxMeasuredValues()* est définie pour permettre la remise à zéro des valeurs *minMeasuredValue* et *maxMeasuredValue*. De la même façon que pour la catégorie précédente, le code généré pour le système de surveillance météo est un squelette à compléter manuellement avant le déploiement et l'exécution sur la Raspberry Pi.

3.1.4 Implémentation du système de chauffage intelligent

Pour implémenter cette application et comme mentionné au Chapitre 2, nous avons utilisé le DHT11 comme capteur de température et une LED pour simuler le chauffage ; i.e. la LED est allumée ou éteinte selon la différence entre la température ambiante (mesurée par le DHT11) et celle désirée. La comparaison de ces deux valeurs et la décision de l'action sur la LED sont faites par une unité de contrôle.

Pour la decription du DHT11, nous avons réutilisé celle définie dans la deuxième catégorie. Nous avons décrit l'unité de contrôle et le chauffage chacun avec un *InfoModel*. Ensuite, nous avons généré le code de notre application en utilisant un des générateurs de code (i.e. en l'occurence, nous avons utilisé celui de Bosch-IoT). Pour chaque *InfoModel*, le générateur de code produit un fichier archive (.zip) (voir figure 3.5). Le code généré est destiné à être intégré à un thing implémenté dans la plate-forme Bosch IoT suite.

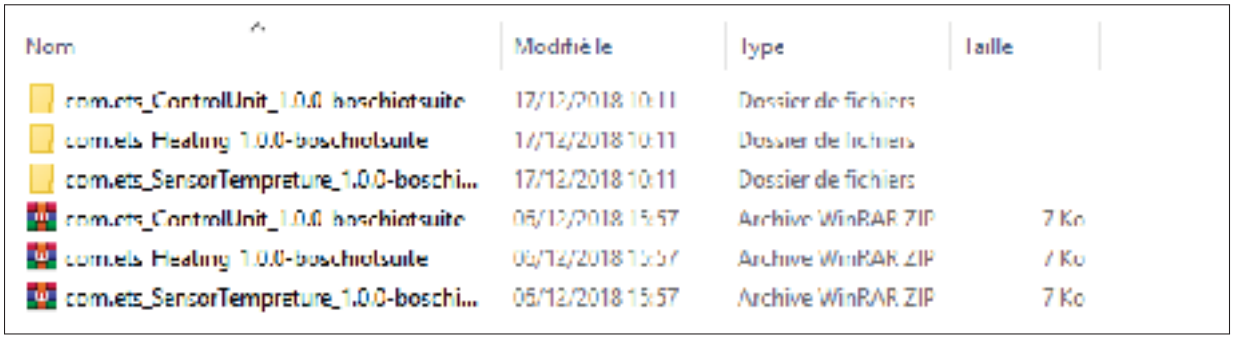


Figure 3.5 Arborescence du code généré par Vorto pour le système de chauffage

3.2 ThingML

3.2.1 Présentation générale du framework

ThingML (F. Fleurey & community, 2016) est un plugin Eclipse qui vise à supporter la conception et l'implémentation de systèmes réactifs distribués (IoT, systèmes embarqués, WSN, etc.). Il cible principalement les objets à ressources limitées. Les principaux objectifs de ThingML sont présentés dans (Vasilevskiy, Morin, Haugen & Evensen, 2016) comme étant de :

- séparer les "préoccupations" ;
- réduire l'écart entre le problème et les solutions technologiques proposées ;
- proposer un ensemble de générateurs de code qui ciblent différentes plates-formes.

ThingML est basé sur un DSL qui combine différents éléments de modélisation incluant :

- un modèle architectural ;
- des machines à état ;
- un langage d'action inspiré d'UML.

En effet, le DSL de ThingML permet de spécifier l'architecture d'une application IoT sous forme d'objets communicants, et de modéliser leur comportement en utilisant des machines à états et un langage d'action. Il repose principalement sur 2 structures clés :

1. Les *Things* qui représentent les composants physiques (les objets). La spécification d'un *Thing* peut comprendre :
   - des propriétés : variables locales accessibles à partir du *Thing* lui-même ;
   - des fonctions : fonctions locales, visibles uniquement par le *Thing* (fonctions privées) ;
   - des ports : C'est l'interface publique du *Thing* ; le *Thing* envoie et/ou reçoit les messages à travers les ports ;
   - des messages : définis au niveau du *Thing* et échangés entre les *Thing* à travers les ports ;
   - des machines à état : elles sont définies par le *Thing* pour décrire son comportement et ses différents états.
2. Les Configurations qui représentent les interconnexions entre les différents *Things*.

Prenons l'exemple "Hello World" : Tel qu'illustré dans le diagramme d'état de HelloWorld (figure 3.6), le *HelloThing* est initialement à l'état "*Greatings*", ensuite, un texte est affiché ("Hello world !") et le *Thing* transite à l'état *Bye*. À l'état *Bye*, un autre texte est affiché ("Bye.").

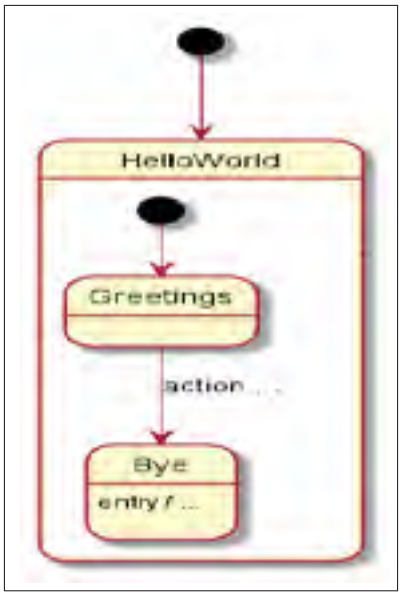


Figure 3.6 Diagramme d'état HelloWorld de l'objet HelloThing

Afin d'instancier l'application, nous avons défini une configuration nommée *HelloCg*, et une instance de la configuration nommée *my_instance* (voir figure 3.7).

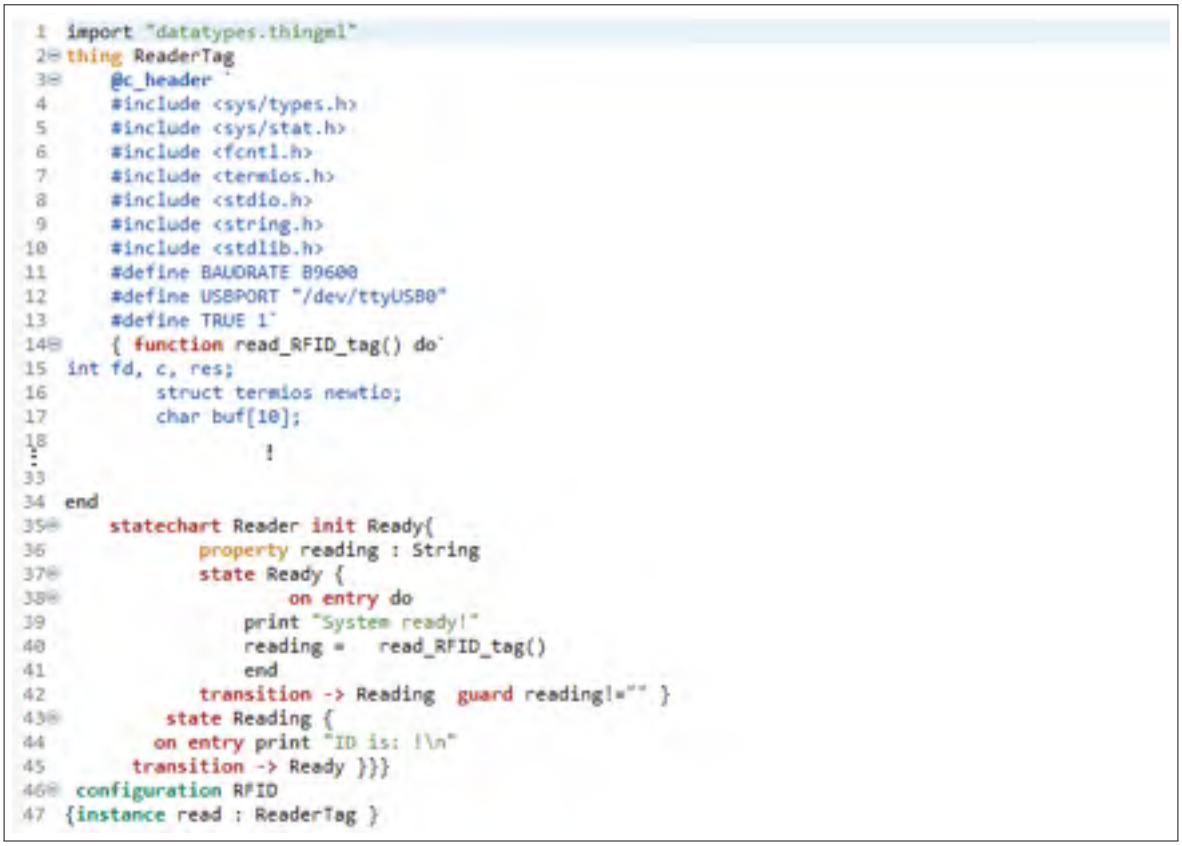
```
thing HelloThing {

statechart HelloWorld init Greetings {
state Greetings {

        transition -> Bye
        action do
            print "Hello World!\n"
        end
        }
state Bye @ignore "sink" {
            on entry print "Bye.\n"}
    }
}
configuration HelloCfg
{
instance my_instance: HelloThing
}
```

Figure 3.7 Code ThingML de l'exemple HelloWorld

ThingML permet la génération d'un code complet, prêt à être exécuté directement sur la plate-forme. Les langages supportés par ThingML sont : C, C++, Java, JavaScript et GO. ThingML offre aussi la possibilité d'intégrer du code C, Java ou Javascript, dans la définition des différents éléments listés précédemment. En fin, ThingML permet la génération de digrammes UML grâce aux plugins Eclipse PlantUML et Graphviz.

3.2.2 Implémentation du système de gestion d'inventaire

Pour déployer la première catégorie avec ThingML, nous avons créé un *Thing* nommé *ReaderTag* (présenté dans la figure 3.8). Nous avons développé une fonction en C nommée *read_RFID_tag()* (lignes 3 à 34 de la figure 3.8, présentée en détail en annexe IV). Cette fonction va écouter le port USB et afficher la valeur du tag RFID dès que ce dernier est lu par le lecteur. Le comportement du lecteur de tags est spécifié comme une machine à états en utilisant le DSL de ThingML



```
1  import "datatypes.thingml"
2  thing ReaderTag
3      @c_header "
4      #include <sys/types.h>
5      #include <sys/stat.h>
6      #include <fcntl.h>
7      #include <termios.h>
8      #include <stdio.h>
9      #include <string.h>
10     #include <stdlib.h>
11     #define BAUDRATE B9600
12     #define USBPORT "/dev/ttyUSB0"
13     #define TRUE 1"
14     { function read_RFID_tag() do"
15 int fd, c, res;
16         struct termios newtio;
17         char buf[10];
18                 !
33
34 end
35     statechart Reader init Ready{
36             property reading : String
37             state Ready {
38                     on entry do
39                 print "System ready!"
40                 reading =   read_RFID_tag()
41                 end
42             transition -> Reading  guard reading!="" }
43          state Reading {
44          on entry print "ID is: !\n"
45        transition -> Ready }}}
46  configuration RFID
47 {instance read : ReaderTag }
```

Figure 3.8 Spécification du Lecteur de Tags RFID en utilisant ThingML

(lignes 35 à 45 de la figure 3.8). Initialement, le *Thing ReaderTag* est à l'état *"Ready"*. Il passe à l'état *"Reading"* lors de la lecture d'un ID de tag.

3.2.3 Implémentation du système de surveillance météorologique

Pour cette deuxième catégorie, nous avons créé un *Thing SenseC* (voir figure 3.9) qui représente le capteur de température DHT11. En premier lieu, nous avons défini une fonction en C nommée *read_dht11_dat()* (lignes 7 à 78 de la figure 3.9, présentée en détail à l'annexe V). Cette fonction permet de lire la température mesurée à partir des pins auxquelles est connecté le DHT11.

Le DHT11 effectue une lecture toutes les minutes grâce à un Timer qui déclenche un compte à rebours d'une minute dès qu'il est appelé dans le processus (grâce au message prédéfini

timer_start () à la ligne 92 de la figure 3.9). Nous avons réutilisé ce Timer d'un projet thingML, proposé dans (TelluIoT, 2016).

```
1  import "lib/_Timer_C.thingml"
2  import "lib/_Datatypes.thingml"
3
4
5  thing SenseC includes TimerMsgs
6
7  @c_header "
8  #include <wiringPi.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <stdint.h>
12 #define MAXTIMINGS  85
13 "
14 {
15
16     readonly property DHTPIN : UInt8 = 7
17
18     function read_dht11_dat() do
19
20     "int dht11_dat[5] = {0,0,0,0,0};
21     uint8_t laststate = HIGH;
22     uint8_t counter = 0;
23     uint8_t j = 0, i;
24     float f; // fahrenheit
25
26     dht11_dat[0] = dht11_dat[1] = dht11_dat[2] = dht11_dat[3] = dht11_dat[4] = 0;
27
                                    [...]
78     end
79         required port clock {
80         sends timer_start
81         receives timer_timeout }
82
83 statechart PiSenseC init Sensing{
84
85
86         state Sensing {
87             property initResult : Int8
88                 on entry do
89                 initResult = `wiringPiSetup()` as Int8
90                 print "sensing...!\n"
91                 read_dht11_dat()
92                 clock!timer_start(0, 1000)
93                   end
94
95
96             transition -> Sensing event clock?timer_timeout guard initResult != -1
97             transition -> Error guard initResult == -1
98
99                 }
100     final state Error {
101     on entry error "Setup wiringPi failed !\n"
102 }
103 }
104 }
105
106 configuration sense
107     @add_c_libraries "wiringPi"
108 {
109     instance sense : SenseC
110     connector sense.clock over Timer
111 }
```

Figure 3.9 Fichier ThingML représentant le capteur DHT11

Dans le *Thing SenseC* (voir figure 3.10), nous avons défini une machine à état décrivant le fonctionnement de l'objet. L'objet se met à lire les valeurs Température et Humidité (état *Sensing*), puis, pour une lecture périodique, un *Timer* s'active. Si les valeurs sont capturées, la lecture se refait dès que le *Timer* se désactive, c'est-à-dire à chaque minute dans notre cas (Boucle sur l'état *Sensing*), sinon, une erreur se produit.

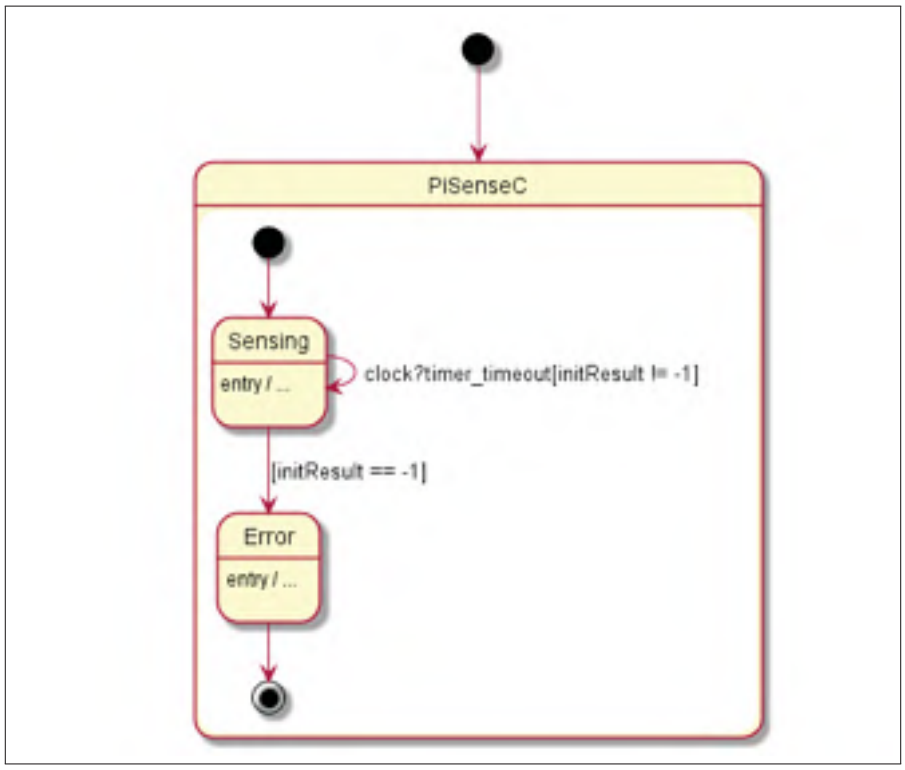


Figure 3.10 Diagramme d'état du Thing SenseC relatif au capteur de température DHT11

3.2.4 Implémentation du système de chauffage intelligent

Pour implémenter le système de chauffage intelligent, nous avons modifié le *Thing* que nous avons défini dans la 2ème catégorie pour le capteur de Température et d'Humidité DHT11 et nous avons ajouté une fonction pour comparer les températures mesurées et celle désirée (figures 3.11 et 3.12). La température mesurée par le DHT11 est désormais comparée à une température prédéfinie (variable *tref* de la ligne 23 de la figure 3.11).

```
1  import "lib/_Timer_C.thingml"
2  import "lib/_Datatypes.thingml"
3
4  thing fragment HeatMsgs{
5      message activate() ;
6      message desactivate() ;
7  }
8
9
10 thing SenseC includes TimerMsgs, HeatMsgs
11
12 @c_header "
13 #include <wiringPi.h>
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <stdint.h>
17 #define MAXTIMINGS  85
18 "
19 {
20
21     readonly property DHTPIN : UInt8 = 7
22     property temp : Double
23     property tref : Double = 25.00
24
25     function read_dht11_dat() : Double do
26
27     "int dht11_dat[5] = {0,0,0,0,0};
28     uint8_t laststate = HIGH;
29     uint8_t counter = 0;
                    [...]
80 return temp
81 end
82
83 function comparer (t1: Double, t2: Double):Boolean do
84 if (t1<t2) return true
85 else return false
86 end
87
88         required port clock {
89         sends timer_start
90         receives timer_timeout
91         }
92
93         provided port temp{
94             sends activate,desactivate
95         }
```

Figure 3.11 Code thingML relatif aux fonctions définies pour implémenter l'application du chauffage intelligent

Nous avons défini un autre *Thing* nommé *Actuator*, relatif à la lampe LED (figure 3.13). Ce *Thing* recevra le résultat de la comparaison des deux températures (*activate* ou *desactivate*, lignes 108 et 109 de la figure 3.12) : Si la valeur capturée est inférieure à celle de référence, la LED reçoit le message *activate* et la LED va se mettre à ON, sinon, la LED reçoit le message *desactivate* et la lampe se met à OFF. Finalement, nous avons défini des instances de chaque

```
statechart PiSenseC init Sensing{


        state Sensing @ignore "sink" {
            property initResult : Int8
            property t : Double
                on entry do
                initResult = `wiringPiSetup()` as Int8
                print "sensing...!\n"
                t = read_dht11_dat()
                clock!timer_start(0, 1000)
                if (comparer(t,tref) == true)
                temp!activate()
                else temp!desactivate()
                end


            transition -> Sensing event clock?timer_timeout guard initResult != -1
            transition -> Error guard initResult == -1

                }
     final state Error {
    on entry error "Setup wiringPi failed !\n"
   }
  }
  }
```

Figure 3.12 Code ThingML relatif au diagramme d'état *PiSenseC* du thing *SenseC*

```
thing Actuator includes HeatMsgs

    @c_header "
    #include <wiringPi.h>"
  {
   readonly property LED_PIN : UInt8 = 11

    required port temp{
        receives activate,desactivate
    }
    statechart Heating init Stop{
        on entry "pinMode(&LED_PIN&, OUTPUT);"
        state Stop{
            on entry do
            "digitalWrite(&LED_PIN&, LOW);"
            end
            transition -> heating event temp?activate
            transition -> Stop event temp?desactivate
        }

        state heating{
            on entry do
            print("got activation")
            "digitalWrite(&LED_PIN&, HIGH);"
        end
        transition -> Stop event temp?desactivate
        transition -> heating event temp?activate
    }

}

}
```

Figure 3.13 Code thingML relatif à la LED

Thing et nous avons défini la connexion entre les deux objets, tel que présenté dans la figure 3.14.

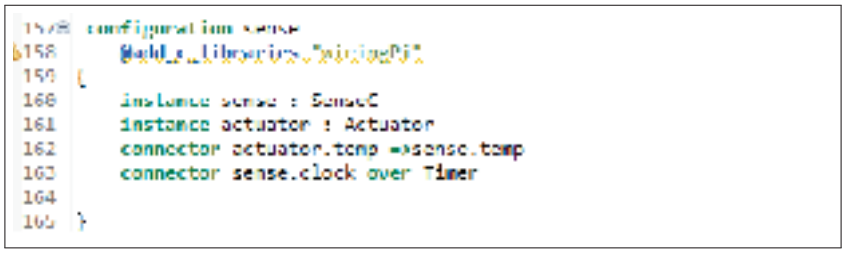


Figure 3.14 Extrait de code thingML relatif à la configuration du DHT11 et de la LED

Pour chacun des deux *things*, nous avons défini un diagramme d'état. La machine a état *PiSenseC* relative au *Thing SenseC* est identique à la machine à état du DHT11 présentée précédemment à figure 3.10). La machine à état *Heating* relative au *Thing Actuator* etst illustrée à la figure 3.15. Le *Heater* est initialement à l'état "Stop". La température mesurée reçue du DHT11 est comparée à une température seuil. Si la température actuelle est inférieure à la température seuil, le chauffage est allumé et il passe à l'état *"Heating"* via la transition *"activate"*. Sinon, il reste à l'état *"Stop"*. Lorsque le *Heater* est dans l'état *"Heating"* et que la température est supérieure au seuil, le chauffage est éteint et il passe à l'état "Stop" via la transition *"desactivate"*.

Finalement, nous avons généré le code de notre application et nous avons déployé et exécuté le code sur la Raspberry Pi.

Il est à noter que ThingML permet aussi de choisir et définir les protocoles. Nous avons tiré profit de cette possibilité pour définir le protocole MQTT pour faire communiquer les deux objets distants DHT11 et la lampe LED (voir figure 3.16). Avec ThingML, on peut utiliser le *MQTT Broker* de Eclipse, Mosquitto. Le capteur de température publie le message au *Broker Mosquitto* avec un *Topic* Temperature. La lampe LED étant souscrite au *Broker* sous le même *Topic* recevra la température, le traitement s'effectuera à ce niveau et une décision sera prise (allumer ou éteindre la lampe LED).

Figure 3.15 Diagramme d'état du Heater

```
115 protocol MQTT @serializer "JSON"
116   @mqtt_broker_address "localhost" @trace_level "2"
117   @mqtt_publish_topic "ThingML/temperature/ctrl"
118   @mqtt_subscribe_topic "ThingML/temperature/status";
```

Figure 3.16 Définition du protocole MQTT avec ThingML

3.3 Node-Red

3.3.1 Présentation générale du framework

Node-Red (OpenJSFoundation, 2019) est un outil graphique pour modéliser les applications IoT sous forme de flux de données. Il permet l'interconnexion des objets physiques, les API, les environnements Cloud et des services web.

L'éditeur de flux de Node-Red est accessible via les navigateurs et est composé de 3 parties (voir figure 3.17) :

- la palette des nœuds : fournit un ensemble de nœuds qui sont les éléments de base pour la création de flux ;
- l'environnement de travail : c'est l'espace de travail où sont créés les flux ;
- la barre d'affichage : peut être utilisée pour afficher les informations, i.e, informations relatives aux noeuds, information de débogage.

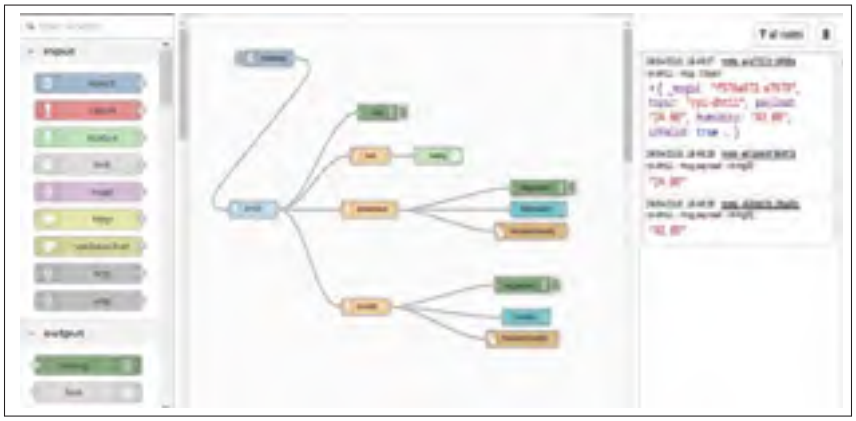


Figure 3.17 Éditeur de flux de Node-Red

Les nœuds sont développés en JavaScript et enregistrés sous format JSON. Il existe une panoplie de nœuds prédéfinis et facilement téléchargeables et réutilisables. Il suffit de glisser et déposer les nœuds depuis la palette des nœuds vers l'environnement de travail pour créer son flux de données. La palette des nœuds est aussi extensible, le développeur peut créer et enregistrer ses propres nœuds. En fait, Node-Red a une communauté très active avec une pléthore de nœuds ayant été développés pour différents appareils. Un nœud notable est le nœud *Function* lequel permet de programmer un comportement donné directement en utilisant du code JavaScript via l'interface graphique.

Node-Red offre un outil graphique puissant qui permet de contrôler et visualiser les différents paramètres et ce en utilisant les nœuds graphiques de Node-Red. La visualisation est aussi possible via la barre d'affichage de l'éditeur de flux de Node-Red (sans utiliser les nœuds graphiques). En plus de la visualisation des résultats, Node-Red permet aux utilisateurs d'agir

directement à travers l'outil graphique sur l'état des différents objets (par exemple, mettre à ON ou OFF une lampe LED.).

3.3.2 Implémentation du système de gestion d'inventaire

Pour implémenter cette application, nous avons suivi les étapes décrites dans le tutoriel en ligne « IOT Tutorial : Read RFID-tags with an USB RFID reader, Raspberry Pi and Node-Red from scratch » (Bink, 2018). Pour commencer, nous avons eu besoin de télécharger la palette de nœuds relative aux objets connectés via les ports USB. Pour pouvoir lire à partir du lecteur de tags, nous devions d'abord l'enregistrer avec ses IDs de fournisseur et d'appareil. Pour ce faire, nous avons utilisé le nœud *getHIDdevices* (*VID* : Vendor ID, *PID* : Product ID) (figure 3.18).

Figure 3.18 Identification de l'ID du lecteur de tags RFID

Nous avons ensuite créé le flux pour traiter les informations récupérées du lecteur de tags (figure 3.19).

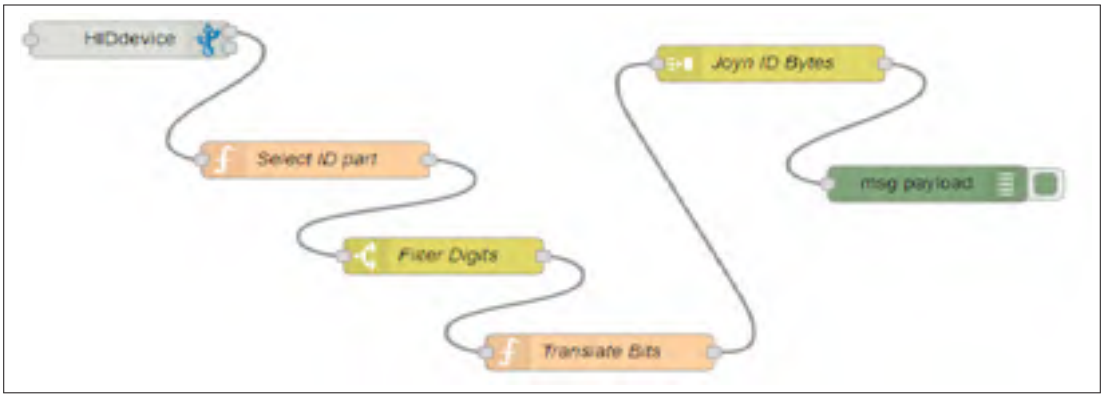


Figure 3.19 Flux de données relatif à la lecture des tags RFID

Pour ce faire, nous avons utilisé le nœud HIDdevice qui a été alimenté par les valeurs des ID de fournisseur et de périphérique du lecteur de tags (*VID* et *PID*) (Figure 3.20). Le nœud HIDdevice lit les informations reçues du lecteur RFID via le port USB de la Raspberry Pi.

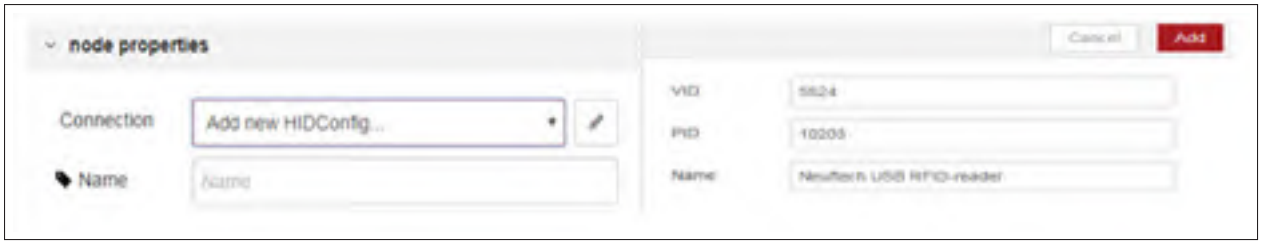


Figure 3.20 Configuration du nœud HIDdevice

Les tableaux d'octets produits par le nœud HIDdevice doivent être traités afin de récupérer les ID des tags. Ce traitement comprend la sélection uniquement de la partie du tableau d'octets contenant l'ID, l'élimination des lignes vides et des sauts de ligne, la conversion des octets en chiffres réels et la fusion des chiffres résultants en un seul ID. Nous avons utilisé un nœud Function *Select ID part* pour extraire la partie qui contient l'ID (le 2e octet, voir figure 3.21).

Figure 3.21 Fonction Select ID part

Puis, nous avons filtré la lecture, pour se limiter aux nombres (0 à 9). Nous avons effectué cela grâce à un nœud de Filtrage *FilterDigit* (voir figure 3.22). Ensuite, la fonction *Translate Bits* (figure 3.23) sert à transformer les octets lues en chiffres. Cette transformation a été proposée dans un projet Github (Maliński, 2019).

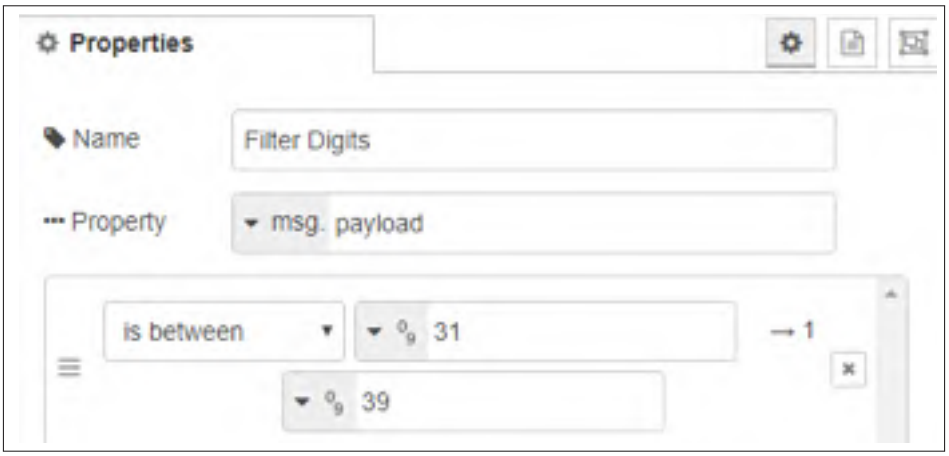


Figure 3.22 Nœud *FilterDigit*

Figure 3.23 Fonction *Translate Bits*

Finalement, nous avons utilisé la fonction join *Join ID Bytes* (figure 3.24) pour lier tous les chiffres et avoir l'ID du tag RFID.

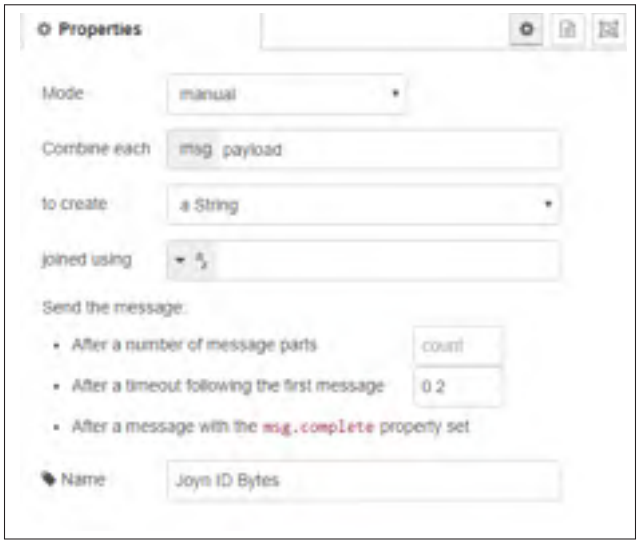


Figure 3.24 Nœud *Join ID Bytes*

Pour tester le flux, nous avons utilisé 3 tags RFID, dont les id sont : 0005951860, 0001216396 et 0006097419 comme montré dans la figure 3.25. Le résultat de notre expérimentation est présenté dans la figure 3.26.

Figure 3.25 Les tags RFID utilisés pour l'expérimentation

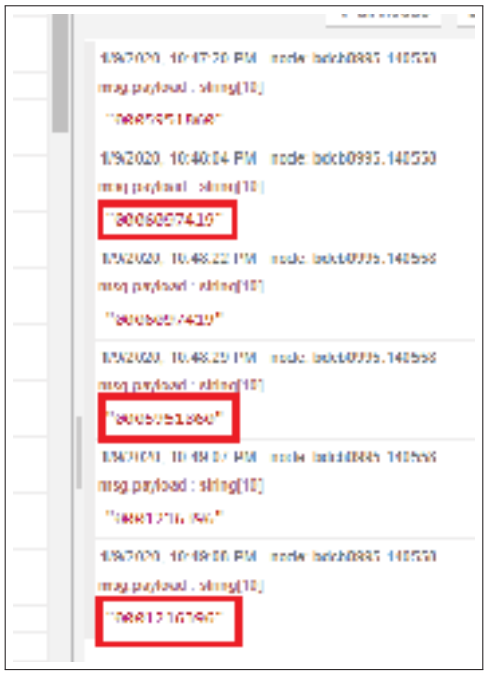


Figure 3.26 Résultat du test du flux final

3.3.3 Implémentation du système de surveillance météorologique

Node-Red cible le développement sur la Raspberry à travers un ensemble de nœuds pré-développés et disponible à télécharger depuis une librairie de nœuds en ligne. Les développeurs peuvent aussi partager leurs propres noeuds avec l'ensemble des utilisateurs de Node-Red. Nous avons utilisé le noeud DHT11/22 qui permet la lecture de la température et l'humidité simplement en configurant le type du capteur (i.e. 11 ou 22) et en indiquant les numéros des Pins de la Raspberry auxquels le capteur est connecté. La figure 3.27 montre le flux de notre application incluant le noeud DHT11. Ce dernier effectue une seule lecture des deux valeurs « température » et « humidité ». Nous avons donc utilisé deux nœuds fonctions écrites en JavaScript (*temperature* et *humidity*) afin de séparer les deux valeurs. Les valeurs capturées sont par la suite enregistrées dans un fichier nommé « *TempAndHumidity* » et affichées graphiquement sur des gauges en utilisant des nœuds prédéfinis, nommés *Temperature* et *Humidity* montrés à la fin du flux.

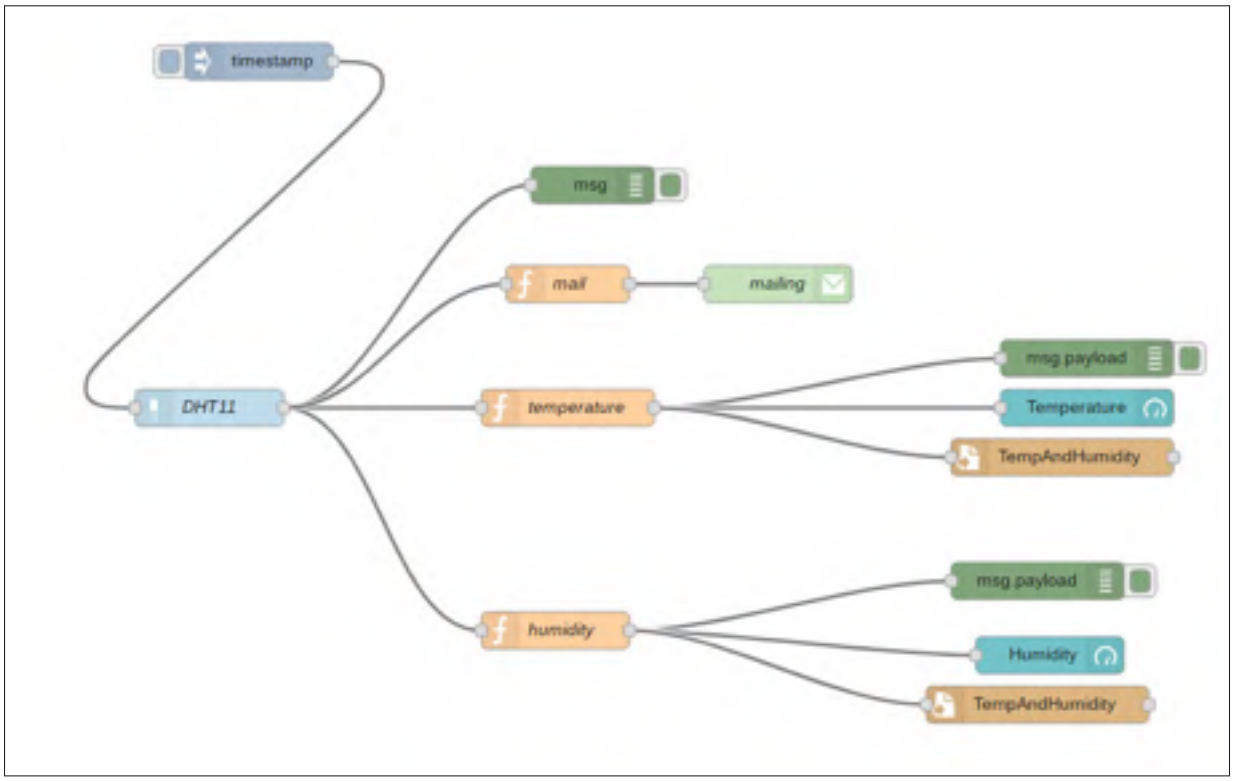


Figure 3.27 Flux de données du système de surveillance météorologique

Parallèlement, nous avons utilisé le nœud prédéfini « *email* » pour envoyer les valeurs capturées par courriel (Figure 3.28). Un nœud « fonction » nous a permis de formater le corps du courriel. Les noeuds *Timestamp* et *Msgpayload* ont pour objectif respectivement d'initialiser et de marquer la fin du processus.

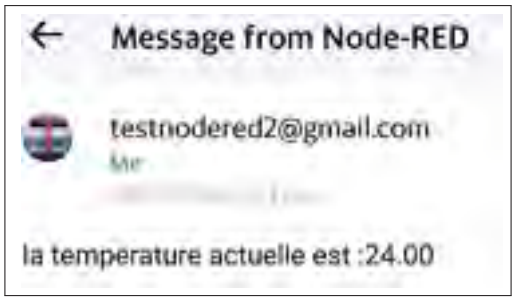


Figure 3.28 Courriel reçu automatiquement depuis Node-Red

Les résultats sont visualisés dans la barre d'affichage, ou bien via l'outil graphique (voir figure 3.29).

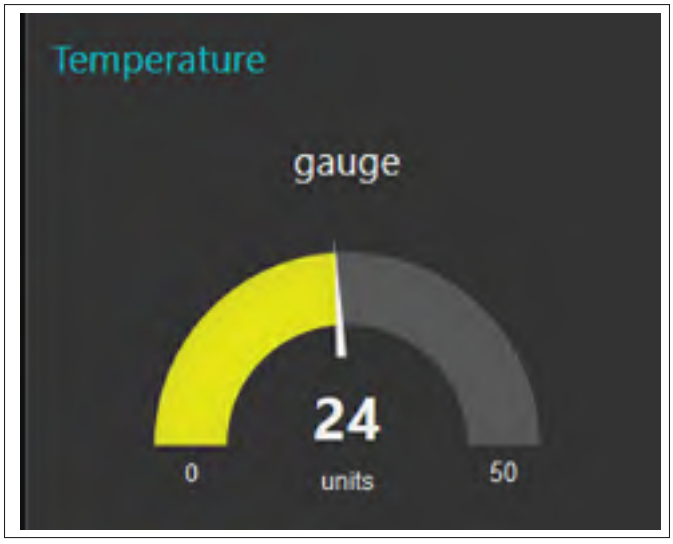


Figure 3.29 Affichage graphique de la Température via l'outil graphique

3.3.4 Implémentation du système de chauffage intelligent

Comme pour la deuxième catégorie, nous avons utilisé les nœuds prédéfinis destinés au développement avec la Raspberry Pi pour implémenter le système de chauffage intelligent. Nous avons utilisé le même noeud DHT11 présenté dans la sous-section précédente. La figure 3.30 présente le flux pour le système de chauffage quand les objets sont tous connectés sur la même Raspberry Pi (i.e. tout est local). La température captée par le noeud DHT11 doit être comparée à une température cible pour prendre une décision d'actionner la LED (qui simule le chauffage). Pour ce faire, nous avons utilisé un nœud de fonction pour convertir les données de température obtenues à partir du nœud DHT11 d'un type "chaîne de caractères" en un type "réel". Nous avons également implémenté un nœud de fonction de décision (en JavaScript), qui compare les deux valeurs de température (i.e. captée et cible), et allume/éteint la LED si le chauffage doit être activé/désactivé.

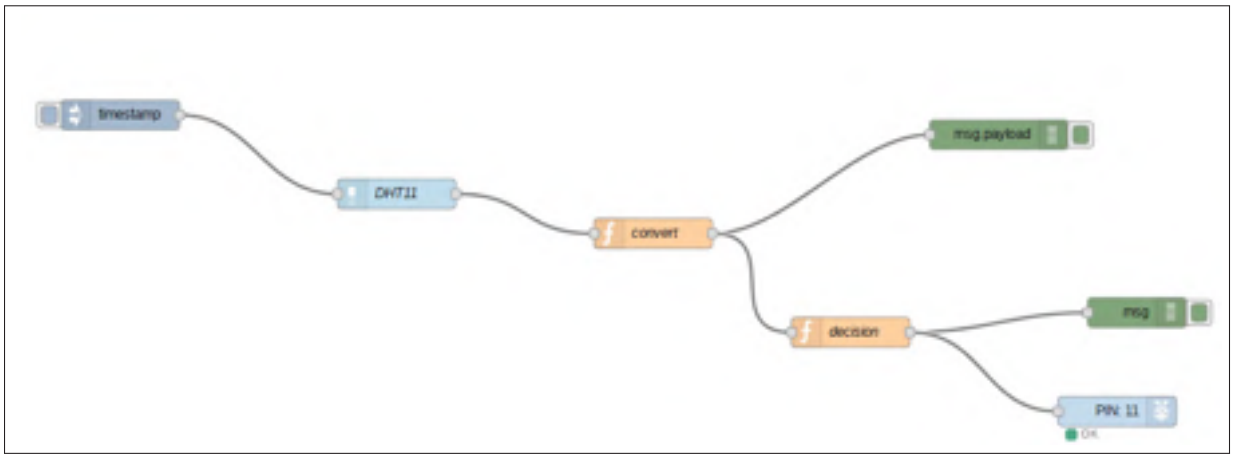


Figure 3.30 Flux de données entre le capteur DHT11 et la lampe LED lorsque connectés sur la même Raspberry Pi

Pour le cas où les objets sont distribués (i.e. ne sont pas connectés localement), Node-Red offre une palette de nœuds supportant des protocoles de communication des objets. Pour ce deuxième cas de figure, nous avons utilisé le protocole MQTT vu sa dominance dans le monde de l'IoT. En particulier, nous avons utilisé le nœud *MQTT in*. Ce dernier permet aux objets de se connecter à un MQTT *broker* (server) et s'abonner aux messages ayant un *Topic* les intéressant. La précision du *Topic* est donc requise dans ce cas (voir figure 3.31).

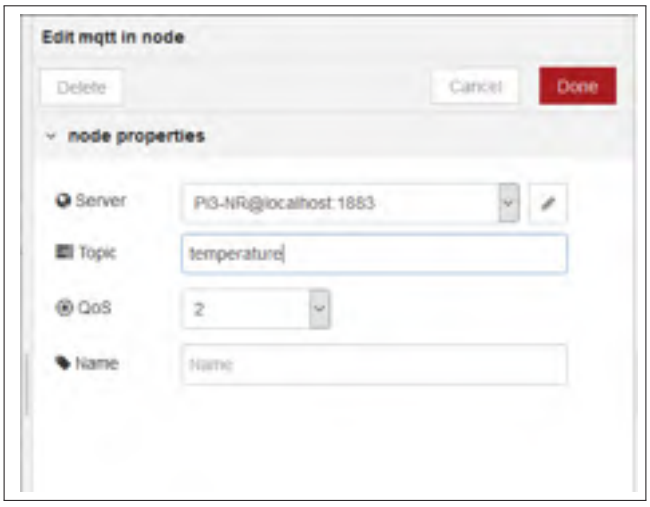


Figure 3.31 propriétés du nœud "MQTT in"

Nous avons aussi utilisé le noeud *MQTT out* qui permet aux objets de publier leurs messages au MQTT *Broker* (voir figure 3.32). Si on précise un *topic* dans le nœud MQTT out, le message sera publié avec ce *Topic*. Si on publie le message avec un *topic* vide, c'est le *topic* de l'objet lui-même qui va être publié avec le message. Dans notre cas, l'objet DHT11 diffuse l'information au MQTT *broker* sous un topic "temperature" et l'objet LED s'enregistre auprès du MQTT *broker* aux messages ayant pour topic "temperature".

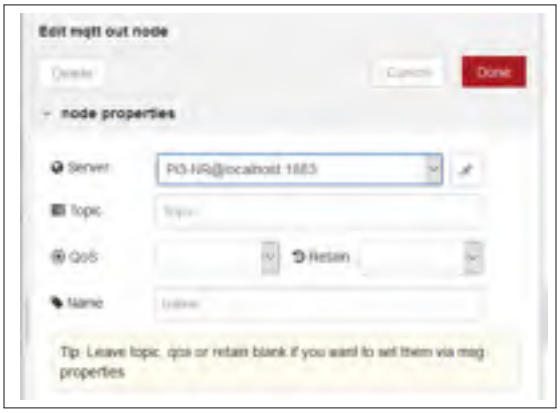


Figure 3.32 Propriétés du nœud "MQTT out"

Le flux de données relatif au nœud MQTT in est illustré dans la figure 3.33. Le nœud DHT11 capte la valeur de la température et l'envoie via le nœud MQTT in.

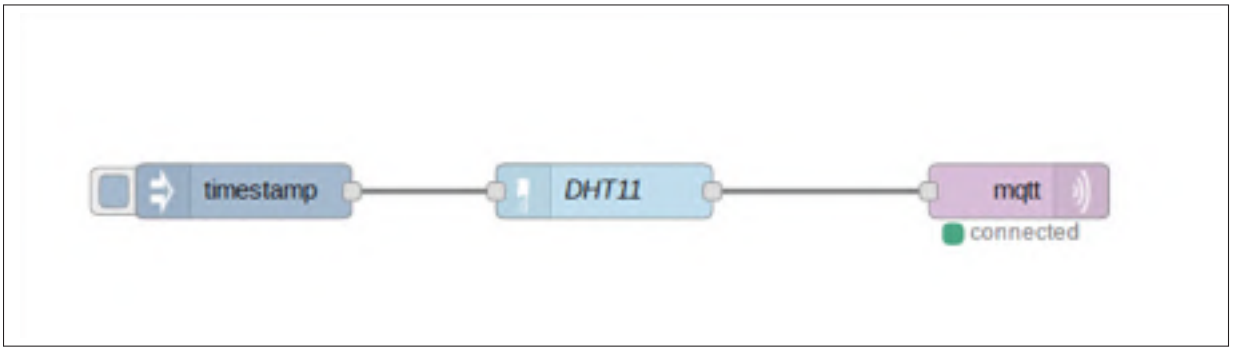


Figure 3.33 Flux de données de l'objet DHT11 dans le cas des objets distribués

Du côté de l'objet LED (voir figure 3.34), la valeur de la température est reçue via le nœud MQTT out. Ensuite, le nœud *convert* convertit cette valeur en un réel qui va être comparé par le nœud *decision* à une température cible. Une décision (0 ou 1) va par la suite être envoyée à la lampe LED pour s'allumer ou s'éteindre.

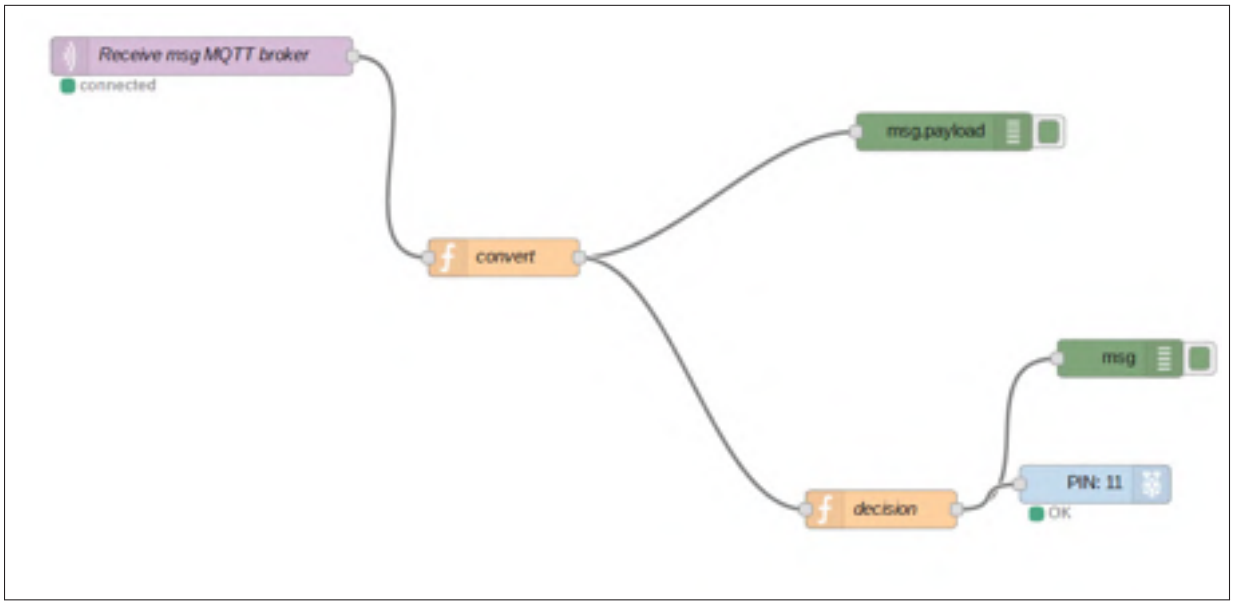


Figure 3.34 Flux de données de l'objet LED dans le cas des objets distribués

3.4 OpenHab

3.4.1 Présentation générale du framework

OpenHab (Open Home Automation Bus) (OpenHAB & the openHAB Foundation e.V., 2019) est un des projets open source de Eclipse IoT destiné au développement des maisons intelligentes. On entend dire par "maison intelligente", des maisons qui offrent le confort, la sécurité, la faible consommation d'énergie (et donc un faible coût d'énergie) et une certaine commodité à leurs résidants (CHEN, 2020). OpenHab permet donc d'intégrer et d'automatiser les différents composants d'une maison intelligente dont, entre autres, lumière, chauffage, capteurs de température, portes et fenêtres.

Essentiellement, OpenHab doit son succès auprès des développeurs aux caractéristiques suivantes :

- il peut intégrer divers objets dans une application : grâce à son architecture, OpenHab peut supporter plus que 200 différentes technologies et systèmes, et des milliers d'objets ;
- il permet une automatisation facile : OpenHab utilise un outil flexible pour l'implémentation de la logique d'automatisation et celà à travers des règles, scripts, actions et notifications, en plus de sa capacité à contrôler la voix ;
- il peut rouler « partout » : le serveur de OpenHab peut être exécuté sur n'importe quel système d'exploitation, fonctionne sur différentes plates-formes hardware telles que la Raspberry Pi et est accessible de presque partout (Web, IOS, Adroid,etc.).

OpenHab fait la distinction explicite entre deux vues d'un système IoT : la vue physique qui représente les appareils et leurs connexions, et la vue logique qui correspond aux informations représentant les appareils et les connexions dans l'application. Concrètement, développer une application IoT revient à :

- définir les *things* et les *channels* et *items* correspondants. Un *thing* est une représentation logicielle d'un objet ou d'un service. Il expose ses fonctions à travers des *channels*. Les *items* représentent les fonctionnalités (interfaces utilisateur et logique d'automatisation) utilisées par l'application. Les *things* font partie de la vue physique tandis que les *items* font partie de la vue logique (également appelée couche virtuelle). Les *channels* sont liés aux *items*, et ces liens sont ce qui établit la relation entre la couche virtuelle (c'est-à-dire les *items*) et la couche physique (c'est-à-dire les *things*). Un lien relie un *channel* à un *item*, mais un *channel* peut être lié à plusieurs *items* et vice versa. Un *thing* réagit aux événements envoyés aux *items* liés à ses *channels*, et il envoie des événements à ces *items* ;
- rechercher les modules complémentaires OpenHab (i.e. *addons*) pour chaque objet faisant partie de l'application et installer les *bindings* pour chaque objet ; un *binding* est un composant OpenHAB qui permet l'interaction avec un objet. En particulier, les *bindings* établissent les connexions entre les objets et leurs *things* ;

- définir le *sitemap* de l'application ; un *sitemap* est l'interface utilisateur générée par OpenHAB qui présente les informations aux utilisateurs et permet l'interaction utilisateur ;
- définir des règles (i.e. *rules*) ; une *rule* est utilisée pour automatiser des processus. Elle définit une condition de déclenchement (par exemple, un item dont le statut a changé) et une logique (c'est-à-dire un script) qui doit être exécutée lorsque la condition est remplie.

Prenons l'exemple d'un système d'éclairage, composé d'une lampe LED. Pour utiliser la LED dans une application développée avec OpenHab, nous avons d'abord installé le *binding GPIO* pour l'objet LED. Ensuite, nous avons défini un *thing* pour la LED. Ce *thing* est stocké dans un fichier nommé *home.things*, Nous avons aussi spécifié l'état de la lampe comme un *item*, dans un fichier nommé *home.items*. Dans ce fichier, nous incluons un bouton «*switch*» pour changer l'état de la lampe (bouton marche/arrêt), et nous indiquons également les numéros des pins auxquelles est connectée la lampe et sur lesquelles va agir le bouton afin d'allumer ou éteindre la lampe (`Switch RaspiLED{ gpio="pin:4"}`.). En dernier lieu, nous avons défini un fichier *sitemap* dans lequel nous avons inclus le bouton *switch* afin de permettre aux utilisateurs d'agir sur la LED à partir de l'interface graphique.

```
sitemap home label="Home" {
Frame label="Rasperry Pi GPIO"
{ Switch item=RaspiLED }
}
```

Cet exemple peut être développé plus par l'intégration d'autres objets et l'interconnexion des différents objets en utilisant des règles qui sont stockées dans des fichiers *.rules*.

OpenHab offre des outils permettant la visualisation des résultats ainsi que le contrôle des états des différents objets ; cela peut se faire à travers un outil web (voir figure 3.35) ou une application mobile (voir figure 3.36) disponible pour Android et IOS, et téléchargeable gratuitement. OpenHab offre un système d'exploitation OpenHabian, basé sur Linux, ayant pour objectif d'être un système auto-configurable en vue de s'adapter aux besoins de chaque utilisateur OpenHab.

Figure 3.35 Interface graphique de OpenHab

Figure 3.36 Application mobile de OpenHab

3.4.2 Implémentation du système de gestion d'inventaire

Nous n'étions pas en mesure de déployer l'exemple de gestion d'inventaire puisque le protocole de communication utilisé par le lecteur de tags RFID (à travers le port USB) n'est pas supporté par la version étudiée de OpenHab.

3.4.3 Implémentation du système de surveillance météorologique

Afin d'implémenter cette application, nous avons utilisé deux méthodes : une qui consiste à suivre les étapes décrites précédemment dans la sous-section 3.4.1 (Présentation du framewok),

et l'autre qui repose sur le protocole MQTT afin de lire et afficher les valeurs mesurées par le DHT11.

Dans la première méthode, nous avons défini un *thing* pour le capteur DHT11 dans un fichier nommé DHT11.things (figure 3.37) et la température en tant qu'*item*, dans un fichier DHT11.items (figure 3.38). Nous avons aussi utilisé un script Python (dht11.py) qui lit les valeurs de température et d'humidité du DHT11 (disponible à (Psyciknz, 2017)). Nous avons également ajouté le *binding GPIO* pour permettre à OpenHab de lire les valeurs capturées par DHT11 via les pins de la Raspberry. Enfin, nous avons développé le *sitemap* pour concevoir l'interface utilisateur de notre application (figure 3.39).

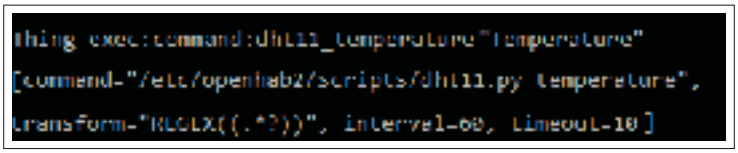


Figure 3.37 Extrait du fichier DHT11.things pour le système de surveillance météo

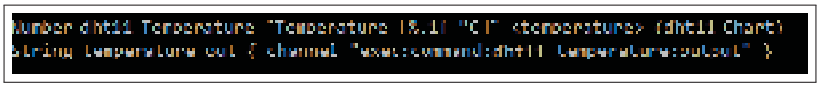


Figure 3.38 Extrait du fichier DHT11.items pour le système de surveillance météo

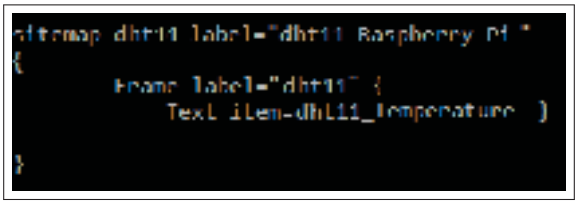


Figure 3.39 Extrait du fichier DHT11.sitemap pour le système de surveillance météo

La deuxième méthode consiste à lire la valeur de température avec un DHT11, puis à l'envoyer à un *Subscriber* via MQTT. Pour ce faire, nous avons d'abord utilisé un script Python pour lire les valeurs de température mesurées par le DHT11 (disponible à (Adafruit, 2019)). Ensuite, nous l'avons testé en exécutant le script en indiquant les numéros des PIN auxquels nous avons connecté le capteur DHT11 dans la ligne de commande OpenHabian (voir figure 3.40). Nous avons utilisé un script python (disponible à (Psyciknz, 2017)), qui lit la valeur de la température et la publie au *MQTT Broker* de Eclipse Hono toutes les 300s dans notre cas avec un topic *Temperature*. Nous avons utilisé Eclise Paho (Paho, 2016) qui est un serveur MQTT et qui fait également partie du projet Eclipse IoT. D'un autre côté, nous avons installé un client MQTT offert par Eclipse Paho qui aura le rôle de *Subscriber*. Le client souscrit au *Broker* avec le même topic *Temperature*, et l'affiche dès qu'il est envoyé par le DHT11.

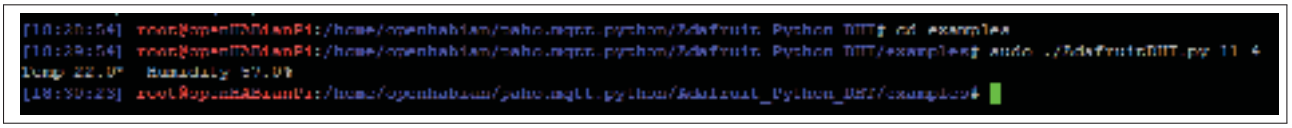


Figure 3.40 Exécution du script python pour la lecture de Température et d'Humidité via DHT11

3.4.4 Implémentation du système de chauffage intelligent

Pour implémenter cette application, nous avons activé le *binding GPIO* pour openHab afin de permettre la lecture et l'écriture via les Pins de la Raspberry Pi. Nous avons spécifié les Pins auxquelles le DHT11 et la LED ont été branchés à l'aide d'un script python (nous avons réutilisé et modifié le script Python disponible sur (Adafruit, 2019)). Nous avons défini deux Things : un pour le capteur DHT11 et un pour la LED (respectivement DHT11.things (figure 3.41) et LED.things). Nous avons défini la température captée comme item pour le DHT11 (figure 3.42) et l'état de la LED (ON, OFF) comme item de l'objet LED. Nous avons également mis en place un ensemble de règles correspondant au processus de chauffage dans un fichier nommé Heater.rules, présenté dans la figure 3.43 : si la température actuelle est inférieure à la

température cible, la LED s'allume, sinon elle s'éteint. Finalement, nous avons défini le *sitemap* de l'application (voir figure 3.44).

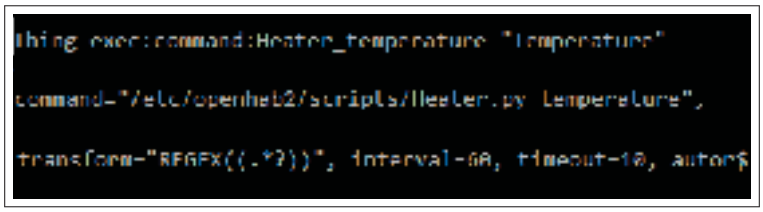


Figure 3.41 Extrait du fichier DHT11.things pour le système de chauffage intelligent

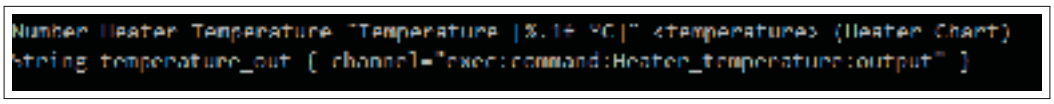


Figure 3.42 Extrait du fichier DHT11.items pour le système de chauffage intelligent

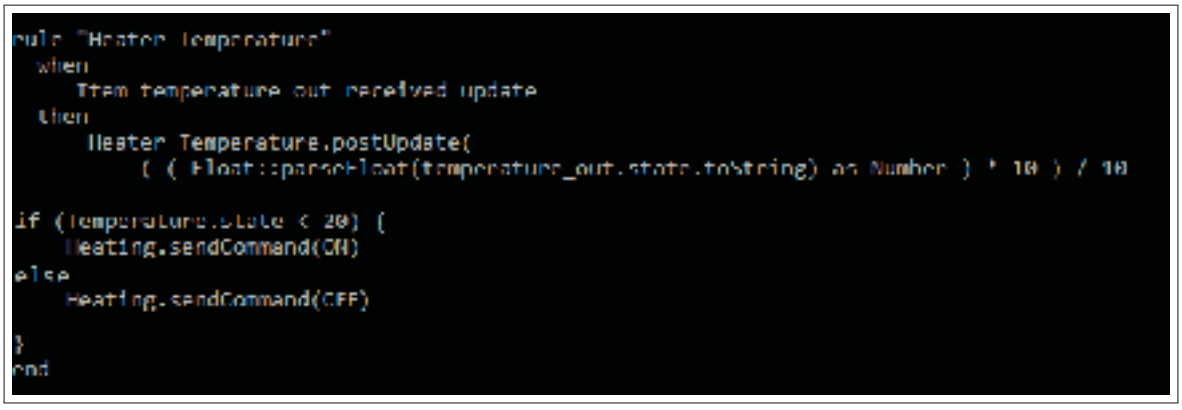


Figure 3.43 Extrait du fichier Heater.rules

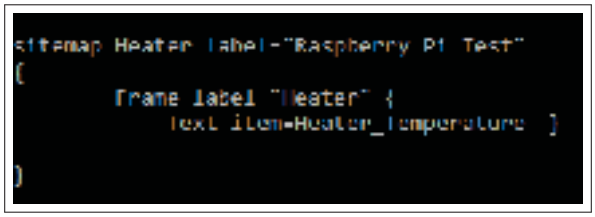


Figure 3.44 Extrait du fichier Heater.sitemap

3.5 Eclipse Kura

3.5.1 Présentation générale du framework

Eclipse Kura (Kura, 2019) est un framework open source faisant partie de l'ensemble des frameworks offerts dans le cadre du projet Eclipse IoT. Basé sur JAVA/OSGI (D'Elia, Viola, Montori, Azzoni & Maiero, 2016), Kura fournit une plate-forme pour la création de passerelles (i.e. Gateways) IoT. On entend dire par *passerelle*, le programme logiciel qui permet la connexion entre le cloud et les objets connectés, ainsi que le transit des données entre les capteurs/sources de données et les plates-formes chargées de les analyser. Ainsi, Kura agit comme un conteneur intelligent d'applications qui permet la gestion à distance des passerelles et fournit une large gamme d'API pour permettre aux développeurs d'écrire et de déployer leurs propres applications IoT. Kura permet également l'extension de son environnement via la possibilité de développer son propre programme JAVA, créer l'exécutable (.jar) et l'utiliser dans l'environnement Kura. Plusieurs programmes sont déjà prêts à l'utilisation et téléchargeables à partir de *Eclipse Marketplace*.

Kura offre plusieurs services offerts présentés dans la figure 3.45.

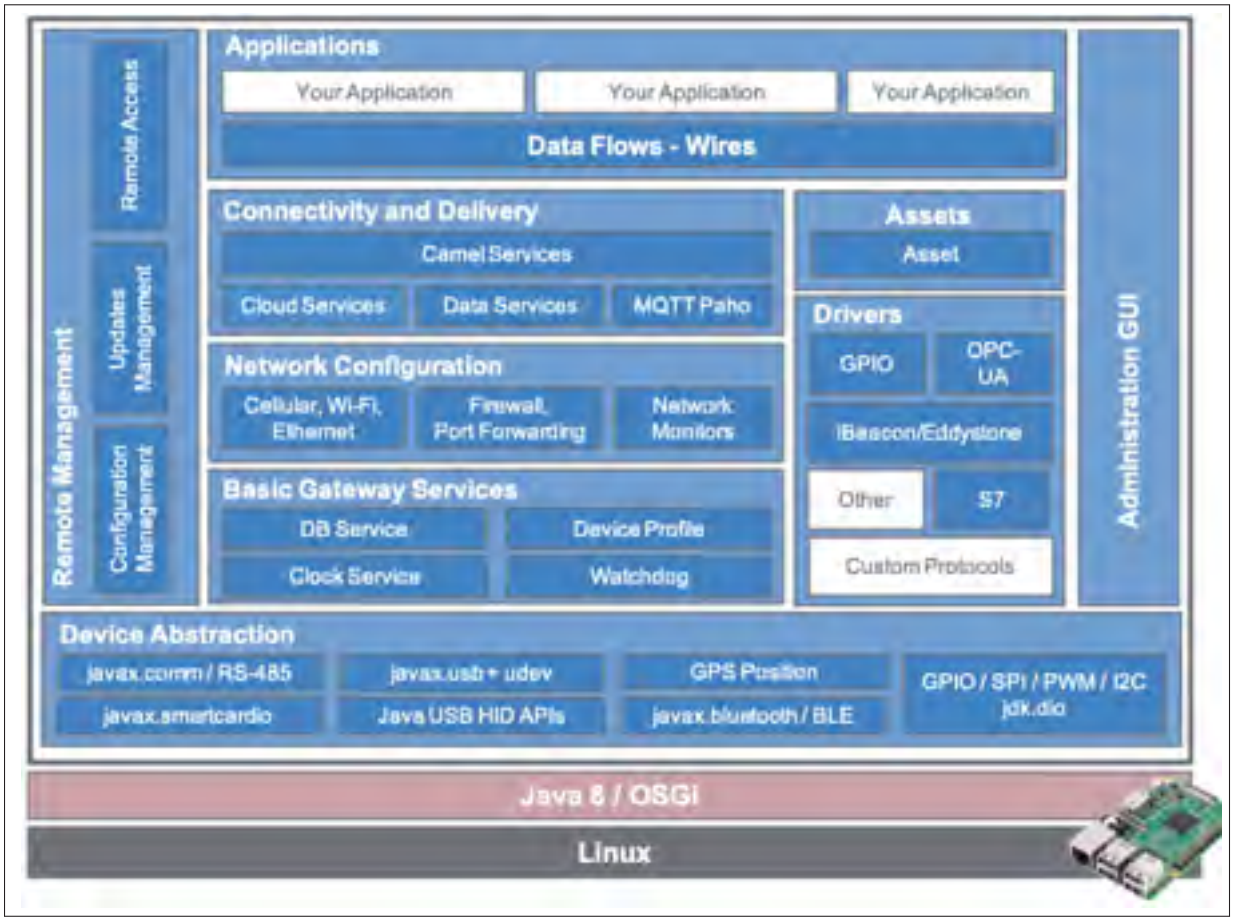


Figure 3.45 Services offerts par Eclipse Kura, extrait de (Kura, 2019)

Ces services incluent :

- services I/O : accès Bluetooth, service de positionnement pour GPS, service de synchronisation de l'heure, des APIs pour l'accès aux GPIO/PWM/I2C/SPI ;
- services de données lesquels comprennent :
  - fonctionnalité de stockage et de transfert des données collectées par la passerelle et publiées sur des serveurs distants ;
  - système de publication axé sur les politiques : permet au développeur d'applications de ne pas se préoccuper de la complexité de la couche réseau et du protocole de publication utilisé. Eclipse Paho et son client MQTT fournissent la bibliothèque de messagerie utilisée par défaut.

- services Cloud : une API pour permettre aux applications IoT de communiquer avec les serveurs distants ;
- services de gestion à distance (i.e. remote management) : permet la gestion à distance des applications IoT ; cela inclut le déploiement, la mise à niveau et la gestion de la configuration ;
- services de réseau : une API pour la configuration des interfaces réseau (Wifi, Ethernet,etc.) ;
- services *Watchdog* : permettent d'enregistrer les objets dits critiques pour de la restauration du système en cas de problèmes/erreurs ;
- interface d'administration Web : une console de gestion basée sur le Web ;
- drivers : supportent la communication entre les objets attachés aux passerelles. Cela comporte le protocole de communication ainsi que ses paramètres de configuration ;
- l'outil *Wires* (figure 3.46) : offre un environnement de développement graphique permettant de créer des flux de données des systèmes IoT.

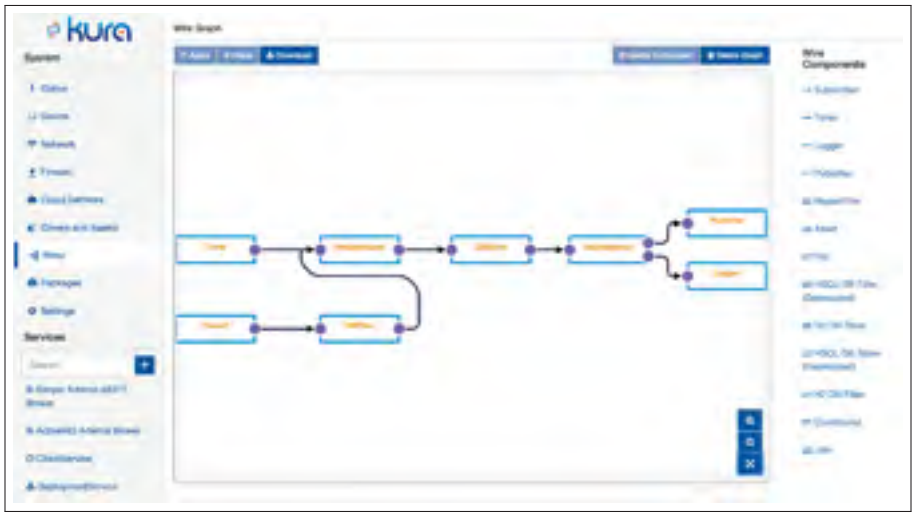


Figure 3.46 Éditeur de flux de données de Eclipse Kura

Développer une application avec Kura revient à développer du code Java, relatif au fonctionnement de chacun des objets (Publisher, subscriber,etc.), produire le fichier exécutable et l'installer dans l'environnement Kura ; ce code peut également être partagé sur Eclipse Marketplace. Ensuite, à l'aide de l'éditeur de flux, définir les interactions entre les différents objets et définir

les configurations requises, par exemple, identifier les numéros des Pins où sont branchés les différents objets.

3.5.2 Implémentation du système de gestion d'inventaire

Nous n'avons pas pu implémenter la première catégorie avec Eclipse Kura. Cela est dû au fait que ce framework ne supporte pas le protocole de communication via USB, requis par le lecteur de tags RFID que nous utilisons dans l'expérimentation.

3.5.3 implémentation du système de surveillance météorologique

Pour implémenter le système de surveillance météorologique avec Eclipse Kura, nous avons utilisé l'éditeur graphique de Kura et nous avons implémenté les différents objets en JAVA. Concrètement, et en premier lieu, nous avons créé l'application de surveillance météorologique graphiquement comme présentée dans la figure 3.47. Le noeud *Timer* permet d'initialiser et de déclencher périodiquement la mesure de la température. Le noeud *TemperatureSensor* représente le capteur de température DHT11. Afin d'effectuer la mesure à partir des Pins de la Raspberry, nous avons configuré le noeud comme présenté dans la figure 3.48. Le noeud *TemperatureSensor* est relié à un noeud *Publisher* qui est responsable de publier les valeurs mesurées via le protocole MQTT. Nous avons utilisé un *package* disponible sur Eclipse Marketplace et qui implémentait un *Publisher*. Nous avons configuré ce *Publisher* selon nos besoins, par exemple pour spécifier la fréquence des publications, tel que présenté dans la figure 3.49).

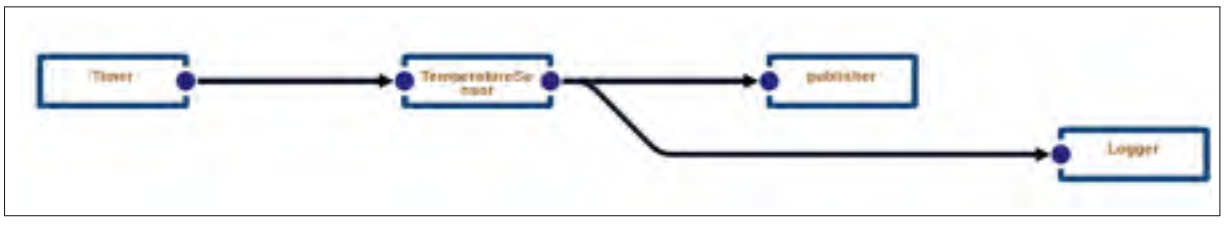


Figure 3.47 Flux graphique réalisé avec Kura

Figure 3.48 Configuration du noeud *TemperatureSensor*

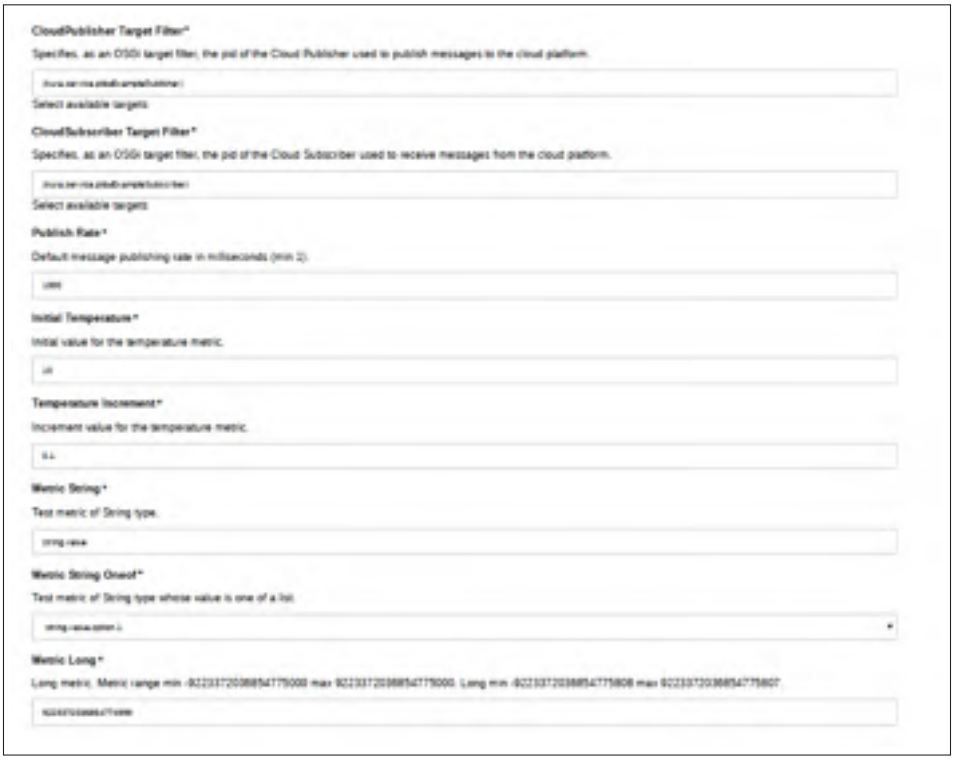


Figure 3.49 Configuration du *Publisher*

Finalement, nous avons établi la connexion entre les différents objets et testé l'application. En plus de la valeur de température mesurée (2ème valeur encadrée en rouge dans la figure 3.50), Kura retourne le *topic* (1ère valeur encadrée en rouge dans la figure 3.50), ainsi que les différentes valeurs des paramètres configurés précédemment.

Figure 3.50 Publication des valeurs configurées et de la température mesurée sur la Raspberry Pi

3.5.4 Implémentation du système de chauffage intelligent

Nous n'avons pas pu implémenter l'application de chauffage intelligent avec Eclipse Kura. Nous avons réalisé plusieurs tentatives sans succès. Nous faisons l'hypothèse que ceci est dû au fait que cette application nécessite que la valeur de la température soit comparée à une autre valeur cible et qu'une décision soit prise pour agir sur la LED (allumer ou éteindre la LED). En d'autres termes, et d'un point de vue développement informatique, un traitement doit avoir lieu sur la valeur mesurée et les deux objets (lampe et capteur) doivent s'échanger des données. Kura permet de publier la valeur mesurée, mais il ne permet pas de stocker temporairement pour permettre le traitement. La seule méthode de stockage des données est de passer par une base de données.

CHAPITRE 4

ANALYSE DES RÉSULTATS EXPÉRIMENTAUX

Dans ce chapitre, nous discutons des résultats de notre étude en fonction de deux questions de recherche (QR1 et QR2) énoncées au Chapitre 2. En réponse à la question QR1, nous discutons de façon générale du support offert par les frameworks étudiés pour développer une application IoT et cela en se basant sur nos expériences avec ces frameworks. En réponse à la question QR2, nous présentons notre évaluation de ces frameworks par rapport à l'ensemble des exigences minimales que nous avons établi au Chapitre 2.

4.1 QR1 : Jusqu'à quel degré un framework open source supporte le développement des applications IoT ?

Dans cette section, nous reportons et discutons des aspects qui ont facilité, ou rendu difficile, notre tache de développer et implémenter les trois exemples d'applications IoT ciblées par notre expérimentation. Nous basons cette analyse principalement sur :

- la facilité d'installation des frameworks ;
- la facilité d'apprentissage et de manipulation du framework ;
- la documentation disponible ;
- les difficultés rencontrées lors du développement et déploiement des différentes applications.

Nous présentons cette analyse par framework.

4.1.1 Eclipse Vorto

Au moment où nous avons commencé nos expérimentations, la version disponible de Vorto, et que nous avons utilisée, ne supporte pas le développement pour la Raspberry Pi. Pour cela, notre travail s'est limité à la définition des "interfaces" des objets avec le DSL de Vorto. Nous n'avons donc pas pu déployer notre code source sur la Raspberry Pi. Le code généré ne nous a pas été utile pour différentes raisons :

- le code généré est un squelette de code. Son utilisation nécessite qu'on le complète manuellement ;
- dans le code généré, certains aspects ne sont pas pris en considération. Par exemple, dans le code vorto de la deuxième catégorie (figure 3.4, chapitre 3), nous avons défini des variables du DHT11 comme étant *ReadOnly* (à travers la définition de `readable : true, writable : false`). Cependant, dans le code généré, une méthode *getter* et une méthode *setter* ont été générées, comme montré dans la figure 4.1 ;
- le code généré est destiné à être utilisé pour une plate-forme bien déterminée ; dans la figure 3.4 présentée au chapitre 3, par exemple, il s'agit de la plate-forme Bosch IoT Suite.

```
package device.sensortempreture.model;

public class SensorTempreture {
	private Sensor sensor;

	private String resourceId;

	public SensorTempreture(String resourceId) {
		this.resourceId = resourceId;
	}

	public Sensor getSensor() {
		return sensor;
	}

	public void setSensor(Sensor sensor) {
		this.sensor = sensor;
	}

	public void setResourceId(String resourceId) {
		this.resourceId = resourceId;
	}

	public String getResourceId() {
		return resourceId;
	}
}
```

Figure 4.1 Extrait du code généré par Vorto pour la *FunctionBlock* décrivant le DHT11

Nous avons étudié la version d'Eclipse Vorto (parue en Fév 2019) qui a suivi celle que nous avons utilisée. Cette nouvelle version offre un DSL plus riche pour décrire des éléments reliés à la Raspberry Pi. Cependant, même avec cette nouvelle version, nous n'étions pas encore capable de déployer nos objets sur la Raspberry. Toutefois, il est à noter que 18 versions ont succédé la

version que nous avons étudiée. Ceci s'explique par le fait que la compagnie mondiale Bosch s'est impliquée dans Vorto en l'utilisant pour définir et connecter ces appareils. En fait, Vorto ne supporte pas le développement sur la Raspberry car la majorité des efforts ont été mis sur le développement et le partage d'objets (sur le répertoire Vorto) correspondant à des équipements électroménagers.

Pour conclure, Vorto est un excellent outil pour l'abstraction et la définition des objets. Grâce à son DSL, l'objet et son mode de fonctionnement sont bien définis. Toutefois, Vorto n'est clairement pas destiné à l'exécution des applications IoT. Notre expérimentation avec les 3 exemples d'applications avec Vorto s'est limitée à la description du fonctionnement des objets (i.e. Lecteur de tags pour la 1ère catégorie, DHT11 pour la 2e catégorie et DHT11, l'unité de contrôle et le chauffage pour la 3e catégorie). Vorto aurait pu être plus puissant s'il fournissait au développeur la possibilité de décrire davantage les relations/interactions des objets. Actuellement et ce, dans le cas de la version étudiée, soit la V0.10.0 M6, seules les entêtes des méthodes sont décrites. Cela explique le fait que les générateurs de code Vorto ne produisent qu'un squelette de code. Le fait de pouvoir définir la méthode et les différentes variables impliquées pourrait permettre de générer un code complet, et donc de pouvoir rouler le code généré sur une plate-forme matérielle.

4.1.2 ThingML

Malgré le fait que nous avons pu implémenter les trois exemples d'applications avec ThingML, ce dernier a été un des frameworks les plus difficiles à utiliser. En fait, le problème majeur que nous avons eu avec ThingML était l'absence de la documentation. Nous avons dû documenter nous-mêmes le DSL de ThingML à partir d'exemples afin de l'utiliser. Cela a présenté des défis et des confusions, car les exemples disponibles utilisaient différentes versions du framework. De plus, les différentes versions du framework ne mentionnaient pas explicitement les changements ni quelles parties de code sont devenues obsolètes. Sans nous en rendre compte, nous avons donc utilisé du code obsolète à certains endroits et cela a complexifié le débogage et la compilation de nos applications.

D'un autre côté, ThingML ne cible pas la Raspberry Pi en particulier. C'est pour cette raison que le DSL de ThingML ne supporte pas la lecture des valeurs des pins de la Raspberry. Cependant, ThnigML permettait d'intégrer du code "propre" à des plate-formes (e.g. des routines écrites en C). Cette caractéristique nous a permis de remédier donc au problème en intégrant du code qui supporte la lecture des valeurs des pins de la Pi. Toutefois, les erreurs reliées à ce code ne peuvent être découvertes que lors de son exécution sur la Raspberry Pi ; c'est-à-dire que le développeur est amené à développer, compiler le code, générer un exécutable et le déployer sur la Raspberry afin de l'exécuter avant qu'il sache si son code contient des erreur. Cela oblige le développeur de répéter tout ce processus plusieurs fois avant d'arriver à exécuter le code avec succès et sans erreurs. La figure 4.2 montre un exemple où la détection des erreurs est faite après l'exécution du code sur la Pi.

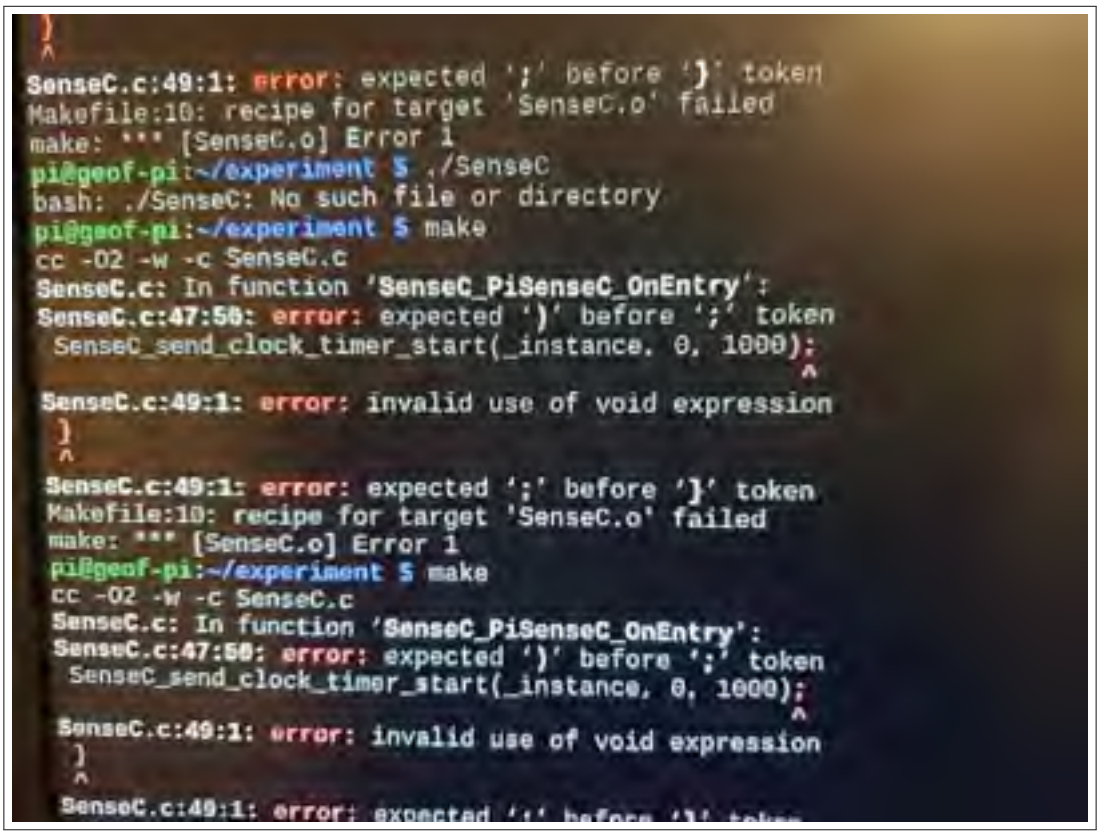


Figure 4.2 Exemples d'erreurs de compilation apparu à l'exécution du programme ThingML

4.1.2.1 Implémentation du système de gestion d'inventaire

Dans la réalisation de la 1ère catégorie d'application, nous avons remarqué que l'utilisation de ThingML n'est pas efficace. En effet, tout le traitement nécessaire a été développé sous forme de fonctions écrites en C ; ThingML est donc sans valeur ajoutée dans ce cas.

Les recherches que nous avons faites ont montré l'existence d'autres solutions plus pratiques et plus efficaces pour ce type d'applications. Les *HyperTerminaux* en sont l'exemple, en effet, le téléchargement du *HyperTerminal « gtkterm »* de la Raspberry nous épargne toutes les étapes décrites précédemment au Chapitre 3 (section 3.2.2), notamment vu que l'exécution de ThingML exige un terminal. Une simple commande permet l'installation du *gtkterm* (`sudo apt-get install gtkterm`), ensuite il suffit d'exécuter le programme (`sudo gtkterm`) et de configurer le port sur lequel le lecteur RFID est branché (Figure 4.3).

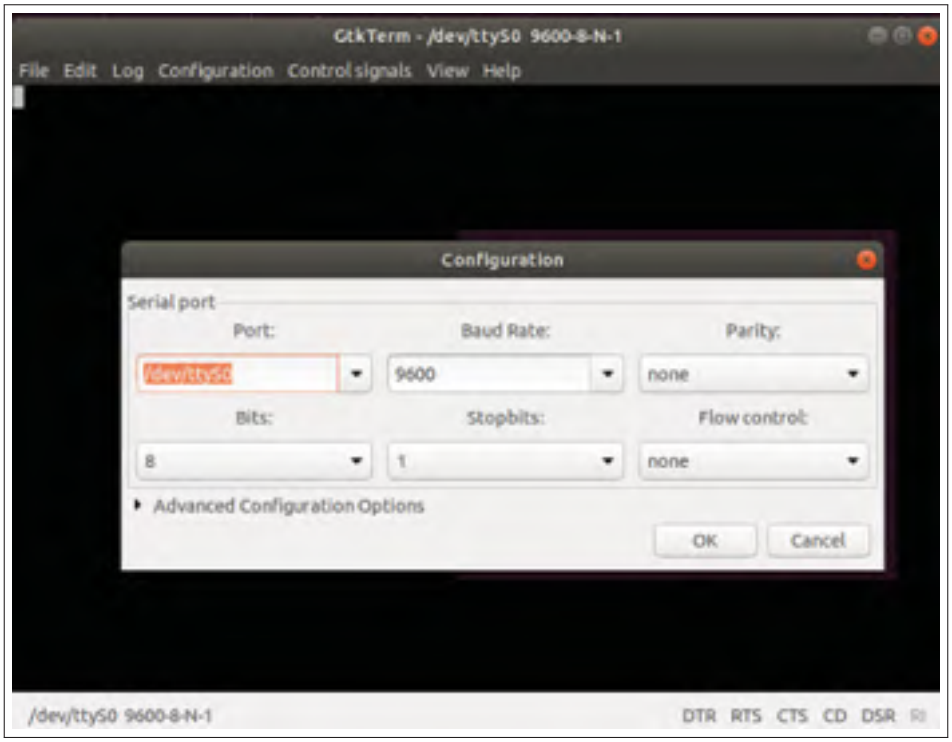


Figure 4.3 *Hyperterminal "Gtkterm"* pour la réalisation de l'application de la 1ère catégorie

4.1.2.2 Implémentation du système de surveillance météorologique

L'implémentation de la deuxième catégorie d'application était faisable avec ThingML. Bien que cela nécessitait une connaissance du code C pour la Raspberry (i.e. les librairies C pour la manipulation des GPIO de la Raspberry), la tâche ne nous a pas posé problème. La visualisation de la température capturée se fait sur le terminal de la Raspberry ; il est à noter que l'aspect visuel n'est pas très mature, mais l'aspect fonctionnel est bien supporté.

Grâce à ThingML, nous avons pu définir le diagramme d'états du système, et nous avons pu ainsi effectuer la mesure de la température selon une fréquence ; i.e. toutes les une minute dans notre cas. Toutefois, cela a révélé un problème au niveau de la synchronisation de la lecture de la température. En effet, la valeur capturée et comparée à la valeur du champ de validation *checksum* n'est parfois pas valide (figure 4.4).

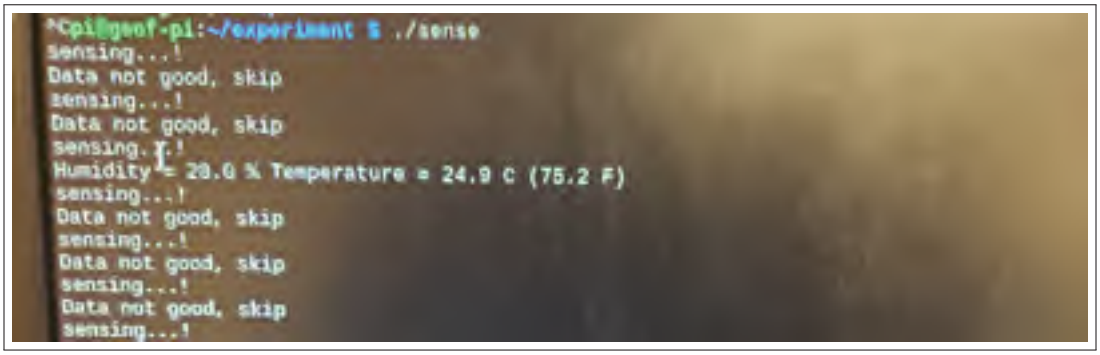


Figure 4.4 Résultats des mesures de la température avec ThingML

4.1.2.3 Implémentation du système de chauffage intelligent

L'implémentation de cette application a posé quelques défis. Ces défis étaient reliés à la difficulté de transférer les valeurs des variables entre les différents things. La valeur capturée par le DHT11 devrait être envoyée au Heater (lampe LED). Cependant, la mise en oeuvre de ce genre de transfert n'était pas documenté ni illustré dans les exemples fournis par ThingML. Nous avons dû demander de l'aide aux développeurs du framework sur github. La solution qui nous a été

proposée consiste à créer un *Thing* qui est responsable de la réception de la valeur de température mesurée via un évènement interne. Nous avons pu échanger les valeurs mesurées grâce à cette méthode.

4.1.3 Node-Red

Du point de vue temps passé pour le déploiement des différents exemples, Node-Red a été le plus pratique et facile. Cela est dû, d'une part, aux nœuds préprogrammés et faciles à utiliser. En effet, Node-Red offre une palette de noeuds dédiés au développement sur la Raspberry Pi, ce qui a facilité énormément notre travail. D'autre part, cela est dû à la documentation « abondante » disponible sur le site de Node-Red. Une communauté importante fournit des exemples d'applications facilitant ainsi la découverte de Node-Red pour les développeurs moins expérimentés.

4.1.3.1 Implémentation du système de gestion d'inventaire

La réalisation de la 1ère catégorie a été facile avec Node-Red. Cependant, nous estimons que la tâche aurait été plus complexe en absence du tutoriel que nous avons trouvé et suivi. En effet, la documentation du lecteur de tags RFID ne fournit pas l'information complète sur le formatage de l'ID. Aussi, l'installation des noeuds relatifs à la gestion des connexions via les ports USB (voir figure 4.5) n'était pas très évidente à cause de certaines dépendances entre paquetages.

Figure 4.5 Noeuds de Node-Red pour la gestion des connexions via le port USB

4.1.3.2 Implémentation du système de surveillance météorologique

La réalisation de l'application de la deuxième catégorie était très facile sur Node-Red et ce, grâce à plusieurs facteurs, dont :

- l'existence d'un noeud prédéfini "DHT11" qui prend en paramètre le numéro de la PIN sur laquelle est branché le capteur DHT11 et qui lit directement les valeurs mesurées ;
- l'outil graphique fourni par Node-Red est très intéressant pour ce type d'application, dont l'ultime but est de mesurer et d'afficher des variables d'environnement ;
- les noeuds pré-développés permettent aux concepteurs des applications de développer encore plus leurs idées. Dans notre cas, nous avions pour but de développer une application qui permet de visualiser la température actuelle. Avec Node-Red, nous sommes allés plus loin en créant un fichier où on stocke les valeurs de température. Aussi, à chaque mesure, un émail est envoyé à l'adresse définie au préalable. Ces ajouts découlent principalement de la disponibilité de noeuds pré-développés et faciles à configurer.

4.1.3.3 Implémentation du système de chauffage intelligent

La réalisation de la 3ème catégorie a aussi été facile grâce aux noeuds dédiés à la Raspberry. En effet, le transfert d'informations entre les différents objets connectés à la Raspberry se fait de façon fluide. Nous avons eu qu'à définir le mode (entrée ou sortie) et préciser les Pins sur lesquelles les différents objets sont connectés.

4.1.4 OpenHab

OpenHab est assez puissant et riche en services fournis. Il offre aussi un excellent support pour l'aspect visuel des application. La gestion est facilitée pour les utilisateurs (Web/Application mobile/Système d'exploitation Linux,etc.). Cependant, OpenHab est restreint au domaine des maisons intelligentes ; il ne supporte pas d'autres secteurs impliqués dans l'IoT. Cela constitue

une limitation et explique pourquoi nous n'avons pas pu implémenter la première catégorie d'application (i.e. le système de gestion d'inventaire).

4.1.4.1 Implémentation du système de surveillance météorologique

Pour implémenter l'application de la deuxième catégorie, nous avons utilisé le système dédié d'OpenHab (OpenHabian). L'application était facile à réaliser grâce à l'implémentation d'un programme Python ainsi que des compléments offerts par OpenHab (*Bindings*).

4.1.4.2 Implémentation du système de chauffage intelligent

Dans la réalisation de l'application de la 3ème catégorie, nous avons eu à implémenter des fichiers de configuration de l'application. Ceci a ajouté un niveau de complexité à la tâche, comparée à la deuxième catégorie d'application.

4.1.5 Eclipse Kura

L'implémentation des différentes applications avec Eclipse Kura n'a pas été facile depuis les premières itérations. En effet, l'installation du framework sur la Raspberry a nécessité un effort particulier pour que le statut du framework soit "Activé". Pour résoudre ce problème, nous avons réessayé l'installation de Kura en suivant à chaque fois une méthode différente.

Théoriquement, Kura offre une multitude de services, très poussés et très puissants pour couvrir les différents aspects en IoT. Cela dit, l'expérimentation des différents aspects de Kura a montré un niveau de complexité assez important et nous avons fait face à plusieurs difficultés. Cela est peut-être dû au fait que Kura est encore sous tests et développement/améliorations. Balakrishnan, Babu, Naiju & Madiajagan (2019) montrent que l'implémentation d'une application IoT avec Kura nécessite l'utilisation d'autres frameworks IoT (tels que Eclipse Hono).

4.1.6 Sommaire de l'évaluation du support des frameworks étudiés

Dans le tableau 4.1, nous récapitulons notre évaluation des différents frameworks étudiés en fonction du support offert pour chacune des 3 catégories d'application visées par notre expérimentation.

Tableau 4.1 Le support fourni par les frameworks pour l'implémentation des exemples d'applications choisis

	1ère catégorie	2ème catégorie	3ème catégorie
Eclipse Vorto	×	×	×
ThingML	−	±	±
Node-Red	±	+	+
OpenHab	×	+	±
Eclipse Kura	×	±	×

+ : Implémentation facile ;
± : Implémentation avec quelques difficultés ;
− : Implémentation très difficile ;
× : Pas faisable

4.2 QR2 : Jusqu'à quel degré un framework open source supporte un ensemble minimal d'exigences fonctionnelles et non fonctionnelles des applications IoT ?

Dans cette section, nous évaluations les frameworks étudiés en fonction d'un ensemble minimal des exigences des applications IoT. Comme discuté au Chapitre 2 (section 2.5), cette liste d'exigences a été établie en se basant sur la littérature et le standard ISO RA IoT. Nous discutons d'abord, de façon générale, les résultats de nos expérimentations selon notre liste des exigences IoT. Ensuite, nous faisons une analyse par framework du support de notre liste d'exigences.

4.2.1 Analyse selon les exigences IoT

Le tableau 4.2 résume les résultats de notre analyse des frameworks étudiés, selon notre liste d'exigences.

Tableau 4.2 Évaluation des frameworks étudiés selon notre liste d'exigences

Critère	Eclipse Vorto	ThingML	Node-Red	OpenHab	Eclipse Kura
Spécification de l'interface de l'objet	✓	✓	×	✓	✓
Spécification du comportement de l'objet	×	✓	×	✓	×
Spécification des propriétés de l'objet	×	×	×	×	×
Sécurité-Authentification	✓	×	✓	✓	✓
Sécurité-Cryptage des données	×	✓	✓	✓	✓
Sécurité-Intégrité des données	×	✓	✓	×	×
Gestion de l'hétérogénéité	✓	✓	✓	✓	✓
Tolérance aux fautes-spécification des erreurs	✓	✓	×	✓	×
Tolérance aux fautes-Détection des erreurs en cours de l'exécution	×	×	✓	✓	✓
Intégration des composants COTS	×	×	✓	✓	✓
Découvrabilité	×	×	×	✓	×
Génération d'un code complet	×	✓	✓	✓	✓
Stockage des données	×	✓	✓	✓	✓
Visualisation	×	×	✓	✓	✓
Simulation	×	×	✓	✓	✓
Facilité de déploiement sur la Raspberry Pi	×	×	✓	✓	✓
Exécution sur la Raspberry Pi	×	✓	✓	✓	✓

De façon générale, nous remarquons qu'aucun des frameworks ne satisfait toutes les exigences. En particulier, Eclipe Vorto est le framework qui fournit le moins de support aux développeurs dans la création des applications IoT. C'est explicable par le fait que Vorto peut être considéré comme un langage de description d'interface axé sur la gestion de l'hétérogénéité. Pour ce qui est des exigences, tous les frameworks supportent la gestion de l'hétérogénéité. Cela est dû au fait que les applications IoT sont généralement composées de divers objets et services et l'interopérabilité entre ces objets et services est nécessaire. À l'inverse, aucun des frameworks ne permet de spécifier les propriétés des objets (c'est-à-dire les caractéristiques matérielles). Ceci pose problème car de nombreux objets IoT peuvent avoir des capacités matérielles limitées

(c'est-à-dire mémoire ou processus) lequelles doivent être prises en considération lors du déploiement, mais également lors de l'exécution des applications IoT.

OpenHab et ThingML sont les frameworks qui offrent le plus de support pour la spécification des objets ; i.e. ils prennent en charge la spécification des objets en termes d'entrées/sorties et de comportement. Bien qu'il soit largement utilisé, Node-Red ne prend pas en charge la spécification de l'interface des objets ni leur comportement. Ceci est dû au fait que Node-Red est basé sur la description du flux de données. Eclipse Kura supporte la définition des entrées/sorties des objets, mais ne permet pas la définition de leurs comportements.

En ce qui concerne la sécurité, la plupart des frameworks fournissent un certain support. Dans Eclipse Vorto, l'utilisateur doit s'authentifier pour se connecter au reposoir Vorto et accéder à la spécification des objets (pour gérer et modifier ses objets). Dans Node-Red, OpenHab et Kura, il existe des APIs/des compléments qui peuvent être téléchargés et installés pour garantir un accès contrôlé aux objets et à l'application en cours de développement. Node-Red est le framework qui facilite le plus la mise en œuvre de la sécurité. Par exemple, Node-Red fournit des nœuds explicites pour prendre en charge le cryptage des données.

En ce qui concerne la tolérance aux fautes, Node-Red et Kura ne permettent pas la spécification explicite des erreurs lors de la conception d'une application. Pour Node-Red, ceci est dû au fait qu'il spécifie le flux des données et non pas le comportement des objets. D'un autre côté, Node-Red, OpenHab et Kura prennent en charge la détection des erreurs lors de l'exécution ; ces frameworks permettent de détecter les erreurs et de créer des fichiers logs.

De plus, Node-Red, OpenHab et Kura prennent en charge la simulation de l'application, la visualisation des données et l'intégration des composants COTS, ce qui rend ces trois frameworks très attrayants pour les développeurs. Quant au stockage des données, ThingML, Node-Red, OpenHab et Kura permettent de conserver les données ; ThingML nécessite de développer une fonction complémentaire pour le faire.

Enfin, ThingML, Node-Red et OpenHab permettent de générer un code source complet. Cependant ThingML nécessite l'implémentation manuelle de plusieurs fonctions spécifiques à la plate-forme visée et le déploiement du code sur la Raspberry Pi nécessite quelques efforts. Quant à Node-Red, OpenHab et Kura, ils sont installés sur la Raspberry Pi ; cela facilite l'exécution du code résultant sur la Pi.

4.2.2 Analyse par framework

Dans cette section, nous discutons du niveau de support offert par chaque framework pour les exigences prises en charge par le framework et nous proposons quelques pistes d'améliorations selon nos expérimentations.

Pour le niveau de support offert, nous utilisons une échelle simple à deux niveaux :

- satisfaisant (illustré dans les tableaux par le symbole +) : nous jugeons satisfaisant le niveau de support d'une exigence si cette dernière est supportée implicitement par le framework ; c'est-à-dire qu'aucune action n'est requise de la part du développeur pour satisfaire cette exigence, ou si les actions requises par le développeur sont minimes (par exemple, activer une option déjà existante, exécuter un script fourni par le framework, etc.) ;
- nécessitant des améliorations (illustré dans les tableaux par le symbole ±) : nous jugeons qu'il y a place pour des améliorations dans deux cas de figures :
  - le développeur doit fournir un effort supplémentaire pour satisfaire l'exigence en utilisant que le framework (par exemple, développer une fonction dans un langage particulier) ;
  - l'exigence est partiellement supportée ; c'est-à-dire que théoriquement, le framework permet de supporter une exigence, mais pratiquement, des actions additionnelles doivent être réalisées pour satisfaire l'exigence (par exemple, pour la sécurité, un framework supporte l'authentification, mais, sur le plan pratique, cette fonctionnalité peut facilement être contournée et l'accès peut être établi).

4.2.2.1 Eclipse Vorto

Vorto était le framework qui supporte le moins d'exigences (4/17). Le tableau 4.3 résume le degré de support offert par Vorto pour les exigences qu'il prend en charge.

Tableau 4.3 Degré de support des exigences prises en charge par Eclipse Vorto

Exigence	Évaluation
Spécification de l'interface de l'objet	±
Sécurité-Authentification	+
Gestion de l'hétérogénéité	+
Tolérance aux fautes-spécification des erreurs	+

Pour la spécification de l'interface de l'objet, Vorto permet la définition des variables qu'un objet aura à traiter dans son processus de fonctionnement. Toutefois, le DSL de vorto ne permet de spécifier si la variable est une entrée ou une sortie. Dans notre cas, nous avons mis cette information en commentaire comme étant une description de la variable. Il est clair que le DSL doit intégrer des moyens pour permettre de différencier ces deux types de variables.

En ce qui concerne l'aspect sécurité, la seule exigence supportée dans Vorto est l'authentification ; un développeur peut accéder et consulter la définition de l'ensemble des objets définis à partir du reposoir Vorto (Figure 4.6). Toutefois, seul l'auteur de la définition de l'objet peut la modifier, et ce, en s'authentifiant soit avec son compte Github (pour les développeurs débutants/n'appartenant pas à la communauté Bosch), soit avec un Identifiant Bosch (Figure 4.7). À ce stade de notre étude, nous évaluons ce processus comme satisfaisant.

Le but ultime du DSL de Vorto (selon sa description dans (Vorto, 2016)) est de gérer l'hétérogénéité des objets en IoT. Notre expérience avec Vorto le confirme. En effet, nous avons pu définir nos objets, quelques soient leurs natures grâce à Vorto et nous avons ainsi pu pallier à leur hétérogénéité.

Pour la tolérance aux fautes, Vorto supporte la spécification des erreurs grâce à son DSL. En effet, Vorto inclut une section dans le FunctionBlock, s'appelant Fault, qui permnet aux développeurs

Figure 4.6 Reposoir des objets de Vorto

Figure 4.7 Méthode d'authentification au reposoir de Vorto

de spécifier les éventuelles erreurs. En d'autres termes, le développeur peut définir les cas où une erreur peut se produire à travers la comparaison des variables (par exemple, si une lampe est à l'état OFF et que sa consommation d'énergie est supérieure à zéro, c'est certainement une erreur), ou le suivi des variables (par exemple, la valeur du taux de glycémie d'un patient mis sous contrôle ne doit jamais atteindre les 2g, sinon c'est une erreur du système de gestion de la dose des médicaments). Nous jugeons cette exigence comme satisfaisante puisqu'il permet au développeur de prévoir les éventuelles erreurs et de les prendre en considération lors du développement.

4.2.2.2 ThingML

Sur un total de 17 exigences, ThingML en supporte 9. Notre évaluation du support fourni par ThingML aux différentes exigences est résumée dans le tableau 4.4 et discutée ci-après.

Tableau 4.4 Degré de support des exigences prises en charge par ThingML

Exigence	Évaluation
Spécification de l'interface de l'objet	±
Spécification du comportement de l'objet	+
Sécurité-Cryptage de données	±
Sécurité-Intégrité des données	±
Gestion de l'hétérogénéité	±
Tolérance aux fautes-spécification des erreurs	+
Génération d'un code complet	+
Stockage de données	±
Exécution sur la Raspberry Pi	+

La spécification de l'interface de l'objet consiste à définir les variables en entrée/sortie à traiter par l'objet. Dans ThingML, la définition et le traitement des données en sortie de l'objet ne nous ont pas posé problème (i.e. était faciles à réaliser). Toutefois, les variables entrantes nous ont posé problème comme discuté précédemment à la section 4.1.2. Des améliorations au framework sont nécessaires pour prendre en charge les entrées d'un objet.

ThingML permet la spécification du comportement d'un objet à travers la définition d'une machine à états. Ceci est un point fort pour ThingML qui pourrait être exploité pour développer davantage ThingML, notamment pour supporter la simulation.

Le cryptage de données n'est pas supporté directement par ThingML. Cependant la possibilité d'intégrer du code propre à une plate-forme (JAVA, C, etc.) permet le chiffrement/déchiffrement des données ; par exemple développer une méthode JAVA pour chiffrer et déchiffrer des données. Toutefois, cela exige qu'on ait de très bonnes connaissances en développement et en cryptage des données, d'où notre évaluation de l'exigence. L'intégrité des données suit la même logique que pour le cryptage des données. Le développeur voulant assurer l'intégrité de ces données aura à développer du code pour le faire. Plusieurs scénarios peuvent être mis en place dans ce cas : le

développeur peut stocker ses données sur une base de données ou dans un fichier sécurisé et pourra y faire référence afin de s'assurer de l'intégrité de ses données. En fait, le stockage de données est une exigence supportée par ThingML, mais à travers l'ajout de code C/Java, etc. Tout cela exige que le développeur ait des connaissances avancées en développement et rend la tâche complexe pour les développeurs qui ont moins d'expérience.

D'après nos résultats, nous avons remarqué que tous les frameworks étudiés supportent la gestion de l'hétérogénéité, mais à des degrés différents. Pour ThingML, la définition des objets un par un ne nous a pas posé problème, ce qui est assez important pour la gestion de l'hétérogénéité (quelles que soient les différences des objets, la méthode de leur définition est la même). Toutefois, nous avons remarqué que faire communiquer les différents objets pose problème dans ThingML. Ceci est au coeur des préoccupations reliées à la gestion de l'hétérogénéité ; comment faire communiquer plusieurs objets de différentes natures ? Cet enjeu a impacté notre évaluation du support de l'exigence de la gestion de l'hétérogénéité par ThingML.

ThingML offre la possibilité de spécifier les erreurs à travers la définition d'un état (par exemple, *error*) qui définit les cas où une erreur peut se produire. Le développeur peut se contenter de cette fonctionnalité ou développer davantage son code (par exemple pour lever des exceptions dans les cas d'erreur, planifier des actions à mettre en place en cas d'erreur,etc.).

ThingML permet la génération d'un code complet et prêt à être exécuté directement sur la plate-forme ciblée (dans notre cas, la Raspberry Pi). Ceci rend l'exécution de l'application sur la Raspberry facile à réaliser.

4.2.2.3 Node-Red

Node-Red supporte 12 exigences parmi les 17 exigences de notre liste. Notre évaluation du degré de support fourni par Node-Red pour ces 12 exigences est résumée dans le tableau 4.5 et discutée ci-après.

Tableau 4.5 Degré de support des exigences prises en charge par Node-Red

Exigence	Évaluation
Sécurité-Authentification	+
Sécurité-Cryptage de données	+
Sécurité-Intégrité des données	+
Gestion de l'hétérogénéité	+
Tolérance aux fautes-Détection des erreurs en cours de l'exécution	+
Intégration des composants COTS	+
Génération d'un code complet	+
Stockage de données	+
Visualisation	+
Simulation	+
Facilité de déploiement sur la Raspberry Pi	+
Exécution sur la Raspberry Pi	+

Par défaut, Node-Red est ouvert et accessible à toute personne ayant l'adresse IP (IP_de_-l'utilisateur :1880, par défaut). Toutefois, afin d'assurer la sécurité, l'authentification peut être activée dans Node-Red, en suivant les étapes décrites dans (Node-red, 2018). La sécurisation comprend 2 parties (voir détails en annexe VI) :

- sécurisation de l'éditeur et de l'API admin ;
- sécurisation des nœuds HTTP et du tableau de bord Node-Red.

Node-Red offre une palette de noeuds permettant le cryptage/décryptage et sécurisation des données (figure 4.8). Les noeuds de cette palette ne sont pas installés par défaut. Ils peuvent être installés à partir de l'interface de Node-Red (voir figure 4.9) ou en exécutant manuellement une commande (figure 4.10)

Node-Red supporte l'intégrité des données à travers la possibilité de stocker les informations, cryptées, dans un support (base de données, fichiers) ou de les envoyer en temps réel par courriel (ou les partager sur un site). Le développeur gardera ainsi trace de ses données et pourra décider de leur intégrité.

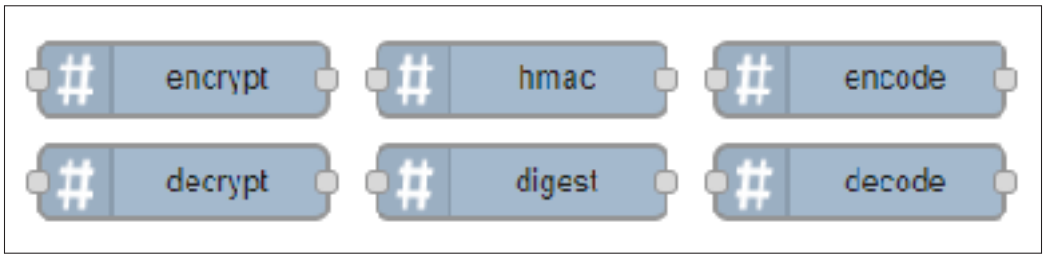


Figure 4.8 Palette de noeuds offerts pour le cryptage/décryptage avec Node-Red

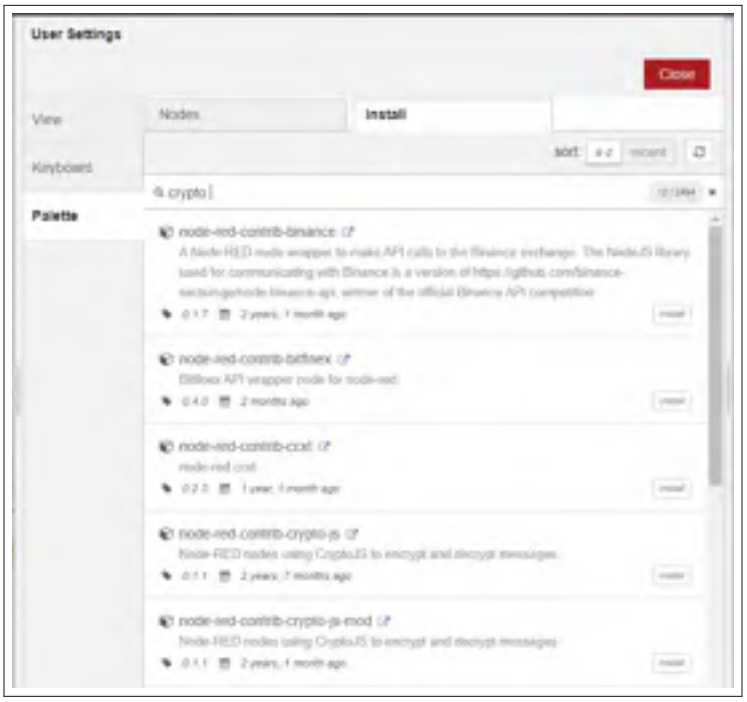


Figure 4.9 Installation des noeuds de cryptage/décryptage à partir de l'interface de Node-Red

Node-Red adopte une programmation basée sur les flux. Belsa, Sarabia-Jacome, Palau & Esteve (2018) expliquent l'efficacité d'une telle méthode de programmation dans la gestion de l'inter-

```
npm install node-red-contrib-crypto-js
```

Figure 4.10 Installation manuelle des noeuds de cryptage/décryptage avec Node-Red

opérabilité des objets IoT. Ceci permet la gestion de l'hétérogénéité. Node-Red est aussi basé sur la programmation événementielle (*event-based*). Le développement avec Node-Red consiste à définir le flux de données sans erreurs. Les erreurs, si elles ont eu lieu, sont enregistrées dans un fichier Log (grâce au noeud *Logger*), ou affichées dans le panneau d'informations et de débogage au moment de la simulation de l'application. Node-Red offre donc un très bon support pour l'exigence de détection des erreurs au cours de l'exécution.

Parmi les frameworks que nous avons étudiés, Node-Red est le meilleur en termes de support de l'exigence d'intégration des composants COTS. Ceci a rendu notre tâche de développement très facile et efficace en terme de temps de développement. Le support de l'exigence de l'intégration des composants COTS dans le cas de Node-Red concerne aussi bien les noeuds pré-développés que les APIs et les services disponibles pour intégration directe. Une large communauté engagée à faire de Node-Red un framework populaire auprès des développeurs est à l'origine de cette diversité.

La génération de code dans Node-Red est supportée de façon implicite. En effet, le fait que nous soyons capables d'exécuter le flux après sa réalisation signifie qu'en arrière-plan, un code a été généré et est en train d'être exécuté (par exemple, en reliant un noeud à un autre, une connexion doit être établie entre les deux composants, en termes de code). Nous évaluons cette exigence comme satisfaisante de par le fait que notre besoin n'est pas d'avoir accès au code, mais de pouvoir l'exécuter sur la Raspberry.

Node-Red permet une communication facile avec les bases de données à travers des noeuds pré-développés et faciles à configurer. Ceci facilite énormément la tâche pour les développeurs moins expérimentés. Node-Red permet aussi le stockage des informations dans des fichiers tant localement que sur le cloud/réseau. Ceci est d'une importante valeur ajoutée aux utilisateurs de Node-Red.

La visualisation sur Node-Red est assurée grâce à une palette de noeuds graphiques. L'utilisateur peut utiliser ces noeuds pour visualiser les données où il souhaite dans le flux de son application et il peut configurer ces noeuds selon ses besoins. En ce qui concerne la simulation, et comme

discuté pour la génération du code, Node-Red permet la simulation fluide de l'application, directement sur la Raspberry ; Node-Red étant lui-même installé sur la Raspberry Pi, la simulation de l'application consiste donc simplement à cliquer sur le noeud "*Timestamp*" qui déclenche l'application.

4.2.2.4 OpenHab

OpenHab est le framework qui supporte le plus grand nombre d'exigences parmi ceux dans notre liste (15 sur un ensemble de 17). Notre évaluation du degré de support fourni par OpenHab pour ces 15 exigences est résumée dans le tableau 4.6 et discutée ci-après.

Tableau 4.6 Degré de support des exigences prises en charge par OpenHab

Exigence	Évaluation
Spécification de l'interface de l'objet	+
Spécification du comportement de l'objet	+
Sécurité-Authentification	+
Sécurité-Cryptage de données	+
Gestion de l'hétérogénéité	±
Tolérance aux fautes-spécification des erreurs	+
Tolérance aux fautes-Détection des erreurs en cours de l'exécution	+
Intégration des composants COTS	+
Découvrabilité	+
Génération d'un code complet	+
Stockage de données	+
Visualisation	+
Simulation	+
Facilité de déploiement sur la Raspberry Pi	+
Exécution sur la Raspberry Pi	+

Comme présenté au Chapitre 3, un fichier *item* dans OpenHab permet la définition des variables manipulées par un objet. La syntaxe est très riche pour permettre la spécification des propriétés des données (type, intervalle de valeurs permises, etc.). Le comportement de l'objet est spécifié grâce au fichier *rules*. Dans ce fichier, le développeur définit les règles qui régissent le fonctionnement de chacun des objets, l'un en fonction de l'autre, ou séparément. À ce stade de

nos expérimentations, nous évaluons le support des exigences de spécification des objets comme satisfaisants. Toutefois, il est nécessaire de valider ce résultat par plus d'expérimentations surtout en implémentant des applications comportant plus que deux objets (Voir notre discussion au chapitre 5).

Par défaut, OpenHab n'est pas sécurisé. L'accès à la plate-forme matérielle sur laquelle OpenHab est installé donne accès à OpenHab lui-même. Ceci dit, OpenHab propose une procédure de sécurisation à installer, disponible dans (OpenHab, 2018b). Cela inclut le cryptage des données, la sécurisation d'accès à OpenHab à travers l'authentification, l'accès VPN, etc. La procédure à suivre est bien expliquée et ne nécessite pas de compétences spécifiques de la part des utilisateurs, d'où notre évaluation des exigences relatives à la sécurité comme "satisfaisant".

OpenHab supporte la gestion de l'hétérogénéité dans le cadre des applications pour maisons intelligentes. Toutefois, comme OpenHab ne supporte pas d'autres types d'applications, il est difficile d'intégrer certains types d'objets. Nous avons observé cela notamment dans l'expérimentation de la première catégorie. En effet, le protocole de communication utilisé par le lecteur de tags RFID n'est pas supporté par OpenHab, ce qui nous a empêché de mettre en oeuvre l'application de gestion d'inventaire. Ces observations expliquent notre évaluation de l'exigence de gestion de l'hétérogénéité.

La tolérance aux fautes est entièrement supportée par OpenHab, aussi bien dans l'étape de conception que dans l'étape d'exécution. Ceci est un point fort pour OpenHab, car il permet aux développeurs de définir eux-mêmes les éventuelles erreurs pouvant avoir lieu (ils peuvent les inclure dans les *rules*). Ensuite, au moment de l'exécution, les erreurs imprévues (qui n'ont pas étaient conçues et prises en considération dans l'étape de conception de l'application) sont enregistrées dans le fichier log de OpenHab. Cela permet aux développeurs de bien gérer les erreurs et donc une aboutir à des applications avec une très bonne tolérance aux fautes.

Dans Openhab, plusieurs composants sont disponibles afin de faciliter l'implémentation de différentes applications. Ces composants sont appelés *Bindings* et permettent à OpenHab de supporter plusieurs types d'applications/plates-formes. À titre d'exemple, l'échange de données

entre OpenHab et les composants intégrés sur la Raspberry Pi (par exemple pour lire la valeur de température, envoyer l'état ON ou OFF à la lampe) a été assuré grâce à l'ajout du *Binding GPIO*. La communauté des développeurs OpenHab peuvent aussi développer de nouveaux *Bindings* et ce, en suivant les instructions proposées dans (OpenHab, 2018a). En effet, l'un des *Bindings* intéressants dans Openhab est le *Thing discovery binding* qui, en l'ajoutant, assure la découvrabilité des objets. Aussi, OpenHab permet la connexion à des bases de données et le stockage de données à travers des *Bindings de Persistance* (voir figure 4.11).

Figure 4.11 Bindings pour la connexion aux bases de données

OpenHab est installé et exécuté sur la Raspberry Pi. Un code est donc généré et exécuté en arrière-plan. D'où le support de l'exigence de génération de code complet et simulation. Le fichier *Sitemap* permet de créer un UI personnalisé ou de modifier et de réutiliser le UI par défaut de OpenHab. Ceci permet une visualisation en temps réel de l'information. D'autre part, une application mobile OpenHab permet aussi la gestion et la visualisation des données de OpenHab en temps réel.

4.2.2.5 Eclipse Kura

Kura supporte 13 exigences parmi les 17 exigences de notre liste. Notre évaluation du degré de support fourni par Eclipse Kura pour ces 13 exigences est résumée dans le tableau 4.7 et discutée ci-après. Notre évaluation des exigences supportées par Kura repose essentiellement

sur l'expérimentation avec notre application IoT de la deuxième catégorie puisque nous n'avons pas pu implémenter les deux autres applications. Toutefois, notre évaluation tient compte aussi des difficultés que nous avons eues à déployer la troisième catégorie.

Tableau 4.7 Degré de support des exigences prises en charge par Eclipse Kura

Exigence	Évaluation
Spécification de l'interface de l'objet	±
Spécification du comportement de l'objet	±
Sécurité-Authentification	+
Sécurité-Cryptage de données	+
Gestion de l'hétérogénéité	±
Tolérance aux fautes-Détection des erreurs en cours de l'exécution	+
Intégration des composants COTS	+
Génération d'un code complet	+
Stockage des données	±
Visualisation	±
Simulation	+
Facilité de déploiement sur la Raspberry Pi	+
Exécution sur la Raspberry Pi	+

Kura nous permis de spécifier des données en sortie d'un objet. Cependant, nous avons eu des difficultés à définir les données en entrée. C'est la raison principale qui nous a empêchés d'implémenter l'application de la troisième catégorie. Cette caractéristique doit être ajoutée dans Kura, ou si elle est déjà supportée, il faudra qu'elle soit bien expliquée et documentée pour permettre aux développeurs de l'utiliser (notamment vu l'importance de l'échange de données en IoT).

Compte tenu des difficultés que nous avons eues avec Kura, nous avons pu définir un seul objet à la fin ; i.e. le capteur DHT11 dans la deuxième application. Pour le DHT11, nous avons pu définir un *subscriber* et un *publisher* afin de mesurer la température, la publier avec un *Topic* Temperature et grâce au *Publisher* (tel qu' expliqué dans le chapitre 3). Toutefois, la spécification du comportement de plus qu'un objet interconnecté n'a pas été faisable, d'où l'évaluation

du support de l'exigence "spécification du comportement de l'objet" comme nécessitant des améliorations.

Kura utilise le client Eclipse Paho MQTT Java. ceci permet l'authentification grâce à un nom d'utilisateur et un mot de passe. D'autre part, Kura prend en charge le protocole SSL, y compris les certificats côté client. Cela rend le canal TCP sur lequel MQTT communique assez sécurisé. Dans le cadre de nos expérimentations, nous estimons que c'est satisfaisant.

Kura permet l'intégration de plusieurs objets ayant différents protocoles de communication. Toutefois, l'échange de données entre deux objets reste un défi dans Kura. La gestion de l'hétérogénéité avec Kura nécessite donc des améliorations.

L'éditeur de flux de Kura permet l'ajout d'un noeud *Logger* qui permet l'enregistrement des détails concernant les erreurs qui ont lieu en cours d'exécution. Il existe plusieurs composants pré-développés et prêts à être déployés ans Kura. Dans notre cas, nous avons utilisé le composant "*Publisher*", que nous avons glissé et déposé à partir de *Eclipse Marketplace* dans l'IDE de Kura.

Le stockage de données et la visualisation sont deux exigences supportées par Kura, mais elles nécessitent des améliorations. En effet, la visualisation avec Kura se fait à travers la linge de commande de la Raspberry Pi. Le stockage de données se fait à travers la liaison de Kura avec une base de données. Nous avons eu des difficultés à mettre en place une base de données avec Kura.

Kura s'exécute directement sur la Raspberry Pi, les exigences de génération de code complet, la simulation, le déploiement et l'exécution sur la Raspberry Pi sont supportées et satisfaisantes.

CHAPITRE 5

SYNTHÈSE DES RÉSULTATS ET LIMITES DE L'ÉTUDE

Dans ce chapitre, nous résumons les résultats de notre étude, discutés dans le chapitre précédent. En particulier, nous faisons la synthèse ces résultats en fonction des questions de recherche que nous avons ciblées dans notre étude. Par la suite, nous discutons les limitations de notre étude.

5.1 Synthèse des résultats

Rappelons en premier lieu les questions de recherche que nous avons définies :

- (QR1) : Jusqu'à quel degré un framework open source supporte le développement des applications IoT ?
- (QR2) : Jusqu'à quel degré un framework open source supporte un ensemble minimal d'exigences des applications IoT ?

1ère question de recherche

Afin de répondre à la première question (QR1), nous avons sélectionné un ensemble de frameworks open source IoT, à savoir : Eclipse Vorto, ThingML, Node-Red, OpenHab et Eclipse Kura. Par la suite, et afin de cibler un large spectre d'applications IoT, nous avons catégorisé les applications IoT en 3 catégories : 1) les applications visant l'identification/localisation des objets à travers des technologies telles que la technologie RFID, 2) les applications pour le suivi d'état des objets et les variables d'environnement en temps réel, et 3) les applications visant à décider et agir sur l'état des objets. Ensuite, nous avons choisi de déployer un exemple d'application par catégorie, en utilisant les différents frameworks sélectionnés. Pour la première catégorie, nous avons choisi de déployer une application de gestion d'inventaire pour un magasin. L'exemple déployé pour la deuxième catégorie est une station météorologique. Finalement, nous avons choisi l'exemple de chauffage intelligent pour la troisième catégorie. Afin de déployer les

différents exemples, nous avons choisi d'utiliser la Raspberry Pi, de par sa popularité dans le monde de l'IoT.

En premier lieu, nous avons analysé notre expérience de déploiement des exemples d'applications par rapport à la faisabilité et la facilité/difficulté de mise en oeuvre de chacune des trois catégories d'applications. En effet, Vorto ne nous a pas permis de déployer les différentes applications. En utilisant Eclipse Kura la seule catégorie qui était faisable est la deuxième catégorie. Node-Red était le framework le plus facile en termes de temps et des efforts d'apprentissage. OpenHab est très riche en services/fonctionnalités. Cependant, son utilisation est limitée au domaine des maisons intelligentes. ThingML nous a permis de déployer toutes les applications, grâce à la possibilité d'intégrer du code spécifique aux plates-formes (C, JAVA,etc.) mais cela a nécessité quelques efforts.

2ème question de recherche

En ce qui concerne la deuxième question de recherche, nous avons analysé les frameworks open source selon un ensemble de critères correspondant à une liste minimale d'exigences des applications IoT. Pour identifier ces exigences, nous avons analysé les travaux qui survolent les applications IoT et construit une liste initiale d'exigences. Ensuite, nous avons complété et raffiné cette liste en analysant le standard ISO-RA IoT.

La première observation tirée de notre analyse est qu'aucun des frameworks ne supporte toutes les exigences de notre liste minimale. En fait, les frameworks IoT disponibles ont des objectifs différents et ciblent différents d'applications IoT. Cela explique la la différence d'un framework à un autre en termes d'exigences supportées. Eclipse Vorto est le framework qui supporte le minimum d'exigences (4/17). Ceci s'explique par le fait que Vorto n'est pas conçu pour déployer une application IoT mais pour fournir un langage simple permettant de définir les objets et leurs capacités dans le but d'assurer l'interopérabilité. ThingML supporte 9 exigences sur 17. Plusieurs exigences sont supportées avec quelques efforts d'implémentation, grâce au fait que ThingML permet l'intégration du code propre aux plates-formes. Node-Red est le seul

framework graphique dans les frameworks que nous avons étudiés. Il supporte 12 exigences sur 17, ce qui le rend très populaire auprès de la communauté des développeurs en IoT. OpenHab est le framework qui supporte le plus grand nombre d'exigences (15 sur 17) grâce à la panoplie de services qu'il offre. Cependant, l'inconvénient de ce framework, comme mentionné avant, est qu'il se limite aux applications du domaine des maisons intelligentes, écartant ainsi un large spectre d'applications en IoT. Malgré le nombre important des exigences qu'il supporte (13 sur 17), Eclipse Kura est le plus difficile et complexe à utiliser. Ceci peut être expliqué par le fait que le framework représente une couche intermédiaire entre les objets physiques et le réseau (edge framework).

5.2 Limites de l'étude

Dans cette section, nous discutons les facteurs qui peuvent avoir un impact sur la validité de nos résultats.

5.2.1 Validité externe des résultats de l'étude

La validité externe fait référence à la capacité d'une étude à produire des conclusions généralisables au-delà du contexte de l'étude. La validité externe des résultats de notre étude peut être limitée par les facteurs suivants :

- l'étude est limitée aux frameworks open source. Ces derniers ne sont peut-être pas représentatifs des frameworks IoT en général. En effet, la différence entre les frameworks open source et les frameworks commerciaux peut être considérable. Cependant, il était difficile de couvrir les frameworks commerciaux dans notre étude, principalement à cause du coût nécessaire pour acquérir ces frameworks ; Toutefois, notre analyse a été faite davantage en profondeur puisqu'elle était multidimensionnelle et prend en considération un bon nombre d'exigences ;
- le nombre de frameworks open source que nous avons étudié est limité, comparé au nombre de frameworks open source sur le marché. Cette limitation volontaire se justifie par notre choix de réaliser des expérimentations pratiques, et en profondeur, avec ces frameworks

contrairement aux études existantes qui se basent uniquement sur une liste de caractéristiques pour analyser les frameworks ;

- le nombre des applications IoT implémentées par catégorie définie selon la méthodologie (Zhang, 2011), peut ne pas être représentatif. Nous nous sommes limités à une seule application par catégorie et nous avons limité le nombre d'objets dans chaque application. Ceci peut limiter la généralisation de nos observations sur les frameworks analysés. Nos travaux futurs visent à couvrir plus d'applications IoT ;
- nous avons choisi d'utiliser le protocole MQTT dans les différentes expériences parce qu'il est un protocole léger et supporté par une grande majorité des frameworks. Cependant, pour un framework IoT qui ne supporte pas ce protocole, nos expérimentations auraient nécessité l'utilisation de "*bridge*" pour convertir d'un protocole à un autre. Cela aurait probablement eu un impact sur les résultats de nos expérimentations. L'étude de tels frameworks fait partie des travaux futurs ;
- le choix de la Raspberry Pi comme plate-forme matérielle a certainement aussi un impact sur la généralisation de nos résultats. En effet, comme mentionné déjà, certains frameworks ciblent explicitement cette plate-forme alors que d'autres non.

5.2.2 Validité interne des résultats de l'étude

La validité interne fait référence à la capacité d'une étude à produire des conclusions précises concernant les aspects ou éléments évalués. Pour assurer la validité interne de nos résultats, nous avons considéré les points suivants :

- toutes les expériences ont été réalisées dans les mêmes conditions ; soit : même réseau internet et même environnement de test ;
- nous avons utilisé le même matériel par catégorie pour toutes les expérimentations pour tous les frameworks.

Toutefois, la validité interne de nos résultats peut être affectée par les facteurs suivants :

- les frameworks étudiés sont divergents en termes de modes de fonctionnement, d'objectifs, etc. Aussi, le niveau de documentation des frameworks est très disparate. Cela peut rendre la comparaison des frameworks difficile et, à la limite, injuste. Des travaux futurs sont nécessaires pour voir comment les frameworks étudiés peuvent être combinés pour se compléter ;
- le matériel utilisé pour l'expérimentation de la catégorie 1 est très limité en termes de fonctionnalités et risque de ne pas représenter convenablement l'ensemble du matériel disponible sur le marché lecteurs RFID, lecteurs de codes (QR, à barres), tags RFID, etc. Son protocole de communication est aussi limité (communication à travers le port USB) et présente un frein dans la réalisation de l'application en utilisant les différents frameworks puisqu'il n'est pas utilisé dans les applications IoT (contrairement aux autres protocoles, tels que MQTT). Des expérimentations avec d'autres matériels et équipements font partie de travaux futurs.

CONCLUSION ET TRAVAUX FUTURS

Le but général de notre étude était d'évaluer le support fourni par les frameworks IoT pour le développement des applications IoT. En particulier, notre travail porte sur l'étude et l'expérimentation de frameworks open source. Nous avons choisi un échantillon de frameworks appartenant au projet Eclispe IoT (Eclipse Vorto, OpenHab et Eclipse Kura), et nous avons sélectionné deux autres frameworks, ThingML et Node-Red pour la diversification de l'échantillon. Afin de couvrir un large spectre d'applications IoT, nous avons adopté une catégorisation des applications IoT et nous avons choisi de déployer un exemple représentatif d'application par catégorie. Pour expérimenter les différents exemples d'applications, nous avons choisi la Raspberry Pi comme plate-forme matérielle de déploiement. Finalement, nous avons défini un ensemble d'exigences sur lesquelles nous avons basé notre évaluation du support fourni par les frameworks étudiés pour le déploiement des exemples d'applications expérimentés.

Notre étude et nos expérimentations nous ont permis d'observer une diversité dans les frameworks IoT en termes de support fourni aux développeurs, fonctionnalités supportées, services disponibles, méthode de développement proposée (graphique, code,etc.), domaine d'application supporté, etc. Aussi, notre évaluation des frameworks IoT selon une liste minimale d'exigences montre que certains aspects ne sont pas encore considérés par ces frameworks ; C'est notamment le cas de la spécification des propriétés des objets, i.e. la spécification des caractéristiques matérielles des objets comme la mémoire ou autre. Cependant, un bon nombre d'objets connectés risque d'avoir des ressources limitées dont il faut tenir compte dans le développement et le déploiement des applications. La spécification des caractéristiques matérielles des objets est un aspect qui nécessite amélioration dans tous les frameworks étudiés.

Nos résultats montrent notamment qu'en IoT, les frameworks propres à un domaine spécifique (e.g. OpenHab) offrent des services considérables et sont mieux adaptés aux besoins des développeurs que les autres frameworks plus généraux. Toutefois, plusieurs frameworks généraux peuvent

apporter une valeur ajoutée aux frameworks spécifiques à un domaine particulier, notamment pour l'intégration des objets qui n'appartiennent pas au domaine d'application visé. À titre d'exemple, Node-Red propose plusieurs noeuds qui facilitent l'intégration avec OpenHab. Aussi, le DSL de Vorto est utilisé pour intégrer des objets du domaine des voitures intelligentes dans des applications du domaine des maisons intelligentes.

En termes de perspectives à court terme, notre étude peut être étendue en : (i) en implémentant des applications supplémentaires avec une plus grande diversité d'objets ; et (ii) en augmentant le nombre de frameworks étudiés et en diversifiant la nature et le type de ces frameworks.

À moyen terme, nous nous proposons d'étudier la complémentarité des frameworks et la possibilité de déployer des applications IoT en intégrant et combinant différents framewoks.

À long terme, nous voulons tirer profit des résultats de cette étude et de notre expérience pour développer un framework IoT fédérateur lequel englobe des points forts des frameworks existants et qui propose des solutions adéquates aux défis rencontrés par les développeurs en IoT.

ANNEXE I

SPÉCIFICATIONS TECHNIQUES DU MATÉRIEL UTILISÉ

1. Raspberry Pi

Les caractéristiques de la Raspberry Pi 3 modèle B+ que nous avons utilisés sont :

- Alimentation : 5 Vcc/maxi 2,5 A* via prise micro-USB (* intensité maxi si toutes les fonctions sont utilisées) ;
- CPU : ARM Cortex-A53 quatre coeurs 1,4 GHz ;
- Wi-Fi : Dual-band 2,4 et 5 GHz, 802.11b/g/n/ac (Broadcom BCM43438) ;
- Bluetooth 4.2 (Broadcom BCM43438) ;
- Mémoire : 1 GB LPDDR2 ;
- Ethernet 10/100/1000 : jusqu'à 300 Mbps ;
- 4 ports USB 2.0 ;
- Port ethernet 10/100 base T : RJ45 ;
- Bus : SPI, I2C, série ;
- Support pour cartes micro-SD ;
- Sorties audio :
  - HDMI avec gestion du 5.1 ;
  - Jack 3,5 mm en stéréo.
- Sorties vidéo : HDMI ;
- Dimensions : 86 x 54 x 17 mm.

2. Capteur de température et d'humidité DHT11

- Voltage : 3.5V à 5.5V ;

- Courant de fonctionnement : 0.3mA (measuring) 60uA (standby) ;
- Sortie : Serial data ;
- Intervalle de température : 0°C à 50°C ;
- Intervalle d'humidité : 20% à 90% ;
- Résolution : Humidité et température : 16-bit ;
- Précision : ±1°C and ±1%.

ANNEXE II

TABLEAU DES CARACTÉRISTIQUES DE L'ARCHITECTURE D'UN SYSTÈME IOT SELON (ISO, 2018)

Tableau-A II-1 caractéristiques de l'architecture d'un système IoT

Caractéristique	Description
Composabilité	Capacité de combiner des composants IoT discrets dans un système IoT afin d'atteindre un ensemble de buts et d'objectifs.
Séparation des capacités fonctionnelles et gestionnelles	Les interfaces et les capacités fonctionnelles d'un composant IoT, sont clairement séparées des interfaces et des capacités de gestion de ce composant. Cela signifie généralement que l'interface de gestion se trouve sur un nœud final différent de celui de l'interface fonctionnelle et que les fonctionnalités de gestion sont gérées par des composants logiciels différents de ceux des interfaces fonctionnelles.
Gestion de l'hétérogénéité	Un système IoT doit gérer un ensemble divers de composants et d'entités physiques qui interagissent de différentes manières.
Système hautement distribué	Les systèmes qui sont constitués de sous-systèmes pouvant être séparés physiquement et situés à distance les un des autres, tout en étant fonctionnellement intégrés.
Systèmes légataires	Concept dont un système IoT a besoin pour incorporer des composants existants même lorsqu'ils incarnent des technologies qui ne sont plus standard ou approuvées.
Modularité	c'est quand un composant est une unité distincte, qui peut être combinée avec d'autres composants ou retirée du système et remplacée par un autre sans nuire aux autres composants
Connectivité réseau	Les objets en IoT sont interconnectés à travers le réseau. Les objets, décrits dans le réseau comme des noeuds, établissent, acheminent et terminent les communications. Les noeuds d'extrémité constituent la source ou la destination de tout type d'information. La connectivité réseau est assurée via des protocoles de communication réseau.
Évolutivité	Capacité d'évolution d'un système en termes de taille et de complexité
Partageabilité	Capacité d'un composant du système à accéder aux informations ou aux ressources allouées entre les différents composants.
Identification unique	Caractéristique d'un système IoT consistant à associer de manière non équivoque et répétée les entités du système à un nom, un code, un symbole ou un numéro unique, et à interagir avec ces entités, suivre ou contrôler leurs activités, en se référant à cet identifiant.

Caractéristique	Description
Composants bien définis	Une entité IoT est considérée bien définie si une description précise de ses capacités et caractéristiques est disponible. Les informations incluent non seulement des informations sur la fonctionnalité du composant spécifique, mais également des informations de configuration, de communication, de sécurité et de fiabilité.

ANNEXE III

TABLEAU DES CARACTÉRISTIQUES FONCTIONNELLES DE L'IOT SELON (ISO, 2018)

Tableau-A III-1 Caractéristiques fonctionnelles de l'IoT

Caractéristique	Description
Précision	Définition de l'incertitude des données, des calculs ou des actions d'un objet, d'un service ou d'un système IoT.
Configuration automatique	Basée sur l'interfonctionnement de règles prédéfinies incluant la connexion et la découverte automatique des objets au réseau, les services prédéveloppés et facilement réutilisables, et le *plug&play* (utilisation de composants COTS). Cela nécessite la mise en place des mécanismes de sécurité et d'authentification pour garantir que seuls les composants autorisés peuvent être configurés automatiquement dans le système. Le mécanisme de sécurité doit être organisé de manière appropriée pour chaque domaine.
Conformité	Caractéristique de se conformer à des règles, telles que celles définies par une loi, un règlement, une norme ou une politique.
connaissance du contenu	Propriété qu'un composant IoT ait suffisamment d'information du système. Les composants IoT ayant connaissance du contenu sont capables d'améliorer la précision de la récupération des données, de découvrir les services et de permettre les interactions appropriées entre les utilisateurs.
connaissance du contexte	Caractéristique d'un objet ou service IoT, où le système est capable de surveiller l'environnement dans lequel il évolue, ainsi que les événements qui se déroulent dans cet environnement pour déterminer les propriétés sur lequel il agit.
Caractéristiques des données 5V	Les systèmes IoT traitent généralement de gros volumes de données, non structurées et à faible densité Les Volume, vitesse, véracité, variabilité et variété
Découvrabilité	Capacité des composants à découvrir les nœuds finaux sur le réseau. Les services de découverte associés permettent de localiser, d'identifier et d'accéder aux points de terminaison en fonction de critères variables, tels que l'emplacement géographique ou le type de service.
Flexibilité	Capacité d'un système, service, objet ou autre composant IoT à fournir différentes fonctionnalités, selon le besoin ou le contexte.
Maniabilité	Concerne des aspects des systèmes IoT tels que la gestion des objets, la gestion du réseau, la gestion du système ainsi que la maintenance et les alertes d'interface. La facilité de gestion est importante pour répondre aux exigences du système IoT.

Caractéristique	Description
Communication réseau	Les protocoles de communication utilisés peuvent varier entre les différents types de réseau. Il est courant que les réseaux de proximité utilisent des protocoles spécialisés adaptés au caractère spécialisé de ces réseaux.
Gestion et exploitation du réseau	Les systèmes IoT nécessitent une gestion de réseau. La forme et le but de la gestion et du fonctionnement du réseau dépendent du type et de la propriété du réseau et du type de communication. La gestion est nécessaire lors de la mise en place d'un réseau, y compris la gestion de l'identité et des adresses de l'appareil, les profils d'utilisation du réseau et l'inclusion de capacités de gestion dynamique. La gestion des réseaux implique le contrôle de la qualité de service, l'extension dynamique des réseaux,, la gestion des pannes et le contrôle de la sécurité.
Capacité en temps réel	Caractéristique d'un système où le calcul est effectué en temps réel. Il peut également désigner la propriété d'effectuer une action, une fonction ou un service dans une période spécifiée, ce qui permet la prise en charge d'opérations déterministes.
Description du soi	Processus par lequel les composants d'un système IoT répertorient leurs capacités afin d'informer d'autres composants IoT ou d'autres systèmes IoT à des fins de composition, d'interopérabilité et de découverte dynamique. L'autodescription inclut la spécification d'interface, les capacités du composant IoT, les types de périphériques pouvant être connectés à un système IoT, les types de service rendus disponibles par le système IoT et l'état actuel du système IoT.
Abonnement au service	Il arrive souvent que les utilisateurs de l'IoT s'abonnent à des services IoT mis à disposition par des fournisseurs de services IoT. Dans ce cas, les fournisseurs de services IoT mettent à disposition un processus d'abonnement permettant aux utilisateurs IoT de s'abonner à un service IoT particulier.

ANNEXE IV

FONCTION C - LECTURE DES IDENTIFIANTS DE TAGS RFID

```
@c_header ‘
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define BAUDRATE B9600
#define USBPORT "/dev/ttyUSB0"
#define TRUE 1‘
{ function read_RFID_tag() do‘
int fd, c, res;
struct termios newtio;
char buf[10];
fd = open(USBPORT, O_RDWR | O_NOCTTY );
if (fd < 0) { perror(USBPORT); exit(-1); }
  bzero(&newtio, sizeof(newtio)); /* clear struct for new port
  settings */
   newtio.c_cflag = USBPORT | CRTSCTS | CS8 | CLOCAL | CREAD;
 /* IGNPAR  : ignore bytes with parity errors
 otherwise make device raw (no other input processing) */
 newtio.c_iflag = IGNPAR;
 /*  Raw output  */
 newtio.c_oflag = 0;
 newtio.c_lflag = ICANON;
```

```
  tcflush(fd, TCIFLUSH);
  tcsetattr(fd,TCSANOW,&newtio);
  while (TRUE) { /* loop continuously */
  res = read(fd, buf, 10);
   buf[res] = 0;
   printf("%s", buf, res); }‘
end
```

ANNEXE V

FONCTION C - LECTURE DE TEMPÉRATURE

```
function read_dht11_dat() : Double do

‘int dht11_dat[5] = {0,0,0,0,0};
uint8_t laststate = HIGH;
uint8_t counter = 0;
uint8_t j = 0, i;
float f; // fahrenheit

dht11_dat[0] = dht11_dat[1] = dht11_dat[2] = dht11_dat[3] =
dht11_dat[4] = 0;

// pull pin down for 18 milliseconds
pinMode( ‘&DHTPIN& ‘, OUTPUT);
digitalWrite( ‘&DHTPIN& ‘, LOW);
delay(18);
// then pull it up for 40 microseconds
digitalWrite( ‘&DHTPIN& ‘, HIGH);
delayMicroseconds(40);
// prepare to read the pin
pinMode( ‘&DHTPIN& ‘, INPUT);

// detect change and read data
for ( i=0; i< MAXTIMINGS; i++) {
        counter = 0;
```

```
        while (digitalRead( '&DHTPIN& ') == laststate) {
                counter++;
                delayMicroseconds(1);
                if (counter == 255) {
                        break;
                }
        }
        laststate = digitalRead( '&DHTPIN& ');

        if (counter == 255) break;

        // ignore first 3 transitions
        if ((i >= 4) && (i\%2 == 0)) {
                // shove each bit into the storage bytes
                dht11_dat[j/8] <<= 1;
                if (counter > 25)
                        dht11_dat[j/8] |= 1;
                j++;
        }
}

// check we read 40 bits (8bit x 5 )
+ verify checksum in the last byte
// print it out if data is good
if ((j >= 40) &&
(dht11_dat[4] == ((dht11_dat[0] + dht11_dat[1] + dht11_dat[2] +
dht11_dat[3]) & 0xFF)) ) {
        f = dht11_dat[2] * 9. / 5. + 32;
        '&temp& '= dht11_dat[2] + dht11_dat[3]*0.1;
```

```
        printf("\%.1f\n",‘&temp& ‘);
        printf("Humidity = \%d.\%d \%\%
        Temperature = \%d.\%d C (\%.1f F)\\n",
        dht11_dat[0], dht11_dat[1], dht11_dat[2], dht11_dat[3], f);
                                    }‘

return temp
end
```

ANNEXE VI

PROCÉDURE DE SÉCURISATION DE NODE-RED

Par défaut, l'éditeur Node-RED n'est pas sécurisé - toute personne qui peut accéder à son adresse IP peut accéder à l'éditeur et déployer les modifications. Cela ne convient que si vous utilisez un réseau de confiance. Ce guide décrit comment sécuriser Node-RED. La sécurité est divisée en deux parties :

- Sécurisation de l'éditeur et de l'API admin
- Sécurisation des nœuds HTTP et du tableau de bord Node-RED

1. Sécurité de l'éditeur et de l'API admin

L'éditeur et l'API Admin prend en charge deux types d'authentification :

- authentification basée sur les informations d'identification du nom d'utilisateur / mot de passe
- authentification contre tout fournisseur OAuth / OpenID tel que Twitter ou GitHub

1.1 Authentification basée sur le nom d'utilisateur / mot de passe

Pour activer l'authentification des utilisateurs sur l'éditeur et l'API admin, décommentez la propriété adminAuth dans le fichier de paramètres (figure VI-1) : La propriété users est un tableau d'objets utilisateur. Cela vous permet de définir plusieurs utilisateurs, chacun pouvant avoir des autorisations différentes.

Cet exemple de configuration ci-dessus définit deux utilisateurs. Un appelé admin qui a la permission de tout faire dans l'éditeur et qui a un mot de passe. L'autre, appelé george, bénéficie d'un accès en lecture seule. Notez que les mots de passe sont hachés en toute sécurité à l'aide de l'algorithme bcrypt.

1.2 Génération du hachage de mot de passe

Figure-A VI-1 Fichier de paramètres

Pour générer un hachage de mot de passe approprié, vous pouvez utiliser l'outil de ligne de commande node-red-admin.

```
node-red-admin hash-pw
```

L'outil vous demandera le mot de passe que vous souhaitez utiliser, puis imprimera le hachage qui peut être copié dans le fichier de paramètres.
Alternativement, vous pouvez exécuter la commande suivante à partir du répertoire d'installation de Node-RED :

```
node -e "console.log(require('bcryptjs').hashSync(process.argv[1], 8));" your-password-here
```

1.3 Authentification basée sur OAuth / OpenID

Pour utiliser une source d'authentification externe, Node-RED peut utiliser un large éventail de stratégies fournies par Passport.
Les modules d'authentification Node-RED sont disponibles pour Twitter et GitHub. Ils récapitulent certains détails spécifiques à la stratégie pour en faciliter l'utilisation. Mais ils peuvent également être utilisés comme modèle d'authentification avec d'autres stratégies similaires.

L'exemple suivant montre comment configurer pour s'authentifier sur Twitter sans utiliser le module d'authentification que nous fournissons.

```
adminAuth: {
    type:"strategy",
    strategy: {
        name: "twitter",
        label: 'Sign in with Twitter',
        icon:"fa-twitter",
        strategy: require("passport-twitter").Strategy,
        options: {
            consumerKey: TWITTER_APP_CONSUMER_KEY,
            consumerSecret: TWITTER_APP_CONSUMER_SECRET,
            callbackURL: "http://example.com/auth/strategy/callback",
            verify: function(token, tokenSecret, profile, done) {
                done(null, profile);
            }
        },
    },
    users: [
       { username: "knolleary",permissions: ["*"]}
    ]
};
```

La propriété strategy prend les options suivantes :

- name : le nom de la stratégie de passeport utilisée
- strategy : le module de stratégie de passeport
- label/icon : utilisée sur la page de connexion. L'icône peut être n'importe quel nom d'icône FontAwesome.
- options : un objet d'options transmis à la stratégie de passeport lors de sa création. Reportez-vous à la documentation de la stratégie pour savoir ce dont elle a besoin. Voir ci-dessous pour un nœud sur le callbackURL.
- verify : la fonction de vérification utilisée par la stratégie. Il doit appeler done avec un profil utilisateur comme deuxième argument si l'utilisateur est valide. Cela devrait avoir une

propriété de nom d'utilisateur qui est utilisée pour vérifier la liste des utilisateurs valides. Passport tente de standardiser l'objet de profil utilisateur, donc la plupart des stratégies fournissent cette propriété.

Le callBackURL utilisée par une stratégie est l'endroit où le fournisseur d'authentification se redirigera vers la suite d'une tentative d'authentification. Il doit s'agir de l'URL de votre éditeur Node-RED avec /auth/strategy/callback ajouté au chemin. Par exemple, si vous accédez à l'éditeur à http : // localhost : 1880, vous utiliserez http : //localhost :1880/auth/strategy/callback.

1.4 Définition d'un utilisateur par défaut

L'exemple de configuration ci-dessus empêchera quiconque d'accéder à l'éditeur à moins qu'il ne se connecte.

Dans certains cas, il est souhaitable de permettre à chacun un certain niveau d'accès. En règle générale, cela donnera un accès en lecture seule à l'éditeur. Pour ce faire, la propriété default peut être ajoutée au paramètre adminAuth pour définir l'utilisateur par défaut :

```
adminAuth: {
    type: "credentials",
    users: [ /* list of users */ ],
    default: {
        permissions: "read"
    }
}
```

1.5 Autorisations aux utilisateur

Avant Node-RED 0.14, les utilisateurs pouvaient avoir l'une des deux autorisations :

- * - accès total
- lecture - accès en lecture seule

À partir de Node-RED 0.14, les autorisations peuvent être beaucoup plus fines et pour prendre en charge cela, la propriété peut être soit une seule chaîne comme précédemment, soit un tableau contenant plusieurs autorisations.
Chaque méthode de l'API Admin définit le niveau d'autorisation nécessaire pour y accéder. Le modèle d'autorisation est basé sur les ressources. Par exemple, pour obtenir la configuration de flux actuelle, un utilisateur aura besoin de l'autorisation flows.read. Mais pour mettre à jour les flux, ils auront besoin de l'autorisation flows.write.

1.6 Expiration du jeton

Par défaut, les jetons d'accès expirent 7 jours après leur création. Nous ne prenons actuellement pas en charge l'actualisation du jeton pour prolonger cette période.
Le délai d'expiration peut être personnalisé en définissant la propriété sessionExpiryTime du paramètre adminAuth. Cela définit, en secondes, la durée de validité d'un jeton. Par exemple, pour définir les jetons pour expirer après 1 jour :

```
adminAuth: {
    sessionExpiryTime: 86400,
    ...
}
```

1.7 Authentification utilisateur personnalisée

Plutôt que de coder en dur les utilisateurs dans le fichier de paramètres, il est également possible de brancher du code personnalisé pour authentifier les utilisateurs. Cela permet de s'intégrer aux schémas d'authentification existants.
L'exemple suivant montre comment un module externe peut être utilisé pour fournir le code d'authentification personnalisé.
Enregistrez les éléments suivants dans un fichier appelé <node-red> /user-authentication.js

```
module.exports = {
    type: "credentials",
    users: function(username) {
        return new Promise(function(resolve) {
            // Do whatever work is needed to check username is a valid
            // user.
            if (valid) {
                // Resolve with the user object. It must contain
                // properties 'username' and 'permissions'
                var user = { username: "admin", permissions: "*" };
                resolve(user);
            } else {
                // Resolve with null to indicate this user does not exist
                resolve(null);
            }
        });
    },
    authenticate: function(username,password) {
        return new Promise(function(resolve) {
            // Do whatever work is needed to validate the username/password
            // combination.
            if (valid) {
                // Resolve with the user object. Equivalent to having
                // called users(username);
                var user = { username: "admin", permissions: "*" };
                resolve(user);
            } else {
                // Resolve with null to indicate the username/password pair
                // were not valid.
                resolve(null);
            }
        });
    },
    default: function() {
        return new Promise(function(resolve) {
            // Resolve with the user object for the default user.
            // If no default user exists, resolve with null.
            resolve({anonymous: true, permissions:"read"});
        });
    }
}
```

Définissez la propriété adminAuth dans settings.js pour charger ce module :

```
adminAuth: require("./user-authentication")
```

1.8 Sécurité des nœuds HTTP

Les routes exposées par les nœuds HTTP In peuvent être sécurisées à l'aide de l'authentification de base.

La propriété httpNodeAuth dans votre fichier settings.js peut être utilisée pour définir un nom d'utilisateur et un mot de passe uniques qui seront autorisés à accéder aux itinéraires. La

```
httpNodeAuth: {user:"user",pass:"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN."},
```

propriété pass utilise le même format que adminAuth. Voir Génération du hachage de mot de passe pour plus d'informations.

L'accès à tout contenu statique défini par la propriété httpStatic peut être sécurisé à l'aide de la propriété httpStaticAuth, qui utilise le même format.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Adafruit. (2019). Adafruit-Python-DHT. Repéré à https://github.com/adafruit/Adafruit_Python_DHT.

Ali, M. (2018). Kura doesn't work properly on Raspberry Pi. Repéré à https://www.eclipse.org/forums/index.php/t/1094212/.

Atzori, L., Iera, A. & Morabito, G. (2010). The internet of things : A survey. *Computer networks*, 54(15), 2787–2805.

Balakrishnan, P., Babu, K. R., Naiju, C. D. & Madiajagan, M. (2019). *Design and Implementation of Digital Twin for Predicting Failures in Automobiles Using Machine Learning Algorithms*.

Bassi, A., Bauer, M., Fiedler, M. & Kranenburg, R. v. (2013). *Enabling things to talk*. Springer-Verlag GmbH.

Bélissent, J. et al. (2010). Getting clever about smart cities : New opportunities require new business models. *Cambridge, Massachusetts, USA*.

Belsa, A., Sarabia-Jacome, D., Palau, C. E. & Esteve, M. (2018). Flow-based programming interoperability solution for IoT platform applications. *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 304–309.

Bink, R. (2018). IOT Tutorial : Read RFID-tags with an USB RFID reader, Raspberry Pi and Node-RED from scratch. Repéré à https://medium.com/coinmonks/iot-tutorial-read-tags-from-a-usb-rfid-reader-with-raspberry-pi-and-node-red-from-scratch-4554836be127.

CHEN, J. (2020). Smart Home. Repéré à https://www.investopedia.com/terms/s/smart-home.asp.

Čolaković, A. & Hadžialić, M. (2018). Internet of Things (IoT) : A review of enabling technologies, challenges, and open research issues. *Computer Networks*, 144, 17–39.

D'Elia, A., Viola, F., Montori, F., Azzoni, P. & Maiero, M. (2016). Electro Mobility automation through the Arrowhead Framework. *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pp. 5246–5252.

Derhamy, H., Eliasson, J., Delsing, J. & Priller, P. (2015). A survey of commercial frameworks for the internet of things. *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–8.

Ericsson. (2018). Internet of Things forecast. Repéré à https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-june-2018.pdf.

F. Fleurey, B. M. & community, O. (2016). Thingml source code repository. Repéré à https://github.com/sintef-9012/thingml.

Ganguly, P. (2016). Selecting the right IoT cloud platform. *2016 International Conference on Internet of Things and Applications (IOTA)*, pp. 316–320.

Großmann, T. (2019). Integrating a Java - based Device with the Bosch IoT Suite using Vorto. Repéré à https://github.com/eclipse/vorto/blob/master/docs/tutorials/connect_javadevice.md.

Guadane, M., Bchimi, W., Samet, A. & Affes, S. (2017). Enhanced range-free localization in wireless sensor networks using a new weighted hop-size estimation technique. *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–5.

Gubbi, J., Buyya, R., Marusic, S. & Palaniswami, M. (2013). Internet of Things (IoT) : A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7), 1645–1660.

Harrand, N., Fleurey, F., Morin, B. & Husa, K. E. (2016). Thingml : a language and code generation framework for heterogeneous targets. *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pp. 125–135.

Hejazi, H., Rajab, H., Cinkler, T. & Lengyel, L. (2018). Survey of platforms for massive IoT. *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, pp. 1–8.

Hermanudin, A., Ekadiyanto, F. & Sari, R. (2019, 02). Performance Evaluation of CoAP Broker and Access Gateway Implementation on Wireless Sensor Network. doi : 10.1109/TENCONSpring.2018.8692050.

ISO. (2018). ISO/IEC 30141 :2018 Architecture de référence de l'Internet des objets (IoT RA).

Kura, E. (2019). Eclipse Kura | the Eclipse foundation. Repéré à https://www.eclipse.org/kura/.

Lamaazi, H., Benamar, N., Jara, A. J., Ladid, L. & El Ouadghiri, D. (2014). Challenges of the internet of things : IPv6 and network management. *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 328–333.

Laverman, J., Grewe, D., Weinmann, O., Wagner, M. & Schildt, S. (2016). Integrating Vehicular Data into Smart Home IoT Systems Using Eclipse Vorto. *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5.

Lee, G. M. & Kim, J. Y. (2010). The internet of things—a problem statement. *2010 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 517–518.

Maliński, P. (2019). github-riklaunim/pyusb-keyboard-alike : Handler for keyboards and keyboard – alike devices like barcode scanner, RFID readers. Repéré à https://github.com/riklaunim/pyusb-keyboard-alike.

Matta, P., Pant, B. & Arora, M. (2017). All you want to know about Internet of Things (IoT). *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 1306–1311.

Miraz, M. H., Ali, M., Excell, P. S. & Picking, R. (2015). A review on Internet of Things (IoT), Internet of everything (IoE) and Internet of nano things (IoNT). *2015 Internet Technologies and Applications (ITA)*, pp. 219–224.

Motta, R. C., de Oliveira, K. M. & Travassos, G. H. (2018). On challenges in engineering IoT software systems. *Proceedings of the XXXII Brazilian Symposium on Software Engineering*, pp. 42–51.

Muccini, H. & Moghaddam, M. T. (2018). Iot architectural styles. *European Conference on Software Architecture*, pp. 68–85.

Node-red. (2018). Securing Node-red. Repéré à https://nodered.org/docs/user-guide/runtime/securing-node-red.

OpenHab. (2018a). Developing a Binding for OpenHAB 2. Repéré à https://www.openhab.org/v2.3/docs/developer/development/bindings.html.

OpenHab. (2018b). Securing access to OpenHAB. Repéré à https://www.openhab.org/docs/installation/security.html.

OpenHab. (2019). Reading a DHT11/22 on a raspberry and send the results by MQTT. Repéré à https://community.openhab.org/t/reading-a-dht11-22-on-a-raspberry-and-send-the-results-by-mqtt/40582.

OpenHAB, C. & the openHAB Foundation e.V. (2019). OpenHab. Repéré à https://www.openhab.org/.

OpenJSFoundation. (2019). Node-Red. Repéré à https://nodered.org/.

Oracle. (2020). Qu'est-ce que l'internet of things. Repéré à https://www.oracle.com/ca-fr/internet-of-things/what-is-iot.html.

Paho, E. (2016). Eclipse Paho - MQTT and MQTT-SN software. Repéré à https://www.eclipse.org/paho/.

Planetscope. (2018). Planetscope-Statistique : Vente mondiale de smartphones. Repéré à https://www.planetoscope.com/electronique/728-ventes-mondiales-de-smartphones.html.

Psyciknz. (2017). OpenHAB-Scripts. Repéré à https://github.com/psyciknz/OpenHAB-Scripts/blob/master/mqtt.dhtsensor.py.

Qutqut, M. H., Al-Sakran, A., Almasalha, F. & Hassanein, H. S. (2018). Comprehensive survey of the IoT open-source OSs. *IET Wireless Sensor Systems*, 8(6), 323–339.

Rahman, L. F., Ozcelebi, T. & Lukkien, J. J. (2016). Choosing your IoT programming framework : Architectural aspects. *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 293–300.

Raspberry-Pi-foundation. (2019). Teach, Learn, and Make with Raspberry Pi. Repéré à https://www.raspberrypi.org/.

Raspberry-Pi-foundation. (2020). Buy a Raspberry Pi. Repéré à https://www.raspberrypi.org/products/.

Razzaque, M. A., Milojevic-Jevric, M., Palade, A. & Clarke, S. (2015). Middleware for internet of things : a survey. *IEEE Internet of things journal*, 3(1), 70–95.

Saad, R. (2016). *Modèle collaboratif pour l'Internet of Things (IoT)*. Mémoire de maîtrise, Université du Quebec à Chicoutimi, Canada.

Salami, A. & Yari, A. (2018). A framework for comparing quantitative and qualitative criteria of IoT platforms. *2018 4th International Conference on Web Research (ICWR)*, pp. 34–39.

Sibiya, S. & Olugbara, O. O. (2019). Reliable Internet of Things Network Architecture Based on High Altitude Platforms. *2019 Conference on Information Communications Technology and Society (ICTAS)*, pp. 1–4.

Singh, D., Tripathi, G. & Jara, A. J. (2014). A survey of Internet-of-Things : Future vision, architecture, challenges and services. *2014 IEEE world forum on Internet of Things (WF-IoT)*, pp. 287–292.

Singh, K. J. & Kapoor, D. S. (2017). Create Your Own Internet of Things : A survey of IoT platforms. *IEEE Consumer Electronics Magazine*, 6(2), 57–68.

Stack, T. (2018). Internet of Things (IoT) Data Continues to Explode Exponentially. Who Is Using That Data and How ? Repéré à https://blogs.cisco.com/datacenter/internet-of-things-iot-data-continues-to-explode-exponentially-who-is-using-that-data-and-how.

Sundmaeker, H., Guillemin, P., Friess, P. & Woelfflé, S. (2010). Vision and challenges for realising the Internet of Things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 3(3), 34–36.

TelluIoT. (2016). The ThingML modelling language. Repéré à https://github.com/TelluIoT/ThingML/blob/master/org.thingml.samples/src/main/thingml/incubator/DynThing/Timer.thingml.

Truong, H.-L. & Dustdar, S. (2015). Principles for engineering IoT cloud systems. *IEEE Cloud Computing*, 2(2), 68–76.

Udoh, I. S. & Kotonya, G. (2018). Developing IoT applications : challenges and frameworks. *IET Cyber-Physical Systems : Theory & Applications*, 3(2), 65–72.

Vashi, S., Ram, J., Modi, J., Verma, S. & Prakash, C. (2017). Internet of Things (IoT) : A vision, architectural elements, and security issues. *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pp. 492–496.

Vasilevskiy, A., Morin, B., Haugen, Ø. & Evensen, P. (2016). Agile development of home automation system with thingml. *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, pp. 337–344.

Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., Jubert, I. S., Mazura, M., Harrison, M., Eisenhauer, M. et al. (2011). Internet of things strategic research roadmap. *Internet of things-global technological and societal trends*, 1(2011), 9–52.

Vorto, E. (2016). Vorto. Repéré à https://www.eclipse.org/vorto/documentation/overview/introduction.html.

Wagner, M., Laverman, J., Grewe, D. & Schildt, S. (2016). Introducing a harmonized and generic cross-platform interface between a Vehicle and the Cloud. *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1–6.

Wohlin, C. et al. (2012). *Experimentation in software engineering*. Springer Publishing Company, Incorporated.

Zhang, Y. (2011). Technology Framework of the Internet of Things and its Application. *2011 International Conference on Electrical and Control Engineering*, pp. 4109–4112.