

PAN-AFRICAN UNIVERSITY
INSTITUTE FOR WATER AND ENERGY SCIENCES
(including CLIMATE CHANGE)

Master Dissertation

Submitted in partial fulfillment of the requirements for the Master degree in

Energy Engineering

Presented by

Abdelkader SALLEMINE

**An Open-Source Approach to Wireless Internet of Things (Iot):
Smart Agriculture Proof-of Concept Using FIWARE**

Defended on 04/09/2019 Before the Following Committee:

Chair	Amine Boudghène Stambouli	Prof. USTO – Oran
Supervisor	Stefano Tondini	Dr. UNITN – Trento
External Examiner	Erick Tambo	Dr. UNU-EHS –Bonn
Internal Examiner	Abdelhalim Benmansour	Prof. UAB – Tlemcen

Pan African University Institute of Water and Energy Sciences (including Climate
Change)

**An Open-Source Approach To Wireless Internet of Things (Iot): Smart Agriculture
Proof-of Concept Using FIWARE**

By: Abdelkader SALLEMINE
B.A Industrial Engineering

This research thesis is submitted in fulfillment of the requirements of the Master of
Science in Energy Engineering at the Pan African University Institute of Water and Energy
Sciences (including Climate Change)-(PAUWES) at the University of Tlemcen in Algeria.

Supervisor : Dr. Stefano ToSndini

Tlemcen, Algeria

September 2019

DECLARATION

I, **Abdelkader SALLEMINE** do hereby declare that this thesis is my original work and to the best of my knowledge, it has not been submitted for any award in any University or Institution.

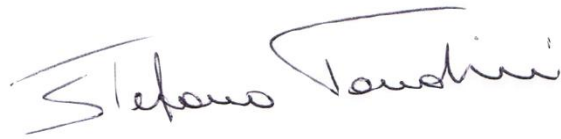


Signed _____ Date 25-08-2019

Abdelkader SALLEMINE

CERTIFICATION

I, **Stefano TONDINI** hereby certify that this thesis has been submitted with my approval as the supervisor

A handwritten signature in black ink that reads "Stefano Tondini". The signature is written in a cursive style with a large, sweeping initial 'S'.

Signed _____ Date 25-08-2019

Dr. Stefano TONDINI

ABSTRACT

Agriculture is considered as a vital activity for providing food to the increasingly growing population reaching 8.6 billion in 2030, according to the UN. Agriculture is directly tied to the availability of water and requires irrigation always. Although the world's freshwater resources are scarce and in limited amounts, many of the existing irrigation systems and practices are highly inefficient, leading to a significant waste of water. Modern problems require modern solutions, the emergence of the cloud computing paradigm supported in shaping many of the existing solutions to tackle water resources management in agriculture, combining the latest technological developments from various fields. The principal purpose of this work is to demonstrate a novel approach for water resources management in precision agriculture by exploiting a context information management framework named FIWARE. The outcome is a complete context-aware system combined of the various Generic Enablers provided by FIWARE, capable of making autonomous irrigation decisions (using Complex Event Processing) which are not solely based on sensor-related measurements, but further on the integration of different sources of information such as weather forecast data from OpenWeatherMap. By doing so, optimal management of water resources according to the predicted weather conditions has been achieved within the system.

Keywords: FIWARE, NGSI, OpenWeatherMap, Orion Context Broker, Context information management, Smart irrigation.

RÉSUMÉ

L'agriculture est considérée comme une activité vitale pour l'alimentation d'une population qui atteint 8,6 milliards d'habitants en 2030, selon l'ONU, et qui ne cesse de croître. L'agriculture est directement liée à la disponibilité de l'eau et nécessite toujours l'irrigation. Bien que les ressources mondiales en eau douce soient rares et en quantités limitées, un bon nombre des systèmes et de pratiques d'irrigation existants sont très inefficaces, ce qui entraîne un gaspillage considérable de l'eau. Les problèmes modernes exigent des solutions modernes, l'apparition du paradigme du cloud computing a contribué à structurer de nombreuses solutions existantes pour gérer les ressources en eau en agriculture, combinant les derniers développements technologiques dans des domaines variés. L'objectif principal de ce travail est de démontrer une nouvelle approche de la gestion des ressources en eau dans l'agriculture de précision en exploitant un framework de gestion des informations de contexte nommé FIWARE. Le résultat est un système complet qui est informé du contexte autour et associe les différents Generic Enablers fournis par FIWARE qui sont capables de prendre des décisions autonomes en terme d'irrigation (en utilisant le traitement des événements complexes) qui ne sont pas seulement basées sur des mesures de capteurs mais également sur l'intégration des différentes sources d'information telles que les données de prévisions météorologiques à partir du OpenWeatherMap au système pour une gestion optimale des ressources hydrauliques selon les conditions météorologiques prévues.

Mots clés; FIWARE, NGSI, OpenWeatherMap, Orion Context Broker, Gestion de l'information contextuelle, irrigation intelligente

ACKNOWLEDGEMENTS

I want to reveal the most profound gratitude to my supervisor Dr. Stefano Tondini, who has been more than a supervisor, always been there for guidance, support, advice, appreciation and honestly without his direction and determined help and encouragement this dissertation would not have been accomplished.

I am grateful to all of those with whom I have had the pleasure to work during my internship in EURAC research, especially the fantastic Center for Sensing Solutions (CSS) team. Each of the team members continuously helped through finding solutions to persisting problems with the software, fix bugs, and overcoming every obstacle. I have learned and inspired a lot from you in both personal and professional life.

I would especially like to thank Dr. Eric Tambo, my host institute supervisor, which helped much in the process of getting the internship as well as getting to the internship place.

Without forgetting the most important persons to me in this life which supported me since day one more than anyone else, my lovely parents and siblings.

I am forever grateful to you all.

Thank you. Grazie mille.

TABLE OF CONTENTS

DECLARATION	ii
CERTIFICATION	iii
ABSTRACT.....	iv
RÉSUMÉ	v
ACKNOWLEDGEMENTS	vi
TABLE OF CONTENTS.....	vii
LIST OF FIGURES	xi
LIST OF TABLES	xiii
NOMENCLATURE	xiv
CHAPTER ONE.....	1
GENERAL INTRODUCTION.....	1
1.1 Introduction	2
1.2 Background Information	2
1.3 Problem Statement	3
1.4 Research Questions	3
1.5 Objectives.....	3
1.5.1 General objective.....	3
1.5.2 Specific objectives.....	4
1.6 Scope of the Study.....	4
CHAPTER TWO	5
LITERATURE REVIEW	5
2.1 Introduction	6
2.2 Context Data Management.....	6
2.2.1 Database Management System.....	7
2.2.2 Popular Database Models.....	7
2.2.3 Context Brings Value	8
2.3 IoT Approaches and Architectures.....	8
2.3.1 Multiagent System Architectures	9
2.3.2 OpenMTC.....	10
2.3.3 SiteWhere	11
2.3.4 AWS IoT	12
2.4 FIWARE Platform.....	13
2.4.1 Context Broker and NGSI data model	15

2.4.2	Multi-Tenancy Headers.....	16
2.4.3	Additional Components.....	17
2.4.3.1	Core Context Management.....	18
2.4.3.2	Interface with Iot, Robots And Third-Party Systems	18
2.4.3.3	Context Processing, Analysis, and Visualisation	18
2.5	Docker	19
2.6.1	LoRaWAN	22
2.6.2	LoRaWAN Device Classes	23
2.6.2.3	Class C devices.....	24
2.6.3	LoRaWAN Regional Frequency Specification	24
CHAPTER THREE		26
METHODOLOGY		26
3.1	Introduction	27
3.2	Architecture Design.....	27
3.2.1	Building Blocks.....	28
3.2.1.1	Core Context Management :.....	28
3.2.1.2	Interface with Iot, Robots, and Third-Party Systems:	28
3.2.1.3	Context Processing, Analysis, and Visualization	30
3.2.2	Deployment View	30
3.3	Components setup	31
3.3.1	Hardware	31
3.3.1.1	Sensor node	33
3.3.1.2	Actuator node	34
3.3.1.3	LoRaWAN gateway	35
3.3.2	Energy harvesting and management.....	35
3.3.3	LoRaServer Network manager.....	37
3.3.3.1	LoRa gateways	38
3.3.3.2	LoRa Gateway Bridge.....	38
3.3.3.3	LoRaServer.....	39
3.3.3.4	LoRa Geo Server	39
3.3.3.5	LoRa App Server.....	39
3.3.3.6	Application.....	39
3.3.4	Orchestrator: Orion Context Broker.....	39
3.3.4.1	Spatial location format	40
3.3.4.2	GeoJSON format	40

3.3.4.3	Geographical Queries	40
3.3.4.4	Subscriptions	41
3.3.4.5	Registration Context Availability Management	42
3.4	IoT Agents: LoRaWAN & Ultralight.....	43
3.4.1	South Bound Traffic (Commands) :	44
3.4.2	North Bound Traffic (Measurement)	46
3.4.3	Device Provisioning	47
3.4.4	Provisioning a service group	48
3.4.5	Provisioning a sensor	48
3.4.6	Provisioning an actuator.....	49
3.5	Data Storage	49
3.5.1	Identified Generic Enablers that use MongoDB	50
3.5.1.1	Orion	50
3.5.1.2	Perseo	50
3.5.1.3	QuantumLeap and CrateDB :	50
3.6	Data Visualization: Grafana	52
3.7	Complex event processing: Perseo.....	55
3.7.1	Perseo front-end	56
3.7.2	Perseo core	56
3.7.3	MongoDB.....	56
3.7.4	Orion context broker	57
3.7.5	Portal	57
3.7.6	Orion Database.....	57
3.7.7	SMS gateway.....	57
3.7.8	SMPP server.....	57
3.7.9	SMTP server.....	57
3.7.10	Generic HTTP server	58
3.7.11	Authorization server.....	58
3.7.12	Rule setting.....	58
3.7.13	Action Request Structure	59
3.7.14	SMS action	60
3.7.15	Email action.....	60
3.7.16	HTTP request action.....	61
3.8	External sources of information:	61
3.8.1	OpenWeatherMap API.....	61

3.8.2	Agro API	61
3.8.3	NGSI Parser Module	61
CHAPTER FOUR.....		63
RESULTS AND DISCUSSION		63
4.1	Introduction	64
4.2	Smart agriculture use-case implementation	64
4.3	Application Mock-Up and Prototype	64
4.3.1	System operating principle.....	67
CHAPTER FIVE		69
CONCLUSION AND		69
RECOMMENDATIONS		69
5.1	Conclusion and discussion	70
5.2	Recommendations: pros and cons of FIWARE.....	71
Bibliography		72
APPENDIX.....		75
2)	method to provision a service group is done as follows:.....	75
3)	request used to provision a sensor:.....	76
4)	Provisioning single devices are done using the following post request:	77
5.3	Research grant use:.....	87

LIST OF FIGURES

Figure 2. 1 The layered structure of a multiagent IoT architecture	9
Figure 2. 2. OpenMTC architecture adapted from [10].....	11
Figure 2. 3. SiteWhere approach adapted from [10]	12
Figure 2. 4. SiteWhere approach adapted from [10]	13
Figure 2. 5 Smart Agrifood Application using FIWARE GE.	14
Figure 2. 6 Context elements in Orion representation adapted from [14].....	16
Figure 2. 7. Context elements of the room adapted from [14]	16
Figure 2. 8. Context elements of the room adapted from [15]	17
Figure 2. 9. How to extend a solution by FIWARE adapted from [1]	18
Figure 2. 10 (left). Application containerized after Docker.	19
Figure 2. 11 Virtual machine representation as an abstraction of physical hardware adapted from [17]	19
Figure 2. 12. The place of LPWAN (LoRa and LoRaWAN) in the IoT wireless connectivity ecosystem as seen in 2015 according to the LoRa Alliance.....	22
Figure 2. 13. Downlink network communication latency as a function of the LoRaWAN device class adapted from [21].....	23
Figure 2. 14 Bi-directional communications uplinks and downlinks in LoRaWAN.adapted from [20]	24
Figure 2. 15. Uplinks and downlinks management in LoRaWAN class C devices.adapted from [20]	24
Figure 3. 1 Architecture design of the IoT prototype accomplished in this work	27
Figure 3. 2 Deployment view of the IoT prototype.	30

Figure 3. 3 (left) Pycom LoPy4.....	33
Figure 3. 4 Microcontroller block diagram adapted from [25].....	33
Figure 3. 5 Connection of the soil moisture sensor to the microcontroller expansion board via the I2C bus.....	34
Figure 3. 6 Main network components of the proposed prototype adapted from [31].....	37
Figure 3. 7 LoRaServer architecture adapted from [33].....	38
Figure 3. 8. Activity Diagram for the Ultralight IoT Agent - South Bound Traffic (Actuation).....	44
Figure 3. 9 Activity Diagram for the LoRaWAN IoT Agent - North Bound Traffic (Measurement).....	46
Figure 3. 10. Usage of QuantumLeap adapted from [55].....	52
Figure 3. 11. Grafana data source setup tab.....	53
Figure 3. 12. Adding a new dashboard for data visualization in Grafana.	54
Figure 3. 13 Query builder of Grafana.	54
Figure 3. 14. Example of Grafana dashboard to visualize the temperature evolution on time.	55
Figure 3. 15. (left) Complex event processing using Perseo (right) Interaction between Perseo and other FIWARE components adapted from [38].....	56
Figure 3. 16. External Data flow Diagram in Perseo.....	58
Figure 3. 17. Interactions between Perseo, MongoDB, and the SMS gateway in the process of triggering the action.....	60
Figure 3. 18. NGSI parser module working principle adapted from [40]	62
Figure 4. 1 Schematics of the complete system put in place.	64
Figure 4. 2 Hardware components of the system prototype.	65
Figure 4. 3 Flowchart of the implemented irrigation routine.	67

LIST OF TABLES

Table 2. 1 Difference from data and information adapted from (“Difference between Information and Data,” n.d.).....	6
Table 2. 2 Principal IoT wireless communication protocols in comparison. Adapted from (Jawad, Nordin, Gharghan, Jawad, & Ismail, 2017)	21
Table 2. 3 LoRaWAN regional specification from LoRa Alliance Technical Committee, LoRaWAN™ 1.0.3 Specification, July 2018.	25
Table 3. 1 Comparison between Direct Communication and MQTT brokering. Adapted from (FIWARE 203, 2018/2019)	29
Table 3. 2 Low-cost water pump technical details.	35
Table 5. 2 Pros and Cons of FIWARE platform	71

NOMENCLATURE

API	Application Programming Interface
CA	Calculation Agent
CB	Context Broker
CEP	Complex Event Processing
CRUD	Create Read Update Delete
CSV	Comma Separated Values
DB	DataBase
DBMS	Database Management System
DCA	Data Capture Agent
DRVO	Data Routing Virtual Organization
DSEs	Domain-Specific Enablers
DTVO	Data Translator Virtual Organization
EPL	Event Processing Language
EUI	End-Device Unique Identifier
GE	Generic Enabler
GPRS	General Packet Radio Services
gRPC	Google Remote Procedure Calls
GUI	Graphical User Interface
HDSA	Heterogeneous Data Supervisor Agent
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICT	Information and Communication Technology

IMA	Irrigation Manager Agent
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
KDVO	Knowledge Discovery Virtual Organization
KMA	Knowledge Manager Agent
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LWM2M	Low Weight Machine to Machine
M2M	Machine to Machine
MLA	Machine Learning Agent
MQTT	Message Queuing Telemetry Transport
NGSI	Next Generation Service Interfaces
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
SMPP	Secure Sockets Layer
SMS	Short Message Peer-to-Peer
SMTP	Short Message Service
SSL	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UL	Ultra-Light

URL	Uniform Resource Locator
UL	Unsupervised Learning
UDP	User Datagram Protocol
UIVO	User Interaction Virtual Organization
UI	User Interface
VO	Virtual Organization
WPA	Wi-Fi Protected Access
WSN	Wireless Sensor Network

CHAPTER ONE

GENERAL INTRODUCTION

1.1 Introduction

The aim of this work is to demonstrate and realize a low-cost and open (from both software and hardware side) IoT proof of concept based on WSN (Wireless Sensor Network) to serve in environmental monitoring and smart farming, as the world is facing significant challenges in terms of food security, water scarcity, growing population and resources demand. A particular focus is given to agriculture, considering the specific aspect of irrigation as a use case. The complete system developed has proven to be reliable in making autonomous decisions upon data collected in the lab, integrated with external context information. Preliminary results demonstrate that a high degree of interoperability between the different components has been achieved. Moreover, the “open” trait of the system can allow a more profound and unrestricted understanding of the recent IoT developments which, in turn, will empower more creativity and innovation in the face of current challenges.

1.2 Background Information

Recent advanced developments in the field of IT allowed the emergence of what is known as cloud computing becoming a trend in the modern business world. Many companies have aligned their activities and businesses with the cloud computing trend hosting their activities on remotely managed infrastructures. In the sector of IoT, the cloud is used to host the various components responsible for processing, filtering, and storing the data. This fast adoption of the cloud computing paradigm led to the appearance of many cloud computing service providers offering services in the form of Software as a Service (SaaS) (López-Riquelme, Pavón-Pulido, Navarro-Hellín, Soto-Valles, & Torres-Sánchez, 2017) , Infrastructure as a Service (IaaS), and Platform as a Service (PaaS). However, many of these providers use a closed-source platform and propose vendor linked solutions which always puts services under the provider’s control, decreasing flexibility for customization or further development and generating extra costs. Therefore alternative open-source platforms supporting smart application commence are taking place, and one example is FIWARE (“Developers Catalogue—FIWARE,” n.d.). It is used to serve as a middleware in a smart application connecting smart devices and managing the sensed data to monitor the environment parameters, identify events and patterns, and take action accordingly. FIWARE features the

concept of consuming not only data acquired from sensors but also context data which is usually acquired from third party sources in order to energize and surround the collected data with more information about the context resulting in more accurate informed decisions.

1.3 Problem Statement

The problem statement is how to switch from the traditional data management consisting of collecting data and storing for analysis to context data management which consists of coupling the collected data with the context data of the surrounding environment for more context-awareness in a smart irrigation application. The topic requires skills related to computer science, such as algorithmic thinking/reasoning, coding, manipulating electronic parts, and building circuits. It also needs for basics in the field of network management and ICT system engineering to allow a good understanding of the current paradigm shift from data management to context information management, which is occurring for the most advanced digital platforms.

1.4 Research Questions

- 1) How can a Wireless Sensor Network (WSN) system be implemented for Environmental Monitoring purposes using an open-source approach to maintain its low cost, but still with a high degree of reliability?
- 2) How can we integrate recent IoT developments and technologies to empower Smart Agriculture applications with an interoperable approach?
- 3) What is the effect of setting an Irrigation Routine based on distributed sensor monitoring and external context data compared to the traditional irrigation ways?

1.5 Objectives

1.5.1 General objective

The main target of this work is to demonstrate a novel approach for water resources management in precision agriculture exploiting a context information management framework. The proof of concept has been carried out by putting together many open-source building blocks in a complementary way. By doing so, a complete system capable of taking

autonomous decisions concerning different sources of information has been achieved, in compliance with the IoT paradigm.

1.5.2 Specific objectives

The specific objectives addressed during the thesis have been:

1. Develop the system architecture design;
2. Select the main hardware components;
3. Investigate the most widespread IoT communication protocols;
4. Gain the basics of network management;
5. Understand the FIWARE framework and select the components to be used;
6. Integrate external sources of information into the system;
7. Set up an elaborate event processing routine;
8. Build a complete prototype to test in the lab.

1.6 Scope of the Study

The scope of this work focuses on the technological benchmark of one of the most advanced frameworks for context information management, namely the FIWARE platform. Because of its novelty, only a few attempts to bring this technology into operational has been carried out. Several examples have been published by the FIWARE community so far, but none of them cover the whole data-lifecycle, from acquisition to action passing through the organization, processing, analysis, and sharing. This work aims at demonstrating that a complete solution, totally based on open source components, is now within reach and can face relevant use cases in an operational environment. Besides, the used approach allowed to address the crucial issues of interoperability with external systems/components and of scalability. Finally, the suggested solution is meant to be a viable alternative to vendor link products, as it demonstrates comparable performances to systems already on the market.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

In this chapter, the state of the current context data management approach in IoT is addressed through a detailed state of the art review. In the second part of this chapter, the FIWARE platform is introduced considering the fundamental building blocks, namely the context broker, the NGSI data model, the additional components available and the Docker containerized platform philosophy.

2.2 Context Data Management

In the beginning, a quick distinction between the meaning of the terms “data” and “information” has to be established, as it is usually confused and is of great importance to the topic. According to Guru99 (“Difference between Information and Data,” n.d.) data and information are compared against defined parameters to indicate the main points of difference that exist (see Tab. 2.1).

Table 2. 1 Difference from data and information adapted from (“Difference between Information and Data,” n.d.)

Parameters	Data	Information
Description	Quantitative, qualitative variables used for developing ideas and conclusions.	It is a set of processed data which is meaningful to the context.
Etymology	The word data originates from Latin, Datum meaning “to give something.”	Originates from French and middle English, meaning “act of informing” used usually in the domain of education and communication.
Format	Found in letters, numbers, set of characters.	Usually, in the form of inferences from the interpreted data.
Meaning	Meaningless, not related to any specific context or purpose.	Holds the meaning of the interpreted data.
Feature	Data alone is raw and meaningless, represented in a single unit.	Information is derived by joining a group of data giving it a logical meaning.
Dependence	No dependence on information	Direct dependence on data.
Measuring unit	In bits and bytes.	In meaningful units like pressure, temperature.

Support for decision making	Not fit for decision making.	Suitable for decision making as it carries a logical meaning.
Knowledge level	Low-level knowledge.	The second level of knowledge.
Characteristic	Property of an organization and is not available for sale to the public.	Available for sale to the public.
Example	10	10 ° Celsius
Usefulness	Data may or may not be useful to the researcher.	Information is valuable and readily available to the researcher for use.

Both data and information have to be stored into appropriate containers, namely databases, which are exploited to get the best out of them, as described in the following.

2.2.1 Database Management System

The database management system is defined as a software package responsible for managing the database by harmonizing the data, its format, record and file structure defining rules to approve and manipulate the data inside the database (Rouse, n.d.). DBMS ensures that the data remains accessible and consistently organized. It also establishes an interface between the end-user or application and the database itself ⁴.

2.2.2 Popular Database Models

Among the different database models, relational database management system (RDBMS) is suitable to most use cases. NoSQL DBMS is well adapted for data structures that are likely to evolve. In-memory database management system (IMDBMS) demonstrates faster response times and improved performance. Columnar database management system (CDBMS) fits well for databases with a large number of related data items. In the cloud-based data management system, the cloud service provider is responsible for providing and maintaining the DBMS ⁴. Eventually, time-series DBMS is designed to efficiently store and query various time series data with high transfer volumes (“Time Series DBMS - DB-Engines Encyclopedia,” n.d.).

2.2.3 Context Brings Value

Context is defined as the set of circumstances, conditions that shape the setting related to an event. Usually, figures in the form of ready to consume information and used to provide clear insights and a surrounding broad view about the conditions of the execution of the process (“Context | Definition of context by Lexico,” n.d.). First, context is not just a state; it is a component of a process. It is not enough for the system to act appropriately at a specified moment: it must act appropriately during the process in which users are engaged. Acting appropriately must be described to the process in question, not only the sequence of states that structure the process (Coutaz, Crowley, Dobson, & Garlan, 2005).

For instance, readings of different parameters such as soil moisture or temperature from the sensor would reveal the current state of the soil, and in case the sensors report values corresponding to “dry” soil, it would indicate the need to trigger irrigation. However, if weather forecasts predict a very high possibility of rainfall the process of irrigation should not be triggered and should be taken into account in order not to lead to a waste in water resources or even flooding the soil with water which could harm the plants. Context seen as a whole process not only a state supports and allows the right actions to be taken at the right time since the fusion of information from sensors with the context information would allow making the “proper” irrigation decisions. In this example, merging the sensed data with the weather forecast data exponentials the value and impact of decision making allowing it to be more efficient in using water resources by applying rules and algorithms when appropriate, like saving water when forecasts indicate rain or threats such as hot arid weather, avoiding risks and damages in real-time. The notion of context is already widely used in the development of web and mobile application. The so-called “customized or personalized services” are used to deliver services or ads according to the user profile, location, and device (Coutaz, Crowley, Dobson, & Garlan, 2005).

2.3 IoT Approaches and Architectures

Many strategies have already been searched in order to bring context management into IoT applications. In order to tailor the state of art analysis to the focus of this work, in this section, we will consider only the approaches/architectures that have already been applied to smart agriculture pilots. (Chiewchan, Anthony, & Samarasinghe, 2019)⁵

2.3.1 Multiagent System Architectures

(González-Briones et al., 2018) Designed and developed a multiagent system based on cloud computing paradigm tested for making efficient irrigation decisions and optimized use of actuators in the system with the help of a WSN system deployed in a rural area field. The system is layered, where each of the layers integrates components responsible for multiple manipulation and handling of data. The different blocks are depicted in the following Fig 2.1.

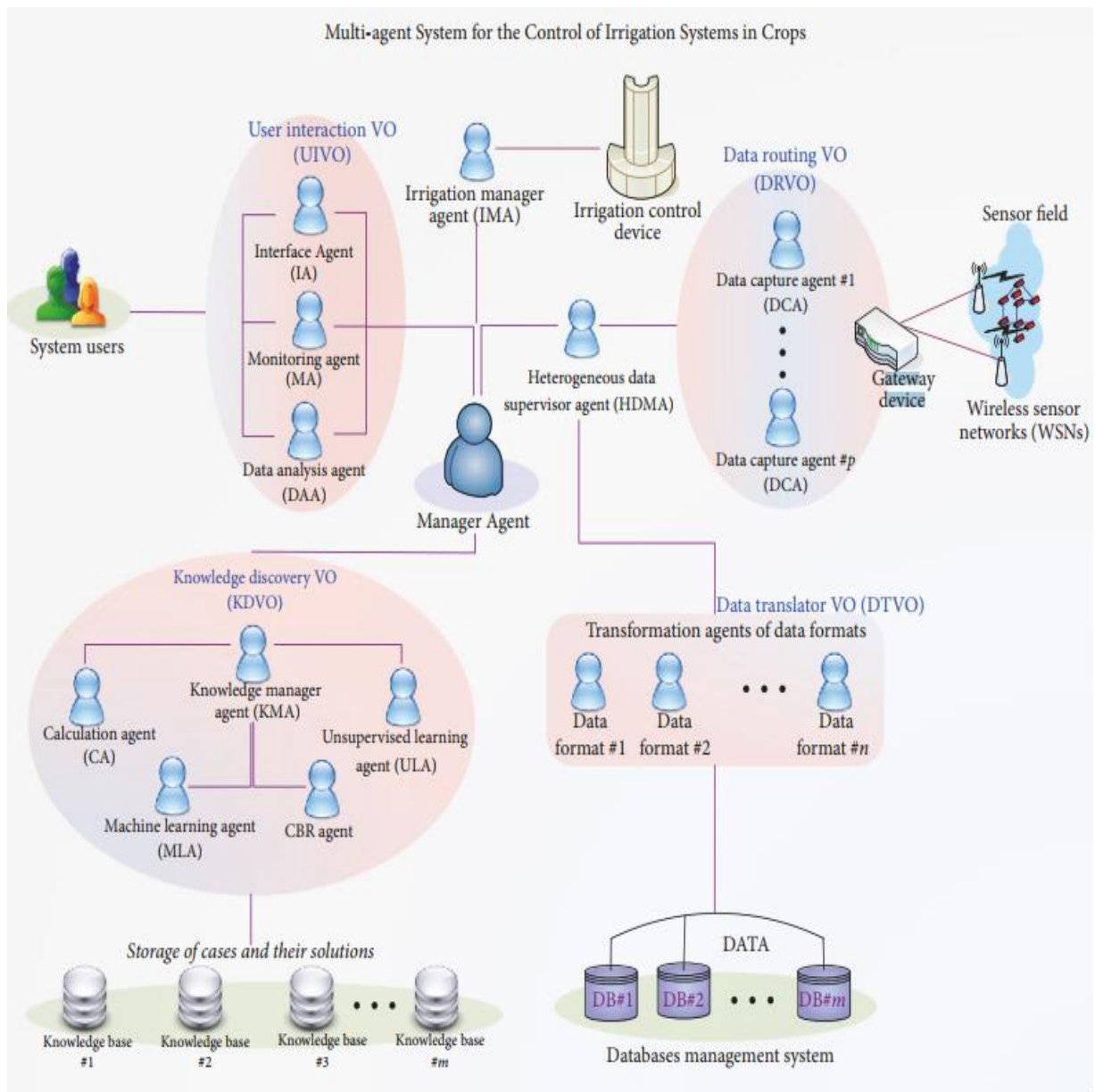


Figure 2. 1 The layered structure of a multiagent IoT architecture

In the schematics, the manager agent (MA) is responsible for managing agents in the surroundings. Heterogeneous data supervisor agent (HDMA) is responsible for converting the communicated data from the sensors into a consistent format as well as persisting them in the database for future queries. The irrigation manager agent (IMA) converts the results given by the KDVO virtual organization into an action that is triggered by the irrigation control device. The UIVO virtual organization is the interface with the system users that allows them to get, analyze information, monitor sensor readings, and state. The DRVO is a group of agents responsible for collecting sensor field data using DCA (Data capture agents). The MAS is designed to handle data for more than one crop type. Consequently, each DCA is assigned to capture data from a particular crop type reported by the corresponding sensor. In order to store the data in compliance with the database supported formats, some converting operations have to be done after capturing the data. The agents of the DTVO are responsible for transforming the data from one format to another due to the heterogeneity of the captured data (different type of crops) also to the standard data format used by MAS. Each agent is assigned to transform data to a specific format besides each crop type is assigned with a particular database. In the end, the KDVO represents the brain of the system that adds intelligence to the system through supervised and unsupervised learning agents. It extracts knowledge and identifies patterns from analyzing the data coming from the sensors, as well as predicting their behavior under different circumstances.

2.3.2 OpenMTC

It is an open-source cloud IoT platform that implements the specific architecture highlighted in Fig. 2.2. It is composed of multiple connected building blocks mainly the applications block, the OpenMTC (“OpenMTC,” n.d.) front-end and back-end, which in turn encompass the Core Feature and Connectivity components. The latter is responsible for managing devices while the former ensures the translation of messages coming from devices heading towards the targeted components and vice-versa. The linking component OpenEPC represented in between also performs filtering data operations and applies rules. The architecture is complete by an interface component with external M2M applications (Ali, Shah, Farooq, & Ghani, 2017) and finally, the sensors and actuators block.

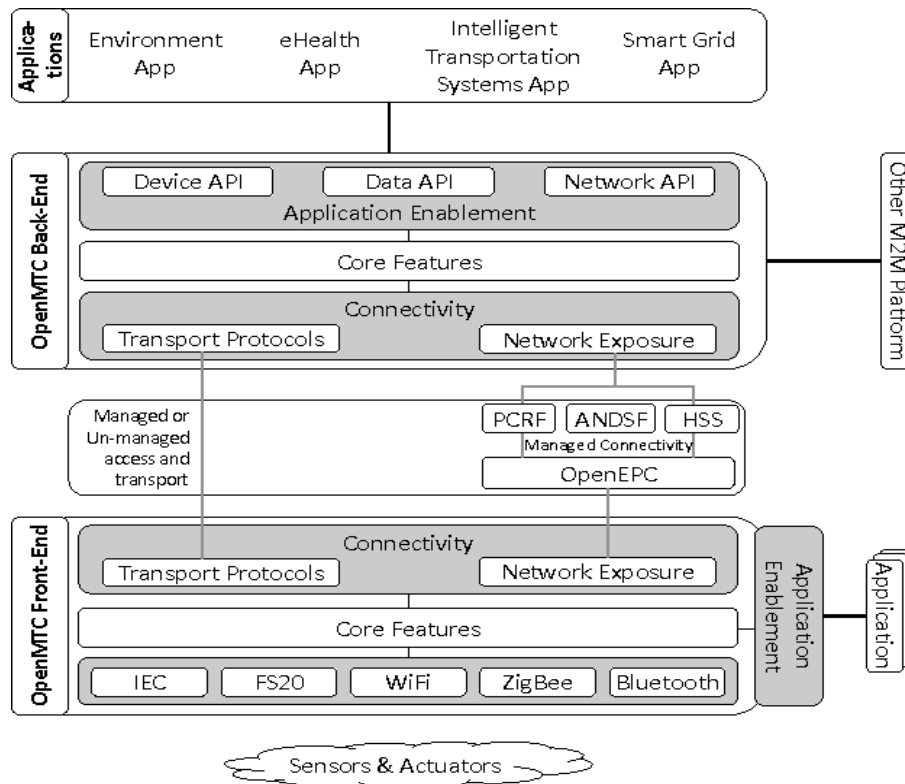


Figure 2. 2. OpenMTC architecture adapted from (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016)

2.3.3 SiteWhere

It is certified open-source IoT platform comprised of multiple building blocks as depicted in Fig. 2.3 (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016). SiteWhere (“SiteWhere Open Source Internet of Things Platform,” n.d.) is built of the central core element that connects applications and encompasses the Tenant Engine, which holds, in turn, a device management and communication engine. The latter takes in charge all possible device interactions ranging from provisioning of new devices to receiving events and sending commands over a set of various protocols. More applications can be linked to the platform through available REST APIs, Asset SPIs, and data storage SPIs.

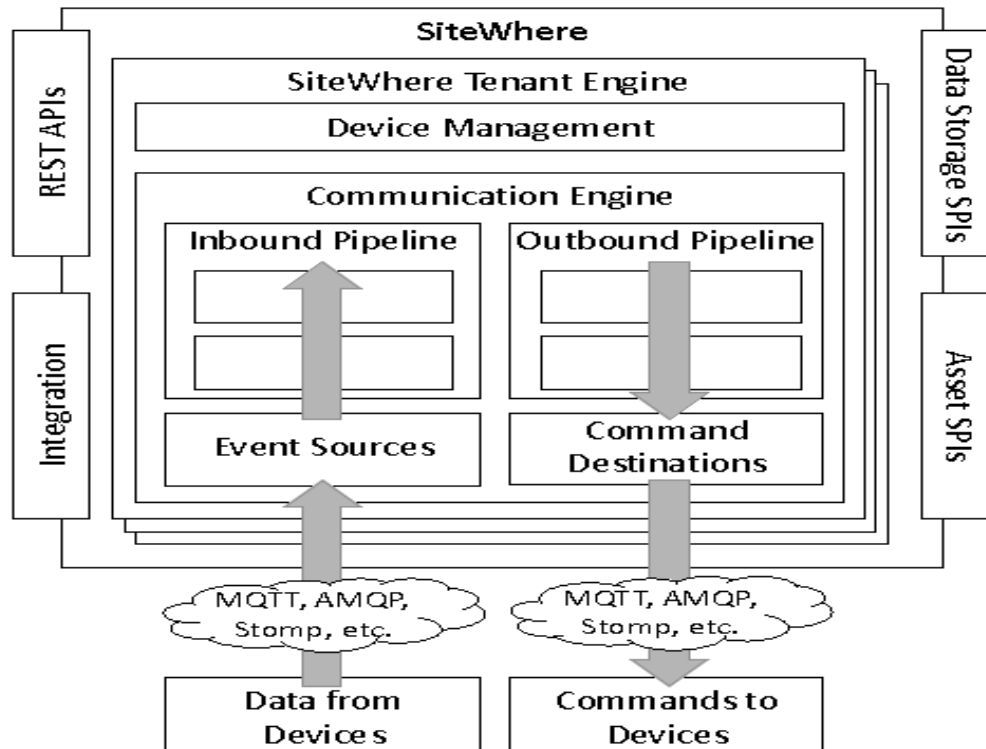


Figure 2. 3. SiteWhere approach adapted from (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016)

2.3.4 AWS IoT

It is another management IoT cloud platform (“Services et produits de cloud Amazon | AWS,” n.d.). It features collecting, storing, and analyzing the data in a reliable and secure manner. Fig. 2.4 depicts the architecture of AWS. It gathers six components working together: the thing shadow, the message broker, the rules engine, the thing registry, the thing shadows service, and service identity. The message broker ensures direct and secure communication with the things through the publish-subscribe mechanism for message exchange supported by communication protocols such as MQTT enabling more effortless scalability and low latency.

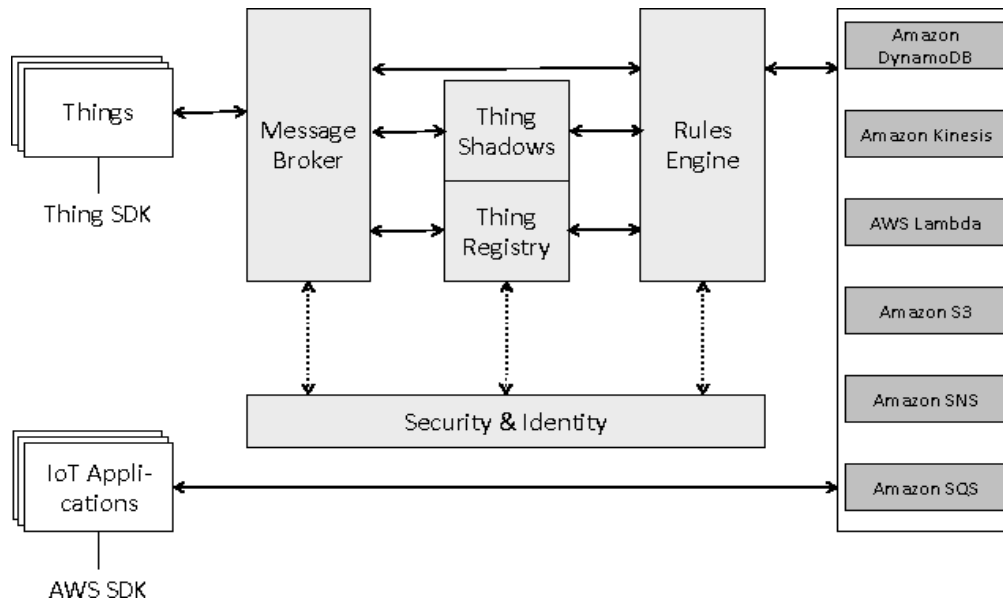


Figure 2. 4. SiteWhere approach adapted from (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016)

Messages received by the message broker are then processed and integrated with other AWS services thanks to the Rules Engine. The latter is also responsible for matching rules against events and invoke actions by sending messages back to the thing whenever a rule matches the condition. The Thing Registry is responsible for handling resources associated with each thing. The Thing Shadow Service persists information related to the things (i.e., current state) in a JSON file format at the AWS cloud. The Security and Identity Service is responsible for securing all data transactions inside the AWS platform from sending data to the things or the rest of AWS services to granting permissions and access to other applications to connect.

2.4 FIWARE Platform

FIWARE is an EU initiative within the EU's Seventh Framework Programme (FP7) initially launched in 2011 to deliver an innovative platform using standardized APIs capable of creating and delivering cost-effective services offering guarantees about security and high QoS (Salhofer, 2018). FIWARE is meant to be the reference implementation for future internet targeting various application domains (Martínez, Pastor, Álvarez, & Iborra, 2016). FIWARE aims at providing a publicly available standard platform while taking into consideration sustainability for the global ecosystem factor (Celesti et al., 2019). FIWARE is

labeled as an ideal open-source solution for smart city implementations that are independent of any vendor lock-in (Araujo, Mitra, Saguna, & Åhlund, 2019).

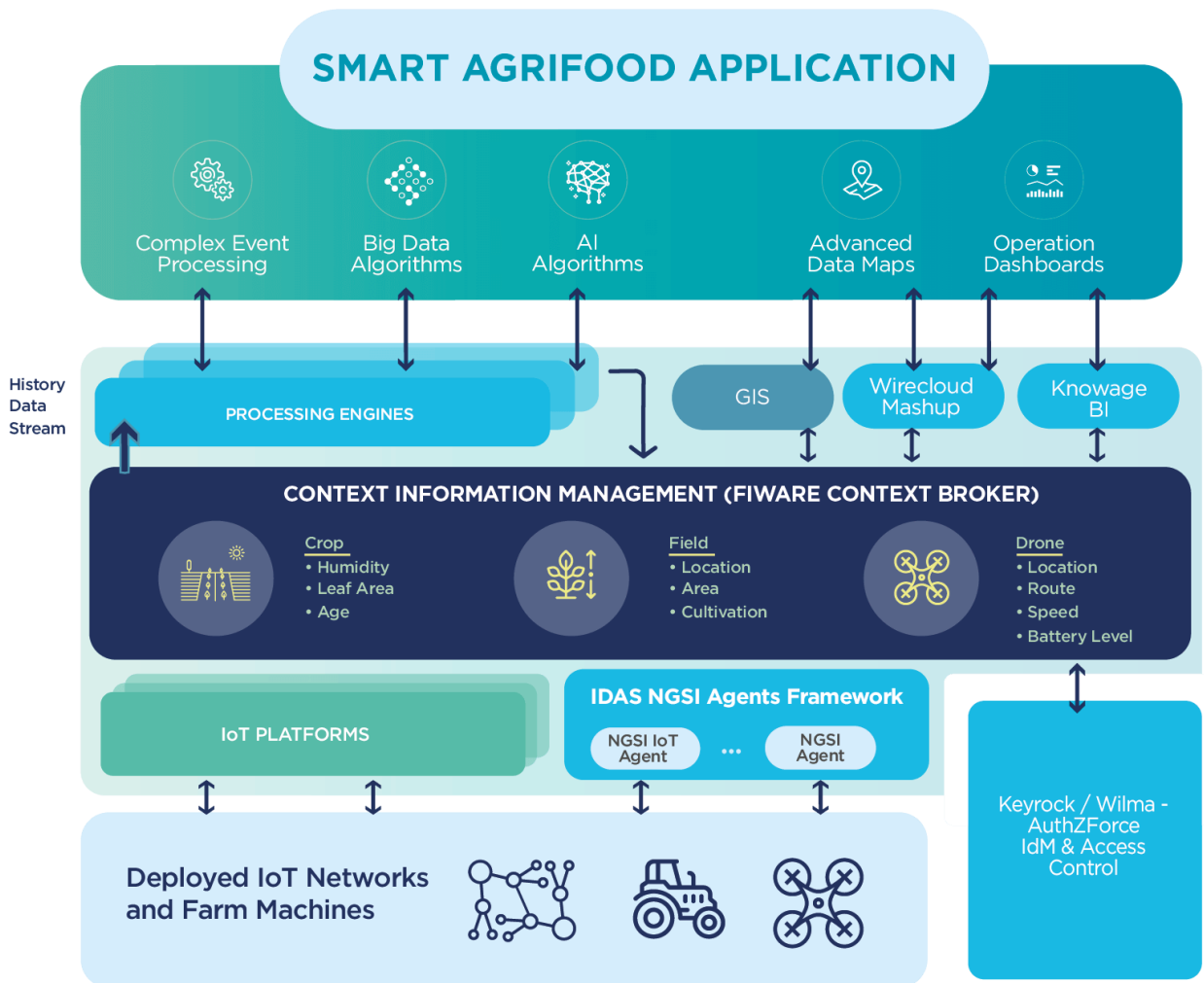


Figure 2. 5 Smart Agrifood Application using FIWARE GE.

The reference architecture of the FIWARE cloud is based on OpenStack, an open-source cloud middleware that is mostly used and adopted in the industry. It incorporates what is called generic enablers (GEs). They are a set of broad platform functions accessible through APIs ensuring data context management, security, advanced middleware interface to devices and networks (Celesti et al., 2019). In addition, for individual domains such as energy, health, there are unique components or more accurately “domain-specific enablers,” which are specifically related to the domain. However, general GEs are organized in this manner:

1. **Data/Context Management:** englobes all the components that afford the process, storage, access and analysis of the data;
2. **IoT Services Enablement:** these components serve as an interface between the sensor networks and the rest of GEs, casting sensor data into proper formats and routing data to the rest of GEs;
3. **Advanced Web-based User Interface:** these components are responsible for creating friendly designed user interfaces to display information such as sensor measurements or geographical information on a map;
4. **Security:** components used to check identities, authenticate, and enforce security;
5. **Applications/Services and Data Delivery:** all components used for generating simple applications and dashboards used to visualize data or remote control;
6. **Cloud Hosting:** secure provisioning and management of services using cloud infrastructures;

All FIWARE components are run as containers using Docker. The file holding configuration for the targeted components to be run is called the docker-compose file and is of a YAML format. It is attached to the Appendix.

2.4.1 Context Broker and NGSI data model

The Context Broker (CB) Generic Enabler is a simple but a robust REST API responsible for operations like establishing queries, updates, subscriptions, and registrations to changes of the current context information in a decentralized and highly scalable manner (“Developers Catalogue—FIWARE,” n.d.). The implementation of the CB in FIWARE is called Orion, as previously mentioned. Orion is a set of open RESTful API based on the NGSI interface that sits on top of a MongoDB database used for the storage of data related to the created context elements, parameter, and configurations for devices, subscription, and registrations (Salhofer, 2018). The context in FIWARE is composed of several sub-blocks called context elements (see Fig 2.6). They have a similar representation to the data elements existing in many existing programming languages.

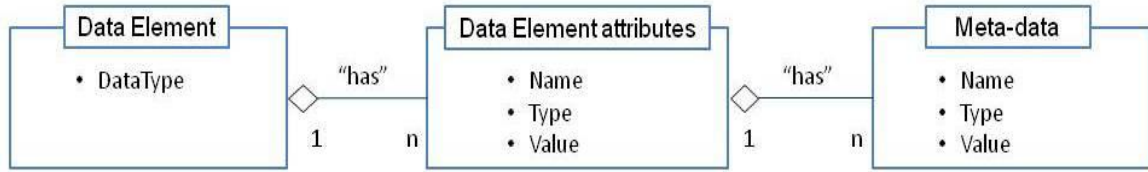


Figure 2. 6 Context elements in Orion representation adapted from (“Context Management Architecture—FIWARE Forge Wiki,” 2016)

Data elements are characterized by a `DataType`, and data element attributes or properties, such as `Name`, `Type`, and `Value` (Salhofer, 2018). Context Elements share the same structure except that it goes a little bit further by addressing the data element as an “entity” that is uniquely defined by an `EntityId` and a type giving it a flexible and easy way to map the desired entity and attribute value in an ecosystem consisting of a countless number of entities (Salhofer, 2018). Entities can be seen as the representation of physical objects existing in our environment, for instance, actuators, sensors, or a room (Martínez, Pastor, Álvarez, & Iborra, 2016). Assuming a room’s temperature and relative humidity parameters are being monitored, the room’s entity would consist of context elements such as `EntityId` “Room-01” and a type of “Room” including 2 active attributes temperature and relative humidity constantly acquiring sensor readings and are reported in a variable of a type “Number” and a value of “10.2”, for instance. The context element (see Fig. 2.7) is then represented in the following structure.

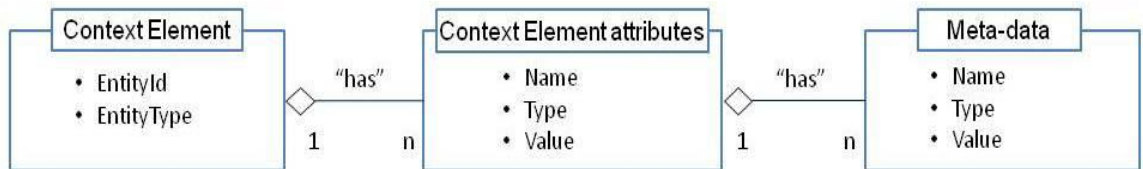


Figure 2. 7. Context elements of the room adapted from (“Context Management Architecture—FIWARE Forge Wiki,” 2016)

2.4.2 Multi-Tenancy Headers

Furthermore, FIWARE uses a unique concept for giving a well-organized hierarchical scope for the various application’s data entities using the “Fiware-service” header found in the header section of every HTTP request structure. The entities are identified using the “fiware-service” header, which is included at the time creating the entity. Operations, like querying or subscribing to an entity, require the usage of the corresponding scopes.

As an example, Fig. 2.8 represents different level scopes, ranging from the high level which is the city “Madrid,” that has as subordinates or children, i.e., “Gardens” and “Districts” forming the second level of scope. The third level consists of “Parquesur,” “ParqueNorte” and “ParqueOeste,” as children of “Gardens,” and so on. Having these different levels of scopes makes it easier to map precisely the desired attribute value found in the corresponding entity. The multi-tenancy header comes in handy, especially in a large and complex implementation that combines many elements and objects. The context broker plays the role of the orchestrator of the flow of data. Thanks to a subscription to event mechanism, other components or GEs can subscribe to changes in values or “events” of a specific entity’s attributes. Events are defined as an identified occurrence or a pattern, that usually leads to changes in the context data enabling other components or event-aware GEs to manage the information and execute rules or algorithms for decision making or actuation.

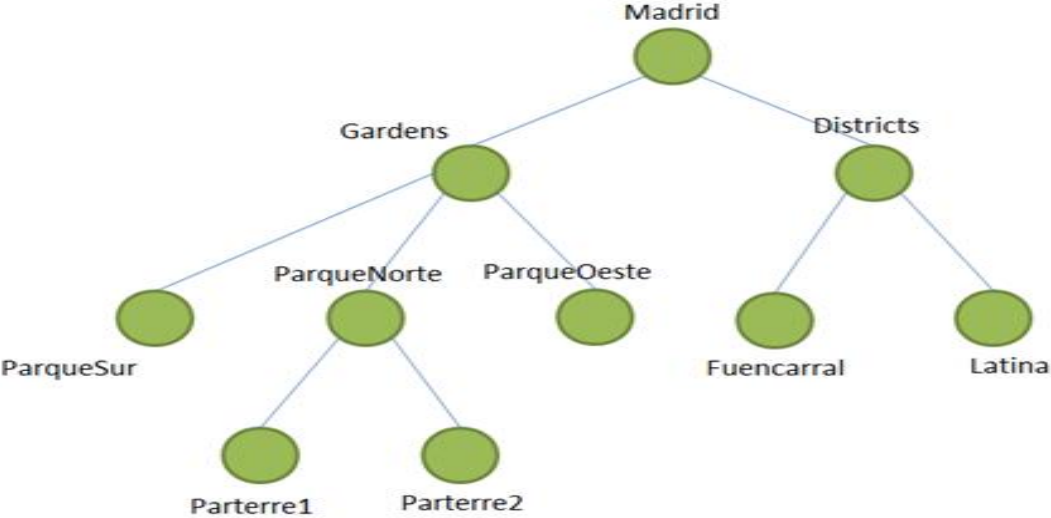


Figure 2. 8. Context elements of the room adapted from (“Entity service paths—Fiware-Orion,” n.d.)

2.4.3 Additional Components

As previously mentioned, FIWARE comes with a set of components or GEs each one of them is responsible for implementing a specific function (see Fig. 2.9). The Orion CB is the only compulsory component for a “Powered by FIWARE” solution (“Developers Catalogue—FIWARE,” n.d.). Other components are optional and deployed based on the need.

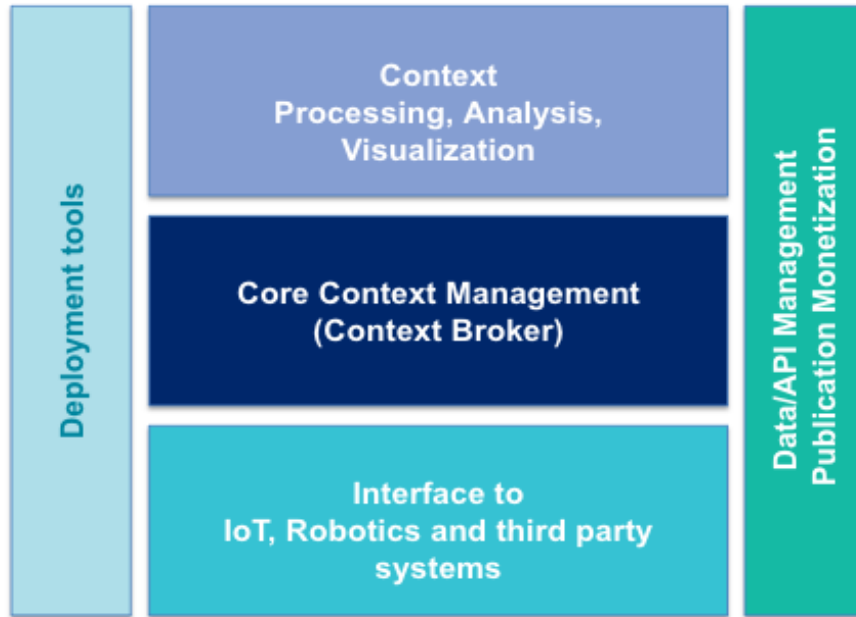


Figure 2. 9. How to extend a solution by FIWARE adapted from (*“Developers Catalogue—FIWARE,” n.d.*)

For this work to demonstrate a prototypal application in the field of smart agriculture using FIWARE, the following components have been selected among all.

2.4.3.1 Core Context Management

- (1) The Orion Context Broker Generic Enabler.
- (2) The Quantum Leap Generic Enabler – responsible for the storage of data into CrateDB, which is a time-series database under the NGSIv2 format.

2.4.3.2 Interface with Iot, Robots And Third-Party Systems

- (3) IoT Ultralight Agent – a bridge between HTTP/MQTT and NGSI.
- (4) IoT Agent for LoRaWAN – a bridge between the LoRaWAN protocol and NGSI.

2.4.3.3 Context Processing, Analysis, and Visualisation

- (5) The Perseo Generic Enabler – responsible for Complex Event Processing (CEP) by reacting to events in real-time using predefined rules stored in a MongoDB database. Perseo uses EPL (Event Processing Language) to define rules that will trigger actions when the event matches the stored rule. Possible actions supported by Perseo are sending an email, tweet, SMS, or an HTTP request.

2.5 Docker

Docker (“Enterprise Container Platform | Docker,” n.d.) is an open-source set of platform-based service products that virtualizes the OS instead of hardware. It clusters the machine’s OS system kernel, packages code, and dependencies in what is known as a container. The software can be quickly built, deployed, shared, and run using containers. A container is seen as a software component that wraps up the developed code with all its dependencies and libraries into one container capable of running reliably on any other machine irrespective of the settings the target machine is implementing.

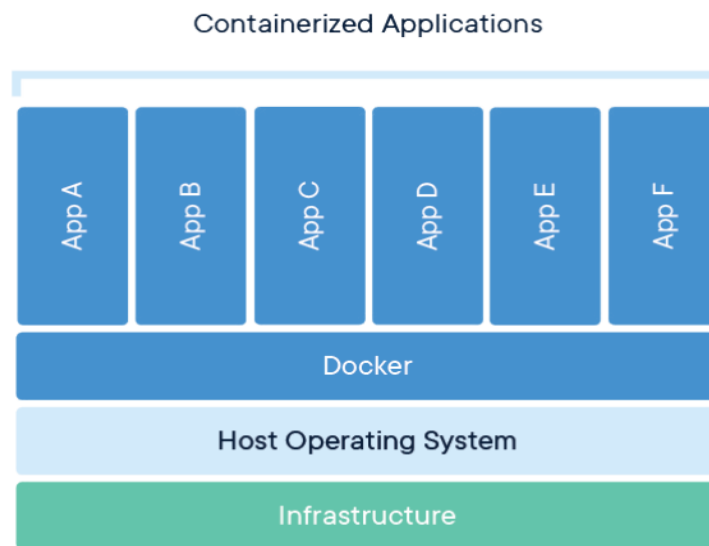


Figure 2. 10 . Application containerized after Docker.

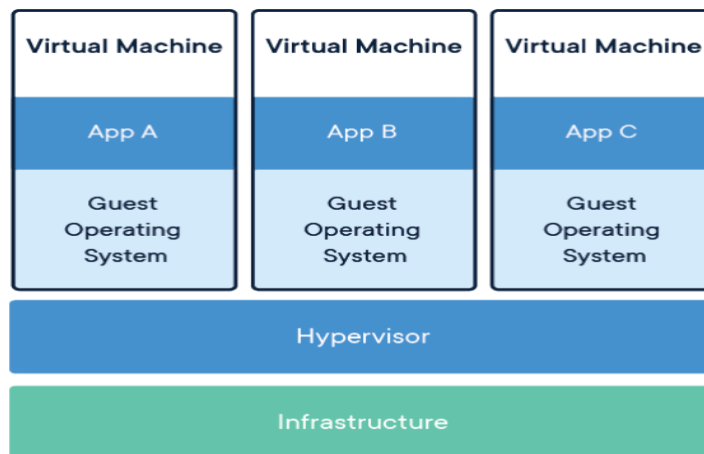


Figure 2. 11 Virtual machine representation as an abstraction of physical hardware adapted from (Definitive Guide to Enterprise Container Platforms”, June 2019)

Containers and virtual machines use equal allocation and isolation of resources. However, containers use a different implementation as it makes it possible to share the OS kernel with the rest of the containers allowing multiple containers to run each as an isolated process (see Fig 2.10). Unlikely, a virtual machine (VM) would require distinct resources and its Operating System running separately from other VMs in order to run an application, resulting in more allocated space and considerable IT infrastructure consumption (see Fig. 2.11). Containers are running instances of images, which is the configuration or setup of the virtual computer. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as an isolated process in userspace. In short, the features of the primary containers are:

- (1) Using A standard approach for increased portability;
- (2) Lightweight for an increased deployment velocity and decreased infrastructure use;
- (3) Secure for the applications as well as containers;
- (4) Sharable with other users through docker hub repository for images (“Docker Hub,” n.d.) ;

As FIWARE comes with many components, several of them are specific to particular applications, FIWARE exploits the Docker-compose approach (“Enterprise Container Platform | Docker,” n.d.).

2.6 IoT Wireless Communication Protocols

Many different wireless communication protocols are nowadays available to IoT applications. They mainly differ from power consumption, coverage range, attenuation (frequency related), and data throughput. These features make the different protocols most suited for specific applications than for others. Tab 2.2 aims at comparing the most popular protocols available for IoT.

Table 2. 2 Principal IoT wireless communication protocols in comparison. Adapted from (Jawad, Nordin, Gharghan, Jawad, & Ismail, 2017)

Parameters	ZigBee	Classic BT	BLE	WiFi	GPRS	LoRa	SigFox
Standard	IEEE 802.15.4	IEEE 802.15.1	IEEE 802.15.1	IEEE802.11a,b,g,n	N/A	IEEE 802.15.4g	IEEE 802.15.4g
Frequency band	868/915 MHz and 2.4 GHz	2.4 GHz	2.4 GHz	2.4 GHz	900–1800 MHz	869/915 MHz	868/915 MHz
Number of RF channels	1, 10, and 16	79	40	11	124	10 in EU, 8 in US	360
Channel bandwidth	2 MHz	1 MHz	1 MHz	22 MHz	200 kHz	<500 KHz	<100 Hz
Power consumption in Tx mode	Low	Medium	Ultra-low	High	Medium	Low	Low
Data rate	20, 40, and 250 kbps	1–3 Mbps	1 Mbps	11–54 and 150 Mbps	Up to 170 kbps	50 kbps	100 bps
Communication range	100 m	10–50 m	10 m	100 m	1–10 km	5 km	10 km
Network size	65,000	8	Limited by the application	32	1000	10,000 (nodes per BS)	1,000,000 (nodes per BS)
Security capability	128 bits AES	64 or 128 bits AES	64 or 128 bits AES	128 bits AES	GEA, MS-SGSN, MS-host	AES 128b Encryption	not supported
Network Topologies	P2P, tree, star, mesh	Scatternet	Star-bus	Point-to-hub	Cellular system	Star-of-stars	Star
Application	WPANs, WSNs, and Agriculture	WPANs	WPANs	WLANs	AMI, demand response HAN	Agriculture, Smart grid, environment control, and lighting control	Agriculture and environment, automotive, buildings, and consumer electronics

Limitations

short communication distance	Short communication range	Concise communication range	High power consumption	Power consumption problem	The low data rate, scalability	Low data rate
------------------------------	---------------------------	-----------------------------	------------------------	---------------------------	--------------------------------	---------------

After an in-depth review of the specification of the beforementioned communication protocols, we will now focus on the LoRaWAN, which is particularly suited for application where low power consumption and full area coverage are crucial, as it is most needed in agriculture.

2.6.1 LoRaWAN

Fig. 2.12. Shows the flow of data from and to the nodes and gateways in the LoRaWAN network protocol. It is deemed to be of Low Power Long-range Wide Area (Sinha, Wei, & Hwang, 2017) network protocol explicitly designed to connect constrained devices to the internet over long-range wireless connections. LoRaWAN is said to be meeting requirements such as end-to-end security encryption of packets, bi-directional communication, mobility, and localization services.

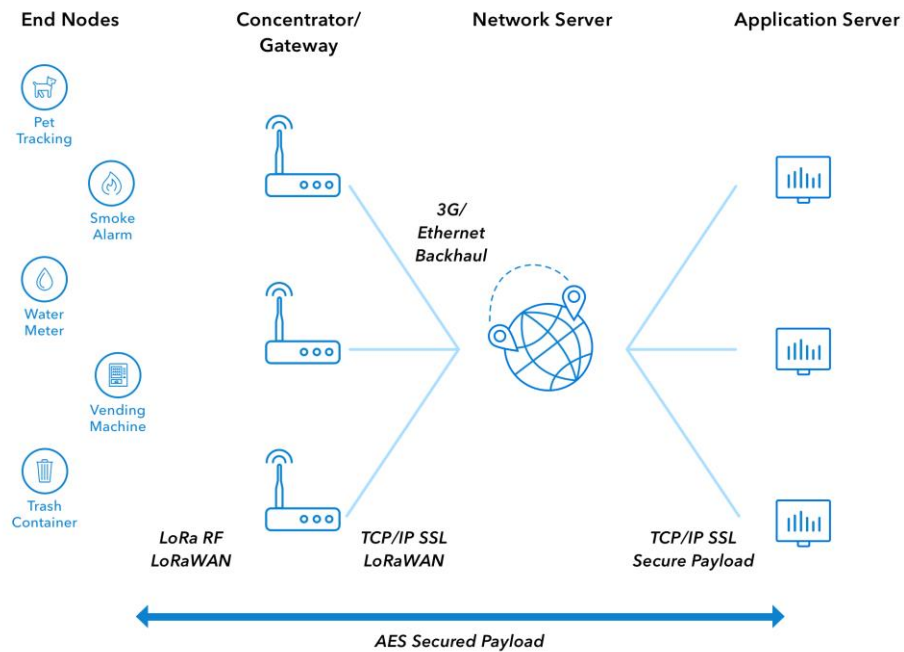


Figure 2. 12. The place of LPWAN (LoRa and LoRaWAN) in the IoT wireless connectivity ecosystem as seen in 2015 according to the LoRa Alliance

2.6.2 LoRaWAN Device Classes

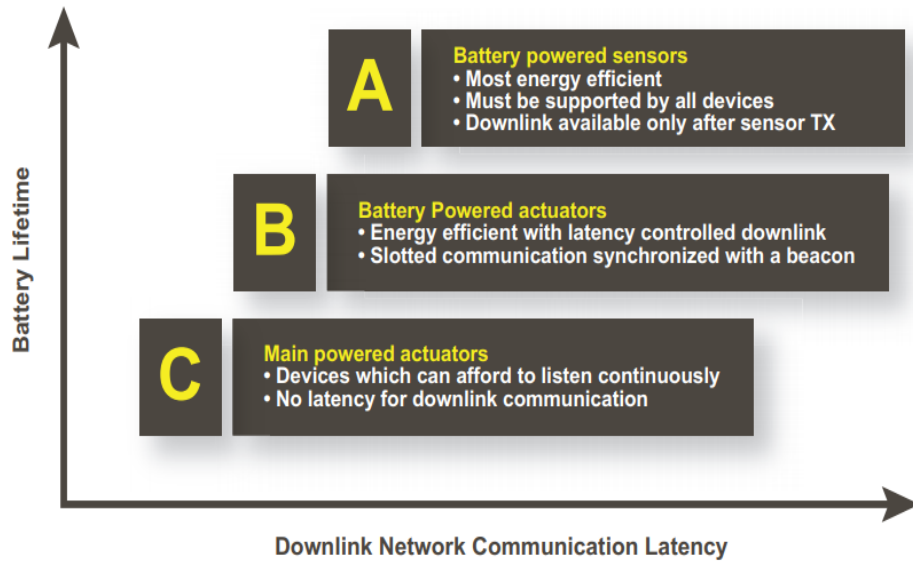


Figure 2. 13. Downlink network communication latency as a function of the LoRaWAN device class adapted from *LoRa Alliance, LoRaWAN What is it?, November, 2015*

LoRaWAN identifies three different types of devices, each grouped in a device class (A, B, C, as shown in Fig. 2.13.) based on the role or function of your device. LoRaWAN specifies that all devices must implement class A while other classes are just extensions to the class A devices (“LoRaWAN,” 2019).

2.6.2.1 Class A Devices

Lowest power end-device supports bi-directional communications uplinks and downlinks. Uplinks are packets transmitted from the sensor or actuator node back to server while downlinks are packets sent from the server to the actuator node. Class A devices are suitable for applications that are ready to receive downlinks communications from the server shortly (1 - 2 seconds) after an uplink transmission (randomly) otherwise it is going to wait until the next uplink to be able to receive the downlink (see Fig. 2.14.).

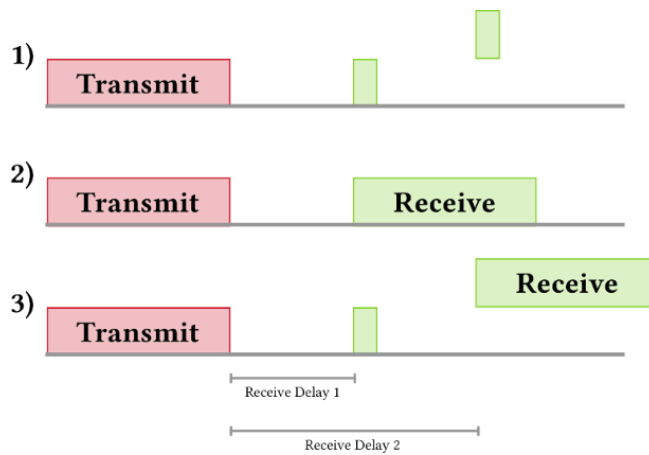


Figure 2. 14 Bi-directional communications uplinks and downlinks in LoRaWAN.adapted from (“LoRaWAN,” 2019)

2.6.2.2 Class B Devices

Energy-efficient end-devices inherits the same behavior from Class A devices (random receive windows); however, it handles downlink differently. Class B devices use scheduled downlink receive time windows by getting a time-synchronized beacon from the gateway, allowing the server to know precisely when the end-device is ready and listening for downlinks.

2.6.2.3 Class C devices

Devices which can afford to continuously listen for incoming packets unless it is transmitting something. The cost to afford is energy. Usually, this kind of devices is powered with a reliable source of power due to the fact that they are always listening for new packets sent from nodes. The C type device configuration is suitable for gateways continually listening for new downlinks from other nodes, additionally being able to continually listen for new packets reduces the latency communication time, as shown in Fig.2.15.



Figure 2. 15. Uplinks and downlinks management in LoRaWAN class C devices.adapted from (“LoRaWAN,” 2019)

2.6.3 LoRaWAN Regional Frequency Specification

Specifications for the LoRaWAN operational frequency band varies from region to another depending on the region’s regulations and requirements. Detailed specifications for the

region of Europe and North America are available, while still under the process of definition for the rest of the regions. Tab 2.3 resumes the different specifications for Europe and North America regions.

Table 2. 3 LoRaWAN regional specification from LoRa Alliance Technical Committee, LoRaWAN™ 1.0.3 Specification, July 2018.

	Europe	North America	China	Korea	Japan	India
Frequency band	867-869MHz	902-928MHz	470-510MHz	920-925MHz	920-925MHz	865-867MHz
Channels	10	64 + 8 +8	In definition by Technical Committee	In definition by Technical Committee	In definition by Technical Committee	In definition by Technical Committee
Channel BW Up	125/250kHz	125/500kHz				
Channel BW Dn	125kHz	500kHz				
TX Power Up	+14dBm	+20dBm typ (+30dBm allowed)				
TX Power Dn	+14dBm	+27dBm				
SF Up	7-12	7-10				
Data rate	250bps- 50kbps	980bps-21.9kpbs				
Link Budget Up	155dB	154dB				
Link Budget Dn	155dB	157dB				

CHAPTER THREE

METHODOLOGY

3.1 Introduction

In this chapter, the methodologies, and strategies adopted to accomplish a complete IoT system for smart irrigation, which exploits a context information management approach are reported. After the architecture description, the hardware and software components' settings are explained in detail.

3.2 Architecture Design

As explained in the previous chapters, the architecture of the system is composed of different operational layers (Core context management, interface with IoT systems) which has different GEs communicating with each other using HTTP requests. Orion is the CB and the central orchestrator of the data flow in the system.

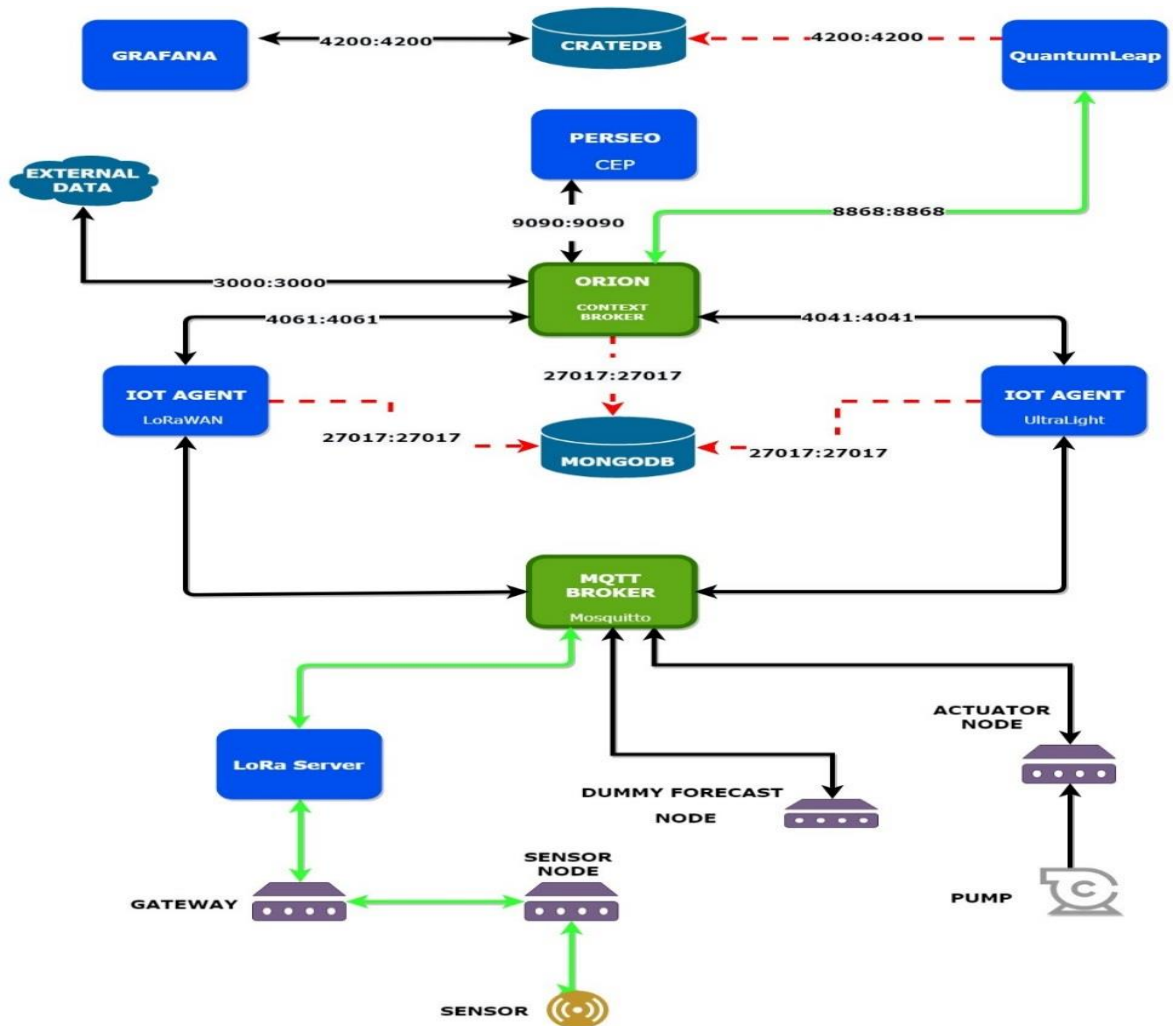


Figure 3. 1 Architecture design of the IoT prototype accomplished in this work

Using the subscription to events, changes in variables and registration of context providers mechanisms, Orion holds a strong position as the rest of the system components depend on it to get source information. The different components and GEs allocate different ports to send and receive notifications and data payloads by initiating HTTP GET and POST requests. It also integrates third-party data sources (or what is called context providers) such as the weather forecast data to be used as context information.

3.2.1 Building Blocks

The building blocks representation of Fig.3.1. Gives an overview of the complete system and its components. Different blocks represent different layers that FIWARE uses:

3.2.1.1 Core Context Management :

- (1) The Orion Context Broker Generic Enabler.
- (2) The Quantum Leap Generic Enabler – responsible for the storage of data into CrateDB, which is a time-series database under the NGSIV2 format and exposes port 8868 for information exchange.

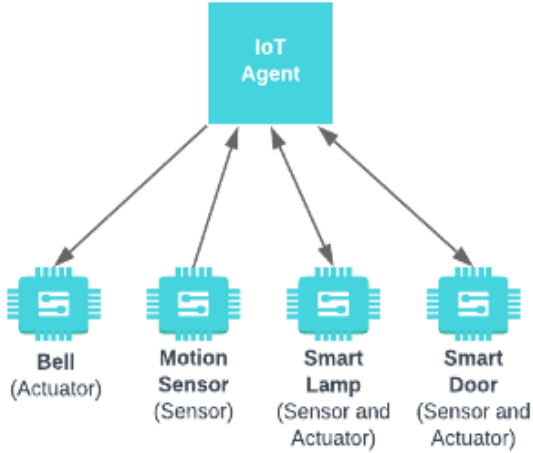
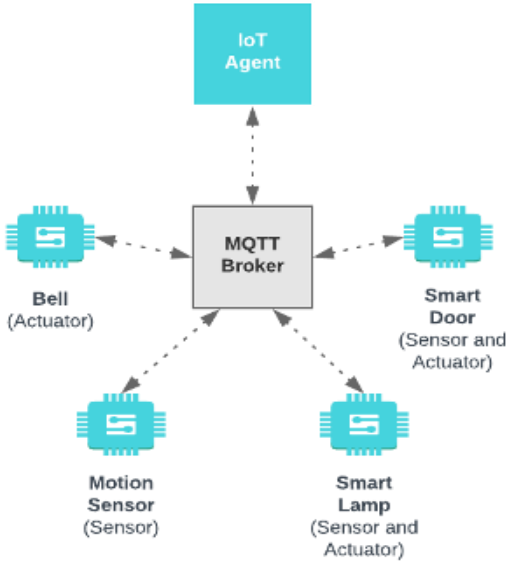
3.2.1.2 Interface with Iot, Robots, and Third-Party Systems:

- (1) IoT Agent for Ultralight – a bridge between HTTP/MQTT and NGSI responsible for parsing the received payload from the actuator node figuring in UltraLight syntax to NGSI format compatible with Orion.
- (2) IoT Agent for LoRaWAN – a bridge between the LoRaWAN protocol and NGSI responsible for parsing the received payload from the gateway in LoRa format to NGSI format compatible with Orion.

MQTT is the communication protocol used to carry the payload from the devices to the IoT Agent. It is open-source and lightweight, suitable for devices with limited resources (low power devices). The MQTT Broker plays nearly the same role as Orion CB in orchestrating the incoming and upcoming information. It implements a subscribe/publish model to exclusive and unique topic addresses structured in a manner allowing devices and the agents to communicate with each other by connecting, subscribing and publishing messages without the need to have direct communication (“MQTT main page,” 2018). Multiple devices can connect to the same topic that they are

interested in and choose to publish messages on that topic to whoever is subscribed and listening for updated messages. Devices that are posting messages and agents fetching them on the same topic do not necessarily need to know each other as long as they are publishing/subscribing to the same topic making it more flexible and more accessible than if both of them knew each other and had direct communication. Some critical differences between MQTT broker and Direct communication with devices are highlighted in Tab 3.1 (from (FIWARE 203, 2018/2019))

Table 3. 1 Comparison between Direct Communication and MQTT brokering. Adapted from (FIWARE 203, 2018/2019)

	
Direct communication between the IoT Agent and device via HTTP.	IoT Agent communicates with IoT devices indirectly via an MQTT Broker
Request-Response mechanism	Publish-Subscribe mechanism
Devices must always be ready to receive messages	Devices can subscribe to the topic at any time to receive messages
High Power demands	Low Power demands

External Data existing in JSON format can be integrated into Orion by parsing it into NGSI format using the NGSI Parser component and using the NGSI proxy would allow

Orion to register it as a context information provider making the query and update context operations an easy task using the port 3000.

3.2.1.3 Context Processing, Analysis, and Visualization

- (1) Perseo is responsible for Complex Event Processing CEP by reacting to events in real-time using predefined rules by the user. It exposes port 9090 to communicate with Orion for update or query context operations.
- (2) Grafana is responsible for the creation of dashboards, visualization, and plot of different sensed parameters. It integrates CrateDB to retrieve historical data and manipulate it to the user need.

3.2.2 Deployment View

In order to provide more insights and understanding of the system, a deployment view model have been designed, as well. It explains the different building blocks, components, and the link between them, as shown in Fig. 3.2.

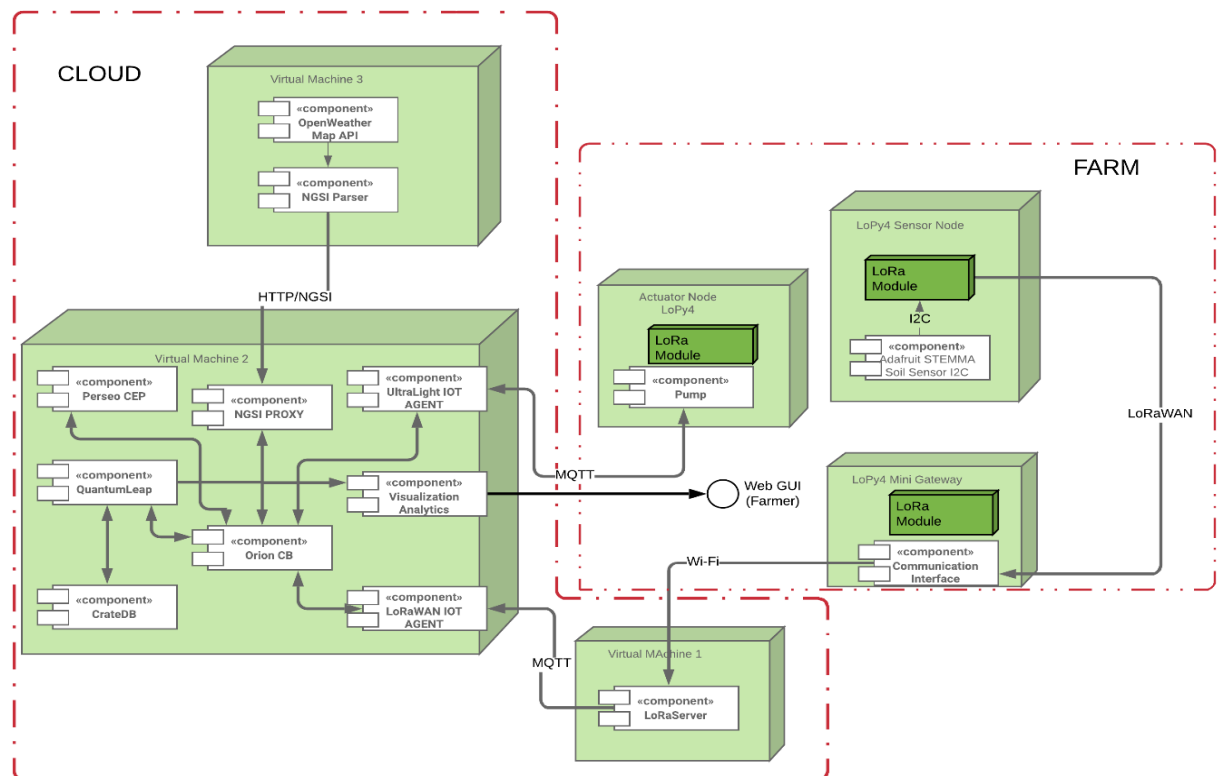


Figure 3. 2 Deployment view of the IoT prototype.

This architecture consists of several building blocks:

- 1) a sensor node deployed locally in the field to acquire environmental data;
- 2) a gateway to communicate to the platform through the LoRaWAN protocol;
- 3) an actuator node that operates a water pump upon context evolution;
- 4) a first remote virtual machine which is fed by the gateway and collects and shares the sensed data;
- 5) a second virtual machine which provides event processing and gives a business representation of information to end-users through a Grafana dashboard;
- 6) Third remote virtual machine which allows enriching the context information with the weather forecast.

3.3 Components setup

3.3.1 Hardware

The choice for the hardware used in our application is of great importance, and the selection of which board, communication protocol to use is also linked to the targeted implementation, which is smart agriculture. Our choice of the targeted board comes considering the environment the hardware is going to operate in (standalone mode) and the various climatic or environmental conditions that it is going to face. Efficiency, durability, and reliability are major factors influencing the node's performance and in turn, the whole system's performance and response time to events. Therefore, we have chosen an industrial class device known as Pycom. We have based our choice on multiple criteria of performance, openness, and cost. Pycom LoPy 4 represents the right balance between performance, power, price, and supported networking protocols. It is a MicroPython ready development board supporting four different network protocols (LoRa, Sigfox, WiFi, Bluetooth) in one small board, recognized as enterprise-grade IoT device ready to be integrated into a smart application. The specs of this microcontroller board are listed in Tab 3.2.

Tab 3.2 Specification of the Pycom LoPy4 development board adapted from Pycom, LoPy4 Datasheet

Pycom LoPy4 - Technical Characteristics	
Processing	-Espressif ESP32 chipset - Dual processor + WiFi radio System on Chip.
Power	- Input: 3.3V – 5.5V 3v3 output capable of providing up to 400mA - WiFi: 12mA in active mode, 5uA in standby - LoRa: 15mA in active mode, 1-uA in standby
Memory	- RAM: 4MB - External flash 8MB - Hardware floating-point acceleration - Python multi-threading
Security	SSL/TLS support - WPA Enterprise security
Supported Networks	LoRa, Sigfox, WiFi, Bluetooth
Interfaces	2 x UART, SPI, 2 x I2C, I2S, micro SD card - Analog channels: 8x12 bit ADCs - Timers: 4x16 bit with PWM and input capture - GPIO: Up to 24
LoRa operating, range Frequencies	- Node range: Up to 40km - Nano-Gateway: Up to 22km - Nano-Gateway Capacity: Up to 100 nodes - 868 MHz (Europe) at +14dBm maximum - 915 MHz (North and South America, Australia and New Zealand) at +20dBm maximum - 433 MHz (Europe) at +10dBm maximum
Size	Size: 55mm x 20mm x 3.5mm weight : 7 g

Due to the high flexibility of the chosen platform, we can program the same development board model, namely a LoPy4 (see figure 3.3), either to act as a sensor node, or an actuator node or eventually a gateway.

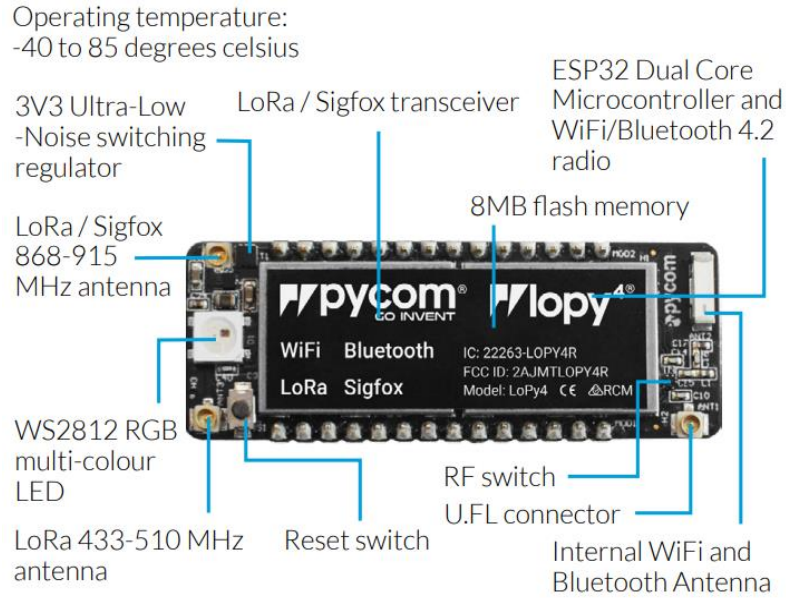


Figure 3. 3 Pycom LoPy4.

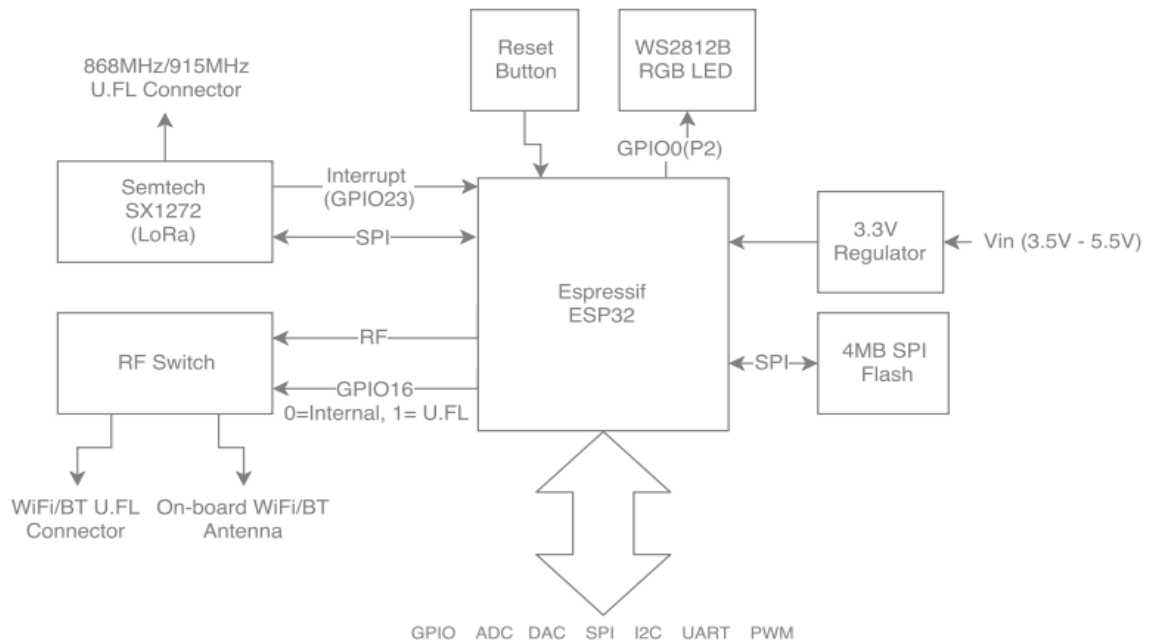


Figure 3. 4 Microcontroller block diagram adapted from Pycom, LoPy4 Datasheet

3.3.1.1 Sensor node

In the proposed prototype, two main parameters will be sensed during the whole monitoring process: temperature and soil moisture. For demonstrative purposes, we have judged using sensors which are capable of measuring temperature, and soil moisture are

suitable for our case. However, it is possible to integrate different types of sensors that could be implemented depending on the user-specific application. The choice of the sensor was Adafruit STEMMA Soil Sensor - I2C Capacitive Moisture Sensor (Adafruit Industries, n.d.), which fits perfectly as a low-cost sensor with acceptable accuracy. It can be calibrated by means of a gravimetric procedure to get a reliable soil moisture read-out. The sensor is of a capacitive type and uses a single probe that fits inside the soil to measure both soil moisture and temperature. The sensor can be interfaced with any microcontroller thanks to the I2C ready interface. The latter allows for a secure connection to the pins of the microcontroller expansion board without the need for any soldering as can be seen in Fig. 3.5.

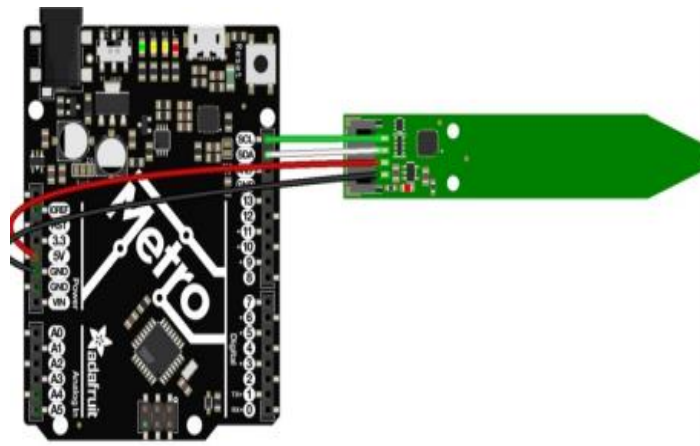


Figure 3. 5 Connection of the soil moisture sensor to the microcontroller expansion board via the I²C bus.

The code that runs inside the board in a closed-loop is attached to the Appendix with the corresponding comments. It mainly establishes a connection to the LoRaWAN gateway using the Device EUI, Application Session Key, and Network Session Key generated by the LoRaServer during the device registration. After successful connection to the network. It ensures acquiring the measurements from the sensor, calibrates the readings, opens a connection with the network, and sends the packet.

3.3.1.2 Actuator node

The actuator node ensures the successful establishment of bi-directional communication with the gateway to start receiving commands from the gateway. The actuator node is also responsible for reporting useful data like the command execution results, the state of

the actuator, and other related information. The actuator node integrates a low-cost DC pump used to pump water to simulate the process of irrigation. It is linked to a relay board that works as a power manager between the microcontroller and the pump since the microcontroller cannot supply the required energy by the pump because using low output current and voltage and are not suitable for operating the pump. The pump comes with the specifications of Tab 3.2 (from (Coutaz, Crowley, Dobson, & Garlan, 2005)). The configuration code for the actuator node is attached to the Appendix with the corresponding comments.

Table 3. 2 Low-cost water pump technical details.

<i>Specifications</i>	<i>Value</i>
<i>Power supply</i>	6~12V DC,65mA-500mA
<i>Interface</i>	DC 5.5-2.1
<i>Pumping head</i>	0-200cm
<i>Capacity</i>	0~550L/H
<i>Power range</i>	4~5W
<i>Dimensions</i>	45x43x30mm(1.77x1.69x1.18")
<i>Weight</i>	300g
<i>Cable length</i>	1m (39.37")



3.3.1.3 LoRaWAN gateway

The programming script for the LoRaWAN gateway has directly been taken from the examples published on the Pycom website (“LoRaWAN Nano-Gateway,” n.d.)). We believe that this piece of code is particularly suited for our purposes, as it implements a LoRaWAN packet forwarder that was easy to couple with the LoRaserver network manager (more details will follow). Moreover, the setting of a more complex LoRaWAN access point goes out of the scope of this work.

3.3.2 Energy harvesting and management

In the proposed prototypal system, we are dealing with two types of nodes, end nodes, and gateway. The end nodes (sensor node and actuator node) can be powered by a

rechargeable battery on a standalone mode. The nodes use a small solar panel to recharge the battery on sunny days for increased autonomy and to avoid yearly battery replacements. Sensor node's primary function is to get measurements from the sensors, encrypt them and transmit them to the gateway or sink node. The process of transmission using the LoRaWAN radiofrequency protocol is energy-consuming. Therefore, nodes are time-scheduled by the user to wake up, collect, and send measurements. When nodes are inactive (not sending measurements), they automatically go on deep sleep mode ("Deep Sleep API," n.d.). Energy consumption in deep sleep mode is deficient with the purpose of preserving the energy of the battery. Deep sleep mode is interrupted by events or triggers that are set by the user (wake up every 10 minutes and transmit the current value of temperature). The actuator node is responsible for receiving and triggering commands pushed from the context processing layer holding the action to be launched (more details will follow in the chapter). In the proposed prototype and for demonstration purposes the actuator node is also powered by a battery. However, in a real implementation that requires the use of a big engine or pump it would require constant high current and voltage, which is usually acquired directly from the grid or using a reliable source of power allowing, in turn, the actuator and the node to operate reliably. The gateway is the most energy-consuming device. This is due to the fact that the gateway is supposed to be connected to internet nearly the whole time in order to upload the readings from the sensor nodes to the cloud for processing, storage, and visualization (or even receive commands back from the unit responsible for triggering actions). The level of energy consumption depends mainly on the way the gateway uses to secure internet connectivity. For instance, if the gateway uses WiFi to connect to the internet, then power consumption is going to be higher compared to using GPRS (Grimm, 2013). Depending on which way the gateway uses to secure internet connectivity varies the level of energy consumption. However, it would not be appropriate to operate the gateway using a battery as the gateway is continuously active and constantly listening for incoming packets from other nodes ready to forward them. That is why the gateway should be plugged to a reliable source of energy. Having said so, various approaches and algorithms for efficient energy use, harvest, and saving exist in the literature and can be easily integrated into the nodes as they can be programmed in python, benefiting of the substantial energy savings.

3.3.3 LoRaServer Network manager

LoRaServer is the network manager that includes a set of applications providing mechanisms for managing registered gateways on the LoRa network, devices, and their associated applications. It also manages the data traffic between the gateways receiving packets from the nodes and the rest of the applications listening for new packets (IoT Agent in the present case). Possible integration with the FIWARE platform is sketched in Fig. 3.6 (“Description- LoRaWAN IoT Agent,” n.d.).

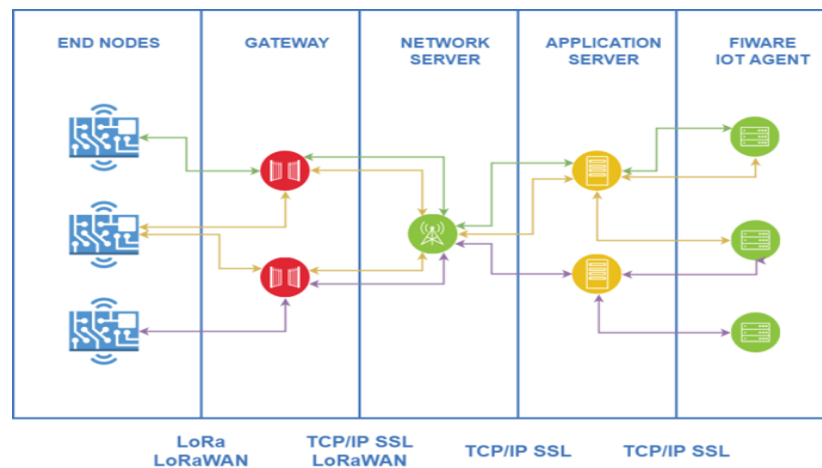


Figure 3. 6 Main network components of the proposed prototype adapted from (“Description- LoRaWAN IoT Agent,” n.d.)

LoRaServer is open-source and comes with a Web UI, the LoRa App Server component, allowing easy integration of devices and gateways configuration profiles, associating devices to applications, and monitoring live data. The network management can also be addressed using programmatic interfaces implemented in gRPC, and JSON REST API (“Project—LoRa Server, open-source LoRaWAN network-server,” n.d.) LoRaServer is built using different components illustrated in Fig. 3.7.

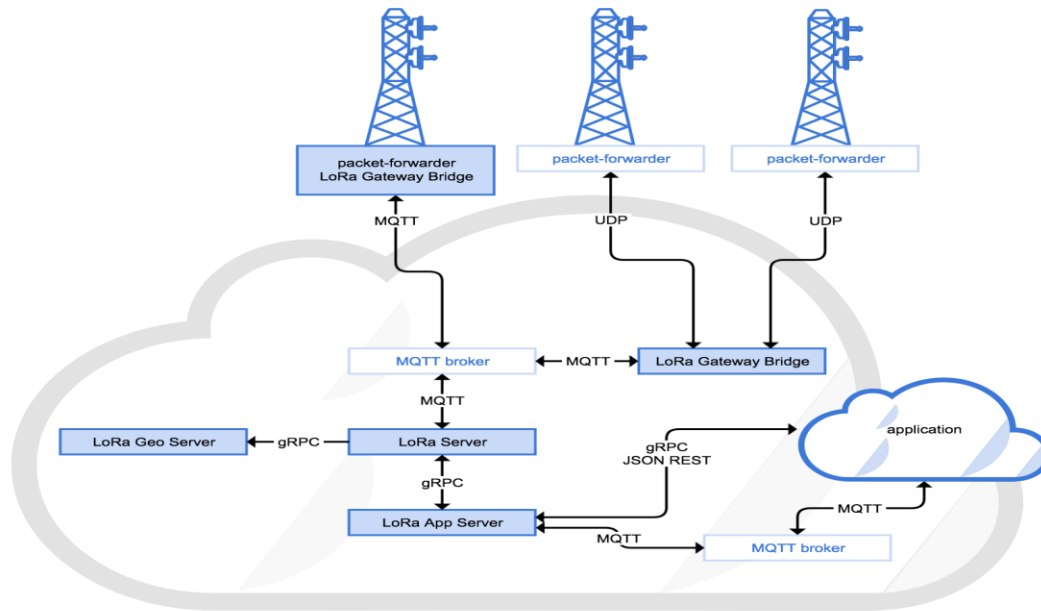


Figure 3. 7 LoRaServer architecture adapted from (“Architecture—LoRa Server, open-source LoRaWAN network-server,” n.d.).

3.3.3.1 LoRa gateways

The gateways receive the data from the end node devices and forward it straight to the LoRaserver using an implementation of the packet-forwarder software which also plays the role of an interface with the Lora hardware on the gateway.

3.3.3.2 LoRa Gateway Bridge

This leading role of the component is to communicate with the gateway and interfaces the UDP protocol used by the packet-forwarder and switch to messages over MQTT protocol for several reasons mentioned below. Debugging using MQTT is smooth compared to while using UDP. Sending a downlink becomes much more comfortable as it only requires knowledge of the target MQTT topic of the gateway and publishes the message. MQTT broker will be responsible then for forwarding it to the LoRa Gateway Bridge associated with the gateway all under secured environment and connections using the MQTT over TLS.

3.3.3.3 LoRaServer

LoRa Server integrates the LoRaWAN network server component capable of managing the state of the network. By knowing which devices are already active on the network and others that have yet joined by handling join-requests from devices willing to join. Gateways often receive identical data pieces from devices and are still forwarded to the application server. However, LoRaServer is going to run some filtering and optimization operations on data when they reach the LoRaServer such as deduplication it and removing any data items that reproduced. For downlinks sent from the app server back to gateways, the LoRaServer uses a queue mechanism to keep data until the targeted gateway is finally reachable.

3.3.3.4 LoRa Geo Server

This component is optional and provides geolocation services to know the location of each device mapped on a map.

3.3.3.5 LoRa App Server

The LoRa App server disposes of a web UI and a set of APIs for handling devices, gateways, applications, networks, and users.

3.3.3.6 Application

The Application, in this case, represents the FIWARE platform that will make use of the data by processing, storing, and deducting actions from it. The first component that is part of FIWARE to interact within this process is the IoT, which we will treat in detail in the next parts.

3.3.4 Orchestrator: Orion Context Broker

Orion (“Home—Fiware-Orion,” n.d.) Context Broker is a C++ implementation of NGSIv2 REST API, designed to manage context information and its availability. It is an NGSIv2 server implementation that is in charge of managing the complete lifecycle and handling the traffic of the context information through the CRUD operations as well as the subscription and registration mechanisms. In order for a system to be certified as “Powered by FIWARE,” it requires only one component to be deployed, which is Orion.

The other components are linked to Orion and depend on it to retrieve the context information. Entities carrying geospatial properties are provisioned in Orion and enables the resolution of geographical queries. It supports two representation formats.

3.3.4.1 Spatial location format

It is a lightweight representation designed to allow quick and easy integration of the properties to the existing entities. It supports underlying geometries representations such as point, line, box, and polygon, and it is not suitable for complex positions representations. Context attributes holding the location coordinates and encoded with the Simple Location Format are entitled to be consistent with the following syntax:

- considered value for the type attribute must conform to one of (geo: point, geo: line, geo: box, geo: polygon);
- all attributes values holding coordinates must be represented in a list and should be defined according to WGS84 Lat Long and EPSG:4326 coordinate reference system (“WGS 84: EPSG Projection—Spatial Reference,” 2007).

3.3.4.2 GeoJSON format

GeoJSON is a geospatial data interchange format (DIF) using the JavaScript Object Notation (JSON). It comes with high power and flexibility for representing points of altitude and even more complex geospatial shapes such as multiple geometries. Context attributes holding the location coordinates and encoded with the GeoJSON Format are entitled to be consistent with the following syntax:

- considered value for the type attribute must conform to GeoJSON, and the attribute’s value must be a valid GeoJSON object;
- in GeoJSON the value of longitude comes first and is followed by the latitude.

3.3.4.3 Geographical Queries

Geographical queries are used to filter, and map entities that match a particular set of parameters. Geographical queries are specified by using the following parameters: “georel” is intended to specify a spatial relationship (a predicate) between matching entities and a reference shape (geometry). It is composed of a token list separated by ‘;.’

The first token is the relationship name, the rest of the tokens (if any) are modifiers which provide more information about the relationship. The following values are recognized:

- 1) georel=near. This means that the entity is located in a certain threshold range distance to the reference geometry and takes:
 - (a) maxDistance. Expresses, the maximum distance to which the entity should be located. Unit meters.
 - (b) minDistance. Expresses, the minimum distance to which the entity should be located. Unit meters.
- 2) georel=intersects. Requires the entities to intersect with the reference geometry.
- 3) georel=equals. The matching entities should have the same position as the reference geometry.

The reference geometry used can be defined using the following supported geometries:

- 1) geometry=point.
- 2) geometry=line.
- 3) geometry=polygon.
- 4) geometry=box.

3.3.4.4 Subscriptions

Orion Context Broker has a powerful feature which allows other components to subscribe to context information, in other words keeping track of changes and updates occurring in the system to the different entities' attributes without the need to continuously repeat query requests. Orion takes in charge notifying the subscribers whenever a change to the specified attribute value occurs, instantly and automatically by sending the latest and recent value change to the components listening for updates. This mechanism goes in line with the smart application's requirements to ensure consistent and coordinated communication between all the components of the smart solution. An example of a subscription is attached to the Appendix: The POST HTTP request is sent to the Orion on the port 1026 followed by the subscriptions' endpoint, the FIWARE-service and service headers are specified. The "entities" and "attrs" subfields describe the set of targeted entities that holds the set of attributes to be sent to the subscriber. However,

“attrs” subfield of “condition” are responsible for firing the notification to the subscriber component in case the specified attribute value changed (“temperatureDegree” in this case). The rest of attributes under the “attrs” subfield do not trigger any notification; they are only attached to the notification sent carrying the attributes’ values. If no attribute is specified under the “condition” field, then Orion assumes that this notification should be triggered if at least one of the attributes belonging to the entity change, as he considers every attribute value change as a notification trigger. It is also possible to tell Orion to only fire a notification if the value of temperature is somewhere greater than 10, for instance. This is possible through using the “expression” field and including "q": "temperature>10" as an attribute. The URL where the component listens for incoming notifications from Orion is specified in the “url” subfield. In this case, the target component is Perseo-fe exposing the port 9090. The throttling element defines the frequency of firing the notification, it is the time between two successful notifications, expressed in seconds, for instance, if the value of throttling is 5 this means that the time between the first notification sent and the second one to be sent has to be 5 seconds even if a value change is reported during the waiting time it will be ignored and the notification will not be sent. It serves to define a notification arrival time for better coordination. All established subscriptions in our system are attached to the Appendix.

3.3.4.5 Registration Context Availability Management

Context availability management is interested in the source or providers of the entities and attributes. Orion implements a registration mechanism capable of integrating the various NGSI sources of information through establishing a registration allowing Orion to run query and updates to the entities whenever requested by the other components. Orion does not persist in any of the information provided by the source. However, Orion knows what is in the source (entities and attributes) and how to get them (the URL of the source). When a component requests certain entity from Orion, he fetches the DB for that specific entity. If it is not stored in the DB, like an entity provided by an external provider, Orion queries and fetches the entity for the requested value through the provided URL.

3.4 IoT Agents: LoRaWAN & Ultralight

IoT Agent GE is a component that serves as an interface between the IoT devices and gateways, ensuring secure connections, authorizing and authenticating the deployed devices for a smart application regarding Orion CB which manages their data, configuration and so on. IoT Agent is capable of “translating” from the device’s native protocol carrying the data to be compatible with Orion NGSI data model. There are various IoT agents who are already implemented or under development for various IoT communication protocols. Among them, we find:

- 1) IoTAgent-JSON –an interface between the MQTT and HTTP protocols (with a JSON payload) and NGSI
- 2) IoTAgent-LWM2M - link Lightweight M2M protocol and NGSI
- 3) IoTAgent-UL – a link between HTTP/MQTT messaging protocol (with an UltraLight2.0 payload) and NGSI
- 4) IoTagent-LoRaWAN – a link between LoRaWAN protocol and NGSI

As previously defined the IoT agent stands in the middle between the FIWARE GEs at his back and the IoT devices and gateways in front of it constantly sending sensed data from the sensors, and after being processed it will be receiving commands or actions (if any) back through the gateway to the actuator. We distinguish two ways of data traffics in this case, measurements coming from the IoT devices passing by the Agent up to the Orion CB, which we call North Bound Traffic. Commands sent from CEP component to Orion are HTTP requests generated by the latter passing through the Agent towards the actuator node are what is called South Bound Traffic.

An IoT agent is supposed to handle both north and south bounds data traffic. However, after running tests and using the LoRaWAN IoT Agent solely, the operation of receiving data measurements from sensors and forwarding them to Orion was successful. The opposite path operation was not due to some critical errors that were encountered and failed to send back commands to actuators. The encountered problem was reported to the FIWARE community and due to limited time and support, we have foreseen the use of another way to send actuations, which is through integrating the UL IoT Agent specifically for South Bound traffic and LoRaWAN IoT agent for North Bound traffic

which is acquiring sensor measurements and forwarding them to Orion in the appropriate format. Each IoT agent disposes of a unique port for the north and southbound traffic as explained in the Activity Diagram below (see Fig. 3.8.). The Activity Diagram represents the southbound traffic (commands) coming from the context processing layer to order the execution of an action, which is switching on the pump for the irrigation process.

3.4.1 South Bound Traffic (Commands) :

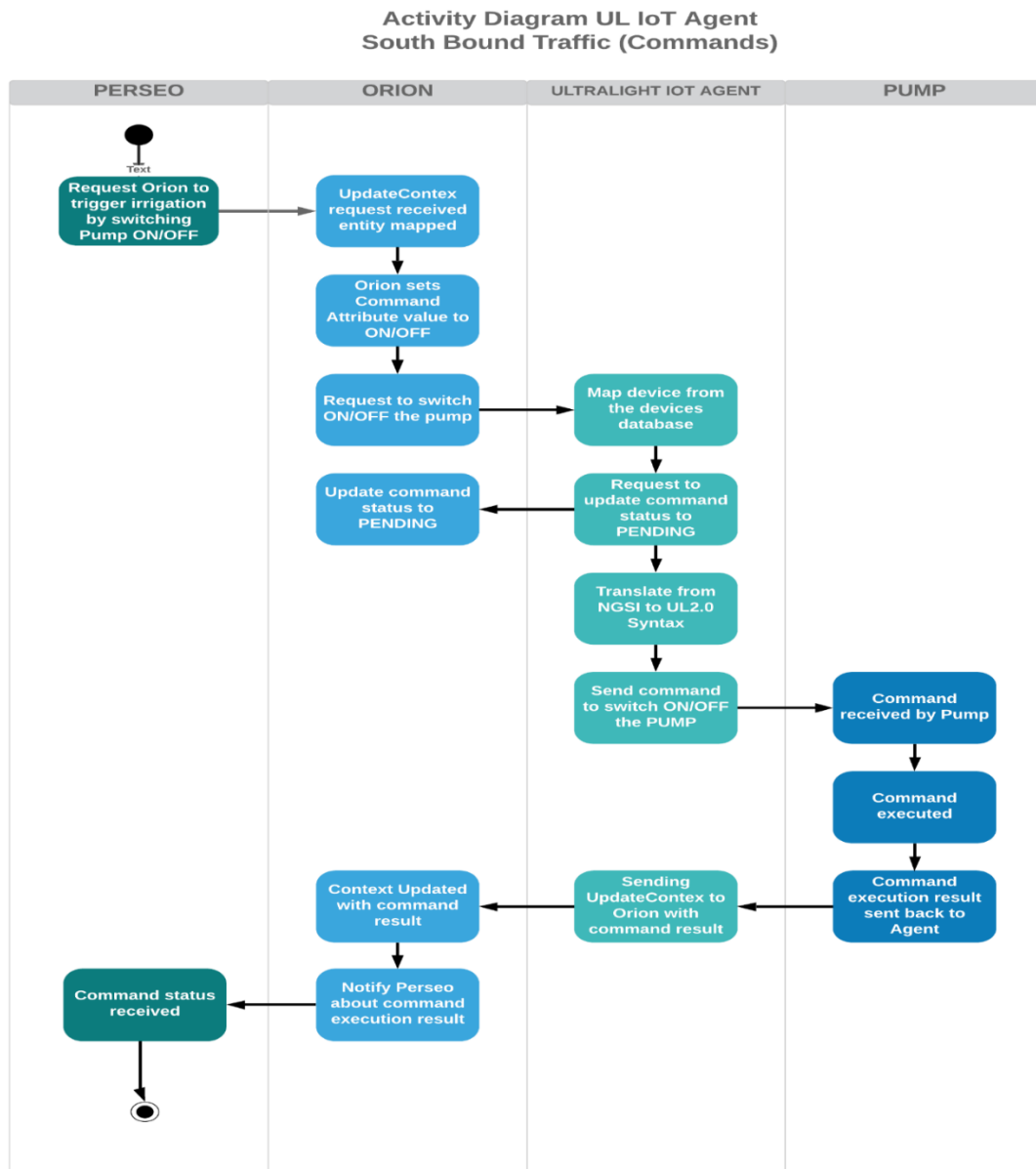


Figure 3. 8. Activity Diagram for the Ultralight IoT Agent - South Bound Traffic (Actuation).

The southbound traffic involves three components and an actuator interacting with each other (Perseo CEP, Orion CB, IoT Agent, and the pump) following these steps:

- 1) Perseo is a CEP GE and is responsible for initiating actions (i.e., switching on the pump) based on the triggering events (i.e., dry soil, very hot and sunny day.). Request to turn on the pump was sent to Orion through the exposed port and using HTTP PATCH request.
- 2) Orion receives the UpdateContext request from Perseo holding relevant information about the targeted entity and the command attribute's value on or off. Orion maps and checks the designated entity.
- 3) After mapping the entity and attribute, Orion Updates the attribute's value to on or off.
- 4) Orion forwards an HTTP request to the Agent on the corresponding port with the required information such as the command attribute value and the entity name and type.
- 5) The UltraLight IoT Agent receives the request and queries the provisioned devices and services database. When the entity id type, FIWARE-service, and service-path headers and attribute match, the Agent converts the message from NGSI format to the matching UL syntax.
- 6) UL Agent sends the command in UL syntax to the actuator node and updates the command status attribute value to "pending" and pushes it back to Orion.
- 7) The actuator node receives the command and executes it.
- 8) The actuator node notifies the agent about the result of execution, successful, not executed.
- 9) The Agent updates back Orion about the result of the command execution.
- 10) Orion notifies Perseo about the change for further actions.

Requests between	Format used
Perseo and Orion CB	NGSI
Orion CB and UL IoT Agent	NGSI
IoT Agent and IoT Device	UL2.0 syntax

3.4.2 North Bound Traffic (Measurement)

The North Bound Traffic represents the data traffic coming from sensor readings passing through LoRaWAN IoT Agent reaching Orion CB. IoT Agent is reaching Orion CB. The Activity Diagram of Fig 3.9. Describes the course of the data from the device until it reaches Orion.

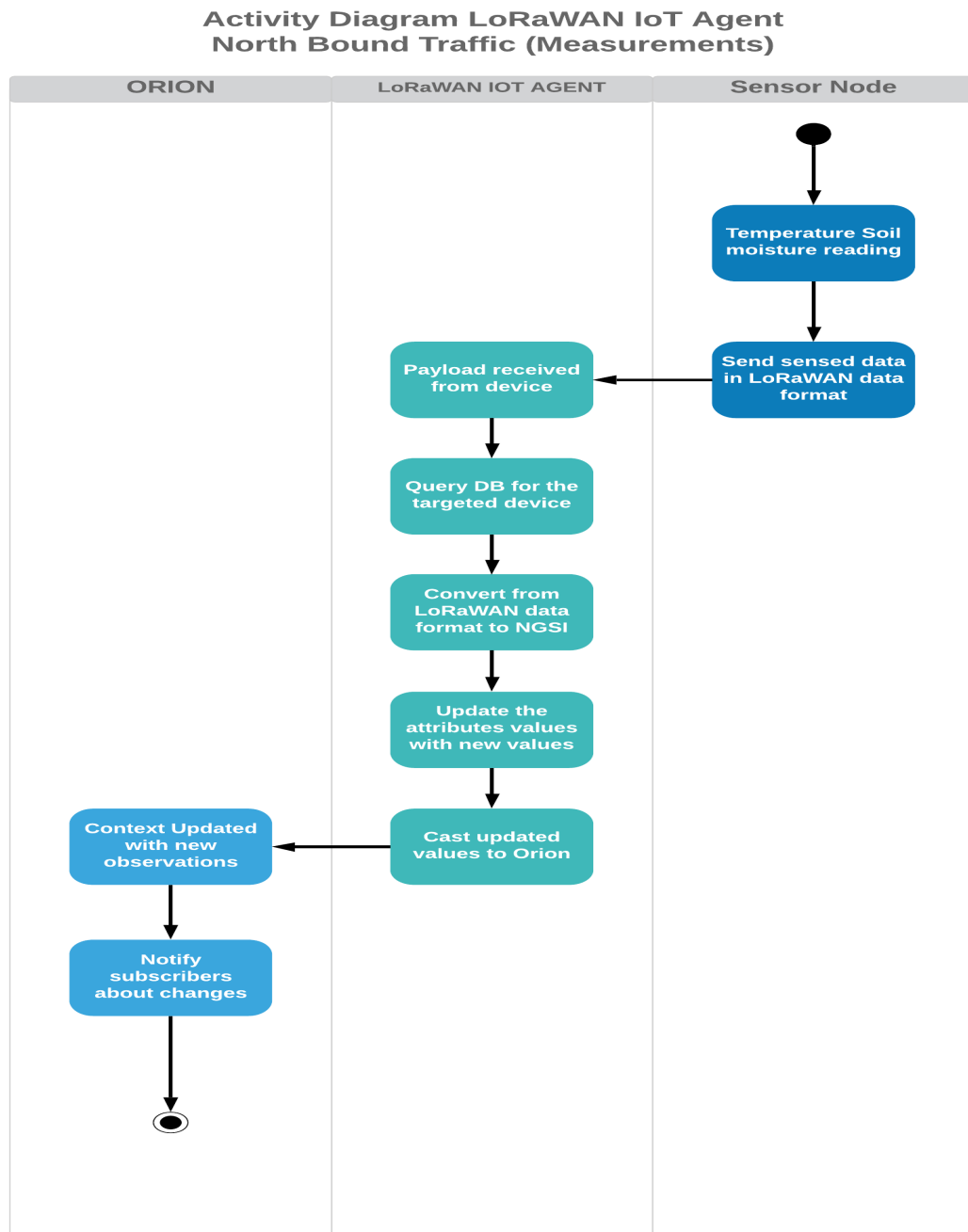


Figure 3. 9 Activity Diagram for the LoRaWAN IoT Agent - North Bound Traffic (Measurement).

The steps are explained as follows:

- 1) The device sensed the current value of temperature and soil moisture from the sensor, wrapped up in a packet and sent it straight to the gateway which, in turns, forwards it to the LoRaServer for decryption, filtering. The packet is then sent to the Agent.
- 2) Once the LoRaWAN IoT Agent, receives the packet, it queries the list of the provisioned devices in the database and searches for the targeted device.
- 3) After finding the targeted device and its entity, the Agent converts the data to be NGSI compliant and casts it into NGSI, it then maps the attribute and updates its value with the acquired readings from the sensor.
- 4) The last task is to send the payload to Orion through HTTP.
- 5) Orion acknowledges the reception of the payload and notifies all subscribed components about the changes of this specific entity's attribute's value.

Requests between	Format used
IoT Agent and IoT Device	LoRaWAN
IoT Agent and CB	NGSI

3.4.3 Device Provisioning

Device provisioning or device registration is the first step to be done for the device to be known and saved in the device collection inside the database. Device provisioning is done through posting a structured HTTP request containing the relevant information about the device such as the attribute types, whether commands or ordinary. Each device will be assigned to an entity with a unique ID and type under the FIWARE-service and service-path headers. All the provisioning requests are made to the IoT Agent (UL IoT Agent for actuators in the current case). Responsible for managing the devices. It is possible to run the CRUD operations for managing devices using HTTP requests sent directly to the corresponding IoT Agent using the correct headers, entity id, and type.

3.4.4 Provisioning a service group

The first thing to do is to provision a service group. Defining a service group is required for UL IoT Agent since it holds information that is crucial for the authentication, the end-point URL where Orion lies. A service group also integrates all devices sending the same type of measurements. For instance, assume there are many nodes deployed on field sensing temperature: all of the devices can be provisioned once in a service group instead of provisioning each device on its own. The method to provision a service group is explained as follows and the code is attached to the appendices:

This request represents a HTTP POST request using cURL for provisioning a service group of devices. The post request is sent to the IoT agent on the ['http://localhost:4041/iot/services'](http://localhost:4041/iot/services) end-point through port 4041 specific to the UL IoT Agent. Lines from three to five represent headers: `'Content-Type: application/json'` is required for post requests and the FIWARE-service and service path headers are specific to the current application. In order to declare services, an API-key is specified for the devices to include it in the authentication process while sending or receiving something. For instance, devices will use the following URL to make any post or get requests, including both the device ID, which is specified in the provisioning of the device step.

1. http://iot-agent:7896/iot/d?i=<device_id>&k=TemperatureNodeServiceGroup

cbroker represents the URL where Orion is reachable for receiving the request, the entity type is required, and its value is Thing by default used to map the correct entity. The resource `"/IoT/d"` means that this specific endpoint will be used to locate the provisioned devices under the specific service group.

3.4.5 Provisioning a sensor

The request used to provision a sensor is attached to the appendices and explained as the following:

It is a POST request sent to LoRaWAN IoT Agent through port 4061. In this case we are not provisioning a service group since we are only dealing with one sensor node. Device Id is mandatory as well as the entity linked to the device its type and ID. For

attributes we distinguish three types of attributes used to carry information and are the following:

- 1) Active Attributes: continuous readings from the device.
- 2) Lazy Attributes: sent only upon request by the Agent.
- 3) Static Attributes: represent static data about the device.

In the sensor node, only the active attribute type is used to acquire readings from the device. Each attribute is characterized by a name, type, and an object_id. Orion uses the name and type to identify the attribute in the corresponding entity while object_id is used to match the attribute ID of the payload in LoRaWAN with the attribute id of the registered device and acquire the exact value. The type must correspond to the supported data types described in FIWARE data models. Internal attributes are specific to using the LoRaWAN IoT Agent and define the authentication parameters required to access the LoRaServer (where devices are already registered and configured) such as the host address username and password as well as the Device EUI. Device EUI and application key are generated inside the LoRaServer when the device is added, both of them are unique to the device and are mandatory to decode the encoded packet coming from the sensor node. Data_model attribute specifies the data model used by the device to post new values; the value application server means the decoding is done by the LoRaServer, which is the app server.

3.4.6 Provisioning an actuator

Provisioning the actuator is not different from the provisioning of the sensor; in fact, the structure is kept the same. What changes is that we define an array “command.” Inside it lies the list of commands that are registered and can be triggered. Each of them has a name, a type, and a value. The endpoint for provisioning devices is “iot/devices” using the port 4041 specifically for UL IoT Agent and the transport protocol is set to MQTT. Provisioning single devices are done using the post request attached to the appendices:

3.5 Data Storage

MongoDB is an open-source, cross-platform database management system listed as one NoSQL database that supports many forms of data. MongoDB is document-oriented,

which means it uses collections and documents instead of tables and rows. It is widely used in applications that require big data processing and management (Rouse, 2018). MongoDB is integrated by many components to hold relevant data used during their activity. The database is queried and updated with new values by components. MongoDB interacts with other components through port 27017.

3.5.1 Identified Generic Enablers that use MongoDB

3.5.1.1 Orion

Orion Context Broker stores four collections in the database, described in the following subsections:

- (a) Entities Collection: this collection stores all the relevant data about entities in the NGSI format; each document in the collection corresponds to an entity.
- (b) Registrations Collection: holds information about established registrations of context providers such as the registered attributes, ID, and the URL of the providing application.
- (c) Subscriptions Collection: this collection holds established subscriptions to Orion about changes in attributes values.

3.5.1.2 Perseo

- (a) Rules: define the “how” to react to events are stored in Rules collection. These rules are defined by users and contain relevant information about the targeted device entity, the set of defined actions as a response to events.
- (b) Executions: results of command executions are stored in this collection with all the relevant information and details such as a timestamp of the execution and the related entity.
- (c) Indexes: indexes guarantee that every execution and rule is identified by its name, service, and sub-services.

3.5.1.3 QuantumLeap and CrateDB :

The collection holds information about the provisioned devices and service groups such as device URL, Apikeys, and device entity. QuantumLeap GE (“QuantumLeap—

QuantumLeap,” n.d.) is the component responsible for persisting data in the NGSII V2 data format inside a time-series database through a set of APIs. QuantumLeap integrates CrateDB (“CrateDB: Simply Scalable SQL Database for IoT & Industrial Time Series,” n.d.) for storing time-series data values from the readings for the following reasons:

- (1) Databases running in a containerized environment are easily scalable.
- (2) Supports Geospatial queries.
- (3) Implements a SQL-Like query language.
- (4) Integrates visualization tool like Grafana.

QL communicates with Orion CB using HTTP requests. QL is interested in acquiring the latest reading and keeps track of it. In order to do so, it needs to be aware of every change in the target attribute value and the time it occurred, it needs to be notified about the change and the corresponding value, that is where Orion kicks in. Knowing about the value change becomes much easier thanks to the subscription mechanism that Orion implements (that is the step one (1) in Fig. 3.10.). QL is going to subscribe to individual entity’s attributes (i.e., temperature change) and provides a URL to Orion for receiving notifications about value changes. Whenever the Agent receives any updates from the nodes about an entity’s attribute value, it immediately forwards it to Orion, that is step two (2). Orion, in turn, will check for any subscribers to the entity in interest and forwards a notification carrying the right information that QL needs. After receiving the notification, QL is going to run some operations or preprocessing on the received payload. It disposes of submodules that serve for this purpose. For instance, the reporter is going to parse and validate the notification supported by the Geocoder, which in turn will contribute by harmonizing the location representation of the different entities. The Reporter sends the preprocessed payload afterward to the Translator in charge of storing the data in the adapted time-series database cluster. When QL is asked to map an absolute value, the Translator assists in querying the database and forwarding the desired value. The user is able to query historical with the help of the existing APIs, step four (4).

Grafana gives the user the possibility to visualize and plot charts, create dashboards, manipulate the data according to the user need step five (5).

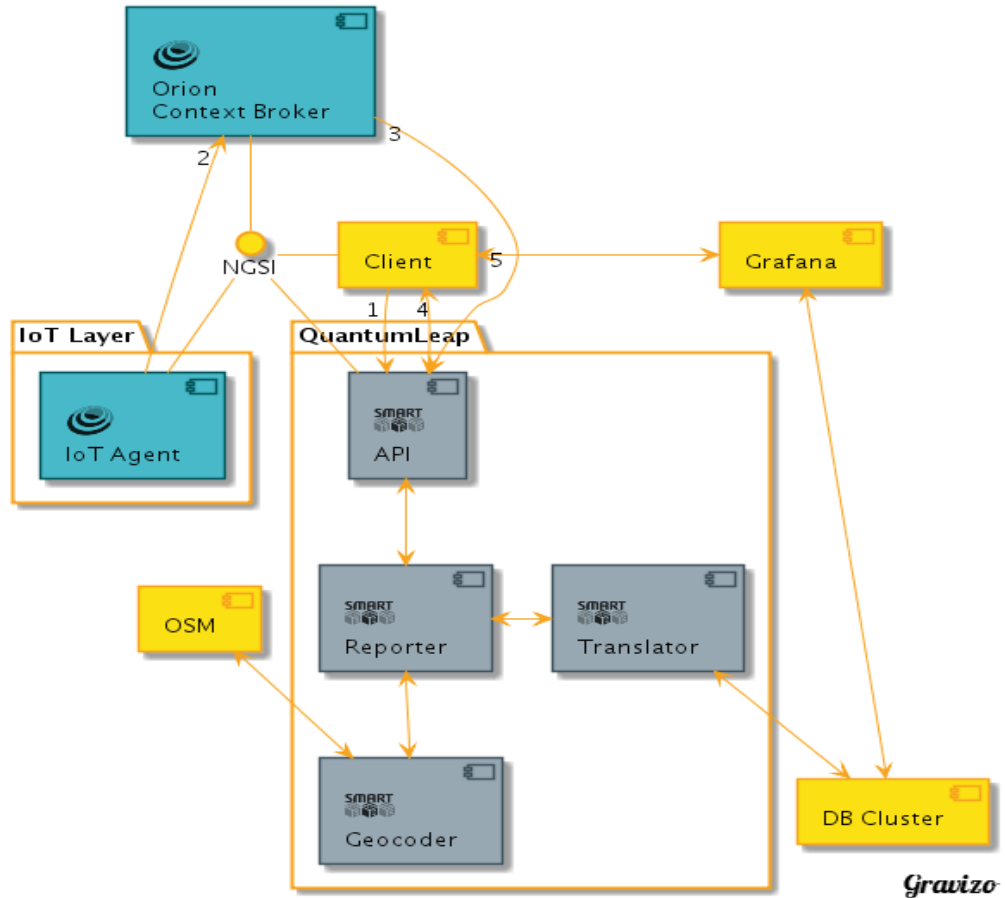


Figure 3. 10. Usage of QuantumLeap adapted from (“QuantumLeap—QuantumLeap,” n.d.)

3.6 Data Visualization: Grafana

Grafana is a powerful open-source visualization and analytics tool deployed in many domains such as home automation, industrial as well as weather. It is generally used to visualize and analyze time-series data (“Docs Home | Grafana Documentation,” n.d.). In order to start visualizing the persisted data in CrateDB Grafana requires to configure a data source, the place where data lies. Grafana is running locally on the virtual machine and is possible to access it from the browser using the following URL: <http://localhost:3000> and then selecting the Postgres Data source and filling the following information related to the database (see Fig 3.11):

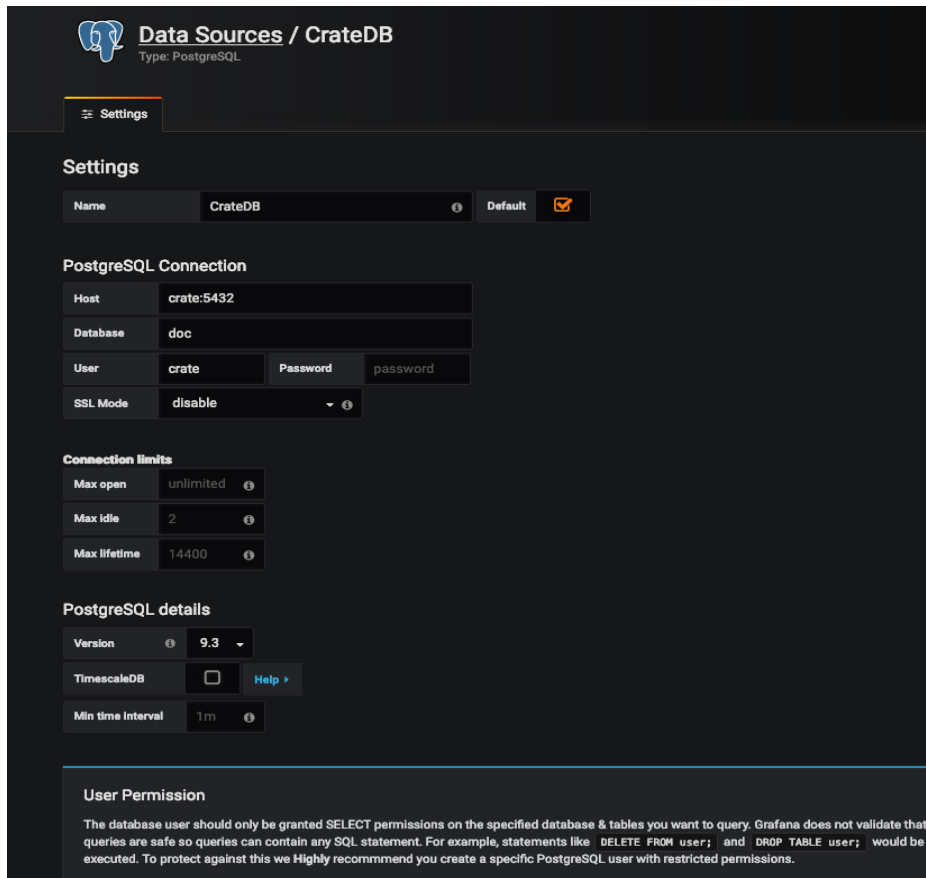


Figure 3. 11. Grafana data source setup tab.

- (1) Name: a name is required then tick default.
- (2) Host: the URL where CrateDB is located, usually crate:5432 when Grafana is deployed using the docker-compose file.
- (3) Database: the database schema names are made using mt prefix followed by the multi-tenancy header specified previously (fiware-service). For instance, mtagriculture is going to be the database schema.
- (4) The IoT agent is using the same schema to forward the IoT data under the agriculture service header.
- (5) User: takes the name and password of user-configured when installing Grafana.
- (6) SSL Mode: disable.

By clicking on the + sign of Fig 3.12. It is possible to create and personalize a new dashboard for visualizing the data.

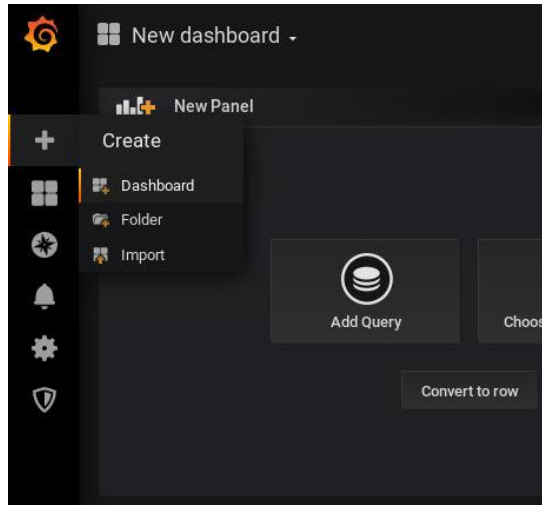


Figure 3. 12. Adding a new dashboard for data visualization in Grafana.

After setting the data source properly, it is time to start using available visualization widgets. Mapping the table and selecting the value of the temperature attribute on the Y-axis while the corresponding timestamp `time_index` on the X-axis all this using the query builder (see Fig. 3.13.). Note that the table names are derived from adding the “et” prefix followed by the entity’s type “tempHumid” in the current case.

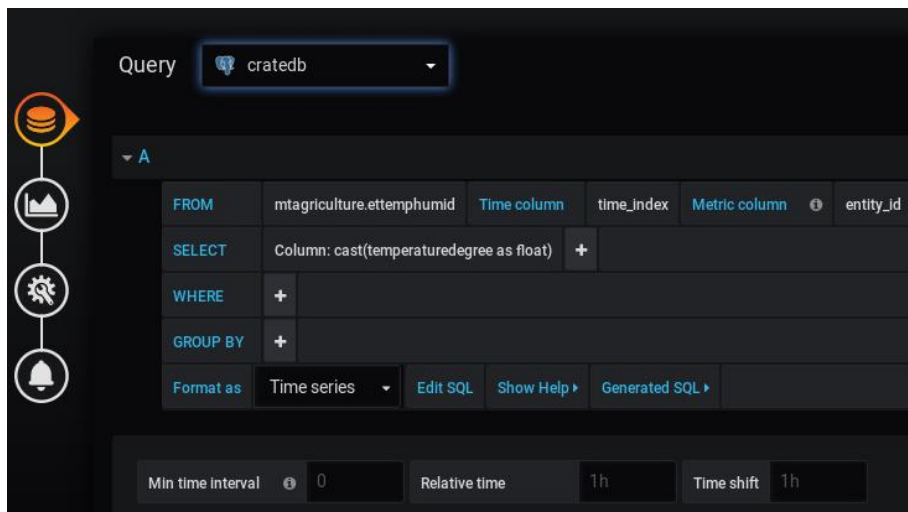


Figure 3. 13 Query builder of Grafana.

The following plot in Fig 3.14. Represents Temperature readings variations in function of the corresponding timestamps.



Figure 3. 14. Example of Grafana dashboard to visualize the temperature evolution on time.

3.7 Complex event processing: Perseo

Perseo (“Introduction—Perseo Context-aware Complex Event Processing,” n.d.) as a GE is designed to identify events patterns and react according to the rule’s logic. Perseo intercepts changes in the environment and matches them against the stored rules for further response by triggering actions if a match was detected. Perseo is built around different components each other is responsible for delivering a specific task; the following diagram (Fig. 3.15. Fig. 3.16.) presents the interactions between internal components of Perseo.

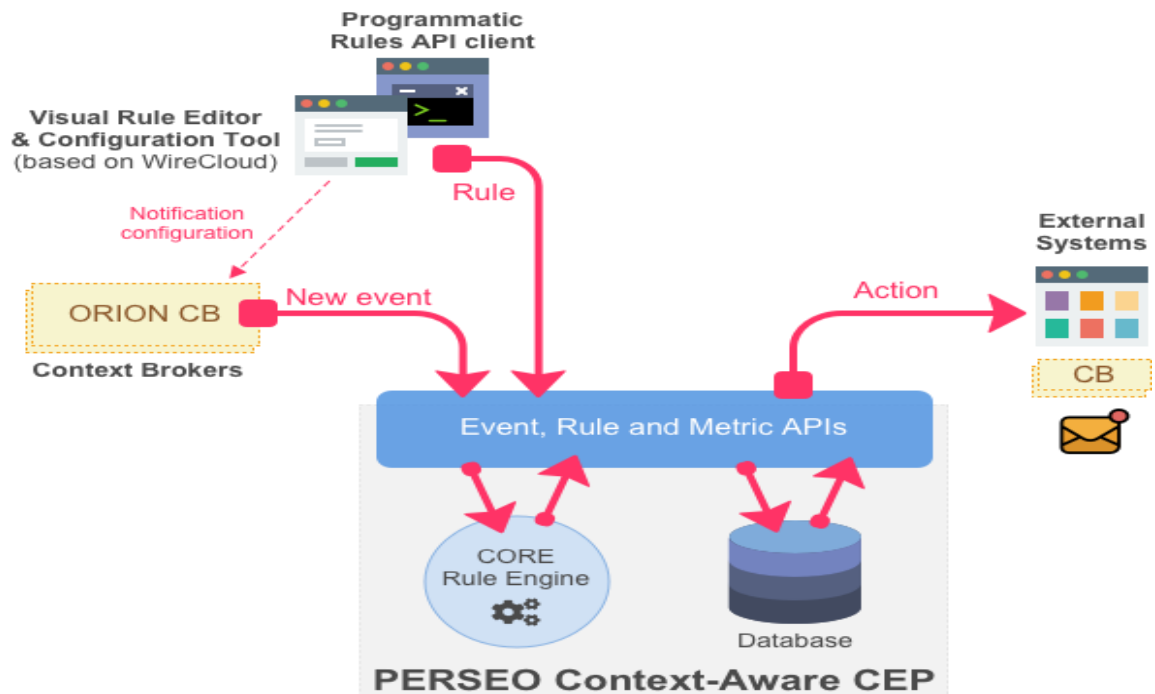


Figure 3. 15 Complex event processing using Perseo adapted from (“Introduction—Perseo Context-aware Complex Event Processing,” n.d.).

Perseo Linking other components

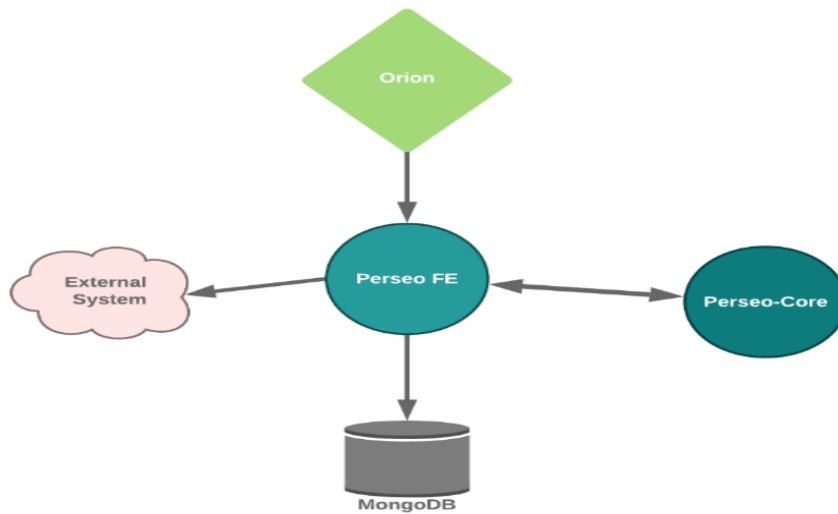


Figure 3. 16 Interaction between Perseo and other FIWARE components adapted from (“Introduction—Perseo Context-aware Complex Event Processing,” n.d.).

3.7.1 Perseo front-end

Perseo front-end or “Perseo-fe” has the main task to intercept and process incoming notifications from Orion about value changes (events) storing rules, triggering actions and storing their execution results. It also uses a check mechanism for entities that have been silent for a maximum time interval; this comes in handy when a node breaks and is no longer transmitting readings. Rules are refreshed periodically by the front-end.

3.7.2 Perseo core

This is the backend, rule-engine for Perseo, this component retrieves the set of stored rules as well as detected events from the front-end. Perseo-Core matches the incoming events against the rule in its disposition, in case there is a match, it invokes the front-end to take actions according to the rule’s defined actions.

3.7.3 MongoDB

MongoDB holds data about the rules and executions of actions.

3.7.4 Orion context broker

Through the notification mechanism, Orion is considered to be the source of events happening in the environment as well as the trigger for actions by updating command attributes of the action entity.

3.7.5 Portal

Represents the graphical user interface capable of creating a select type of rules called visual rules by using the FIWARE WireCloud GE (“Introduction—WireCloud,” n.d.).

3.7.6 Orion Database

For entities that have not communicated any values for a maximum time interval, Perseo queries Orion for the no-signal entity check that it runs periodically for silent entities. Perseo allows the definition of a no-signal rule as well to react for this specific kind of situation.

3.7.7 SMS gateway

Among the possible actions, Perseo can trigger there is sending an SMS to recipient number, and it is done through an HTTP post directly to the SMS Gateway (SMPP Adapter).

3.7.8 SMPP server

Among the possible actions, Perseo can trigger sending an SMS to recipient number, and it is done through an HTTP post directly to the SMS SMPP server. The corresponding SMPP server is configured with the other environment variables in Perseo.

3.7.9 SMTP server

It is possible to react to events by sending an email (i.e., notify me by email if no temperature measurements reached in the last one hour). This is done by configuring the SMTP with other environment variables in Perseo.

3.7.10 Generic HTTP server

Perseo can interact with other external components through HTTP requests providing the endpoint URL as parameter Perseo can send posts request.

3.7.11 Authorization server

All interactions of Perseo with Orion require an access token refreshed periodically. In order to execute a rule that updates an entity in Orion, a trust token is associated with the executed rule and needs to be exchanged by an access token at the Authorization Server.

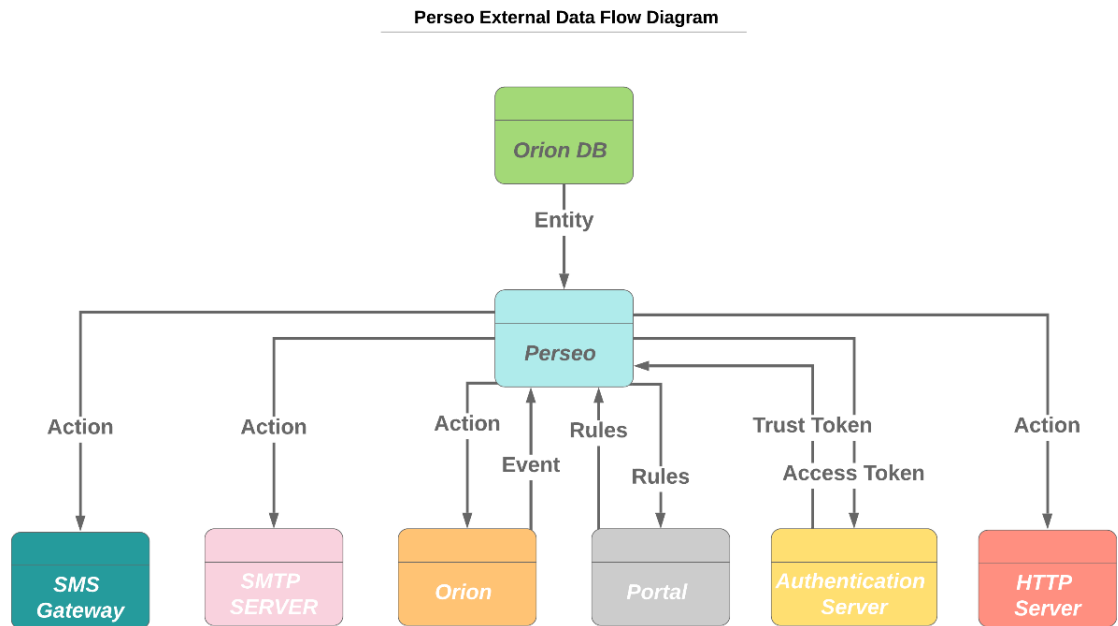


Figure 3. 17. External Data flow Diagram in Perseo

3.7.12 Rule setting

Rules are meant to define how things should go when an event occurs; they are aware of events and comes as a response to them by triggering actions. Rules are defined by the user to match certain identified patterns and react accordingly to them. Different types of rules exist in Perseo, but the most exploited ones are plain rules.

Plain rules are represented in a simplified JSON format. A request to add a rule is structured this way and is attached to appendices:

Perseo is going to be listening for incoming rule provisioning requests on port 9090. The post request is sent to this URL '<http://perseo-fe:9090/rules>' with "/rules" as an endpoint. The same multi-tenancy headers FIWARE-service and service path are used to declare rules; this way, it will be easier to identify which rule belongs to which application service and service path in an organized and accurate manner. A rule holds a specific structure and is combined with the following required attributes:

- 1) Name: the name of the rule consisting of only ASCII characters, digits, underscore, and dash with a maximum length of fifty char.
- 2) Text: This field is where the user defines the conditions to be met. When this condition fulfills the action is triggered afterward. It should contain valid EPL (Event Processing Language) statements. EPL is a domain language of Esper ("Esper," n.d.). It is used by Perseo-core for processing and matching events against the rules for any matches and generates actions to be taken accordingly. It uses an SQL-like language.
- 3) Action: the set of actions to be executed by Perseo-fe after triggering the rule.

The example beforementioned runs the condition specified in the text field, and in case of a match it triggers an update action to an entity's command attribute (for example to set the value to ON). Inline ten values of different sensor readings are retrieved, and a condition, For instance, if the moisture level is under five 5% and the predicted rainfall for tomorrow is 10 mm, then tell the actuator node to trigger irrigation. The condition variable carries the value of the predicted rainfall from external weather forecast data sources. Line 30 represents the minimum time interval attribute between two action executions expressed in milliseconds.

3.7.13 Action Request Structure

Perseo uses different attributes for building actions, and the structures differ from one kind of action to another. The sequence diagram of Fig 3.18 explains the interactions between Perseo, MongoDB, and the SMS gateway in the process of triggering the action.

SEQUENCE DIAGRAM PERSEO TRIGGERING ACTION

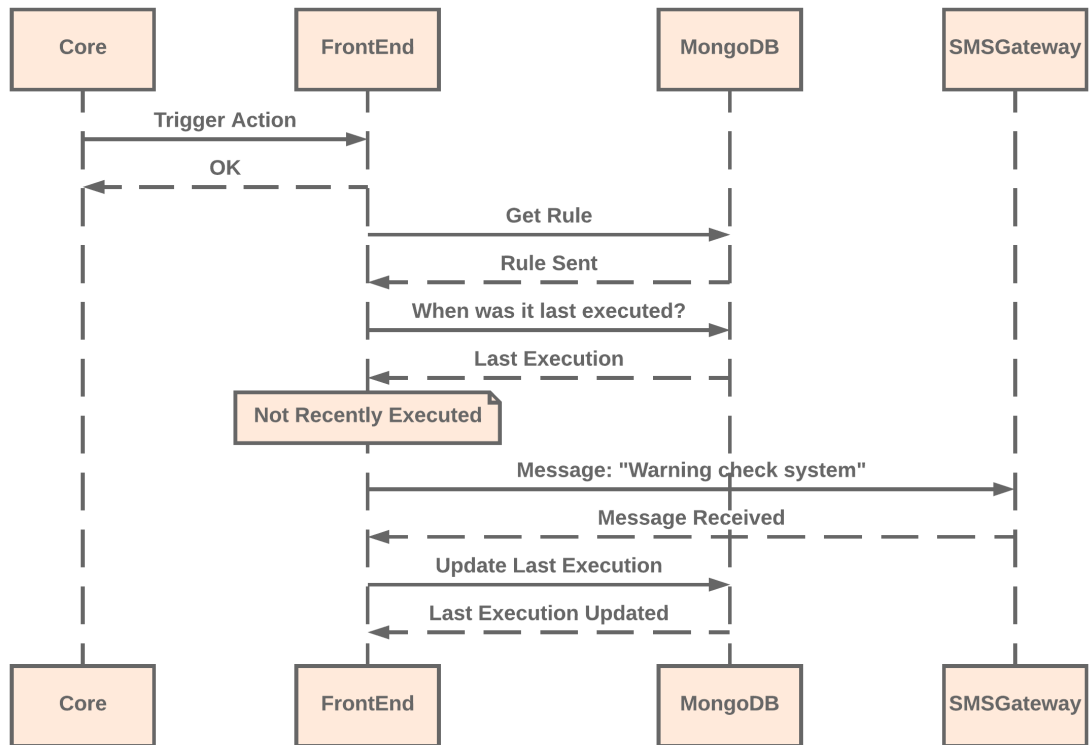


Figure 3. 18. Interactions between Perseo, MongoDB, and the SMS gateway in the process of triggering the action.

3.7.14 SMS action

The action attribute to send an SMS is attached to the appendix:

Line 3 represents the body of the SMS. It is possible to precise the soil moisture variable value that triggered the rule directly by putting it inside `{soilMoisture}`: it is known as string substitution. The phone number of the receiver is highlighted in line five as the value for the “to” attribute. In case of many receivers, the numbers are separated using whitespace character.

3.7.15 Email action

The triggered action will send an email to one or more recipients. The body of the mail is built in the “template” attribute and attached to the appendix as well as the address of

the sender and receivers to be added in the “from” and “to” attributes respectively. It is possible to perform string substitution for the “template,” “from,” “to,” and “subject.”

3.7.16 HTTP request action

Among the available actions, figures sending a HTTP request to any external entity exposing its port using the corresponding URL the body of the request is attached to the appendix. The following parameters are crucial:

- method: POST by default.
- URL: target URL mandatory.
- headers: FIWARE-service and service-path headers.
- JSON: the body of the request that will be sent as JSON.

3.8 External sources of information:

3.8.1 OpenWeatherMap API

OpenweatherMap (“OpenWeatherMap About company,” n.d.) is a company which is active in the domain of IT, created in 2014 by a group of engineers and experts in Big Data, data processing, and satellite imagery processing. They provide a set of fast, easy to use API with databases of weather data available for use, as well as the satellite imagery and other environmental data. It provides current weather data as well as forecasts tailored to fit every need (for instance, they provide forecasts for 16 days as well as for five days every three hours).

3.8.2 Agro API

Agro API provides crucial data used for agriculture for the desired location, ranging from NDVI (normalized difference vegetation index) to soil moisture, to Ultraviolet index (UVI).

3.8.3 NGSI Parser Module

The NGSI Parser (CENIDET, 2017/2019) library for JavaScript is a piece of software designed to convert JSON entities into NGSI data model to be used by Orion. In order to adapt the data format acquired from the weather forecast provider, which is formatted

in a JSON file format and needs to be formatted according to NGSI format, the Parser role comes to do the necessary transformations. Fig. 3.19 explains the process to convert into NGSI and feed Orion with the data:

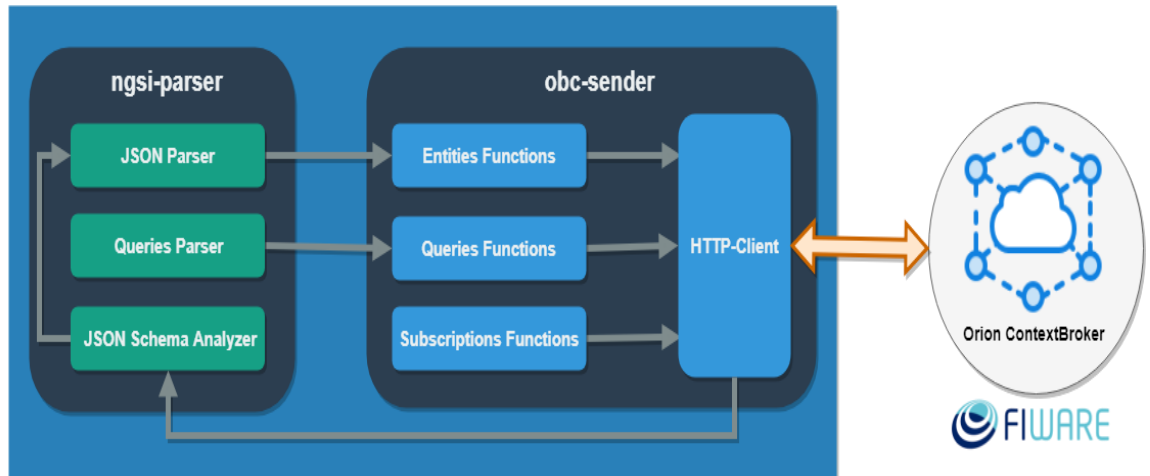


Figure 3. 19. NGSI parser module working principle adapted from (CENIDET, 2017/2019)

Ocb-sender oversees communicating the parsed entities to Orion through HTTP request and many other functionalities like querying, creating entities and subscriptions. The NGSI-parser uses a JSON schema Analyzer to tell whether the data structure follows a reference JSON schema. In the case of interest to get the weather forecast data, it will be necessary to send an HTTP request using a GET method to the corresponding server in order to be able to retrieve the data sent by the server in a JSON file format.

After parsing the JSON, it will be possible to feed Orion directly with the updated data as well as sending it to another receiver or vice versa, which can be a mobile phone application, for instance.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, the whole system architecture is recalled, and the operating performances of the complete prototype are detailed.

4.2 Smart agriculture use-case implementation

In order to prove the effectiveness of the FIWARE platform in managing real use cases, a smart agriculture proof-of-concept has been set up. Although many different pilots for smart agriculture application have already been put in place, the unique selling point of this approach is that all the system building blocks concur to update the FIWARE orchestrator, i.e., the Orion context broker (green cube in Fig. 4.1). This is the core unit of FIWARE and all the other components of the system build, in a way, the context where the application runs into.

4.3 Application Mock-Up and Prototype

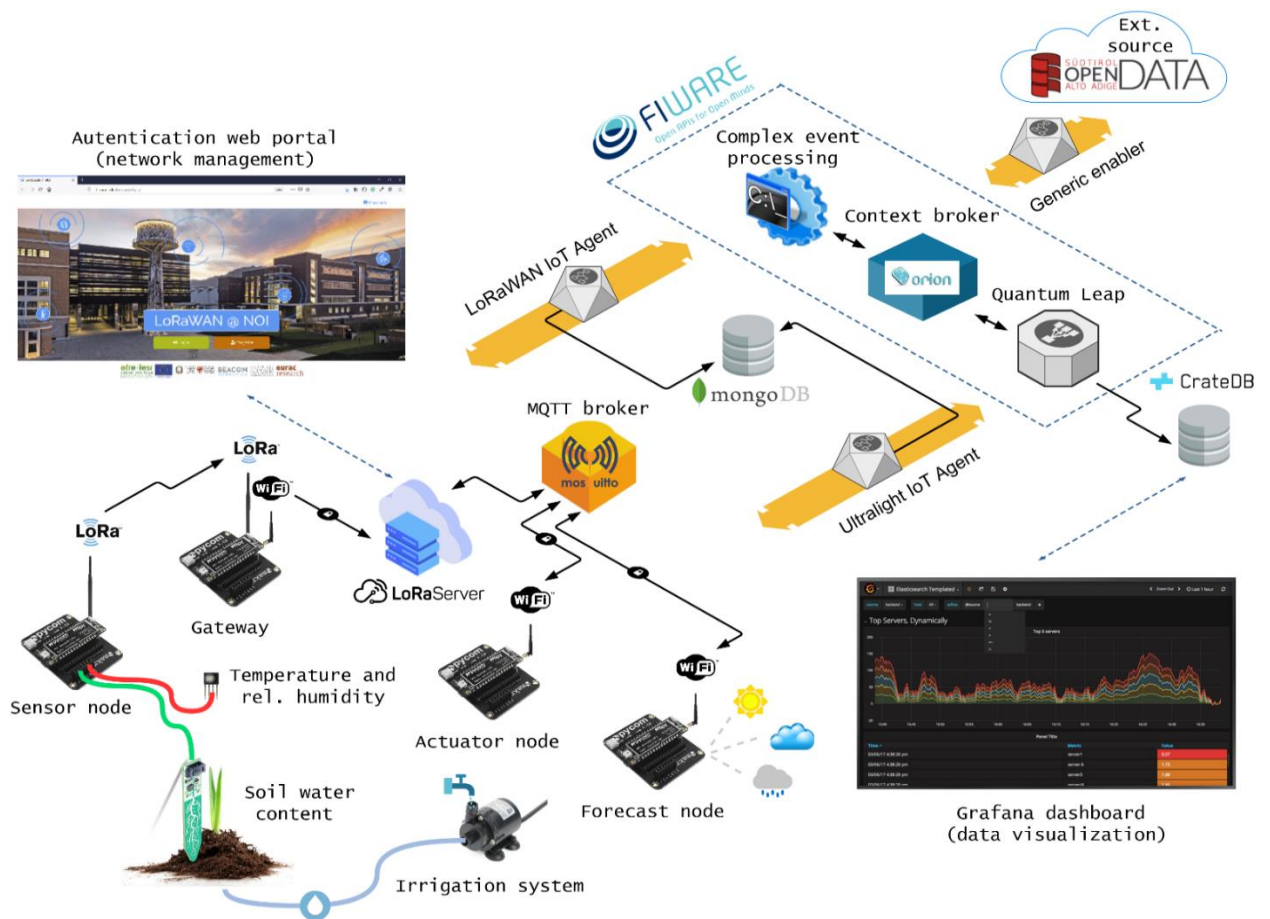


Figure 4. 1 Schematics of the complete system put in place.

Fig. 4.1 represent the mock-up of the complete systems. It encompasses the hardware part, down the left section of the figure, the back-end part, in the center of the schematics, and a couple of web interfaces, in the top left and the bottom right of the figure. The external source of information is represented by the Open Data portal of the province of Bolzano (“OpenData Portal Bolzano,” n.d.) which has been used to integrate remote data into the irrigation system. For a more detailed description of the single hardware and software components see Component Setup in Chapter 3.

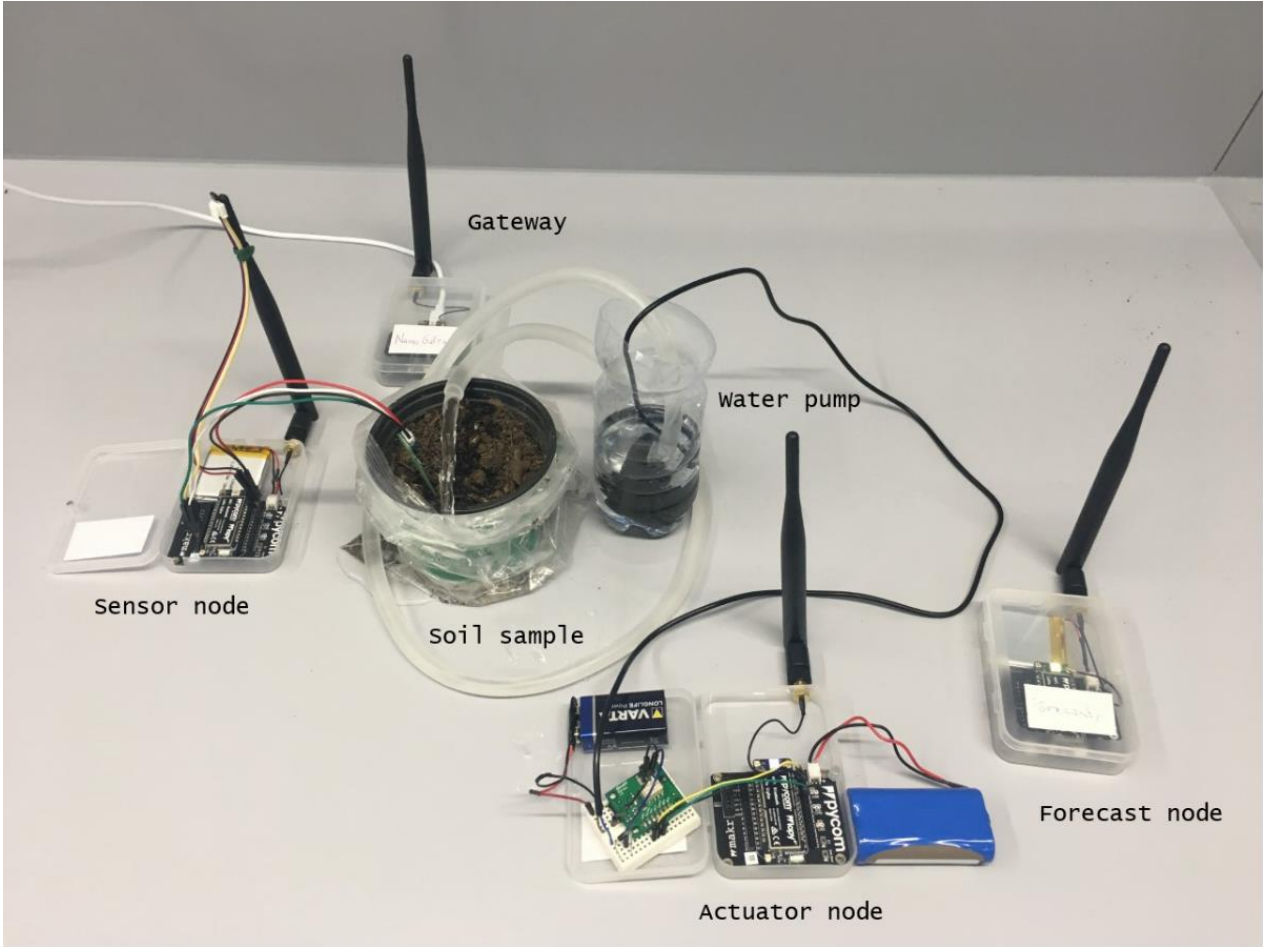


Figure 4. 2 Hardware components of the system prototype.

Fig. 4.2 shows the system prototype during operation in the lab. The main hardware building blocks are four microcontrollers, which serve as sensor node, LoRaWAN gateway, actuator node, and a node that simulate the forecast change. The system is completed by a capacitive soil water content sensor connected to the sensor node and a water pump connected to the actuator node.

The picture shows the system irrigating the soil sample, as a consequence of the current context condition: soil water content acquired & weather forecasts for the next 6 hours. The water pump is triggered when the soil water content in the soil pot is less than 20% under sunny condition. Instead, the drought threshold is lowered to 10% when the weather forecasts predict a rain event within 6 hours. The forecast node pretends to simulate sudden weather changes when a push-button is pressed. Such action produces an update into the Orion context broker, which, in turn, provides this information to all the components that have subscribed to the forecast topic. In particular, this notification reached the Perseo complex event processor, which fires the irrigation action according to the beforementioned rule. This way, we are able to test the prototype in all the possible working conditions.

As can be seen from the picture, only the LoRaWAN gateway is powered by USB, while the other three microcontrollers of the system are powered by LiPo batteries and can be distributed in the field with ease. The reason why the gateway needs to be powered in a different way is that this element of the network is continuously awake, waiting for new LoRa packets from the sensor node. On the one hand, this limits the deployment options for such a system. By the other hand, this attribute allows for collecting data from sensors that are within 10 km from the gateway (in-sight distance).

4.3.1 System operating principle

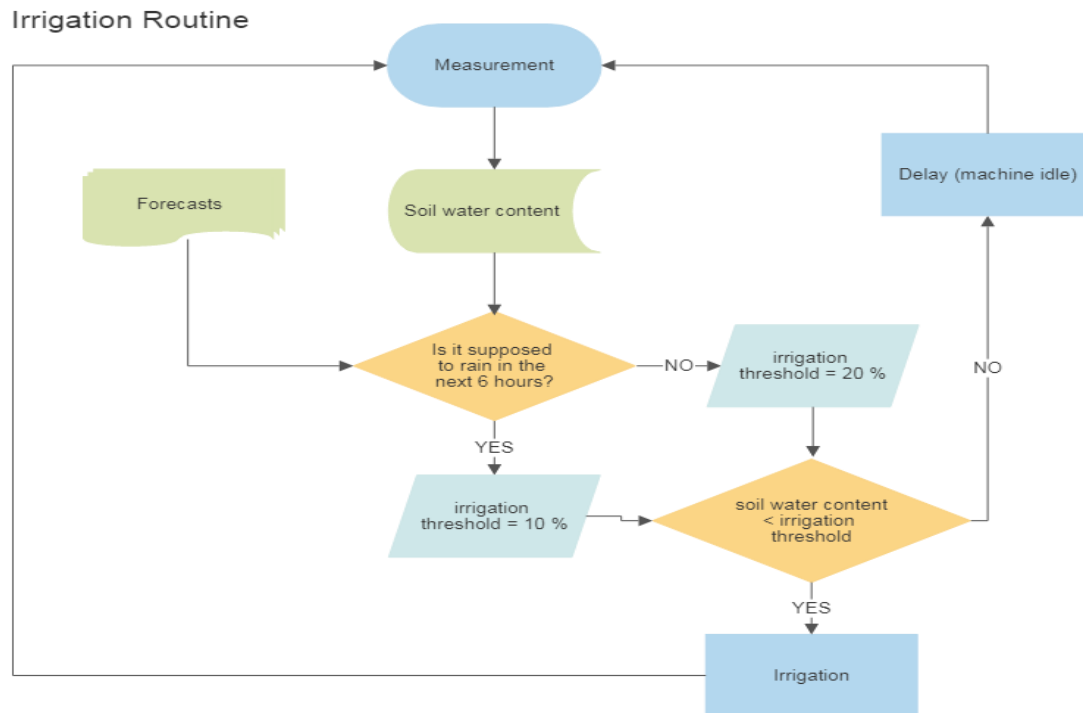


Figure 4. 3 Flowchart of the implemented irrigation routine.

Even though FIWARE relies on subscriptions and updates to build actions, as it has been described in chapter 3, a simplified flowchart of the prototype working principle is here reported. The aim of the following schematics is to highlight the role that the external sources of information play during the operation. As can be seen from Fig. 4.3, after each measurement, the acquired value is stored in the soil water content variable. Then the system chooses which rule to apply, depending on the last available forecast. Once the threshold value has been set to 20% or 10%, it is compared with the measured soil water content. If the condition is met (i.e., soil water content < threshold value), the water pump is triggered. Otherwise, the system goes into sleep mode (machine idle) until the next scheduled measurement. Special attention must be paid to the fact that the flowchart is not meant to represent an algorithm where subsequent processes/tasks are executed step by step. More appropriately, each update from the forecast node or the sensor node triggers the flow of the decision process. Despite the simplicity of the implemented application, this proof of concept demonstrates that FIWARE is suitable for implementing the most advanced approaches of the IoT paradigm. The transition from data management to context

information management allows for easier and more efficient implementation of crucial tasks like autonomous tuning and predictive maintenance, among others. Preliminary tests have shown that centralizing the decision kernel into a single kernel (Orion context broker) comes with several benefits. Firstly, the latency of the system is lowered by more than one order of magnitude (from seconds to sub-seconds) with respect to similar applications which exploit traditional data, management models. Moreover, the scalability and the integration of other sources of information is facilitated as many connectors (agents) are already available on the FIWARE website, and other target-specific connectors (generic enablers) can be developed through a dedicated syntax. Lastly, the computational power of the peripheral building block of the system can be entirely dedicated to internal tasks, for example, averaging the sensor read-out to enhance the measurement accuracy (at the sensor node) or run predictive models upon environmental data (in the forecast node) (Herrera, Torgo, Izquierdo, & Pérez-García, 2010) Of course, the complexity of the system makes the use of such an approach not well suited for small size pilots. The added value of using FIWARE becomes evident as the number of the system components increases, i.e., the context becomes broader and more complex. In this case, the application can be upgraded or scaled up without the need to interrupt the current system operation.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion and discussion

The objective of this work was to develop a system architecture design based on FIWARE platform by identifying and selecting the most adherent components for a smart irrigation application as well as selecting the appropriate and most suitable hardware and most widespread IoT communication protocols fitting into this specific application. The work has been completed by setting up a prototype to showcase and validate the proper functioning of the proposed architecture consisting of the different components. The added value of using the FIWARE platform comes with the ability to integrate multiple external context information from third-party data providers, that, in turn, increases awareness about context which is considered crucial for novel smart applications. It is necessary to capture all relevant events related to the specific application (weather forecasts, for instance, in the case of smart agriculture) and to be able to react in an appropriate manner. The complex event processing (CEP) unit of FIWARE allows for doing so. FIWARE comes with many complex components or Generic Enablers architectures which can be adopted and reused in an agile way, due to the fact that FIWARE is not linked to a specific application or function. It could adapt to many ranges of applications by tailoring components to adapt to the application's need. Handling many components can be seen as tedious, especially for a small-scale application. Though the complexity of FIWARE, the subscription and registration mechanisms used by the Context Broker to report events adapts significantly to the smart application requirements, saving a considerable amount of time and resources compared to the traditional ways (report/react). However, the only issue faced was lack of support and the absence of documentation on many details regarding the implementation and use of components which have made solving a simple problem a significant challenge.

The implemented prototype was able to

- record sensor measurements in a reliable and timely manner;
- forward them to the cloud instantly for further processing, visualization, and storage;
- acquire context information from external sources to deduce and predict events;
- allow the system to react to events immediately after they occur by triggering actions defined by user-set rules.

The prototype was designed and deployed for a smart agriculture application to trigger irrigation process based on the sensor readings as well as external weather forecast data. The use of this prototype in this specific application does not necessarily mean it cannot be deployed in other applications depending on the expressed need. For instance, the same prototype could also be customized and deployed in case of monitoring the temperature of PV panels or measuring real-time output current and voltage to analyze the collected data and assess the performance of the system. Furthermore, it will still have the chance to trigger actions according to the situation. Future perspective of this work would be to integrate GIS (geographic information system) data and extract relevant information such as thermal satellite images in order to perform and ensure uniform irrigation to the whole field (Roopaei, Rad, & Choo, 2017). It would also be interesting to establish a connection with a MAS (Multi-Agent System) in order to extract knowledge from the persisted data about the different environmental components (i.e., knowledge about soil) by a big data analysis.

5.2 Recommendations: pros and cons of FIWARE

After familiarizing and acquiring deeper insights about FIWARE and its Generic Enablers, it is safe to say that there is much more to discover and experience about the platform and the possible integrations and developments to smart solutions. Tab 5.2 represents the pros and cons based on the author’s know-how with FIWARE. *Table 5. 1 Pros and Cons of FIWARE platform*

Pros of FIWARE	Cons of FIWARE
Flexible development and easy deployment over the cloud network.	Complex architecture linking many components together, making it a tedious task to configure and debug.
Secure, supports big data analysis tools, complex event processing, and efficient context information management.	Documentation lacking details, relatively hard to understand outdated examples, components with small or no tutorials.
Allows the development of powerful Apps and data fed in real-time.	A small community, lack of technical support.
Supplies different sources of context information in a straightforward manner	Would fit perfectly for a big scale implementation, not very interesting for small applications.
Exploits Open API standards.	
Easily scalable and upgradeable.	

Bibliography

- Developers Catalogue—FIWARE. (n.d.). Retrieved August 25, 2019, from <https://www.fiware.org/developers/catalogue/>
- Difference between Information and Data. (n.d.). Retrieved August 25, 2019, from <https://www.guru99.com/difference-information-data.html>
- What is a Database Management System (DBMS)? - Definition from Techopedia. (n.d.). Retrieved August 25, 2019, from <https://www.techopedia.com/definition/24361/database-management-systems-dbms>
- Rouse, M. (n.d.). What is a Database Management System? - Definition from WhatIs.com. Retrieved August 25, 2019, from What is a Database Management System website: <https://searchsqlserver.techtarget.com/definition/database-management-system>
- Time Series DBMS - DB-Engines Encyclopedia. (n.d.). Retrieved August 25, 2019, from Time Series DBMS website: <https://db-engines.com/en/article/Time+Series+DBMS>
- Context | Definition of context by Lexico. (n.d.). Retrieved August 25, 2019, from <https://www.lexico.com/en/definition/context>
- Coutaz, J., Crowley, J. L., Dobson, S., & Garlan, D. (2005). Context is key. *Communications of the ACM*, 48(3), 49. <https://doi.org/10.1145/1047671.1047703>
- Kunzler, F., Kramer, J.-N., & Kowatsch, T. (2017). Efficacy of mobile context-aware notification management systems: A systematic literature review and meta-analysis. 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking, and Communications (WiMob), 131–138. <https://doi.org/10.1109/WiMOB.2017.8115839>
- González-Briones, A., Castellanos-Garzón, J. A., Mezquita Martín, Y., Prieto, J., & Corchado, J. M. (2018). A Framework for Knowledge Discovery from Wireless Sensor Networks in Rural Environments: A Crop Irrigation Systems Case Study. *Wireless Communications and Mobile Computing*, 2018, 1–14. <https://doi.org/10.1155/2018/6089280>
- Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., & Reinfurt, L. (2016). Comparison of IoT platform architectures: A field study based on a reference architecture. 2016 Cloudification of the Internet of Things (CIoT), 1-6.
- Salhofer, P. (2018). *A Case-Study on Implementing Smart Application with FIWARE*. 9. <https://doi.org/10.24251/HICSS.2018.726>
- Martínez, R., Pastor, J., Álvarez, B., & Iborra, A. (2016). A Testbed to Evaluate the FIWARE-Based IoT Platform in the Domain of Precision Agriculture. *Sensors*, 16(11), 1979. <https://doi.org/10.3390/s16111979>
- Celesti, A., Fazio, M., Galán Márquez, F., Glikson, A., Mauwa, H., Bagula, A., ... Villari, M. (2019). How to Develop IoT Cloud e-Health Systems Based on FIWARE: A Lesson Learnt. *Journal of Sensor and Actuator Networks*, 8(1), 7. <https://doi.org/10.3390/jsan8010007>
- Context Management Architecture—FIWARE Forge Wiki. (2016). Retrieved August 25, 2019, from Context Management Architecture website: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Data/Context_Management_Architecture
- Entity service paths—Fiware-Orion. (n.d.). Retrieved August 25, 2019, from Entity service paths website: https://fiware-orion.readthedocs.io/en/master/user/service_path/index.html
- Enterprise Container Platform | Docker. (n.d.). Retrieved August 25, 2019, from <https://www.docker.com/>

“Definitive Guide to Enterprise Container Platforms”, June 2019, <https://www.docker.com/resources/white-paper/the-definitive-guide-to-container-platforms>

Docker Hub. (n.d.). Retrieved August 25, 2019, from <https://hub.docker.com/>

Jawad, H., Nordin, R., Gharghan, S., Jawad, A., & Ismail, M. (2017). Energy-Efficient Wireless Sensor Networks for Precision Agriculture: A Review. *Sensors*, 17(8), 1781. <https://doi.org/10.3390/s17081781>

LoRaWAN. (2019, August 23). Retrieved August 25, 2019, from The Things Network website: <https://www.thethingsnetwork.org/docs/lorawan/>

LoRa Alliance, LoRaWAN What is it?, November, 2015, A Technical Overview of LoRa and LoRaWAN, *technical marketing workgroup 1.0*, retrieved from <https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf>

LoRa Alliance Technical Committee, LoRaWAN™ 1.0.3 Specification, July 2018, retrieved from <https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf>

MQTT main page. (2018, September 20). Retrieved August 25, 2019, from Eclipse Mosquitto website: <https://mosquitto.org/man/mqtt-7.html>

FIWARE 203: IoT over MQTT. *Contribute to FIWARE/tutorials.IoT-over-MQTT development by creating an account on GitHub* [Shell]. (2019). Retrieved from <https://github.com/FIWARE/tutorials.IoT-over-MQTT> (Original work published 2018)

Pycom, LoPy4 Datasheet, retrieved from https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_LoPy4_v2.pdf

Industries, A. (n.d.). Adafruit STEMMA Soil Sensor—I2C Capacitive Moisture Sensor. Retrieved August 25, 2019, from <https://www.adafruit.com/product/4026>

Immersible/submersible Water Pump (6V~12V)—DFRobot. (n.d.). Retrieved August 25, 2019, from <https://www.dfrobot.com/product-1710.html?search=water%20pump&description=true>

LoRaWAN Nano-Gateway. (n.d.). Retrieved August 25, 2019, from LoRaWAN Nano-Gateway website: <https://docs.pycom.io/tutorials/lora/lorawan-nano-gateway/>

Deep Sleep API. (n.d.). Retrieved August 25, 2019, from Deep Sleep API website: <https://docs.pycom.io/datasheets/boards/deepsleep/api/>

Grimm, C. (Ed.). (2013). *Embedded systems for smart appliances and energy management*. New York: Springer.

Description- LoRaWAN IoT Agent. (n.d.). Retrieved August 25, 2019, from ARCHITECTURE website: <https://fiware-lorawan.readthedocs.io/en/latest/index.html#description>

Project—LoRa Server, open-source LoRaWAN network-server. (n.d.). Retrieved August 25, 2019, from <https://www.loraserver.io/overview/>

Architecture—LoRa Server, open-source LoRaWAN network-server. (n.d.). Retrieved August 25, 2019, from <https://www.loraserver.io/overview/architecture/>

Home—Fiware-Orion. (n.d.). Retrieved August 25, 2019, from <https://fiware-orion.readthedocs.io/en/latest/index.html>

WGS 84: EPSG Projection—Spatial Reference. (2007). Retrieved August 25, 2019, from epsg projection 4326—Wgs 84 website: <https://spatialreference.org/ref/epsg/wgs-84/>

Rouse, M. (2018, August). What is MongoDB? - Definition from WhatIs.com. Retrieved August 25, 2019, from What is MongoDB? website: <https://searchdatamanagement.techtarget.com/definition/MongoDB>

Docs Home | Grafana Documentation. (n.d.). Retrieved August 25, 2019, from <https://grafana.com/docs/v4.3/>

Introduction—Perseo Context-aware Complex Event Processing. (n.d.). Retrieved August 25, 2019, from PERSEO CONTEXT-AWARE CEP website: <https://perseo.readthedocs.io/en/latest/#perseo-context-aware-cep>

About company. (n.d.). Retrieved August 25, 2019, from <https://openweather.co.uk/about>

CENIDET. (2019). *ngsi-parser is an npm module for parsing and converting a simple unstructured JSON or value to an NSGI-compliant object: Cenidetiot/ngsi-parser* [JavaScript]. Retrieved from <https://github.com/cenidetiot/ngsi-parser> (Original work published 2017)

OpenData Portal Bolzano. (n.d.). Retrieved August 25, 2019, from <http://daten.buergernetz.bz.it/>

Herrera, M., Torgo, L., Izquierdo, J., & Pérez-García, R. (2010). Predictive models for forecasting hourly urban water demand. *Journal of Hydrology*, 387(1-2), 141–150. doi:10.1016/j.jhydrol.2010.04.005

Roopaei, M., Rad, P., & Choo, K.-K. R. (2017). Cloud of Things in Smart Agriculture: Intelligent Irrigation Monitoring by Thermal Imaging. *IEEE Cloud Computing*, 4(1), 10–15. <https://doi.org/10.1109/MCC.2017.5>

López-Riquelme, J. A., Pavón-Pulido, N., Navarro-Hellín, H., Soto-Valles, F., & Torres-Sánchez, R. (2017). A software architecture based on FIWARE cloud for Precision Agriculture. *Agricultural Water Management*, 183, 123–135. <https://doi.org/10.1016/j.agwat.2016.10.020>

Araujo, V., Mitra, K., Saguna, S., & Åhlund, C. (2019). Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities. *Journal of Parallel and Distributed Computing*, 132, 250–261. <https://doi.org/10.1016/j.jpdc.2018.12.010>

Chiewchan, K., Anthony, P., & Samarasinghe, S. (2019). Agent Based Irrigation Management for Mixed-Cropping Farms. In R. Alfred, Y. Lim, A. A. A. Ibrahim, & P. Anthony (Eds.), *Computational Science and Technology* (Vol. 481, pp. 471–480). https://doi.org/10.1007/978-981-13-2622-6_46

Sinha, R. S., Wei, Y., & Hwang, S.-H. (2017). A survey on LPWA technology: LoRa and NB-IoT. *ICT Express*, 3(1), 14–21. <https://doi.org/10.1016/j.icte.2017.03.004>

Ali, A., Shah, G. A., Farooq, M. O., & Ghani, U. (2017). Technologies and challenges in developing Machine-to-Machine applications: A survey. *Journal of Network and Computer Applications*, 83, 124–139. <https://doi.org/10.1016/j.jnca.2017.02.002>

Kunzler, F., Kramer, J.-N., & Kowatsch, T. (2017). Efficacy of mobile context-aware notification management systems: A systematic literature review and meta-analysis. 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking, and Communications (WiMob), 131–138. <https://doi.org/10.1109/WiMOB.2017.8115839>

Coutaz, J., Crowley, J. L., Dobson, S., & Garlan, D. (2005). Context is key. *Communications of the ACM*, 48(3), 49. <https://doi.org/10.1145/1047671.1047703>

SiteWhere Open Source Internet of Things Platform. (n.d.). Retrieved August 25, 2019, from <https://sitewhere.io/fr/>

OpenMTC. (n.d.). Retrieved August 25, 2019, from <https://www.openmtc.org/>

Services et produits de cloud Amazon | AWS. (n.d.). Retrieved August 25, 2019, from <https://aws.amazon.com/fr/>

CrateDB: Simply Scalable SQL Database for IoT & Industrial Time Series. (n.d.). Retrieved August 25, 2019, from <https://crate.io/>

QuantumLeap—QuantumLeap. (n.d.). Retrieved August 25, 2019, from <https://quantumleap.readthedocs.io/en/latest/>

Introduction—WireCloud. (n.d.). Retrieved August 30, 2019, from <https://wirecloud.readthedocs.io/en/stable/>

Esper. (n.d.). Retrieved August 30, 2019, from EsperTech website: <https://www.espertech.com/esper/>

APPENDIX

1) example of a subscription is the following:

```
1. curl -iX POST \  
2. 'http://orion:1026/v2/subscriptions/' \  
3. -H 'Content-Type: application/json' \  
4. -H 'fiware-service: agriculture' \  
5. -H 'fiware-servicepath:/irrigation' \  
6. -d '{  
7.   "description": "Notify Perseo on Humidity and Temperature changes on Device 1",  
8.   "subject": {  
9.     "entities": [  
10.      {  
11.        "type": "tempHumid",  
12.        "idPattern": ".*"  
13.      }  
14.    ],  
15.    "condition": {  
16.      "attrs": [  
17.        "temperatureDegree"  
18.      ]  
19.    }  
20.  },  
21.  "notification": {  
22.    "http": {  
23.      "url": "http://perseo-fe:9090/notices"  
24.    },  
25.    "attrs": [  
26.      "moistureLevel", "temperatureDegree"  
27.    ],  
28.    "metadata": ["dateCreated", "dateModified"]  
29.  },  
30.  "throttling": 1  
31. }'
```

2) method to provision a service group is done as follows:

```
1. curl -iX POST \  
2. 'http://iot-agent:4041/iot/services' \  
3. -H 'Content-Type: application/json' \  
4. -H 'fiware-service: agriculture' \  
5. -H 'fiware-servicepath: /irrigation' \  
6. -d '{  
7.   "services": [  
8.     {  
9.       "apikey": "servicegroup",  
10.      "cbroker": "http://orion:1026",  
11.      "entity_type": "actionEntity",  
12.      "resource": "/iot/d"  
13.     }  
14.   ]
```

```
15. }'  
16.
```

3) request used to provision a sensor:

```
1. curl -iX POST \  
2. 'http://iot-agent:4061/iot/devices' \  
3. -H 'Content-Type: application/json' \  
4. -H 'fiware-service: agriculture' \  
5. -H 'fiware-servicepath: /irrigation' \  
6. -d '{  
7.   "devices": [{  
8.     "device_id": "01_FIWARE_DEVICE",  
9.     "entity_name": "urn:ngsi-ld:tempHumid:001",  
10.    "entity_type": "tempHumid",  
11.    "transport": "MQTT",  
12.  },  
13.  "attributes": [ {  
14.    "object_id": "humidity",  
15.  }, {  
16.    "name": "humidityLevel",  
17.    "type": "Number"  
18.  }, {  
19.    "object_id": "moisture",  
20.  }, {  
21.    "name": "moistureLevel",  
22.    "type": "Number"  
23.  }, {  
24.    "object_id": "temperature",  
25.  }, {  
26.    "name": "temperatureDegree",  
27.    "type": "Number"  
28.  } ],  
29.  "internal_attributes": {  
30.    "lorawan": {  
31.      "application_server": {  
32.        "host": "10.8.244.170",  
33.        "username": "abdul",  
34.        "password": "123456",  
35.        "provider": "loraserver.io"  
36.      },
```

```

37.         "dev_eui": "af8db228120c1cdc",
38.         "app_eui": "70B3D57ED000985F",
39.         "application_id": "191",
40.         "application_key": "722b28e7777bbaa31b3d3ef8adfba52f",
41.         "data_model": "application_server"  } } } ] } '

```

4) Provisioning single devices are done using the following post request:

```

1. curl -iX POST \
2. 'http://localhost:4041/iot/devices' \
3. -H 'Content-Type: application/json' \
4. -H 'fiware-service: agriculture' \
5. -H 'fiware-servicepath: /irrigation' \
6. -d '{
7.   "devices": [
8.     {
9.       "device_id": "pycom1",
10.      "entity_name": "urn:ngsi-ld:actionEntity:001",
11.      "entity_type": "actionEntity",
12.      "protocol": "PDI-IoTA-UltraLight",
13.      "transport": "MQTT",
14.      "timezone": "Europe/Berlin",
15.      "attributes": [
16.
17.      ],
18.      "commands": [
19.        { "name": "on", "type": "command" }, { "name": "off", "type":
"command" } ] } ] }'

```

5) request to add a rule is structured this way:

```

1. curl -iX POST \
2. 'http://perseo-fe:9090/rules' \
3. -H 'Content-Type: application/json' \
4. -H 'fiware-service: agriculture' \
5. -H 'fiware-servicepath:/irrigation' \
6. -d '{
7.
8.   "name": "irrigation_rule",
9.

```

```

10. "text": "select *, ev.humidityLevel? as humid, ev.temperatureDegree? as temp,
    ev.moistureLevel? as moisture, condition? as cond from pattern [every
    ev=iotEvent(cast(moistureLevel?,float)<5 and cast(condition?,float)>10)]"
11.
12. ,"action":{
13.     "type":"update",
14.
15.     "parameters":{
16.
17.         "id":"urn:ngsi-ld:actionEntity:001",
18.
19.         "type":"actionEntity",
20.
21.         "attributes": [{
22.             "name":"ON",
23.
24.             "type":"command",
25.
26.             "value":"on"
27.         }]
28.     , "interval" : "30e3",
29.     "actionType": "UPDATE" }}}}

```

6) action attribute to send an SMS:

```

1. "action": {
2.     "type": "sms",
3.     "template": "Soil moisture has reached the level of ${soilMoisture}.",
4.     "parameters": {
5.         "to": "0664415142 0554415142"
6.     }
7. }

```

7) body of the mail is built in the “template”

```

1. "action": {
2.     "type": "email",
3.     "template": "Temperature has reached the level of ${temperatureDegree}.",
4.     "parameters": {
5.         "to": "me@eurac.edu",
6.         "from": "cep@student-pauwes.dz",

```

```

7.         "subject": "Temperature Warning"
8.     } }

```

8) HTTP request to any external entity

```

1.     "action": {
2.         "type": "post",
3.         "parameters": {
4.             "url": "http://iot-agent:4041/v1/updateContext",
5.             "method": "POST",
6.             "headers": {
7.                 "Content-Type": "application/json",
8.                 "fiware-service": "agriculture",
9.                 "fiware-servicepath": "/irrigation"
10.            },
11.            "json": {
12.                "contextElements": [{
13.                    "isPattern": "false",
14.                    "id": "urn:ngsi-ld:actionEntity:001",
15.                    "attributes": [{
16.                        "name": "on",
17.                        "type": "command",
18.                        "value": "USING HTTP POST FROM PERSEO"
19.                    }] }],
20.                "type": "actionEntity" } } }

```

9) Configuration Code for Actuator Node :

```

1. device = "pycom1" #Defining device ID
2. topic=("/servicegroup/"+device+"/cmd") # The MQTT Topic used to publish messages in
3. state = 0
4.
5. # Function that does message formatting to extract value of command
6.
7. def sub_cb(topic, msg):
8.     global state
9.     print((topic, msg))
10.    if msg == (device+"@on|"+"on").encode('UTF-8'):
11.        pycom.rgbled(0x000010)
12.        p0.value(1)
13.        time.sleep(1)
14.        pycom.rgbled(0x100000)
15.        p0.value(0)
16.        client.publish(topic, msg="off")

```



```

17.     elif msg == (device+"@on|"+"off").encode('UTF-8'):
18.         pycom.rgbled(0x100000)
19.         p0.value(0)
20.
21. wlan = WLAN(mode=WLAN.STA) # Defining the type of Wireless Local Area Network.
22. wlan.connect("ABZN18056W10 7381", auth=(WLAN.WPA2, "6i8/K387"), timeout=5000) #
    Specifying the SSID of the network, and password
23.
24. while not wlan.isconnected(): #
25.     machine.idle()
26. print("Connected to WiFi\n")
27. pycom.rgbled(0x100000)
28.
29. #client      =      MQTTClient("device_id",      "10.8.244.180",user="your_username",
    password="your_api_key", port=1883)
30. client = MQTTClient("pycom1", "10.8.244.180", port=1883)
31. client.set_callback(sub_cb)
32.
33. # OS Error handling, reboot when detected
34. try:
35.     client.connect()
36. except OSError:
37.     print("reboot\n")
38.     machine.deepsleep(100)
39.
40. #Subscribing to the topic
41. client.subscribe(topic)
42.
43. # Listening for incoming messages
44. while True:
45.     try:
46.         client.wait_msg()
47.     except OSError:
48.         print("reboot\n")
49.         machine.deepsleep(100)

```

10) Configuration code for the sensor node

```

1. import sys
2. import time
3. import pycom
4. import struct
5. import socket
6. import machine
7. import ubinascii
8. from network import LoRa
9. from machine import I2C
10.
11. pycom.heartbeat(False)
12.
13. # Initialise LoRa in LORAWAN mode
14. lora = LoRa(mode=LoRa.LORAWAN, device_class=LoRa.CLASS_C, region=LoRa.EU868)
15.
16. # Create an ABP authentication
17. dev_addr = struct.unpack(">I", ubinascii.unhexlify('00ccc852'))[0] # your device
    address here
18. app_swkey = ubinascii.unhexlify('722b28e7777bbaa31b3d3ef8adfb52f') # your application
    session key goes here
19. nwk_swkey = ubinascii.unhexlify('2a5dc5e89cecc44c3716e85c35a48633') # your network
    session key goes here

```

```

20.
21. # Join the network using ABP (Activation By Personalisation)
22. lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey))
23.
24. # Remove all the non-default channels
25. for i in range(3, 16):
26.     lora.remove_channel(i)
27.
28. # Set the 3 default channels to the same frequency
29. lora.add_channel(0, frequency=868100000, dr_min=0, dr_max=5)
30. lora.add_channel(1, frequency=868100000, dr_min=0, dr_max=5)
31. lora.add_channel(2, frequency=868100000, dr_min=0, dr_max=5)
32.
33. """ Temperature and humidity code part 1 (initialization) """
34. TEMP_HUM_SENSOR_ID = 0x27 # identifies the sensor id
35. i2c = I2C(0, I2C.MASTER, baudrate=115200)
36. print("Found the following addresses: ")
37. for i in range(len(i2c.scan())):
38.     print(hex(i2c.scan()[i]))
39.
40. """ Soil moisture sensor code part 1 (initialization) """
41. SOIL_SENSOR_ID = 0x36 # identifies the sensor id
42. reg_1_soil = const(0x0F)
43. reg_2_soil = const(0x10)
44. reg_soil = bytearray([reg_1_soil, reg_2_soil])
45.
46. # Main loop
47. i = 0 # iteration index
48. while True:
49.
50.     # LED blinking
51.     pycom.rgbled(0x000033)
52.     time.sleep(1)
53.     pycom.rgbled(0x000000)
54.
55.     # Send uplink
56.     s = socket.socket(socket.AF_LORA, socket.SOCK_RAW) # create a LoRa socket
57.     s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5) # set the LoRaWAN data rate
58.     s.setblocking(False) # make the socket non-blocking
59.
60.     #s.send(pkt)
61.
62.     #s.send(temp_hum) # uncomment when inserting temperature and humidity code
63.     s.send(temp_hum_soil) # uncomment when inserting soil moisture code
64.
65.     # output print section
66.     print(pkt + " sent")
67.
68.     time.sleep(6) # this dead-time is needed by the network to gather to your LoRa
        packets
69.
70.     # downlink receiving (if enabled)
71.     rx, port = s.recvfrom(4096)
72.     if rx:
73.         print('Received: {}, on port: {}'.format(rx, port))
74.
75.     s.close() # close the LoRa socket
76.     time.sleep(10)

```

11) Subscription of Perseo to Orion

```
1. curl -iX POST \  
2. 'http://localhost:1026/v2/subscriptions/' \  
3. -H 'Content-Type: application/json' \  
4. -H 'fiware-service: agriculture' \  
5. -H 'fiware-servicepath:/irrigation' \  
6. -d '{  
7.   "description": "Notify Perseo on Humidity and Temperature changes on Pycom 1",  
8.   "subject": {  
9.     "entities": [  
10.      {  
11.        "type": "tempHumid",  
12.        "idPattern": ".*"  
13.      }  
14.    ],  
15.    "condition": {  
16.      "attrs": [  
17.        "temperatureDegree"  
18.      ]  
19.    }  
20.  },  
21.  "notification": {  
22.    "http": {  
23.      "url": "http://perseo-fe:9090/notices"  
24.    },  
25.    "attrs": [  
26.      "humidityLevel", "moistureLevel", "temperatureDegree"  
27.    ],  
28.    "metadata": ["dateCreated", "dateModified"]  
29.  },  
30.  "throttling": 1  
31. }'
```

12) Subscription of QuantumLeap to Orion

```
1. curl -iX POST \  
2. 'http://localhost:1026/v2/subscriptions/' \  
3. -H 'Content-Type: application/json' \  
4. -H 'fiware-service: agriculture' \  
5. -H 'fiware-servicepath:/irrigation' \  
6. -d '{  
7.   "description": "Notify quantumleap on Humidity and Temperature changes on Pycom  
8.   1",  
9.   "subject": {  
10.     "entities": [{  
11.       "type": "tempHumid",  
12.       "idPattern": "tempHumid.*"  
13.     }],  
14.     "condition": {  
15.       "attrs": [  
16.         "humidityLevel", "moistureLevel", "temperatureDegree"  
17.       ]  
18.     }  
19.   }  
20. }'
```

```

17.     }
18.   },
19.   "notification": {
20.     "http": {
21.       "url": "http://quantumleap:8668/v2/notify"
22.     },
23.     "attrs": [
24.       "humidityLevel", "moistureLevel", "temperatureDegree"
25.     ],
26.     "metadata": ["dateCreated", "dateModified"]
27.   },
28.   "throttling": 1
29. }'
30.

```

13) Simple rule in Perseo

```

1. curl -iX POST \
2.   'http://localhost:9090/rules' \
3.   -H 'Content-Type: application/json' \
4.   -H 'fiware-service: agriculture' \
5.   -H 'fiware-servicepath:/irrigation' \
6.   -d '
7.
8.   {
9.     "name": "irrigation_rule5",
10.    "text": "select *, ev.humidityLevel? as humid, ev.temperatureDegree? as temp,
11.           ev.moistureLevel? as moisture, condition? as condition from pattern [every
12.           ev=iotEvent((cast(moistureLevel?,float)<10) and condition?=1)]"
13.    , "action": {
14.      "type": "update",
15.      "parameters": {
16.        "id": "urn:ngsi-ld:actionEntity:001",
17.        "type": "actionEntity",
18.        "attributes": [
19.          {
20.            "name": "on",
21.            "type": "command",
22.            "value": "on"
23.          }
24.        ]
25.      }
26.    }
27.    ,
28.    "actionType": "UPDATE" }}}'

```

14) Docker-compose configuration file

```

1. version: "3.5"
2. services:
3.
4.   orion:
5.     image: fiware/orion:2.2.0
6.     hostname: orion
7.     container_name: fiware-orion
8.     depends_on:
9.       - mongo-db
10.    networks:
11.      - default
12.    expose:

```

```

13.     - "1026"
14.   ports:
15.     - "1026:1026"
16.   command: -dbhost mongo-db -logLevel DEBUG -noCache -logForHumans
17.   healthcheck:
18.     test: curl --fail -s http://orion:1026/version || exit 1
19.
20.   perseo-core:
21.     image: telefonicaiot/perseo-core:1.3.0
22.     ports:
23.       - "8080:8080"
24.
25.     networks:
26.       default:
27.         aliases:
28.           - perseo-core
29.     command: -perseo_fe_url perseo-fe:9090
30.
31.   perseo-fe:
32.     image: telefonicaiot/perseo-fe:1.9.0
33.     ports:
34.       - "9090:9090"
35.     networks:
36.       default:
37.         aliases:
38.           - perseo-fe
39.     depends_on:
40.       - perseo-core
41.     environment:
42.       - PERSEO_MONGO_ENDPOINT=mongo-db
43.       - PERSEO_CORE_URL=http://perseo-core:8080
44.       - PERSEO_LOG_LEVEL=debug
45.       - PERSEO_ORION_URL=http://orion:1026
46.       - PERSEO_SMTP_HOST=smtp.gmail.com
47.       - PERSEO_SMTP_PORT=465
48.       - PERSEO_SMTP_SECURE=true
49.       - PERSEO_SMTP_AUTH_USER=XXXXX@XXXXX.com
50.       - PERSEO_SMTP_AUTH_PASS=XXXXX
51.
52.
53.   iot-agent:
54.     image: fiware/iotagent-ul:latest # iotagent
55.     hostname: iot-agent
56.     container_name: fiware-iot-agent
57.     depends_on:
58.       - mongo-db
59.       - mosquito
60.     networks:
61.       - default
62.     expose:
63.       - "4041"
64.       - "7896"
65.     ports:
66.       - "4041:4041"
67.       - "7896:7896"
68.
69.     environment:
70.       - IOTA_CB_HOST=orion # name of the context broker to update context
71.       - IOTA_CB_PORT=1026 # port the context broker listens on to update context
72.       - IOTA_NORTH_PORT=4041

```

```

73.     - IOTA_REGISTRY_TYPE=mongodb #Whether to hold IoT device info in memory or in a
      database
74.     - IOTA_LOG_LEVEL=DEBUG # The log level of the IoT Agent
75.     - IOTA_TIMESTAMP=true # Supply timestamp information with each measurement
76.     - IOTA_CB_NGSI_VERSION=v1 # use NGSIv2 when sending updates for active attributes
77.     - IOTA_AUTOCAST=true # Ensure Ultralight number values are read as numbers not
      strings
78.     - IOTA_MONGO_HOST=mongo-db # The host name of MongoDB
79.     - IOTA_MONGO_PORT=27017 # The port mongoDB is listening on
80.     - IOTA_MONGO_DB=iotagentul # The name of the database used in mongoDB
81.     - IOTA_MQTT_HOST=mosquitto # The host name of the MQTT Broker
82.     - IOTA_MQTT_PORT=1883 # The port the MQTT Broker is listening on to receive
      topics
83.     #- IOTA_DEFAULT_RESOURCE=
84.     - IOTA_PROVIDER_URL=http://iot-agent:4041
85.     healthcheck:
86.     test: curl --fail -s http://iot-agent:4041/iot/about || exit 1
87.
88.     mongo-db:
89.     image: mongo:3.6
90.     hostname: mongo-db
91.     container_name: mongo-db
92.     expose:
93.     - "27017"
94.     ports:
95.     - "27017:27017"
96.     networks:
97.     - default
98.     command: --bind_ip_all --smallfiles
99.     volumes:
100.    - mongo-db:/data
101.
102.    mosquitto:
103.    image: eclipse-mosquitto
104.    hostname: mosquitto
105.    container_name: mosquitto
106.    expose:
107.    - "1883"
108.    - "9001"
109.    ports:
110.    - "1883:1883"
111.    - "9001:9001"
112.    volumes:
113.    - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
114.    networks:
115.    - default
116.
117.
118.    # Quantum Leap is persisting Short Term History to Crate-DB
119.    quantumleap:
120.    image: smartsdk/quantumleap:0.7.0
121.    hostname: quantumleap
122.    container_name: fiware-quantumleap
123.    ports:
124.    - "8668:8668"
125.    depends_on:
126.    - crate-db
127.    environment:
128.    - CRATE_HOST=crate-db
129.    healthcheck:
130.    test: curl --fail -s http://localhost:8668/v2/version || exit 1

```

```
131.
132.
133.     crate-db:
134.         image: crate:${CRATE_VERSION:-3.3.5}
135.         hostname: crate-db
136.         container_name: db-crate
137.         ports:
138.             # Admin UI
139.             - "4200:4200"
140.             # Transport protocol
141.             - "4300:4300"
142.         command: crate -License.enterprise=false -Cauth.host_based.enabled=false
-Ccluster.name=democluster -Chttp.cors.enabled=true -Chttp.cors.allow-origin="*"
143.         volumes:
144.             - crate-db:/data
145.
146.         # Other services
147.         grafana:
148.             image: grafana/grafana:latest
149.             container_name: grafana
150.             depends_on:
151.                 - crate-db
152.             ports:
153.                 - "3003:3000"
154.             environment:
155.                 - GF_INSTALL_PLUGINS=grafana-clock-panel,grafana-worldmap-panel
156.             volumes:
157.                 - grafana:/var/lib/grafana
158.
159.         networks:
160.             default:
161.                 ipam:
162.                     config:
163.                         - subnet: 172.18.1.0/24
164.
165.         volumes:
166.             mongo-db: ~
167.             crate-db: ~
168.             grafana: ~
```

5.3 Research grant use:

Abdelkader SALLEMINE : Energy Engineering

Item	price	Column1	currency price	exchange
Travel Insurance	7324.22	DZD	119.01	\$61.54
Bus ticket (Tlemcen-Algiers-Adrar-Tlemcen)	5350	DZD	119.01	\$44.95
Flight tickets (Algiers-milan)	45465	DZD	119.01	\$382.03
VISA fees	21876	DZD	119.01	\$183.82
Airport transportation	24	EUR	1 EUR = 1.1397 USD	\$27.35
Bus ticket (Bolzano-Milan)	42.5	EUR	1 EUR = 1.1397 USD	\$48.43
Train tickets (Milan-verona-Bolzano)	49.3	EUR	1 EUR = 1.1397 USD	\$56.18
Bank exchange fees (400 USD)	7.01	EUR	1 EUR = 1.1397 USD	\$8.00

Sim Card + 1 month internet	30	EUR	1 EUR = 1.1397 USD	\$34.19
stay permit	116.46	EUR	1 EUR = 1.1397 USD	\$132.72
Bank exchange fees	17.49	EUR	1 EUR = 1.1433 USD	\$20.00
field transportation	15	EUR	1 EUR = 1.1433 USD	\$17.14
Internet charge 1 month	15	EUR	1 EUR = 1.1433 USD	\$17.15
Bank exchange fees	2.5	EUR	1 EUR = 1.1458 USD	\$2.86
tickets (Bolzano- verona) 2 ways	17.59	EUR	1 EUR = 1.1433 USD	\$20.11
entrance	10	EUR	1 EUR = 1.1433 USD	\$11.43
Train tickets (Milan- trentino) 2 ways	14.6	EUR	1 EUR = 1.1433 USD	\$16.70

Thesis printing	4390	DZD	1 USD = 119.01 DZD	\$36.88
				Total : \$1,121.48

Research grant was spent according to the table above, mostly on transportation, visa, printing. It was not possible to obtain the devices due to many financial and rule complications.