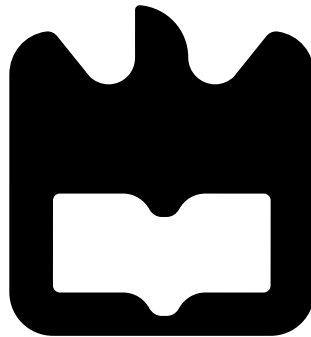




**Ivo Daniel
Pinto da Silva**

**Standards IdC para Cidades Inteligentes
IoT Standards for Smart Cities**





**Ivo Daniel
Pinto da Silva**

**Standards IdC para Cidades Inteligentes
IoT Standards for Smart Cities**

“If you cannot fail, you cannot
learn.”

— Eric Ries



**Ivo Daniel
Pinto da Silva**

**Standards IdC para Cidades Inteligentes
IoT Standards for Smart Cities**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica de Diogo Gomes, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor Joaquim João Estrela Ribeiro Silvestre Madeira
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Diogo Nuno Pereira Gomes
Professor Auxiliar da Universidade de Aveiro (orientador)

Professora Doutora Ana Cristina Costa Aguiar
Professora Auxiliar da Faculdade de Engenharia da Universidade do Porto

**agradecimentos /
acknowledgements**

I want to thank my family. Without them I would not have the needed support to enroll, pursue and conclude this degree.

On another note, professor Diogo Gomes and Ricardo Vitorino from Ubiwhere were excellent supervisors. I want to thank Ricardo for the availability and understanding and professor Diogo for good advice and for always having my best interest in mind.

I am also very thankful for all the support given by the Ubiwhere team, especially André Duarte and Francisco Monsanto that were always available to help with technical issues and Ricardo Machado that gave me the opportunity to grow, as a person and developer, with the company.

From the University, André Marques was definitely an enormous help for technical discussions and for good insight on IoT related subjects.

On a personal level I want to thank all my closest friends. Thank you Fabiana for keeping me grounded and always being there when I need. Thank you Diogo Cardoso, Daniel Silva, Tiago Henriques, Tiago Magalhães, Ricardo Martins and André Jerónimo for letting me share my experiences and always taking my opinions into account.

Resumo

Hoje em dia, o panorama da Internet das Coisas enfrenta um grande obstáculo ao tentar estabelecer um ecossistema global devido ao surgimento de vários standards de dados que se adequam às necessidades de diferentes use cases e/ou indústrias.

No cenário das Cidades Inteligentes é agora mais claro do que nunca que é absolutamente necessário traçar o caminho para a interoperabilidade de diferentes soluções verticais de modo a tornar as cidades verdadeiramente inteligentes.

No âmbito desta dissertação o objetivo é partir de uma implementação desenvolvida de uma solução IdC vertical e possibilitar a integração de aplicações de terceiros de uma forma transparente.

Abstract

Nowadays, the IoT panorama faces a great obstacle in establishing a global ecosystem, due to the emergence of a number of data standards created to fit the needs of different use cases and/or industries.

In the Smart Cities scenario it is now clear that paving the way for the interoperability of different vertical solutions is an absolute necessity in order to make them truly intelligent.

It is in the interest of this dissertation to take a developed implementation of a vertical IoT solution and enable the integration of third-party applications in a transparent way.

CONTENTS

CONTENTS	i
LIST OF FIGURES	v
ACRONYMS	vii
1 INTRODUCTION	1
1.1 Internet of Things	1
1.2 Development in a business environment	3
1.3 Involvent Project/Product	4
1.4 Motivation	6
2 STATE OF THE ART	7
2.1 Internet of Things	7
2.1.1 Cisco	8
2.1.2 AT&T	8
2.1.3 IBM	8
2.1.4 AllSeen Alliance	9
2.1.5 Open Mobile Alliance	9
2.1.6 European Telecommunications Standards Institute (ETSI) M2M . .	10
2.1.7 Eclipse Internet of Things (IoT)	10
2.1.8 FIWARE	11
2.2 Smart Cities	12
2.2.1 Porto, Portugal	13
2.2.2 Santander, Spain	13
2.2.3 Array of Things	14
2.3 IoT Protocols	14
2.3.1 Hypertext Transfer Protocol (HTTP)	14
2.3.2 Constrained Application Protocol (CoAP)	15
2.3.3 MQ Telemetry Transport (MQTT)	15
2.3.4 Lightweight Machine-to-Machine (LwM2M)	16
2.3.5 Extensible Messaging and Presence Protocol (XMPP)	16
2.3.6 Overview and Comparison	17
2.4 Interoperability	17
2.4.1 Meshblu	18
2.4.2 Ponte by eclipse	18
2.4.3 Hypercat	19

2.4.4	FIWARE's enablers	20
2.4.4.1	IDAS	21
2.4.4.2	Orion Context Broker	22
2.4.4.3	NGSI 10	23
2.4.5	OpenMTC	24
2.4.6	Overview and Comparison	24
3	CITIBRAIN	27
3.1	General Description	27
3.1.1	Devices	28
3.1.2	Message broker	30
3.1.3	Core components	31
3.1.4	Backends & Application Program Interface (API)s	32
3.1.5	Client applications	33
3.2	Hypercat Interoperability	35
3.2.1	Reasons to support Hypercat	35
3.2.2	Solution	35
3.2.3	Implementation and tests	37
3.2.4	Benchmarks	41
3.3	FIWARE support	43
3.3.1	FIWARE at Ubiwhere	43
3.3.2	NGSI at Citibrain	46
3.3.3	IDAS at Citibrain	47
3.3.4	Implementation and Tests	49
3.3.5	Benchmarks	51
3.4	LwM2M Support	52
3.4.1	Experiments	52
3.4.2	First approach	54
3.4.3	Integration with Meshblu	55
3.4.4	Citibrain's object specification	56
3.4.5	Communication flow	58
3.4.6	Tests	61
3.4.6.1	Device registration	62
3.4.6.2	Device observation	66
3.4.6.3	Resource update	67
3.4.7	Benchmarks	69
4	OPEN-SOURCE CONTRIBUTIONS	73
4.1	LwM2M	73
4.2	Hypercat	73
4.3	Fiware	74
5	CONCLUSION	75
5.1	Future work	76
6	APPENDIX A	77
6.1	Citibrain's Hypercat "/cat" method response	77
7	APPENDIX B	81
7.1	Citibrain's Hypercat "/cat/parking/assets" request response	81

8	APPENDIX C	83
8.1	LwM2M waste event specification	83
	BIBLIOGRAPHY	87

LIST OF FIGURES

2.1	ETSI M2M architecture reference points	10
2.2	Ponte's wide range of protocols support	19
2.3	FIWARE enablers for IoT platforms	21
2.4	Fundamental use-case for the IDAS component	22
2.5	An example architecture diagram for the use of the Orion Context Broker	22
2.6	An NGSI compliant APIs resource tree	23
3.1	Citibrain's architecture overview	28
3.2	Citibrain's range of sensors communicating with a Gateway	29
3.3	Citibrain's broker and brokerage nodes	30
3.4	Citibrain's core components	31
3.5	Citibrain's backends and APIs	32
3.6	Citibrain's client applications	33
3.7	Citibrain's mobility API serving parking information to app	34
3.8	Citibrain data translation into/from the hypercat specification	35
3.9	Citibrain architecture diagram after the hypercat wrapper implementation	36
3.10	Citibrain's Hypercat API GET methods	37
3.11	Parking events request	40
3.12	Time spent processing a Hypercat GET request in comparison to Citibrain	41
3.13	Time spent processing a Hypercat POST request in comparison to Citibrain	42
3.14	FIWARE Porto's demo	46
3.15	Vertical APIs exposing NGSI nodes	47
3.16	LwM2M client creation	48
3.17	IDAS receiving a registration request successfully	48
3.18	IDAS reporting an error due to the fact that it can't find the Context Broker	49
3.19	IDAS client registration flow	49
3.20	Waste event response in proprietary format	50
3.21	Waste event response in NGSI format	50
3.22	Elapsed time on HTTP requests to the Citibrain API endpoints in comparison to NGSI endpoints	51
3.23	Leshan server detecting the LwM2M client	52
3.24	Leshan server reading/subscribing client's resources	52
3.25	LwM2M client creation and resource assignment	53
3.26	LwM2M first approach	54
3.27	LwM2M integration in Meshblu	55
3.28	LwM2M registration flow	58
3.29	LwM2M registration flow saving the Meshblu's credentials	59

3.30	LwM2M object observation	60
3.31	LwM2M complete message flow example for parking sensor	61
3.32	LwM2M client creation and registration.	64
3.33	Meshblu registering the device and printing the generated device Universally Unique Identifier (UUID) and authentication token.	64
3.34	Client's Meshblu credentials being written in appropriate resources.	65
3.35	Client's resource value list.	65
3.36	Meshblu's HTTP registered clients list method.	66
3.37	Meshblu establishing an observer connection with the valuable client's resources.	67
3.38	A LwM2M client updating its "water_flow" resource.	67
3.39	Meshblu's output for a device registration and following update.	68
3.40	HTTP list devices request to Meshblu.	69
3.41	Time elapsed registering devices into Meshblu via HTTP	70
3.42	Time elapsed updating devices into Meshblu via HTTP	70
3.43	Time elapsed registering devices into Meshblu via LwM2M	71
3.44	Time elapsed updating devices into Meshblu via LwM2M	72
5.1	Overview of the Citibrain platform	75

ACRONYMS

IoT	Internet of Things
M2M	Machine-to-Machine
H2M	Human-to-Machine
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
CoAP	Constrained Application Protocol
LwM2M	Lightweight Machine-to-Machine
MQTT	MQ Telemetry Transport
API	Application Program Interface
XMPP	Extensible Messaging and Presence Protocol
ETSI	European Telecommunications Standards Institute
OMA	Open Mobile Alliance
GSCL	Gateway Service Capability Layer
NSCL	Network Service Capability Layer
SME	Small and medium-sized Enterprises
JSON	Javascript Object Notation
UUID	Universally Unique Identifier
XML	Extensible Markup Language
OpenXPS	Open XML Paper Specification
REST	Representational State Transfer
3GPP	Third Generation Partnership Project
CEP	Complex Event Processing
IFTTT	If-This-Than-That
CRUD	Create, Read, Update and Delete
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
OASC	Open and Agile Smart Cities
DSCL	Device Service Capability Layer

INTRODUCTION

1.1 INTERNET OF THINGS

Nowadays, the Internet of Things panorama is suffering from a severe problem that is preventing the paradigm from establishing itself and becoming the truly interconnected ecosystem that it aims to be.

IoT is an idea that predicts the interconnection of billions of devices worldwide. The goal is to turn every device smart, which means being able to get information from the outside world and surrounding devices and being able to act upon it.

One can imagine how a device gets the ability of sensing its surroundings, how can it be aware of the temperature, for example. Through sensors, sensors have a big role on the Internet of Things scenario. A sensor is capable of measuring something or detecting a change in the state of what it is monitoring.

This leads us to the Smart Cities scenarios. The term Smart City refers to a city that is sustainable and has monitored useful information about itself.

A smart city takes advantage of the IoT use-cases to enhance the quality of the services that it provides to the community while reducing costs by controlling the waste, of the used resources and actual garbage, that it generates.

A couple of examples of how this can be achieved are:

- Building systems that detect water level rises and predict floods;
- Monitoring traffic in certain areas so that, in case of emergency, ambulances and police cars take less time getting to a desired location.

The possibilities are endless.

It is not hard to visualise the overlap of the the Internet of Things and Smart Cities.

In a place filled with sensors, one is able to extract so much knowledge about what surrounds us that we can build very useful applications. But there is a big question: how can developers get their systems and platforms communicating with these sensors?

Every sensor is different. A temperature sensor can provide measurements in Celsius and the other in Fahrenheit. There is a need of context on the received information in order to understand it in an automatic way. First, if a device is communicating with another, whether it is a sensor or another device, it will interpret the received information the way the developer intends it to.

Following this line of thought, a developer decides to build an app that reads temperature information from a sensor. It is expected for the sensor to provide a number that corresponds to the temperature. Also, there is the need of knowing if the said temperature is in Celsius, Fahrenheit or any other unit. One can predict the outcome of testing the same application using different sensor models. Possibly it will not work because it will not make the information available in the same way that the previous one did. As it is implied, this is not scalable. The developer would have to develop an application for each sensor type that would be installed in the different locations to do the same exact operation. This is a big problem, “telling a device” how he should interpret the information. A device is only as intelligent as we program it to be. It does not have the capability of interpreting information and decide if it is good or bad like humans do.

This is why standards are needed in the IoT world. If sharing information with devices around us is a true goal to achieve, a valid, homogeneous interface from one device to another must be set so that they become able to “talk” and “understand” each other.

The fact is that, for the Internet of Things to scale and reach the potential that developers and big corporations are intending for it, there must exist one single communication standard. This will enable a large range of devices to communicate with one another. With this in mind, several early-adopters started to develop standards that they hope will define the communication and enable IoT interconnection.

Until this point, all was progressing as expected. Everyone agreed with what has been mentioned previously. From this point forward, it can be said that the competition has started.

The competition is between different companies that want to set the trend and get developers to build platforms upon their systems. Every effort should be invested in moving forward with technology and making the world a better place but many companies have the ambition of holding a big part of the market share and increasing their profit.

Yes, there are many initiatives that aim to turn the IoT dream to reality. But there are corporations, consortia and foundations which for them it is a business. Right now there are different “teams” with their standard competing to be the most interoperable out of all. Because of this, now a number of standards exist and not just one. Having different systems using different standards is what is available today. In what will it benefit developers having multiple standards for the same thing? If there are many ways of doing something, and people do it differently from one another, than there is no point in calling that a standard.

The Internet of Things needs one standard, that is agreed upon by experts, in order to truly benefit of the interoperability of devices. Until then, it will be impossible for devices to share information and

create the desired cross-functionality.

These different teams of companies that are gathering to create their own standard, often do not agree with the decisions of other conglomerates and insist that their implementation should be the standard. So now, multiple consortia exist and companies join not only one but several of them in the same way as they would place bets on which standard will prosper. These “bets” also allow them to analyse the situation from the inside, getting a better understanding of the vision and how the standard is being developed.

Knowing that, it is clear that the ideal situation for the Internet of Things, which is the implementation of one, and only one, standard for communication between devices will not become reality as soon as we wish. It becomes obvious that this delays the deployment of true Smart Cities.

In the Smart City scenario, it is necessary to guarantee the interoperability of the different endpoints while enabling the creation of a unified ecosystem for the cities, in order for them to become truly intelligent.

This dissertation has the goal of paving the way to make the IoT developers lives easier when using the Citibrain platform. It is important that developers are able to integrate their applications seamlessly with the IoT solutions already implemented and working in the field, potentiating the transparency and interoperability of the Smart Cities solutions.

1.2 DEVELOPMENT IN A BUSINESS ENVIRONMENT

This dissertation was developed in conjunction with an internship at Ubiwhere, a company based in Aveiro. The opportunity arose and it felt like a better suited environment for me. It also gave me the chance to position myself better in the market and provided me a smooth transition from the academic field into the business environment.



Ubiwhere is a company based in Aveiro, Portugal. It had its origin in the University of Aveiro’s Business Incubator (IEUA) and its creation arose from the enthusiasm of three young researcher’s from the Institute of Telecommunications and PT Inovação, SA in pursuing a successful career.

The proximity to the Institute of Telecommunications of Aveiro, the headquarters of PT Inovação and Nokia Siemens Networks division was an important fact in the choice of Aveiro for the company’s headquarters.

One of the main goals of the company is to research and develop bleeding edge technologies, design state-of-the-art solutions and create valuable intellectual property.

Ubiwhere’s vision is to be an international reference in smart cities and future internet and its mission is to improve people’s lives by developing usable technologies.

The company is involved in several projects regarding Smart Cities and Telecommunications but the biggest interest in this particular theme comes from the fact that Ubiwhere is a major part of the Citibrain consortium which specialises in smart solutions for today's cities.

Citibrain's solutions for Smart Cities includes parking, waste management, environmental quality control, traffic management, vending and smart cards. These solutions are very great and have been implemented singularly but the goal here is to connect the different verticals and in this way provide a unique product that fits most of the needs of Smart Cities, all in one platform.

My first contact with Ubiwhere was when I applied for a Summer Internship. I spent that same Summer working at the company and learned everything I know about web development, working in a team, continuous integration and overall work life. Because I liked the in-house environment and the projects proposed were in the fields I wished for, I took the opportunity of doing my dissertation along with this team and implementing something that may actually be a part of a shipped, on the field platform is very rewarding.

1.3 INVOLVENT PROJECT/PRODUCT



The Citibrain platform offers six main services that are:

Smart Parking

Brings together all aspects of parking management technology into one integrated system. It is a parking management system in real time that collects performance indicators and translates them into knowledge so that the managers are able to formulate better policies. Smart Parking is also a robust way to let users know about the location of free parking spaces. This solution reduces costs of parking, accidents and traffic congestion as it improves parking operations.

Smart Waste Management

Citibrain introduces a system that makes the collection of the city's urban waste easier. This is done by placing sensors with low energy consumption and high durability in the traditional trash bins and containers. By doing this, it is possible to keep a tight control on the state of the container, its location and security, thus increasing the effectiveness and efficiency of the waste management teams while aiming to make the cities greener.

Smart Environmental Quality

Through sensing stations positioned in the current urban infrastructure, it is possible to draw indicators on air quality, noise pollution levels, temperature, humidity among others. This data is useful to respond to the problems of the citizens, to improve urban planning and obtain objective data on the quality of life provided.

Smart Traffic Management

Solves the problem of traffic management in urban environments, done in an adaptive and non-invasive way. This happens through the installation of low cost sensors throughout the city. The sensor data is transmitted in real time and combining it with information from the drivers' mobile devices, it is possible to adapt the traffic flow of the city. It is meant to operate in real time, setting the states of vertical signs, informative panels and even sending alerts to the citizens' mobile devices.

Smart Vending

The solution offers its costumers flexible payment options and allows to achieve high levels of efficiency in the management of their assets by monitoring the machines remotely in real time. The vending machines accept payments in money, debit/credit card and Citibrain Smart Card. They also contain multiple sensors that alert the owners about their location, inventory and maintenance issues. This solution simplifies the business, enables the automation of routine tasks and allows companies to increase efficiency in the management of their assets.

Smart Card

The Smart Card monitors physical access to buildings and controlled spaces and can be used in a wide range of applications. It is intended to support organisational management bodies such as companies, governmental institutions and schools in security, control, payments and information management areas. Each user is assigned a single and multifunctional card, which acts as their identification, installations access, parking and substitute for money.

However, the main and core services where Citibrain is the most efficient and revolutionary are parking, waste management, environmental quality and traffic management.

1.4 MOTIVATION

With no end in sight for this “standard war” and as an agreement between companies is not a likely outcome, IoT developers are in need of a tool that is capable of doing the standard conversion for them.

This is Ubiwhere’s way of getting closer to what is truly the desired consequence of the Internet of Things as a whole. While the one universal standard is still in development and being fought for among the big players in the corporate world, this project comes to force the integration of different standards within the platform in order to move one step closer to interoperability. This can add value in the sense that it makes Citibrain a more available platform allowing the team to connect different verticals and also decouple them to integrate in other deployed platforms that are using different communications protocols and formats from our proprietary one.

STATE OF THE ART

2.1 INTERNET OF THINGS

The next generation of the Internet is expected to be the Internet of Things. Now society takes advantage of e-commerce, the cloud, social media among others. But the next step is to connect things and devices. This will enable the interconnection of sensors, vehicles, mobile phones and other devices instantly without the need for pairing nor connection to the cloud.

Developers will benefit from this by being able to build context aware applications. For example, if the trash bins in one's house are full, the company that is in charge of the garbage collection is notified to pick up the trash earlier than it was supposed to. This is only one use case for the IoT era.

The Internet of Things panorama is quite undefined at this moment. Universities and companies know what they are supposed to do but no one does it well. This results in everyone implementing their IoT systems in the way that they consider correct, which leads to small interoperability.

In order for the "Things" to communicate with each other, they must know how to do it. This means that there has to exist a common interface to which every device must obey in order to achieve the full interoperability. What is happening is that, nowadays, many groups of companies have different visions on what that interface should look like. Different consortia are building different standards interpreted to be the best-fitting one and refuse to compromise with the view of others.

One standard is perfect. If there is more than one standard for the same purpose, then there is no standard at all.

Some companies are really thriving and trying to push the Internet of Things to the next level. There are the big players that are always looking for the next big thing. They are IBM, AT&T, Cisco, General Electric, Apple, Google, Amazon and other major influencers.

IBM's effort is seen in the form of its Watson platform: "IBM's well-known machine learning platform Watson is a big part of IBM's vision of the IoT world. The super computer renders actionable insight from massive amounts of data produced by IBM IoT sensors at a pace no human could ever match." [1]

AT&T predicts that the IoT is a way for the company to grow while providing ways for the sensors, devices and platforms to communicate with each other. "In order to function, the "Internet of Things"

relies on wireless broadband connections and AT&T sees IoT applications in a number of verticals as the natural next step in the evolution of the company.”[2]

Google is a company that always wants to be the first to achieve the most innovative solutions as seen with extravagant initiatives such as project Loon[3] and the space elevator[4]. So the IoT is, by no means, a field that they are forgetting as “The ubiquitous technology giant is often synonymous with forward thinking innovation and the Internet of Things space is no exception.”[5]

Right now, IDC (International Data Corporation) predicts that the market for the Internet of Things will nearly triple, reaching \$1.7 trillion in four or five years.[2]

The following subsections will provide a more detailed analysis of the above mentioned companies and consortia.

2.1.1 CISCO

Cisco aims to help push IoT scenarios into the real world by featuring solutions such as:

- Managing and giving intelligence to multiple enterprise-class networks;
- Optimising scalability of solutions. This is extremely crucial if the rising number of connected devices is taken into account. And the number is only going to keep rising;
- Moving to IoT connected network without the need of upgrading your current infrastructure solution;
- Making the network solutions perform well in extreme conditions;
- Promoting normalisation and connectivity across multiple network environments, which means: to support multiple communication protocols.

According to Cisco, the big benefits of an Internet of Things world are that data-driven efficiencies are able to reduce inventory, downtime and time to market in a company, will provide new business opportunities and revenue streams and better decision making through informed prioritisation.[6]

2.1.2 AT&T

From AT&T’s point of view, IoT solutions will also help lowering costs, performing more efficiently and improve competitive advantage.

They state that “Harnessing data to predict, learn, and make real-time decisions can create a distinct competitive advantage for your business”. Which means that data is much more valuable than what society imagines right now and recognising it is the first step into taking advantage and welcoming a positive change using IoT.

2.1.3 IBM

When it comes to IBM, they state that “The early applications of IoT are undoubtedly delivering great value. They are reshaping customer experiences by putting consumers into context, and offering

new avenues for engagement.”

They go so far as sharing that data is the currency of IoT and the capturing and transmission of this data in a secure way is critical to the success of any IoT strategy. Their vision includes having products and services with cognition. This will enable the creation of products that sense, reason and learn from their users and the world around them. The main goal of this is to allow these companies to develop “things” that are in constant adaptation and improvement.

Yet, their position is that the Internet of Things is nothing without cognitive computing. “It is essential in realising the true value of the IoT. And in so doing, together we will discover answers to questions we never thought to ask.”[1]

2.1.4 ALLSEEN ALLIANCE

The AllSeen Alliance has the mission of enabling the interoperability between different products and brands to provide intelligent experiences for the Internet of Things. The initiative includes more than two hundred members in which are included leading consumer electronic manufacturers, home appliance makers, automotive companies, cloud providers. . .

This alliance has created an open source software framework[7] that makes it easy for devices and apps to discover and communicate with each other. The main goal is to enable the developers to write apps that are interoperable without knowing the transport layer, manufacturer and without needing internet access.

AllSeen framework uses a client-server model to organize itself. Each “information producer” on the network has an Extensible Markup Language (XML) file called introspection that advertises the devices abilities and what it can be asked to do.

2.1.5 OPEN MOBILE ALLIANCE

The Open Mobile Alliance is an association that develops open standards for the mobile communications industry.[8]

Open Mobile Alliance (OMA)’s goals are:

- Provide open technical specifications that take into account market requirements and improve the extensibility and modularity of the systems while reducing the industry’s implementation efforts;
- Ensure their enablers provide interoperability across a wide range of endpoints, locations, service providers, networks. . . ;
- Strive for the consolidations of standards in the mobile industry by working with other existing standards organisations;
- Value and benefit members in OMA regardless of where they stand in the value chain;

The OMA maintains a large number of specifications including MMS (multimedia messaging), OMA IMPS (instant messaging and presence service) and OMA LwM2M (Light Weight Machine-to-Machine).

2.1.6 ETSI M2M

ETSI M2M is an initiative that thrives to standardise the Internet of Things. In order to do so, ETSI proposed a solution to manage, process, store and transfer big amounts of data in a secure fashion.[9]

An ETSI compliant platform needs to consist of three service capability layers:

- Gateway Service Capability Layer (GSCL);
- Network Service Capability Layer (NSCL);
- Device Service Capability Layer (DSCL);

The **GSCL** is a flexible Machine-to-Machine (M2M) gateway that supports various M2M area network technologies and communication protocols such as ZigBee and Wireless M-Bus.

The **NSCL** is a cloud-based M2M platform that aggregates and stores data from various devices and acts as a device management and abstraction layer providing intuitive APIs.

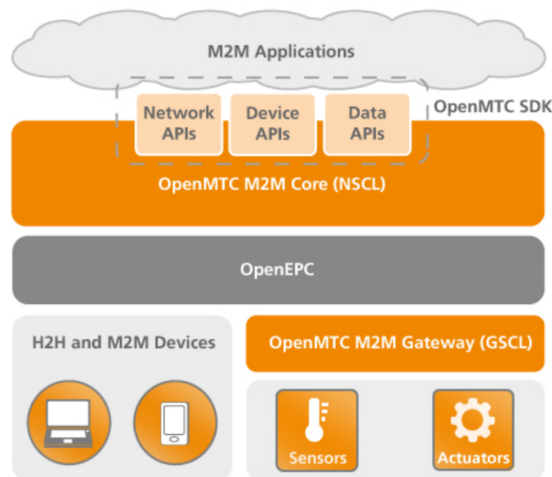


Figure 2.1: ETSI M2M architecture reference points.

2.1.7 ECLIPSE IoT

The IDE giant Eclipse also wants to be part of the IoT future. With this in mind they are trying to push a set of standards that they feel that are definitely important in the development and maturing of the ecosystem.[10]

They state that there are many aspects of an end-to-end solution where it's important to rely on standards like protocols that enable device-to-device and device-to-server communication, device management and gateway interface discovery. Eclipse believes that open standards are essential but it is also important to make available open-source implementations of these standards in order to motivate their adoption by IoT developers and the whole industry.

The project provides implementations of CoAP, ETSI SmartM2M, MQTT and LightWeightM2M.

2.1.8 FIWARE

FIWARE is a middleware platform that's driven by the European Union with the goal of developing and deploying applications for the Future Internet and Smart Cities.[11]

“The involvement of users and developers is critical for this platform to become a standard and reusable solution.”

Its purpose is to facilitate a cost-effective creation and delivery of Internet of Things applications and services in a variety of areas that include smart cities, sustainable transport, renewable energy and environmental sustainability.

FIWARE's main targets are small and medium entrepreneurs and startups with the intention of leveraging opportunities in areas as the Internet of Things, Open Data, Big Data, Smart Cities. . .

The platform is entirely open-source and based on openstack and the project attempts to not only setup european but also worldwide standards.

There already are thirty one cities that agreed on using FIWARE standards to adopt open APIs to gather, publish, query and subscribe to context information describing what happens in the city at any point in time. This translates into an achievement in their attempt at the establishment of a standardised environment. These municipalities that are adopting the standard are striving to create big market to attract small and medium entrepreneurs to invest in high quality, cheap and scalable applications.

FIWARE is a result of a public-private partnership between the EU and professional IT companies.[12] The EU's interest in the digital market comes from the desire to be more independent of big US commercial players and wants to promote innovation by common standards. The initial program includes many accelerators to help small teams that have ideas and, hopefully, transform them into a business.

In a Smart City that supports FIWARE, they would implement and maintain the setup for the application developers, hiding the complexity of the data gathering and processing from the developer, this way the application would only connect to the front-end of the FIWARE ecosystem that is the context broker. In this situation, the app developers do not need to worry about how the data is collected and is always sure that the app is aware of the context of the city that it is built for.

Adopting FIWARE means making developed applications “aware”. What FIWARE does to help this task is giving the developer the tools to produce, gather, publish and consume context information in a very large scale. Data from thousands of sensors is being made available in their interfaces. Additionally, the more developers start supporting FIWARE, the more data is made available and, consequently, applications become more and more context aware due to the fact that they become able to sense their surroundings better.

The ultimate goal for the FIWARE project is to be installed in every city around the globe. This would mean that every device supporting their standards would be interoperable. They could create content at the same time that they would be consuming it. Every device would be able to use each other's context information in a way that would make applications truly “smart” and “aware”.

2.2 SMART CITIES

A smart city is one that uses information technologies, consumes and makes relevant data available. Relevant data can be information regarding the weather, the water level, traffic conditions, among others. There is a lot of information referring to a city that can be used by developers to build applications.

This data is, for the most part, collected by sensors placed in strategic locations. These sensors need to communicate their events to a centralised platform that gathers all the information. This is known as a broker.

A broker serves as a mediator between producers and consumers of the information, which in this case are sensors and user applications.

The main goal of the cities when they make this kind of data publicly available is to motivate developers to build software that can help the citizens of that same city in their daily life improving the quality of the services provided by the municipalities.

Services like public transports can be improved by knowing the time that they get more attendance, the traffic situation of the route and information of that nature.

In this scenario, there are already several cities investing and disclosing data for developers to build these smart platforms for them.

“Imagine walking or driving through a city and the city itself tells you which trendy spots to check out, where to park your car in that moment, or which areas to avoid because of air quality or traffic congestion. In Porto, Portugal, this vision has become a reality and the city itself is already communicating directly with residents, tourists and even startup businesses using FIWARE standards and the UrbanSense platform.”[13]

The quote above refers to the implementation of the FIWARE standards in Porto, Portugal. Porto is serving as a test bench for FIWARE developers to test the applications they build in a real world scenario. Porto is positioning itself to become a very attractive city for SmartCity development and establishment. “Porto has been a pioneer city adopting FIWARE standards with the support of Ubiwhere”.

In the same article we can read that “Lack of standard interfaces for accessing real-time data of cities becomes a rather huge challenge for Small and medium-sized Enterprises (SME)s and startups because they cannot afford to repeat the development of adapters in each city. While the benefits for end users can be great, they are too high to be passed on through a low-costing app, which has held back the smart cities, Internet of Things, and civic tech industries so far.”

So, here lies the cause of the delaying of the Smart Cities and IoT solutions in general. There are no standards and until there is an establishment of a technology that allows for different devices to communicate with each other, being from different manufacturers or not, there is no way that the true Smart City concept becomes reality. Right now, even if there are cities that make data available and invest in the development of Smart City solutions, this effort is made in different directions in different cities. An example of this is that Porto and other cities like Amsterdam and Eindhoven support the FIWARE ecosystem and that’s great because they’re allowing FIWARE developers to take advantage of their data and build useful solutions. While this is happening, the city of Chicago is running a project called Array of Things.

“IoT will provide real-time, location-based data about the city’s environment, infrastructure and activity to researchers and the public. This initiative has the potential to allow researchers,

policymakers, developers and residents to work together and take specific actions that will make Chicago and other cities healthier, more efficient and more livable.” It is extremely unlikely that this Array of Things specification for the openly available data communication is the same that FIWARE is using in Porto and Amsterdam. This means that if a developer builds a fantastic app for Porto and everyone loves it there, it is not possible to deploy it in Chicago.

2.2.1 PORTO, PORTUGAL

In 2016 Porto has been a pioneer adopting FIWARE’s smart city standards. This leap has already leveraged the development of the UrbanSense infrastructure that was developed under the Future Cities project, which is a partnership between the University of Porto and the city council.[14]

With the participation of Citibrain, the city of Porto and Ubiwhere have developed platforms that bring access to real-time environmental data from seventy five monitoring stations located across the city and more than two hundred scanners installed on the Council’s fleet of cars.

External service providers like water suppliers, transports data providers, social media data and business statistics are connected into the platform allowing the city to guide citizens as they go about their lives.

To demonstrate their commitment to the Smart Cities initiative, the city of Porto created a competition called “Desafios Porto” where they challenged developers to build applications that would take advantage of this data and provide useful use cases for the city’s population. These challenges included building apps to discover what is happening in town at any given moment or one to report real-time issues that would happen in Porto so that their resolution could be performed faster.

2.2.2 SANTANDER, SPAIN

There was a time where the Spanish city of Santander had little interaction with the outside world. This happened before it was chosen to be Europe’s test bed for a sensor-based smart city.

More than ten thousand sensors were placed around the city with the goal of measuring everything from the amount of trash in containers, number of parking spaces available, affluence of citizens in a given area. . . In addition, there are sensors on police and taxi cars that measure air pollution levels and traffic conditions.[15]

The wide number of sensors allows for a huge data collection and analysis that proceeds to give the city the possibility to adjust their different services in order to better suit the needs of the citizens. The goal for Santander is to make it a more attractive place to visit, shop and get around while allowing the city to save significant amounts of money.

This sunny Spanish city is now technologically advanced due to the SmartSantander project, funded by the European Union, that started when a professor at the University of Cantabria installed sensors in the city’s downtown area to have a better management of the limited parking spaces.

People that are responsible for the smart city implementation in Santander are concerned about what to share and what should remain private. Security is a big issue when it comes to public

sharable information. They want to create a cooperative relationship between the people and the city government in order for the citizens to embrace technology and not fear it.

2.2.3 ARRAY OF THINGS

The Array of Things is an initiative of an urban sensing project. It will take place in the city of Chicago, in the United States of America, and it consists of a network of interactive and modular sensor boxes installed around the city to collect real-time data on the environment, infrastructure and activity for public use of that information. They make a great analogy of it being a fitness tracker for the city.[16]

Its creators claim that this data will help make Chicago a truly “smart city”, allowing it to operate more efficiently as well as save on unnecessary costs and address potential urban problems. All the data will be published openly without any charge in order to motivate developers to build innovative applications that use the city’s real-time data.

2.3 IOT PROTOCOLS

The internet of Things is widely considered the next internet evolution and many companies and universities are in the race for the establishment of its standards.

Setting these standards has become so complex that has triggered comparisons to other technological competitions of the past, for example VHS and Betamax. Many do not believe that devices can seamlessly connect without this battle being resolved once and for all. While there is not one standard to rule them all, we will see a lot of debate and competing parties.

2.3.1 HTTP

HTTP is the foundation of data communication for the World Wide Web. The development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C).

HTTP is a protocol that follows the client-server model. The client sends an HTTP request to the server and the server performs certain actions, on behalf of the client that requested them, and returns a response message to the client. The response code is an indicative of the completion of the desired task and the response may also contain some requested content in the message body.

It is designed to enable communications between clients and servers and it is commonly used when it comes to the Internet of Things world. This happens because of the very common Representational State Transfer (REST) architectures in which systems have a central server that answers clients’ requests, this way making possible for the processing to be held server side instead of on the end-user devices reducing the load and speeding applications. These REST services are based on HTTP and are widely used in order to communicate between remote devices.[17]

It is not hard to imagine a use case where a sensor detects some kind of event and communicates it to some sort of message broker. In this implementation, for example, the sensor can be the client and the broker the server. When the event is triggered, the sensor sends an HTTP POST request to a certain address signalling that said event occurred.

2.3.2 CoAP

The Constrained Application Protocol's specification allows devices to communicate over the internet and is objectively targeted for low power sensors that require remote supervision, making it a very useful protocol for Smart Cities and Internet of Things scenarios.[18]

CoAP was designed to easily translate to HTTP for an easier integration with the web and, at the same time, meeting requirements like multicast support and simplicity. These requirements are of an extreme importance for the IoT because the devices that are going to use them often have very scarce resources than regular internet devices. This protocol is able to work on most devices that support UDP.[19]

When it comes to developers, CoAP feels very familiar. To obtain a value from a sensor is not that much different from what they are used to with HTTP, using web APIs.

Additionally, there is a protocol extension that enables CoAP clients to observe resources, retrieve representations of the said resource and keep it updated over a period of time.

In 2014, CoAP was analysed and compared to HTTP [20] and the results indicate that a wide variety of situations exist where the use of CoAP is more cost-efficient than HTTP.

It was concluded that one special case where CoAP is especially beneficial is when the smart objects are kept asleep in between communications instead of scenarios where they need to be active for most of the time.

CoAP is also friendly in the volume of data transferred in communications, when compared to HTTP, because "the small overhead of the protocol and its reliance on the UDP enable a manifold reduction in the transferred data volume." [20]

2.3.3 MQTT

MQ Telemetry Transport is a publish-subscribe lightweight messaging protocol that runs on top of the TCP/IP protocol. The "MQ" in "MQTT" comes from IBM's MQ message queuing products but the queuing itself is not a required standard feature. Its characteristics make it an ideal choice for constrained environments implementations like machine to machine communications and IoT contexts where small footprints are a must.[21]

MQTT uses a publish/subscribe message pattern that enables one-to-many message distribution and the decoupling of applications.

It supports three types of quality of service control for the message delivery that are: "At most once", where message loss can occur but the receiver will never receive the same message twice; "At least once", where it's ensured that messages will arrive but duplicates may occur; "Exactly once", where messages arrive to the destination only exactly once.

The protocol also supports a mechanism to notify interested parties when irregular disconnections happen.

2.3.4 LwM2M

Lightweight M2M is a protocol defined by the Open Mobile Alliance with the purpose of managing M2M or IoT devices. It defines the application layer communication protocol between a LwM2M server and client. As happens with other standards, the target devices are the ones that are resource constrained because it makes use of a light protocol and it strives for an efficient resource data model and it is frequently used with CoAP.

LwM2M provides device management functionalities, transfers service data to devices and is extensible to meet other application requirements.[8]

However, a LwM2M server only recognises attributes that are in its object specification file. Every resource that a developer wishes to use has to be present in the LwM2M object specification because it is the main way for servers and clients to discover information and what that data pertains to.

2.3.5 XMPP

The XMPP is an XML-based protocol that was originally designed for instant messaging and online presence detection.[22] There is an experimental extension that specifies how Things can be installed and safely discovered being, therefore, connected into networks of Things.

The installation of enormous amounts of Things into public networks needs to be simple yet secure to prevent hacking or hijacking.

A specification exists that specifies a network architecture based on XMPP that provides a way to install, configure, find and connect Things together. It also provides information on how each individual step in can be performed aiming at having no manual configuration.

There are many use cases for the usage of this technology in an IoT scenario but the main ones are production, installation, finding an XMPP server and connecting to one. The production is the phase of assigning responsibility for the parameters on the creation of the Thing in the network. Installation is the part of the process where a Thing might require extra values that could not be set in the production environment. Any manual configuration should be avoided. But manual entry of parameter might allow for Things to use local resources that cannot be found in the production phase. Finding an XMPP server is almost self explanatory and it is the attempt of the Thing finding an XMPP server in its local surrounding by several methods like DHCP. The connection to an XMPP server happens when it has already been found. If there are multiple servers available, the client is free to choose the one that best suits its purposes.

As this is an experimental use of the XMPP protocol, some of the security features are still to be decided and implemented and the use of this specification for a production environment is not advised by its developers.

2.3.6 OVERVIEW AND COMPARISON

While HTTP is commonly used in web scenarios, protocols like CoAP and MQTT were designed to work in more constrained environments where the processing power of the device is much lower and the data volume of communications matter.

MQTT features three Quality of Service policies to avoid packet loss, which can be a very important feature for real-world reliable Smart City applications.

CoAP has several similarities to HTTP, being built with a RESTful approach. It makes use of the same verbs and response status code's are very similar and it features an observe mechanism that will be discussed meticulously further.

LwM2M is a standard that is often found hand-in-hand with CoAP and It provides M2M platforms a simple and efficient device resource model. The advantage of the CoAP implementation of this standard is that it enables the observation of certain clients' resources from the server eliminating the need of querying them for changes.

For this specific point in time, XMPP is not a recommended solution for any platform as it is still in very experimental conditions. However, the community is starting to see some potential in the use of this protocol for IoT scenarios.

2.4 INTEROPERABILITY

A true smart city can only be achieved if the different parts of the solution reach a state of interoperability. By this it is meant to explain that the different parts of the architecture need to be able to understand and communicate with one another.

There is the need of a specification that enables the true intercommunication between these IoT devices. This necessity comes from the fact that there are countless manufacturers of devices, with them being sensors, actual devices where consumers can run software or any other kind of IoT meant device. Whichever category the device may fall in, by coming from different manufacturers they are bound to work with different mechanisms and standards. If this happens and they cannot understand each other, true interoperability is not attainable.

Every manufacturer could just agree on a specification and use it exclusively. However this does not happen because different companies disagree on what they think that the best specification is. Also, the big manufacturers are always trying to profit, as it is their main goal at the end of the day, so they try to push their own proprietary specifications in hope that it will be adopted by the IoT community some day. If they're successful, they are in control of the market and can explore new sources of income.

The main goal of this dissertation is to provide an alternative to the traditional exclusive adoption of one technology and get to a point where the platform is compatible with more than one standard in order to achieve true interoperability between them.

It is being made a true effort to push Smart Cities scenarios and deploy them but the companies that are the most persistent with this approach want to profit as much as they can. Knowing this, they are trying to control IoT panorama with their ecosystem. In an example we can see that Google is working on smart watches[23] and smart cars[24] that run some modified version of Android and work with other Android running devices. Meanwhile, Apple is on the other end doing the same thing with

their iOS platform. So, by implementing these features using their mobile operating systems, they are trying to buy costumers into the Google/Apple ecosystems. They provide these IoT/Smart services and devices but they are all dependant and only compatible with the devices in their ecosystem.

Components and standards that drive the IoT and M2M communication will be analysed and discussed below.

2.4.1 MESHBLU

Meshblu enables machine-to-machine instant messaging. It provides an API that is available on HTTP REST, Web Sockets via remote procedure calls, MQTT and CoAP. The main focus is to seamlessly bridge all the protocols so that devices that support, for example, CoAP can communicate with devices that support only MQTT.[25]

Each registered device is assigned a UUID and secret tokens so that they are used as credentials to authenticate with Meshblu and maintain the device in the device history.

With Meshblu, we are allowed to discover and query devices and send messages to one or several of them as well as subscribing to messages and sensor activities.

2.4.2 PONTE BY ECLIPSE

Ponte allows the developers to receive and publish data using either HTTP, MQTT or CoAP. It even enables sending and receiving the said data in different formats. Devices can get real-time updates thanks to MQTT and CoAP subscribe and observe methods, respectively. After that, the users can also get near real-time notifications due to the implementation of MQTT-over-Websockets.[26]

It also aims to support multiple data formats including Javascript Object Notation (JSON), MsgPack and XML.

Like Meshblu, Ponte also proceeds to authenticate the Thing that is talking to him. So, there's no need for the developers to prepare custom authentication.

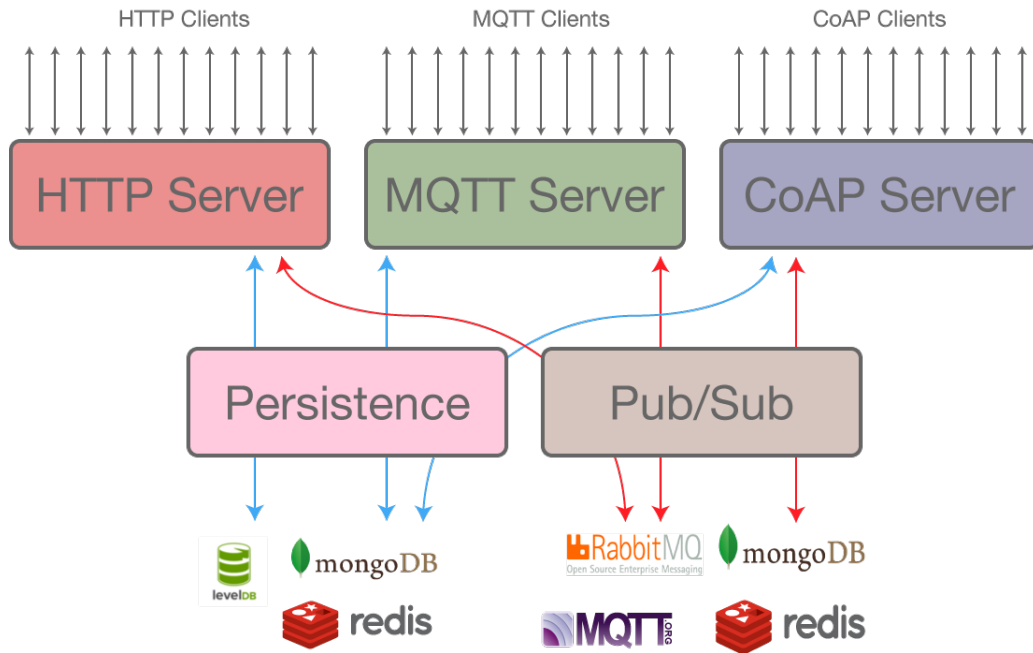


Figure 2.2: Ponte’s wide range of protocols support.

As the diagram explains, Ponte is able to bridge the gap between developers and Things. Using this platform, programmers that are used to build REST-based applications are now closer to the Internet of Things protocols while the core publish/subscribe methods that are more appropriate for IoT scenarios continue to work extremely well side by side. The current implementation of Ponte is built on top of the Node.js framework because it provides a fast event loop.

2.4.3 HYPERCAT

Hypercat is a consortium and standard that aims to push a secure and truly interoperable Internet of Things. The specifications intends to enable IoT clients to discover what resources are available in an IoT server. Its implementation is tightly related to common web standards like Hypertext Transfer Protocol Secure (HTTPS), REST and JSON. It’s described as “the most that forty companies could agree on”.[27]

These forty companies began to look for similarities between the architectures of the platforms that we’re implemented already.

First, they started to look at the lower level interfaces, which they found out to be very different from one case to another. It was quickly realised that trying to achieve the desired interoperability at this low-level specification would be just too hard due to the facts that there are many different protocols used to connect Things.

Instead, the focus went “up”. To connect with client applications, these platforms used a pretty homogeneous Web standards like HTTPS, REST and JSON which are widely used in the majority of information systems today. This seemed a lot more achievable than the first attempt. The main goal now was to make it possible for an applications built for an information source, to also work with another information source.

Now that the common interfaces for the applications were found, other problems rose:

- The way that information is displayed and organised is different from source to source. The data is shared in a way that makes sense for the data source at hand but not necessarily for everyone else. An example is that to find a temperature reading from different data sources can be different according to the purpose of the information. A building-management service would display it on the endpoint ‘/country/customer/site/building/temperature’, while a connected home service would organise it as ‘/customer/home/devicetype/device/temperature’;

- The “semantics” vary from source to source. The same thing is not always called by the same name in different data sources. For example, a european company would name their temperature resource “temperature_celsius”, while an american company would name it “temperature_fahrenheit”. Again, there is not a standard way to provide these attributes.

The current solution to overcome these obstacles is for the developer to read the documentation, learn how the data is organised and build software according to that implementation of the data source. This is not how the Internet of Things idealised the interconnection of the devices as it requires a developer to build software for each different source of data and this is not scalable. As the number of apps grows with the number of data hubs, the interaction between apps and data hubs explodes creating the need to develop different software for each of these interactions. It is clear that this is not feasible.

The solution would be to get every API to be organised in the same way and get everyone to use the same semantics. This was considered to be an unviable approach because there are reasons for these differences.

Knowing this, the Hypercat consortium quickly decided that the way to go was to create a way for machines to automatically solve the problem for themselves, without humans, embracing the differences between data organisation and semantics.

The hypercat standard aims to implement a way for an app to discover Resources, on the data hub, that it can understand. “If it understands only temperatures in degrees Centigrade, it needs a way to find such on a hub.”

As the goal was to approach the problem in such a way that it would be applicable to any modern web interface, built on the same web standards. By doing it this way, it is facilitating the adoption of the standard by the data source and app developers.

From these principles, Hypercat was born.

It is described as a “very thin layer which allows apps to either explore what is available on a hub, or search for particular types of resource outright”.

2.4.4 FIWARE’S ENABLERS

The most interesting and pivotal FIWARE enablers for the Citibrain consortium are the three specified below.

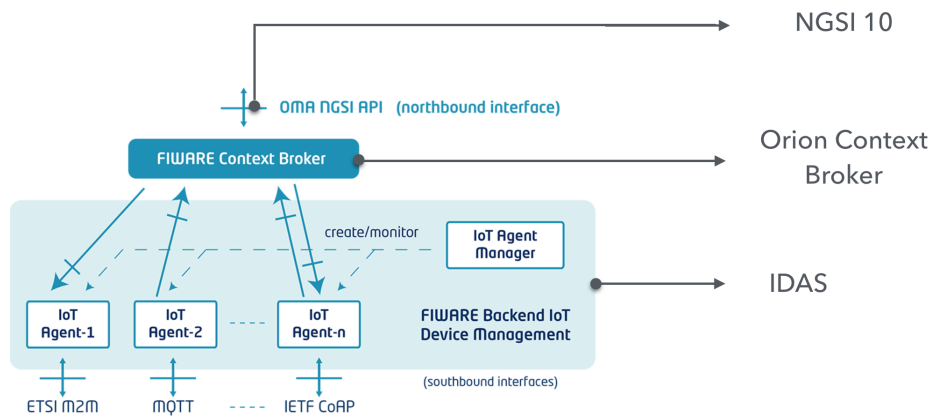


Figure 2.3: FIWARE enablers for IoT platforms

From the bottom up, IDAS could help us support various protocols like HTTP, CoAP and MQTT. The Orion Context Broker would provide us a smooth transition into delivering our data according to the FIWARE NGSI 10 specification and, also, integrating it with IDAS should be a pretty seamless process as the two of them are often bundled together.

2.4.4.1 IDAS

The IDAS component handles all the backend device management for the platforms and according to FIWARE, it is needed if there is the necessity of connecting devices/gateways into FIWARE-based ecosystems. Its main focus is to translate IoT-specific protocols into the NGSI context information standard.

IDAS provides a smooth way of connecting regular IoT devices to platforms that use FIWARE's generic enablers, more specifically the Orion Context Broker. On a very brief note, the goal of this component is to get messages from external devices and transform their data into something that the broker is able to understand and vice-versa.

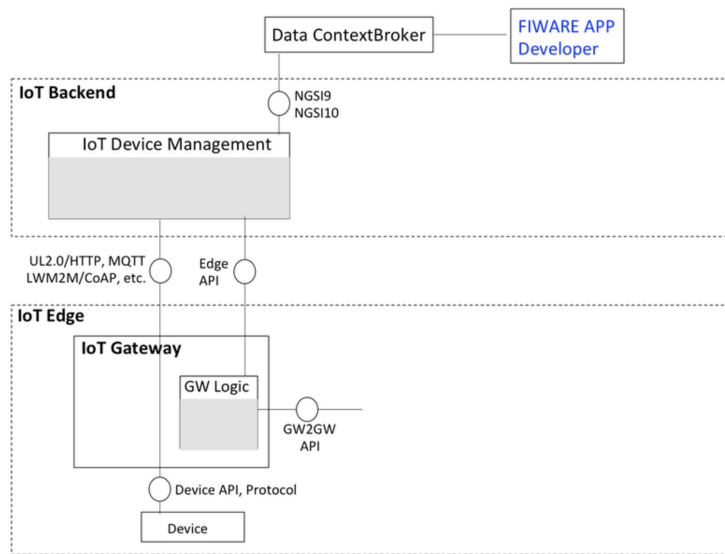


Figure 2.4: Fundamental use-case for the IDAS component

Analysing the image above, IDAS corresponds to the “IoT Device Management” component. It receives data from devices and gateways through various IoT adopted protocols and bridges the communication to the context broker by providing the device information in an NGSI specification for the broker to “understand”.

2.4.4.2 ORION CONTEXT BROKER

Orion is an implementation of a Publish/Subscribe broker. It allows several operations such as registering context producer devices, update, subscribe and query context information. This component stores the context information and the queries are based on this data.

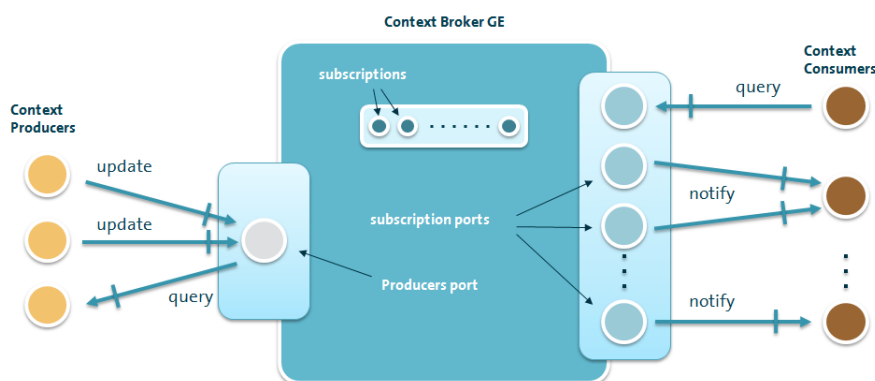


Figure 2.5: An example architecture diagram for the use of the Orion Context Broker

A context broker such as Orion is very useful to mediate the context between context producer and context consumer devices. It saves and transforms the received data from, for example, sensors and delivers it to the consumers, such as, web applications that use the context data to perform certain operations.

2.4.4.3 NGSI 10

NGSI is an open RESTful API specification for exchanging context information. This specification was created by FIWARE in order to make possible the interaction between their architecture and other endpoints that want to communicate with its ecosystem.

The three main interaction types it supports are:

- One-time queries for context information;
- Subscriptions for context information updates;
- Unsolicited updates.

The main goal of this API is to ensure interoperability with the FIWARE enablers that expose NGSI interfaces.

It enables different actors in the environment to provide/consume context information and discover context entities.

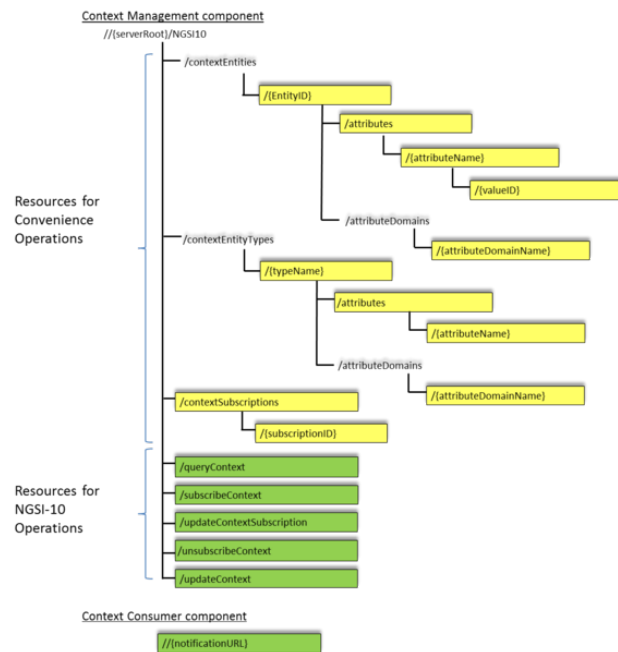


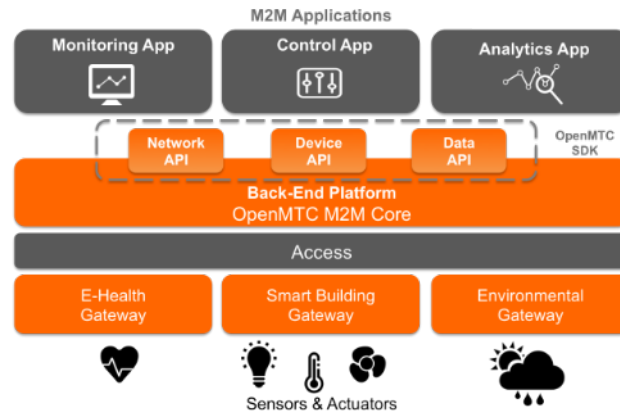
Figure 2.6: An NGSI compliant APIs resource tree

The main interaction types are supported exclusively via POST methods. These are listed above as the green part of the tree.

The yellow part corresponds to the convenience operation resources that are available through more than just POST HTTP methods, typically supporting GETs, PUTs and DELETEs also. These operations support a subset of the functionality of the corresponding NGSI operations but they provide a simpler navigation and more straightforward access.

2.4.5 OPENMTC

The Open Machine Type Communications platform implements a machine-to-machine middleware that aligns with international M2M standards like oneM2M, OMA LwM2M, ETSI M2M and Third Generation Partnership Project (3GPP). Developers use OpenMTC to interconnect various sensors and actuators from different vertical domains. It is a cloud-enabled and open platform that forwards data to the applications and enables event-based control to devices.[28]



It promises to:

- Optimize the network by integrating 3GPP elements in order to get connectivity status without having a constant keep-alive channel;
- Scale because it can be deployed in low resource and also Linux and Android compatible devices while the backend platform can be deployed on cloud infrastructure allowing for a smooth horizontal scaling;
- Converge M2M with Human-to-Machine (H2M) by integrating the Open XML Paper Specification (OpenXPS) which is the Open XML Paper Specification format for documents that enables the transformation of the handled data into human-readable documents;
- Offer multi-transport protocols like HTTP, CoAP and Websockets to allow different domain-specific applications to interact with the platform.

2.4.6 OVERVIEW AND COMPARISON

In this section, several components that promote interoperability were displayed. However, most of them do not follow the same approach.

Comparing, for example, Hypercat to Meshblu is not appropriate because they do not have many similarities. Hypercat works at the northbound of platforms specifying discoverable APIs, thriving to enable resource discovery for devices using the available data, while Meshblu is a message broker that aims to make lower-end devices (sensors, mainly) interoperable in a way that the communication protocol that they use becomes irrelevant.

It is obvious that there is a long way to go until full M2M interoperability is achieved and it is very good to see a number of initiatives that are trying to solve different problems in this scenario.

Ponte aims to be a solution for receiving and publishing data using multiple protocols and supporting also a range of data formats.

FIWARE takes an ambitious approach and has the goal of providing different components and enablers for most of the necessities of an IoT platform backend.

CITIBRAIN

Citibrain's value proposal is to deliver a unified solution for Smart Cities, covering diverse vertical domains like parking, environment and traffic management while crossing information and adding intelligence to the multiple city life's domains.

This means that the architecture of the platform is going to have a clear separation between each vertical solution while still providing ways to infer certain behaviours by gathering and examining information from different origins.

3.1 GENERAL DESCRIPTION

On a very brief note, the platform works as follows:

1. Sensors send state data to gateways;
2. Gateways send messages to the Meshblu broker;
3. Broker authenticates sender and sends messages to their respective queues to be processed. Simultaneously, data is collected and analysed by the Complex Event Processing (CEP) and If-This-Than-That (IFTTT) modules and events are triggered, if necessary;
4. Messages are processed in their arrival order on their backend system;
5. Data is made available in the backends' APIs and ready for analysis on the web portals.

This provides for a unified, centralised system that aggregates data from different vertical applications. Crossing this data is, the company believes, an advantage to better understand the necessities of a city and provide the needed help as quickly as possible.

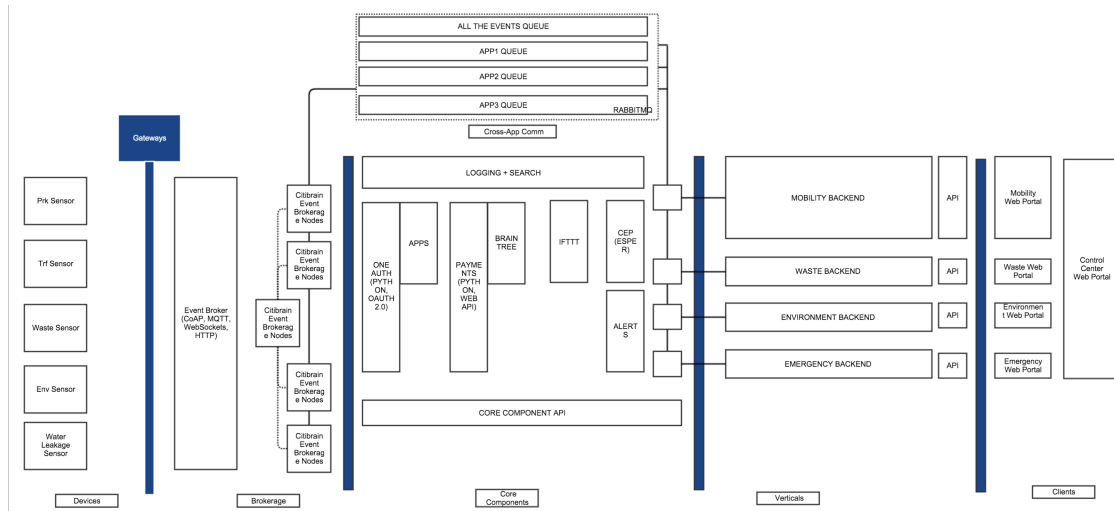


Figure 3.1: Citibrain's architecture overview

3.1.1 DEVICES

At this moment, the Citibrain platform is focused on four main areas that are waste, environment, traffic and parking. In order to gather information about these four scenarios, the basic component that the platform needs are sensors. The types of sensors that exist at the moment are:

- Parking sensors that detect whether a parking spot is occupied or free;
- Traffic sensors which monitor the vehicle congestion in strategic places of the cities;
- Waste sensors in order to capture the state of trash bins to conclude if they're full or still have capacity for more;
- Environment sensors to monitor temperature, humidity, pollution and other statistics about the city;
- Water metering sensors;

All these sensors communicate with their respective gateways and they, on the other hand, communicate with the rest of the platform on the sensor's behalf.

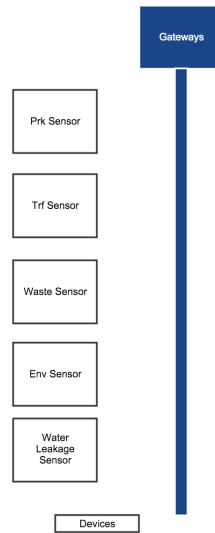


Figure 3.2: Citibrain’s range of sensors communicating with a Gateway

Afterwards, the gateway machines message the rest of the platform through HTTP messages. This ignores the physical location of the servers that will process the messages. An event in Aveiro can be processed in a server in New York, where the platform may be deployed. Better yet, heading for the path of a decentralised solution, handling these messages using web protocols seems like a logical approach as it provides a certain transparency so that the physical location of the platform deployments stop influencing as much the overall workflow of the applications.

3.1.2 MESSAGE BROKER

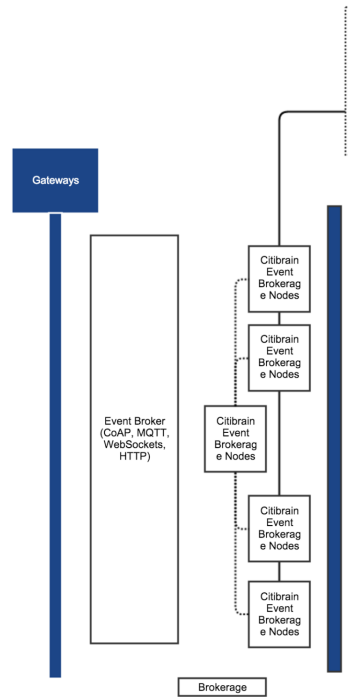


Figure 3.3: Citibrain's broker and brokerage nodes

In the second tier we have the message broker. Right now, our message broker is a machine-to-machine instant messaging platform for the internet of things called Meshblu¹.

Following Meshblu's description, they provide a "secure, cross-protocol scalable cloud-based system enabling communication between smart devices, sensors, cloud resources. . ." [25]

This broker helps us by creating an instant messaging network and API. The API is available through HTTP, WebSockets, MQTT and CoAP, which are widely used protocols in the Internet of Things use cases. Meshblu also bridges the gap between devices that use different protocols allowing for devices that support HTTP to communicate with devices that use CoAP.

Authentication of the sensors and gateways is also taken care of by the broker, which maintains a JSON description of the devices in the device directory.

When a device registers into the Meshblu server, it will be assigned a pair of authentication and identification parameters, which are "meshblu_auth_token" and "meshblu_auth_uuid", respectively. These fields are necessary in every request to the broker, after the registration. If the device performing the request cannot be identified and authenticated, no request will be processed. This means that at the registration, the devices will be responsible for storing their credentials for future interactions.

Additionally, Meshblu allows for discovery and query of devices as well as the subscription to messages being sent to or from devices (very useful for tracking sensor activities).

The broker is the component that receives the sensor updates from the gateways and then is in charge of authenticating who sent the message and filter them, according to their type (waste, environment. . .), to the brokerage nodes.

¹<https://meshblu.readme.io/>

Right now, this message filtering is done by scanning one field of the content received on the requests. A field called “devices” stores a unique identifier that corresponds to one of five areas:

- Parking sensor devices;
- Traffic sensor devices;
- Waste sensor devices;
- Air sensor devices;
- Water sensor devices.

Each one serves a specific vertical solution in the Citibrain platform. So, “devices”: “df5b2380-7a4d-11e4-bff6-ffd7eef4967c”, means that the device that triggered the message is a parking sensor and that this information belongs to the parking solution use-case.

3.1.3 CORE COMPONENTS

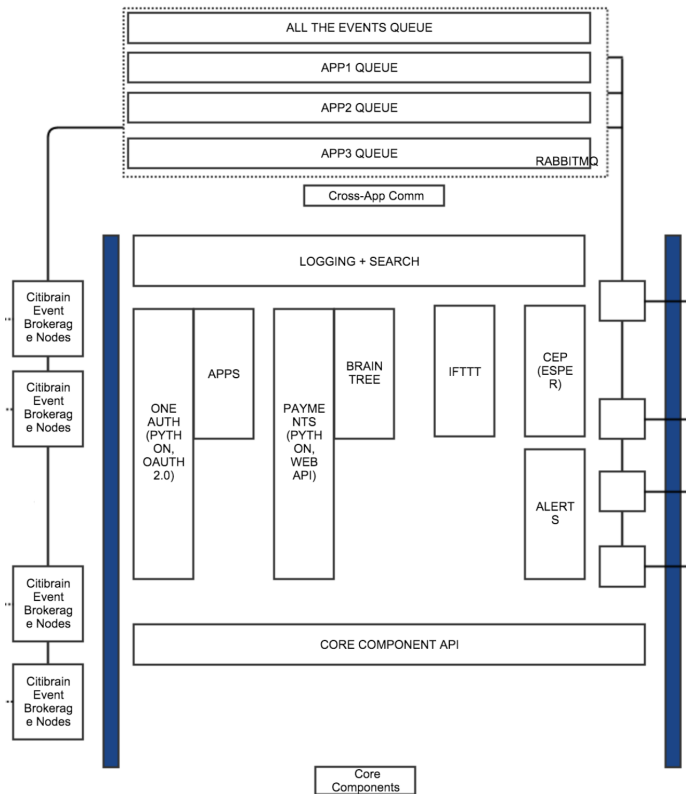


Figure 3.4: Citibrain’s core components

From the brokerage nodes, the events and messages are placed into their respective queue. Each queue represents a segment of the platform. That means that we have one queue for waste events, another one for parking, environment and traffic. Said message queues are very useful mainly to improve the overall scalability of the platform by helping with the load distribution and the asynchronous message processing.

These queues are powered by the RabbitMQ technology which implements the Advanced Message Queueing Protocol[29]. RabbitMQ allows the messages to arrive to their destination in the correct order to maintain the information consistency and provides message delivery guarantees like at-most-once and at-least-once.

This is also the stage where the logging and system-wide user authentication is done.

Ubiwhere is also doing extensive research on our IFTTT feature, which is also being subject of a Master's degree dissertation that is being taken into development by Eduardo Duarte along with the University of Aveiro. This feature goes hand in hand with our CEP engine. This engine combines data, like sensors' events and messages, from multiple resources and infers patterns. These patterns are used to act and respond to the identified problems as quickly as possible.

3.1.4 BACKENDS & APIS

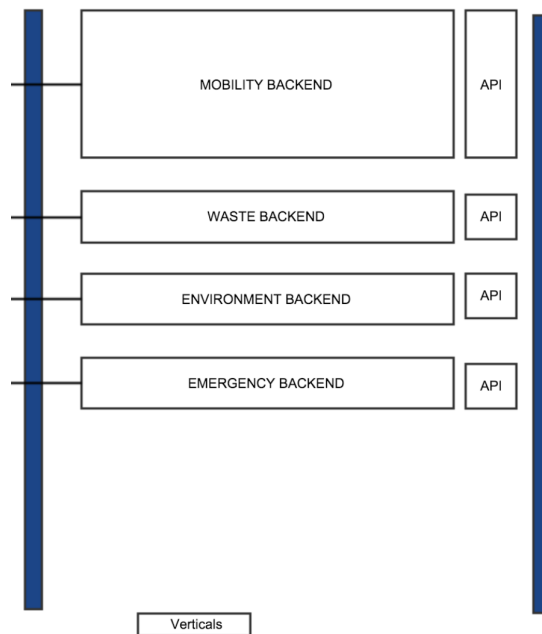


Figure 3.5: Citibrain's backends and APIs

This is where the verticals are differentiated. Each backend receives information from its respective queue and processes it.

The information is accessible from other applications through their APIs. They allow for the retrieval of asset and events informations as well as the registration and edition of assets.

3.1.5 CLIENT APPLICATIONS

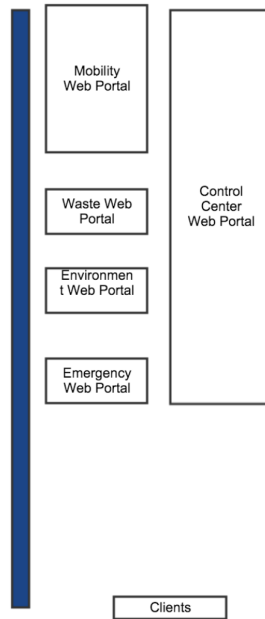


Figure 3.6: Citibrain's client applications

Finally, we have web portals where we can manage and view the information regarding the different vertical solutions. We've also built a unified control center that is a centralised web portal to control the different settings of the overall platform.

The Citibrain APIs make it possible to build applications that make use of this real-time information provided by the users.

An example of it is an smart parking application developed in Ubiwhere that lets the user know which parking spots are available and which are not.

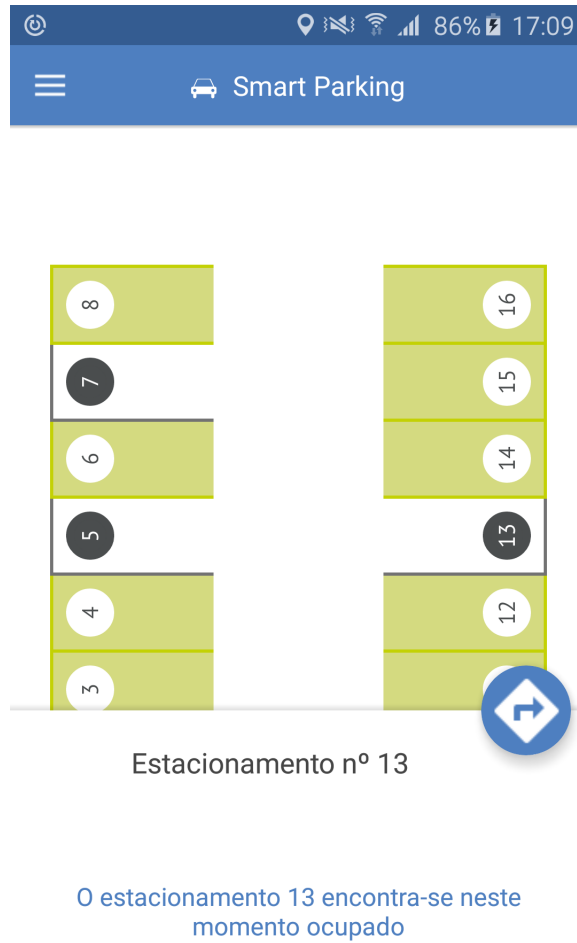


Figure 3.7: Citibrain's mobility API serving parking information to app

When selecting a parking spot, the app will tell the user if it is available or occupied and provide directions to its location.

The above application is just an example and a use case for the available APIs.

3.2 HYPERCAT INTEROPERABILITY

3.2.1 REASONS TO SUPPORT HYPERCAT

Hypercat was used in order to build a wrapper to a set of APIs that were created to serve the Citibrain's services.

These APIs serve information about parking, traffic, environment and waste. They are used to provide services such as informing someone that the waste bins are full or giving the municipalities information about traffic in their cities.

So, these services can be enhanced if they provide a discoverable way to share the information. The intention is that this information enables the communication in a standard way. As it was discussed earlier, there is no standard way. The idea here is to provide an alternative and support multiple standards.

3.2.2 SOLUTION

With that in mind, the solution found was to implement a Hypercat server that serves the information requested from the APIs in the Hypercat specification and performs the Create, Read, Update and Delete (CRUD) operations of the desired items.

The Hypercat server gets the request and, depending on the desired operation, serializes the data received into or from a compliant Hypercat catalogue.



Figure 3.8: Citibrain data translation into/from the hypercat specification

Essentially there are two types of data that we want to manipulate: Events and Assets.

Assets are mostly sensors and Events are notifications of the state change of an asset. The desired operations for this Hypercat server are to list event and asset information and also register and update

assets. There is no point in inserting and editing events because they are triggered by the assets' state change.

By using this implementation, we are able to support architectures using the Hypercat standard while not taking the risk of migrating all the interaction with our system to it and interfering with the work done until now, which uses the proprietary data model.

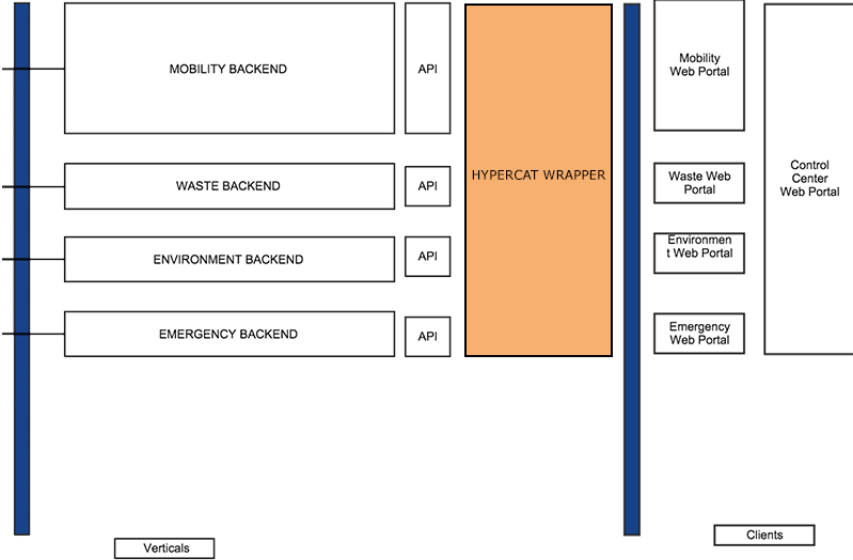


Figure 3.9: Citibrain architecture diagram after the hypercat wrapper implementation

3.2.3 IMPLEMENTATION AND TESTS

cat		Show/Hide	List Operations	Expand Operations	Raw
GET	/cat/				Gets Catalogue Info in Hypercat format
GET	/cat/parking/events/				Gets Parking Events in Hypercat format
GET	/cat/parking/events/{href}/				Gets Parking Event in Hypercat format
GET	/cat/parking/assets/				Gets Parking Assets in Hypercat format
GET	/cat/parking/assets/{href}/				Gets Parking Asset in Hypercat format
GET	/cat/waste/events/				Gets Waste Events in Hypercat format
GET	/cat/waste/events/{href}/				Gets Waste Event in Hypercat format
GET	/cat/waste/assets/				Gets Waste Assets in Hypercat format
GET	/cat/waste/assets/{href}/				Gets Waste Asset in Hypercat format
GET	/cat/environment/events/				Gets Environment Events in Hypercat format
GET	/cat/environment/events/{href}/				Gets Environment Event in Hypercat format
GET	/cat/environment/assets/				Gets Environment Assets in Hypercat format
GET	/cat/environment/assets/{href}/				Gets Environment Asset in Hypercat format
GET	/cat/traffic/events/				Gets Traffic Events in Hypercat format
GET	/cat/traffic/events/{href}/				Gets Traffic Event in Hypercat format
GET	/cat/traffic/assets/				Gets Traffic Assets in Hypercat format
GET	/cat/traffic/assets/{href}/				Gets Traffic Asset in Hypercat format

Figure 3.10: Citibrain’s Hypercat API GET methods

The above image reflects our APIs GET methods. These cover all the Citibrain’s vertical solutions for operations involving events and assets. Asset related methods also support POST and PUT method which correspond to asset creation and update, respectively.

Every Hypercat server is required to have a “/cat” endpoint where the API is described. This is mandatory in the implementation and its goal is to make the rest of the endpoints discoverable by other machines.

With this in mind, it is likely that the response contains references to the other API Uniform Resource Locator (URL)s so that they are quickly accessible. It also features a human readable description in English and other languages are also supported but not mandatory.

The JSON serialized data in Appendix A represents a response from the “/cat” HTTP method in the Citibrain API. A snippet can be found below.

```
{
  "catalogue-metadata": [
    {
      "val": "application/vnd.hypercat.catalogue+json",
      "rel": "urn:X-hypercat:rels:isContentType"
    },
  ],
}
```

```

{
  "val": "Citibrain Hypercat Catalogue",
  "rel": "urn:X-hypercat:rels:hasDescription:en"
}
],
"items": [
  {
    "href": "/cat/parking/assets/",
    "item-metadata": [
      {
        "val": "application/parking_assets",
        "rel": "urn:X-hypercat:rels:isContentType"
      },
      {
        "val": "Parking Assets",
        "rel": "urn:X-hypercat:rels:hasDescription:en"
      }
    ]
  },
  {
    "href": "/cat/parking/events/",
    "item-metadata": [
      {
        "val": "application/parking_events",
        "rel": "urn:X-hypercat:rels:isContentType"
      },
      {
        "val": "Parking Events",
        "rel": "urn:X-hypercat:rels:hasDescription:en"
      }
    ]
  }
]
}

```

The "items" array is a collection of all the endpoints available in this server that obey the Hypercat specification. Every "items" object has two mandatory fields:

- "href" : The Uniform Resource Identifier (URI) to the desired Hypercat endpoint. It refers to a further catalogue;
- "item-metadata" : Metadata of the resource which provides a human readable description and a content type to be recognised by other machines.

When there is a request to one of the items' href URLs, it will trigger a request to the actual Citibrain API. After getting the response that is serialized in a proprietary fashion, it is transformed into a Hypercat format response so that the data provided by the mobility backend is interoperable with Hypercat supporting devices.

As an example, accessing the "/cat/parking/assets" endpoint returns the Hypercat formatted JSON response in Appendix B. A snippet can be found below.

```
{
  "items": [
    {
      "i-object-metadata": [
        {
          "val": "application/asset",
          "rel": "urn:X-hypercat:rels:isContentType"
        },
        {
          "val": "30-0001",
          "rel": "urn:X-hypercat:rels:hasDescription:en"
        },
        {
          "val": "",
          "rel": "additional_fields"
        },
        {
          "val": "30-0001",
          "rel": "uuid"
        },
        {
          "val": "True",
          "rel": "is_active"
        }
      ]
    }
  ]
}
```

It is noticeable that the responses show a pattern. All the Hypercat messages come in the form of catalogues of information. By showing consistently the same message structure, it becomes easier for machines to “predict” what to expect and to handle the information in the best way possible.

Again, there's a content type field specifying what the catalogue is actually characterising, in this case an asset, that is a general object on all the verticals. The human readable description shows what asset is being listed (30-0001). Other dictionaries in the object's metadata correspond to the real

attributes of the resource being specified by the catalogue like, for example, the UUID, location (split in latitude and longitude), type, among others.

To better illustrate the process and the data transformation, an example is given where a parking request is performed to the Hypercat server.



Figure 3.11: Parking events request

Figure 3.11 serves to demonstrate the serialisation of the Citibrain data into the required Hypercat specification. The original data undergoes quite a bit of processing, being because of this reason a rather slower operation. On top of that, there is an additional overhead of performing, in fact, two HTTP requests for each operation. The original one for the Hypercat API and the one performed by the actual server to one of the vertical’s APIs.

The Hypercat catalogue is actually very large in terms of content when compared to the original data. But it is the necessary price to pay for discoverability. The reason behind it is that the attributes need to be announced in order to become discoverable by other machines. In this format other machines supporting Hypercat know what to expect from the messages and have the ability to parse through the items metadata to, hopefully, be able to understand what that attribute actually means and how it is presented.

3.2.4 BENCHMARKS

To benchmark the Hypercat server, it was decided to perform a series of the same operation using nothing but the proprietary Citibrain API and then exactly the same operations were executed via the Hypercat interface.

It is to be reminded that every method of the Hypercat API requires a call to the Citibrain one. Knowing this, it expected that every Hypercat method takes a longer time interval for processing. Additionally, as the proprietary data is converted into the required Hypercat format, the transformation time also makes this request more expensive.

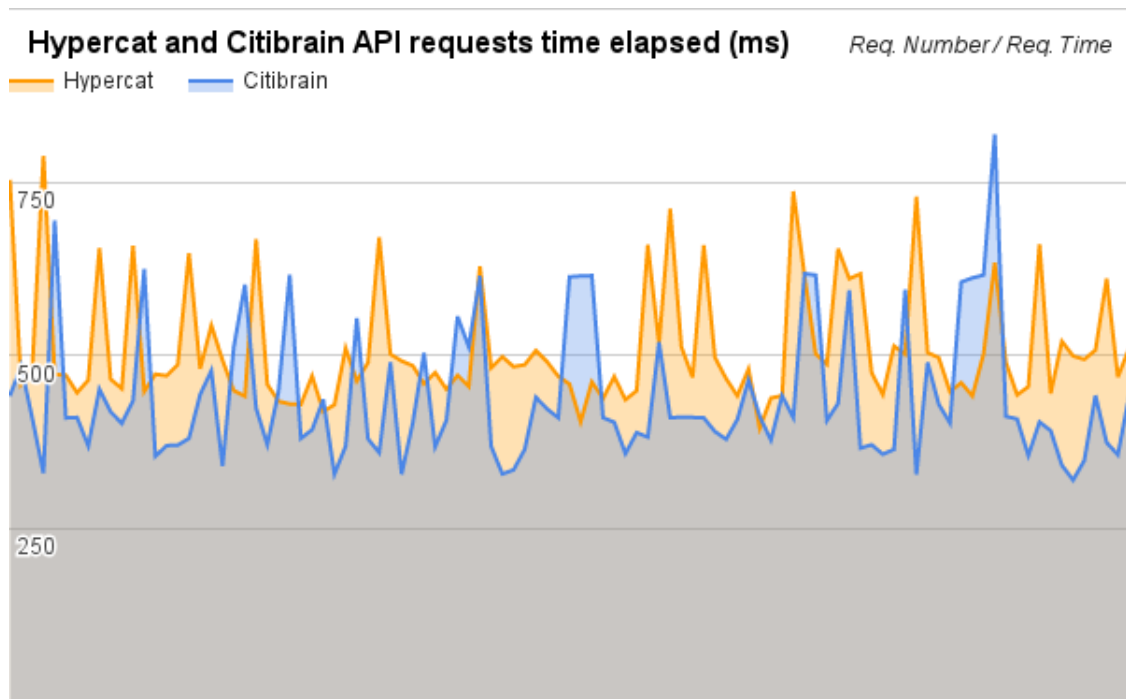


Figure 3.12: Time spent processing a Hypercat GET request in comparison to Citibrain

Figure 3.12 proves the reasoning above. The expected behaviour is displayed in the graph. It is visible that the majority of the Citibrain's API requests take less processing time than the Hypercat requests.

However, in this particular situation the Hypercat server was being executed on the machine's localhost. This means that a lot of the delay of the client's API call is not taken into account in these requests, as it would in a regularly deployed platform. This means that the Hypercat requests can be slower when the system is deployed in a regular server instead of a local machine where the actual request is also being executed.

In this case, the overhead percentage that the Hypercat requests have when compared to the regular Citibrain ones is roughly 14% on average.

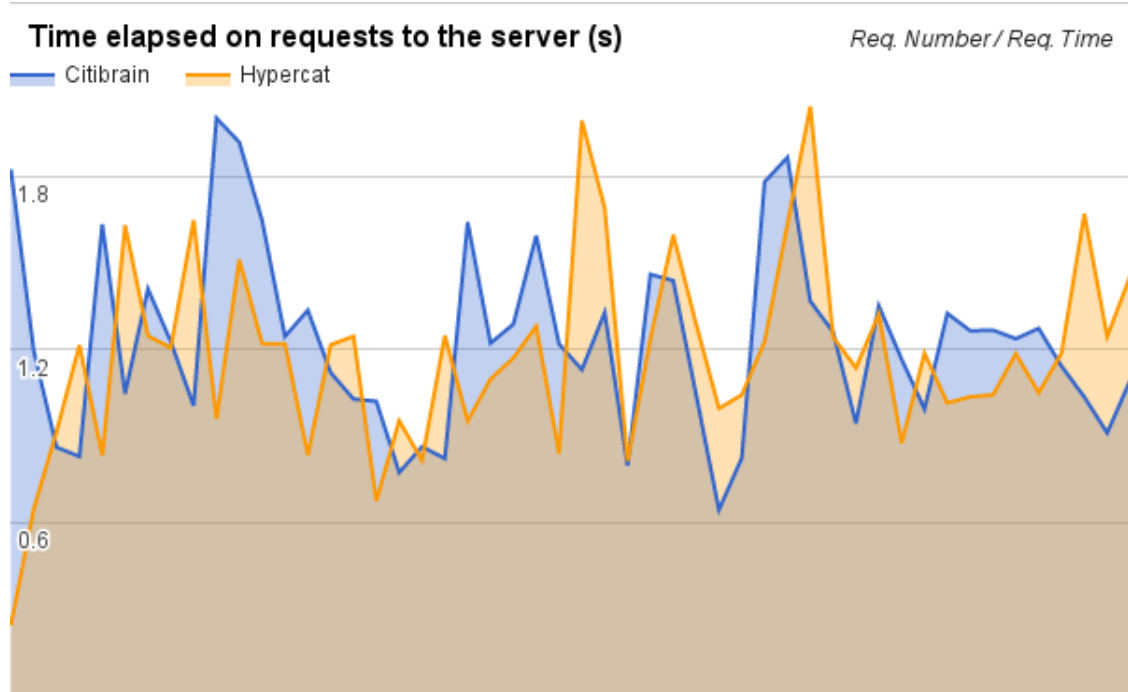


Figure 3.13: Time spent processing a Hypercat POST request in comparison to Citibrain

Above, it is shown what happens with HTTP POST requests. Requests made to the Hypercat server are roughly as expensive as the ones made directly to the Hypercat server, in terms of processing time.

Hypercat's requests peaks are slightly more expensive but the difference is not very significant.

3.3 FIWARE SUPPORT

3.3.1 FIWARE AT UBIWHERE

Ubiwhere is one of the early adopters of the FIWARE's standards in its Citibrain consortium and smart city solutions. By making our systems compatible and compliant with the FIWARE specifications we are enabling the interoperability of all devices that also use this European Union supported IoT enablers.

Taking by example Porto and Santander, two cities that are supporting and making their data available according to FIWARE's rules:

One is able to prepare applications and whole architectures to work in Porto, which is more convenient due to Ubiwhere's geographical situation, configure it, deploy it and test it there and, afterwards, take it to Santander and deploy it with zero effort. This happens because the data that is distributed in each city obeys the same rules and is propagated in the same way. Our backend service is developed once taking into account these directives and then just deploy it in any city that supports FIWARE.

In this case it was only being referred the high level information availability through public APIs. This information is listed using the NGSI specification but this is not the only enabler that FIWARE has to offer.

According to the organisation's website they provide a "rather simple yet powerful set of APIs that ease the development of Smart Applications in multiple vertical sectors."

Some of the most relevant "pieces of the puzzle" for Citibrain include the above mentioned NGSI 10 API specification that is used on the northbound of systems to disseminate the knowledge obtained by the whole backend, the IDAS that handles everything related to device management and the Orion context broker that bridges the gap between the devices and the northbound API, sitting right in the middle of IDAS and the NGSI 10 API.

As a proof of Ubiwhere's involvement in the dissemination of FIWARE's ideologies, the organisation itself published an article[13] where it details the project.

"As one of the first cities that joined the Open and Agile Smart Cities (OASC) initiative back in March this year, Porto has been a pioneer city adopting FIWARE standards with the support of Ubiwhere, a Portuguese company experienced in the development of middleware and platforms."OASC

"The Future Cities Project is a partnership between the University of Porto and the City Council aiming to create a Competence Centre for Future Cities in the city of Porto. Together, but also with the participation of the Citibrain joint-venture, the city of Porto and Ubiwhere have developed the interfaces bringing access to real-time, contextual environmental data from 75 fixed and mobile units (monitoring stations) located across the city. The data is augmented by scanners installed on the city Council's 200+ fleet of vehicles, creating a large-scale mobile scanner. External providers like the city's water supplier, transport data providers, social media data and business startup statistics are all plugged in to the platform to allow the city itself to guide you as you explore, travel, and work."OASC

"Following the steps taken by Porto, and again with the support of Ubiwhere, several other Portuguese cities (e.g. Águeda, Aveiro, São João da Madeira and Torres Vedras) are starting to provide real-time data on mobility/ transportation and environment."OASC

Ubiwhere is the main force thriving to standardise the IoT world and working towards a truly interoperable smart city scenario. This is being done by creating several testbeds in different cities in

order to test the detachment of the platform to the data we use, meaning that one platform should work seamlessly in different cities.

```
{
  "contextElement":{
    "attributes":[
      {
        "type":"coords",
        "name":"coordinates",
        "value":"41.1579, -8.58516",
        "metadatas":[
          {
            "type":"string",
            "name":"location",
            "value":"WGS84"
          }
        ]
      },
      {
        "type":"datetime",
        "name":"date",
        "value":"2016-04-08T07:15:18.000000Z"
      },
      {
        "type":"integer",
        "name":"distance",
        "value":"215"
      },
      {
        "type":"integer",
        "name":"hdop",
        "value":"10"
      },
      {
        "type":"float",
        "name":"latitude",
        "value":"41.1579"
      },
      {
        "type":"float",
        "name":"longitude",
        "value":"-8.58516"
      },
      {
        "type":"integer",
```

```

        "name": "movement",
        "value": "1"
    },
    {
        "type": "integer",
        "name": "speed",
        "value": "12"
    },
    {
        "type": "string",
        "name": "vehicle",
        "value": "a1bb8a923iw8jcellqk10xo9sjamsndj2j283sisjcem1qsa9xo"
    }
],
"type": "TrafficEvent",
"id": "a1bb8a923iw8jcellqk10xo9sjamsndj2j283sisjcem1qsa9xo",
"isPattern": "false"
},
"statusCode": {
    "code": "200",
    "reasonPhrase": "OK"
}
}

```

Above is an example of the data that the city of Porto is making available to developers. It is according to the NGSI specification and, as we can see, it translates into a traffic event. By this, the developers have information of what vehicle triggered this event, the speed that it was going at, its position, among other data.

But this is just an example of the resources that are public. An API like this is extremely useful to enable smart city application developers build their software and make the city a livable and friendly as possible for the citizens.

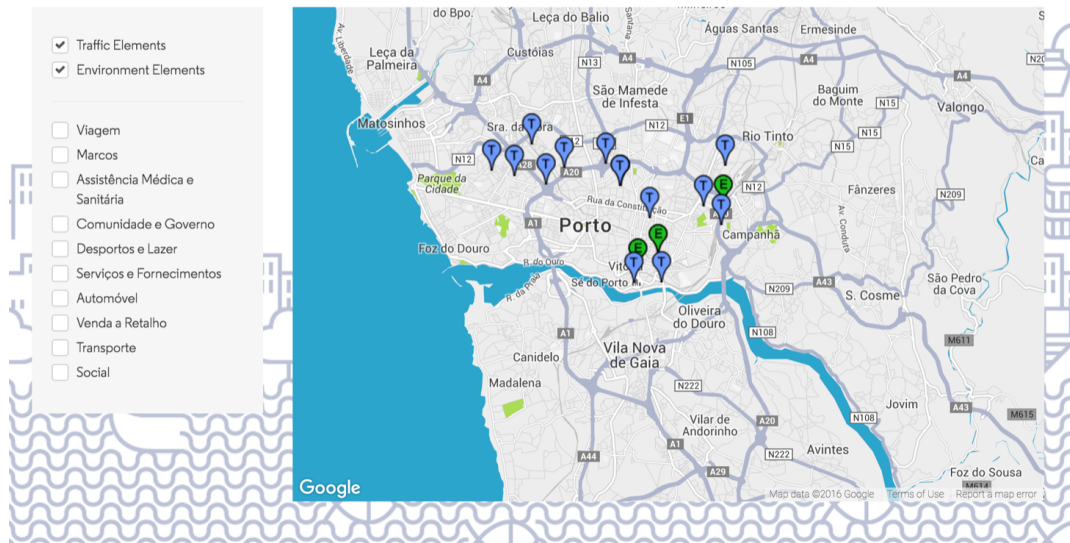


Figure 3.14: FIWARE Porto's demo

For demonstration purposes, a small web app was built using this data that Porto publicly publishing on the web. The app featured a map that showed the information of traffic and environmental elements. The user was able to see where in the city were these sensors and, by clicking on them, its information and measurements were displayed in real-time. It needs to be reminded that, as this app is working in the city of Porto, it can work anywhere that supports the NGSI API specification.

3.3.2 NGSI AT CITIBRAIN

In the Citibrain platform, we have every interest in supporting such an API specification. By doing so, it enabled the deployment of our solutions wherever there are already apps working with this standard. Not only this but installing Citibrain solutions in a city means that every app built around the FIWARE specification is able to be ported very easily into that city.

This portability means that if some developer is having a huge success with his smart city solution, he is able to scale his application quickly. If some municipality is having a huge success in saving water due to a system that takes the city's water data, that same system can be made available to their neighbours, their neighbour's neighbours and so on. This is the ultimate achievement and the true consequence that defines the concept of interoperability, and is what we, the developers and the citizens, hope to have one day. Obviously, this is not going to solve all the problems. It will only work with FIWARE compliant platforms but it is an initiative and at Ubiwhere, the goal is to future-proof the platform making it "talk and understand" every language we can.

By making the platform "talk" NGSI, we are already making our data available in three formats:

- Proprietary;
- Hypercat;
- NGSI.

And like so, the company is competing in three leagues. No one knows which "standards" are going to persist, so the best bet is to try and understand them all so the platform does not become

obsolete, no matter what the future looks like.

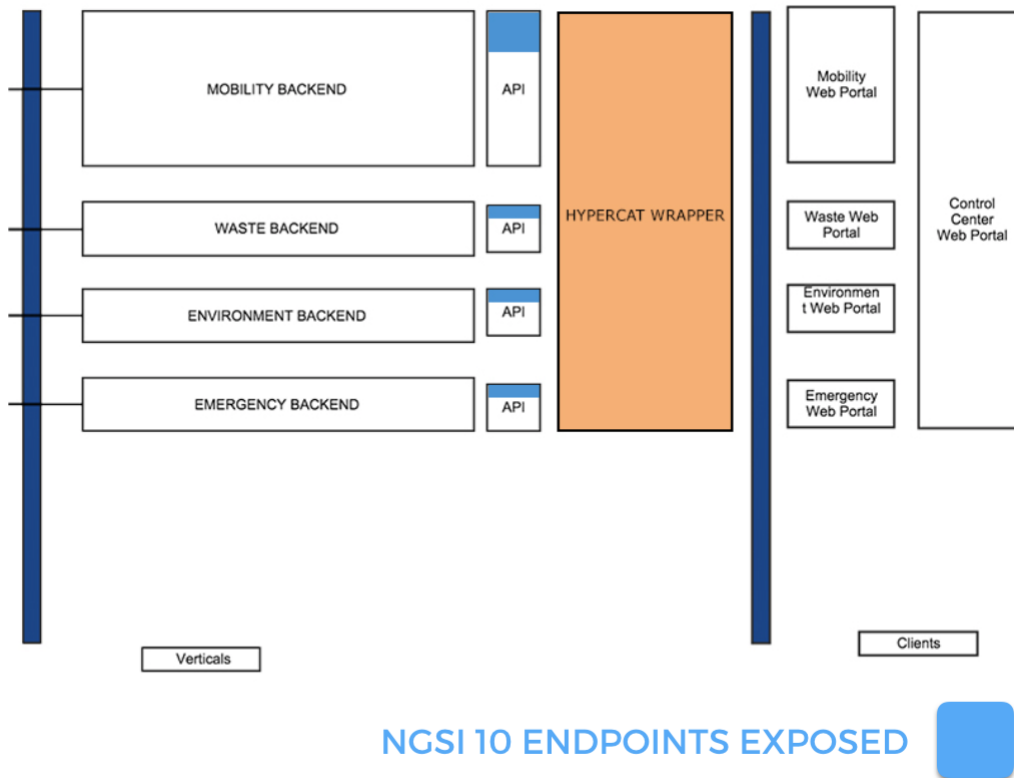


Figure 3.15: Vertical APIs exposing NGSI nodes

The figure shows a slightly different implementation from the Hypercat implementation to the NGSI one. This is because the Hypercat wrapper is literally a proxy server that receives Hypercat requests, serializes the information into Citibrain’s proprietary specification and interacts with the proprietary APIs, while the NGSI endpoints are a Django add-on that is used by the own APIs to expose different methods.

3.3.3 IDAS AT CITIBRAIN

IDAS is a rather interesting component. It enables the interaction with devices using different communication IoT protocols like HTTP, LwM2M and MQTT. That’s a good thing but the Meshblu broker that is being used in Citibrain already supports HTTP, MQTT and CoAP. So, maybe it could be useful using IDAS for adding the LwM2M device compatibility that we want to have.

Knowing this, some experiments with IDAS started being made. The official FIWARE code repository was used to perform these experiments. Two main repositories exist that implement the IDAS behaviour. One for the LwM2M support and the other one to offer compatibility for HTTP and MQTT devices. Having no interest, for now, on the HTTP or MQTT we went with the LwM2M version of the code base.

The description of this component (OMA Lightweight M2M IoT Agent):

“An Internet of Things Agent is a component that lets groups of devices send their data to and be managed from a FIWARE NGSI Context Broker using their own native protocols. This project provides the IoT Agent for the Lightweight M2M protocol, i.e. the bridge between OMA Lightweight M2M enabled devices and a NGSI Context Broker.”

In other words, it provides the possibility of LwM2M devices communicating with the FIWARE's context broker.

So, after studying these facts and having good knowledge of what IDAS actually is, it was decided to run a small experiment. It consisted in deploying an IDAS instance, creating a regular LwM2M client and trying to register the client into the device manager. For this it was used a completely independent library, not affiliated with FIWARE in any way, that gives us the ability of creating a LwM2M client and perform the supported operations according to the OMA specification.

```
[LWM2M-Client> create /100/1

Object:
-----
ObjectType: 100
ObjectId: 1
ObjectUri: /100/1
[LWM2M-Client> set /100/1 1 itlife

Object:
-----
ObjectType: 100
ObjectId: 1
ObjectUri: /100/1

Attributes:
  -> 1: itlife

[LWM2M-Client> connect localhost 60001 teste /

Connecting to the server. This may take a while.

LWM2M-Client> |
```

Figure 3.16: LwM2M client creation

Here, the client creation is demonstrated, using a node.js library. It is also defined the resource with the “/100/1” URI just to see what happened when this request got to the IDAS instance. The next step was to register the client into the server, meaning, IDAS.

The registration is done successfully into the device management system.

```
time=2016-03-10T13:55:22.958Z | lvl=DEBUG | corr=n/a | trans=n/a | op=LWM2MLib.Registration | msg=Handling registration request
time=2016-03-10T13:55:22.959Z | lvl=DEBUG | corr=n/a | trans=n/a | op=LWM2MLib.COAPUtils | msg=Checking for the existence of the following parameters [{"ep"}]
time=2016-03-10T13:55:22.962Z | lvl=DEBUG | corr=n/a | trans=n/a | op=LWM2MLib.Registration | msg=Storing the following device in the db:
{
  "name": "teste",
  "lifetime": "85671",
  "address": "127.0.0.1",
  "port": 62035,
  "creationDate": "2016-03-10T13:55:22.962Z"
}
time=2016-03-10T13:55:22.963Z | lvl=DEBUG | corr=n/a | trans=n/a | op=LWM2MLib.Registration | msg=Registered device [teste] with type [Device]
```

Figure 3.17: IDAS receiving a registration request successfully

However, it breaks when trying to send that same information to the Orion Context Broker because we didn't connect one to its northbound.

```
time=2016-03-10T13:56:38.386Z | lvl=ERROR | corr=n/a | trans=n/a | op=IoTAgentNGSI.NGSIService | msg=Connection error creating initial entity in the Context Broker: Error: connect ETIMEDOUT 192.168.56.101:1026
time=2016-03-10T13:56:38.387Z | lvl=DEBUG | corr=n/a | trans=n/a | op=LWM2MLib.Registration | msg=Registration request ended up in error [Error] with code [ETIMEDOUT]
```

Figure 3.18: IDAS reporting an error due to the fact that it can't find the Context Broker

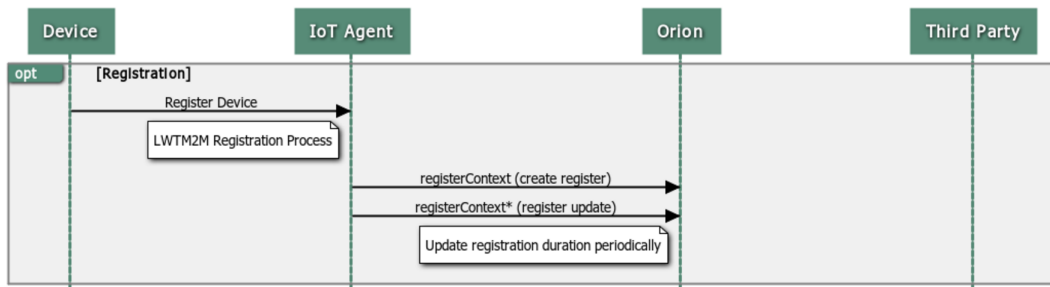


Figure 3.19: IDAS client registration flow

So, the conclusion was that the system really does not need IDAS at the southbound of Citibrain. The reasons for that are:

- Meshblu, the platform's broker, handles the device management;
- Citibrain's core architecture does not use the Orion Context Broker.

IDAS handles the IoT protocols and feeds the information in them in the NGSI format to the Orion Context Broker. Meshblu doesn't need the context information in the NGSI specification. As long as we can provide that same format in the northbound to external applications, the data format we use while communicating between different components of our architecture becomes transparent to the developers. In other words, Meshblu already supports HTTP and MQTT. The only advantage IDAS would bring the platform would be the LwM2M support but with the drawback of having to change our message broker, which would mean altering the core architecture quite a bit.

In the next chapter it will be explained how the LwM2M support was achieved on the platform.

3.3.4 IMPLEMENTATION AND TESTS

In the end, it was decided that the real need was for the platform to support FIWARE's NGSI in its northbound. This means that our platform will be usable by any application that's built with its basis on the FIWARE NGSI 10 API specification.

Integrating this feature in our APIs means one more interoperability point between Citibrain and smart city application developers.

For this purpose, new endpoints were exposed in the already existing APIs.

All the URLs available in the `{domain}/NGSI10/{...}` are obeying the NGSI 10 specification. It was not implemented in the same way as the Hypercat wrapper but rather like the original proprietary

```

{
  "id": 1,
  "asset": "3000",
  "container": 1,
  "created": "2016-05-19T14:30:50.423105Z",
  "modified": "2016-05-19T14:30:50.432167Z",
  "notification_id": "1",
  "fill_level": 0.25,
  "temperature": 20.0,
  "is_turned_over": true
}

```

Figure 3.20: Waste event response in proprietary format

```

{
  "contextElement": {
    "attributes": [
      {
        "type": "autocreated",
        "name": "created",
        "value": "2016-05-19 14:30:50.423105+00:00"
      },
      {
        "type": "autolastmodified",
        "name": "modified",
        "value": "2016-05-19 14:30:50.432167+00:00"
      },
      {
        "type": "char",
        "name": "notification_id",
        "value": "1"
      },
      {
        "type": "float",
        "name": "fill_level",
        "value": 0.25
      },
      {
        "type": "float",
        "name": "temperature",
        "value": 20.0
      },
      {
        "type": "boolean",
        "name": "is_turned_over",
        "value": "true"
      },
      {
        "type": "string",
        "name": "asset",
        "value": "1 (3000)"
      },
      {
        "type": "string",
        "name": "container",
        "value": "WasteContainer #1"
      }
    ],
    "type": "wasteevent",
    "id": 1
  },
  "statusCode": {
    "code": 200,
    "reasonPhrase": "OK"
  }
}

```

Figure 3.21: Waste event response in NGSI format

APIs, this way eliminating the existing overhead of the two HTTP requests that exist in the Hypercat solution.

As an example, two responses from different endpoints are listed above in figure 3.20 and 3.21.

They roughly correspond to the same request in the sense that they carry the same attributes in their payload and developers can extract identical information about the waste event that they are characterising.

Technically, achieving this implied implementing a new Django app. The waste API was built in Django, so it was only logical to follow the same approach. This app was built in the waste API project. It could be done in the same app or even in the same file where the original interface is specified but, for the sake of clearness and organisation, it was decided that a separate app would suit our needs the best.

3.3.5 BENCHMARKS

For this subsection, a benchmark was performed where one hundred requests were performed to a Citibrain API endpoint and another one hundred to an NGSI endpoint.

Both requests were very similar as they're goal was to list one event's attributes. One in the proprietary JSON format and the other obeying the NGSI specification.

It is not expected a big difference in the time intervals that both requests take to process. Unlike what happened in Hypercat, in this case there are no additional overheads in comparison to the proprietary implementation so the processing cost should be similar.

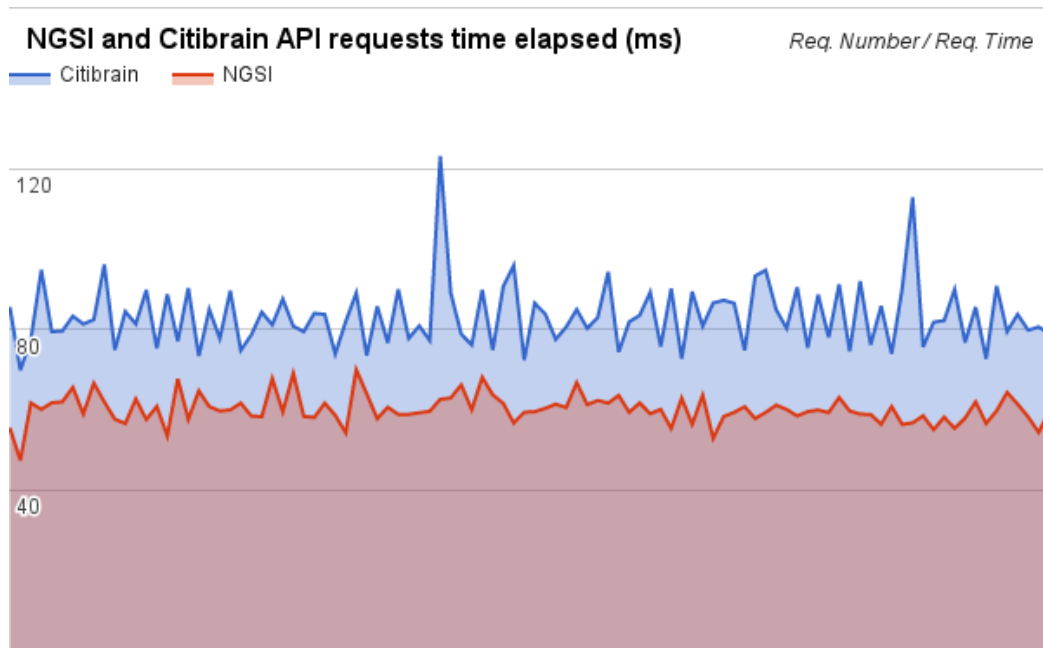


Figure 3.22: Elapsed time on HTTP requests to the Citibrain API endpoints in comparison to NGSI endpoints

It is somewhat surprising to discover that the NGSI implementation is actually faster than the original Citibrain one. As the graph shows, there is not much of a difference (roughly 20 ms) but the NGSI requests always seem to be faster.

One reason for this can be the implementation of both. The original API is built with the support of the Django Rest Framework, while the NGSI also is but very few features of that framework are used and a part of the necessary implementation is actually overridden to display the data in the NGSI format, which may save quite a bit of processing time.

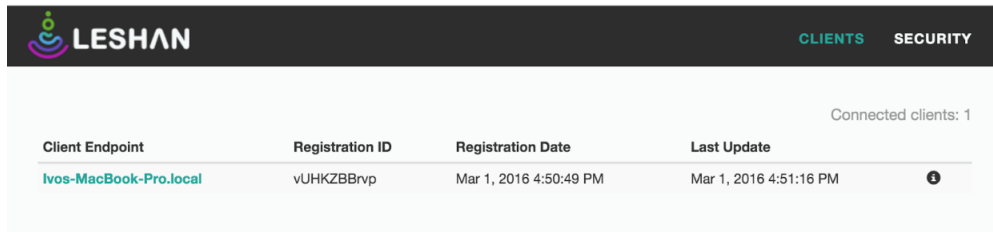
On average, the NGSI requests are 28% faster than the regular Citibrain ones.

3.4 LWM2M SUPPORT

3.4.1 EXPERIMENTS

Experimentations started with a Java library that implements a Lwm2m server and client. The project is called Leshan and is hosted by the Eclipse Foundation[30].

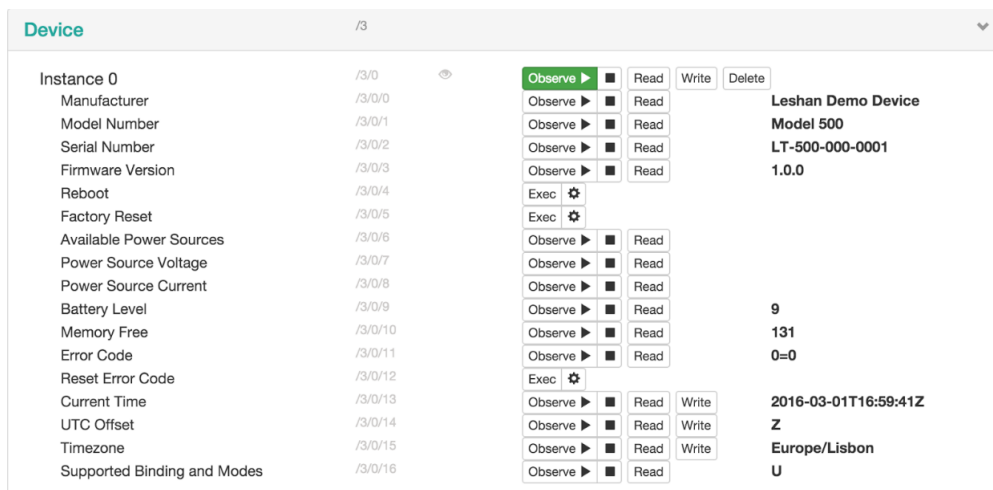
A big feature that this implementation provides is a demo web portal which makes the Lwm2m connections easily visible. It also allows client subscription and information query for a variety of fields.



The screenshot shows the Leshan server web portal. At the top, there is a navigation bar with the Leshan logo and the text "CLIENTS SECURITY". Below the navigation bar, there is a header indicating "Connected clients: 1". The main content area displays a table with the following columns: "Client Endpoint", "Registration ID", "Registration Date", and "Last Update". The table contains one entry for the client "Ivos-MacBook-Pro.local" with registration ID "vUHKZBBrv", registration date "Mar 1, 2016 4:50:49 PM", and last update "Mar 1, 2016 4:51:16 PM".

Client Endpoint	Registration ID	Registration Date	Last Update
Ivos-MacBook-Pro.local	vUHKZBBrv	Mar 1, 2016 4:50:49 PM	Mar 1, 2016 4:51:16 PM

Figure 3.23: Leshan server detecting the Lwm2m client



The screenshot shows the Leshan server web portal displaying the details of a device's resources. The device is identified as "Leshan Demo Device" with model "500" and ID "LT-500-000-0001". The resources are listed in a table with columns for "Resource ID", "Resource Name", and "Resource Value". The resources include "Instance 0", "Manufacturer", "Model Number", "Serial Number", "Firmware Version", "Reboot", "Factory Reset", "Available Power Sources", "Power Source Voltage", "Power Source Current", "Battery Level", "Memory Free", "Error Code", "Reset Error Code", "Current Time", "UTC Offset", "Timezone", and "Supported Binding and Modes". Each resource has a corresponding "Observe" button and a "Read" button. The "Observe" button is highlighted in green, indicating that the resource is being observed. The "Read" button is highlighted in blue, indicating that the resource is being read. The "Write" and "Delete" buttons are also visible for some resources.

Resource ID	Resource Name	Resource Value
/3/0	Instance 0	
/3/0/0	Manufacturer	
/3/0/1	Model Number	
/3/0/2	Serial Number	
/3/0/3	Firmware Version	
/3/0/4	Reboot	
/3/0/5	Factory Reset	
/3/0/6	Available Power Sources	
/3/0/7	Power Source Voltage	
/3/0/8	Power Source Current	
/3/0/9	Battery Level	9
/3/0/10	Memory Free	131
/3/0/11	Error Code	0=0
/3/0/12	Reset Error Code	
/3/0/13	Current Time	2016-03-01T16:59:41Z
/3/0/14	UTC Offset	Z
/3/0/15	Timezone	Europe/Lisbon
/3/0/16	Supported Binding and Modes	U

Figure 3.24: Leshan server reading/ subscribing client's resources

The above images show the most relevant Lwm2m features for our architecture which are device registration and resource observation and reading.

Breaking this experiment into pieces, it contains:

- A Lwm2m server running;
- A Lwm2m client named "Ivos-Macbook-Pro.local";
- The client registers into the server and appears in the connected clients list displayed in figure 3.23;
- Scanning the Lwm2m object specification, the server is able to query for certain resources of the client and know what they mean;
- The server has five available operations to act in the clients, which are observing, reading, writing, deleting and executing;

- Tapping the observe command on the instance makes the server observe all the available resources of that client.

The Leshan demo worked but both the client and the server integrated seamlessly and, being built in the scope of the same project, they should. To go a bit further and have more control over the whole device registration/creation part, it was decided to test the same exact server with a completely different implementation of a LwM2M client built with Node.js.

Doing this experiment helped to know if the LwM2M protocol is truly outlined and implementation-agnostic as well as how things are done “under the hood”. Many things appeared on the Leshan screen but, if we want to implement it in the company’s platform, a good knowledge of the core workflow is needed.

Going from the LwM2M node.js library, the test was started by tweaking the client implementation slightly by making it register in the port in which the server was listening, the same address, among other slight modifications.

In the beginning, a LwM2M client was created and some resources were assigned to it.

```
LWM2M-Client> create /1/1
Object:
-----
ObjectType: 1
ObjectId: 1
ObjectUri: /1/1
LWM2M-Client> set /1/1 student ivo
Object:
-----
ObjectType: 1
ObjectId: 1
ObjectUri: /1/1
Attributes:
-> student: ivo
LWM2M-Client> set /1/1 company ubiwhere
Object:
-----
ObjectType: 1
ObjectId: 1
ObjectUri: /1/1
Attributes:
-> student: ivo
-> company: ubiwhere
LWM2M-Client> set /1/1 thesis IoTStandardsForSmartCities
Object:
-----
ObjectType: 1
ObjectId: 1
ObjectUri: /1/1
Attributes:
-> student: ivo
-> company: ubiwhere
-> thesis: IoTStandardsForSmartCities
```

Figure 3.25: LwM2M client creation and resource assignment

In the above image, it is shown the following resources being created:

- “/1/1/student”: “ivo”;
- “/1/1/company”: “ubiwhere”;

- “/1/1/thesis”: “IoTStandardsForSmartCities”.

This was a very valuable experience because it was when it was discovered that LwM2M’s URIs don’t support non-numerical characters. So when this client device was registered into the Leshan server, the registration was concluded perfectly but, however, no resources were recognised. Here was the right moment to dive into the LwM2M object specification. It was learned that one LwM2M server only recognises attributes that are in its object specification file.

Concluding, this experiment was very insightful as it showed us how LwM2M servers handle the reading of resources from its clients and how can we make the clients’ resources known to the servers.

Afterwards, it was clear to understand how the server processes the messages that arrive to him. This enabled capturing the received LwM2M messages and paved the way for the first integration approach.

3.4.2 FIRST APPROACH

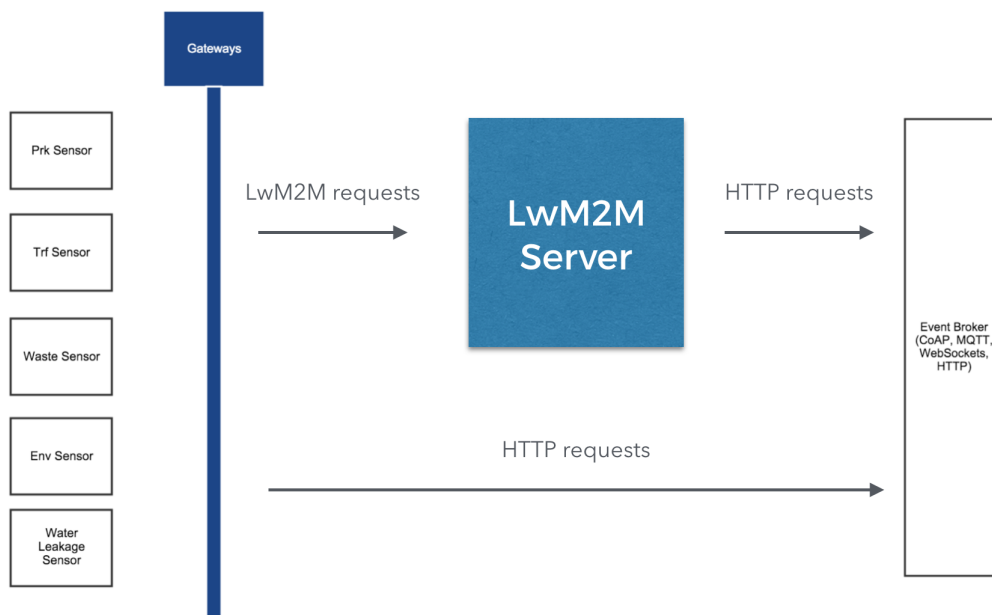


Figure 3.26: LwM2M first approach

The first approach was to ease the integration process by installing a proxy LwM2M server. The idea behind this was to make this proxy server receive the LwM2M specific requests and messages, parse them to build a similar message to one that the HTTP server would receive and forward it to out meshblu message broker as a regular HTTP request.

However, this approach could have been better planned and executed as I was not familiar with how the LwM2M standard and CoAP protocol worked at the time.

Firstly, we acknowledged that Meshblu does not take advantage of CoAP’s main selling features when compared to HTTP, such as clients’ observation on behalf of the server. The biggest benefit of using CoAP instead of HTTP is that the server has the possibility to subscribe to the client’s resources. Each time the client’s (sensor’s in our case) attributes are modified, the server gets notified and acts accordingly. Allied to the fact that CoAP is better suited for low-power devices, it would be a shame not to use it to its full potential.

In Meshblu’s situation, this does not happen. The devices use CoAP to communicate with Meshblu in an HTTP style. Every time there is a change to be made in a device, the device itself has to POST a request to change its attributes in the broker. This is a rather ineffective way of doing things.

Knowing this, it would be profitable to take advantage of CoAP’s advantages and make the least amount of changes we could in the broker that was already being used. The goal is to not break features that we have already have in production and add LwM2M support in the most unobtrusive way possible.

3.4.3 INTEGRATION WITH MESHBLU

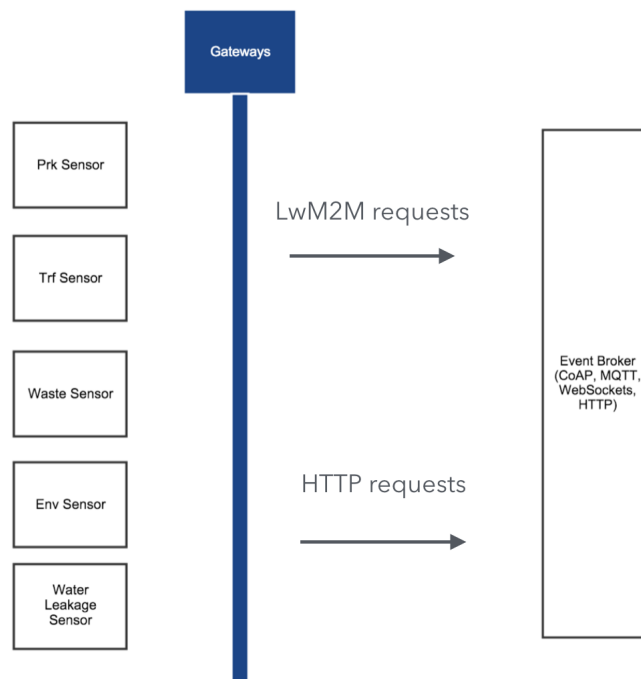


Figure 3.27: LwM2M integration in Meshblu

With the updated approach, CoAP’s advantages in contrast with HTTP were a possibility.

Citibrain uses Meshblu to “talk” with the devices at its Southbound, this means that the most critical communication for us is between the broker and the sensors.

The easiest process for this communication would be to register the devices in Meshblu and then subscribe to all their resources. This method would make the implementation of LwM2M almost

transparent to the broker. Each sensor sends a “register device” request to Meshblu and it would be done.

During the registration process, the broker is in charge of all the normal registering that occurs with any other protocol and in the end it would just observe the device. Each time a sensor value would update, our Meshblu instance would get notified and proceed to update its registry for that device, being the attributes affected by the LwM2M notification up-to-date and available in all the other protocols without interfering with the regular workflow that is already available in the production phase.

3.4.4 CITIBRAIN’S OBJECT SPECIFICATION

LwM2M works in such a way that resources are accessed by their URI.

A typical LwM2M resource URI is usually defined by its object identifier, the instance identifier and the resource identifier. This means that it is not possible at first sight to decipher what attribute does the URI “/3/3/1” represent. For this purpose, there is the need of an object specification to map resources into their specific URI.

The Open Mobile Alliance produced a specification with several generic objects that usually appear in the problems LwM2M tries to solve. Although this specification is very broad and covers all the major resources needed, it does not suit the system we being built.

We have some measurements from the sensors we use that are not covered by the OMA specification and, beyond that, our broker needs to receive a series of arguments so that it knows what kind of sensor is sending information, to what vertical solution does it belong. . .

Also, Meshblu handles all the device authentication and registration for the Citibrain platform. This means that there are certain fields that are needed to prove the identity of the device trying to send data to the broker. This is not a predicted use case in the object specification.

Our solution was to build our own specification. This translates roughly into something of what can be seen in Appendix C. A small snippet is shown below.

```
{
  "name": "waste_event",
  "id": 7300,
  "instancetype": "multiple",
  "mandatory": false,
  "description": "Description: Citibrain’s waste message spec",
  "resourcedefs": [
    {
      "id": 7301,
      "name": "name",
      "operations": "R",
      "instancetype": "single",
      "mandatory": true,
      "type": "string",
      "range": "",
    }
  ]
}
```

```
    "units": "",
    "description": "The name of the signal"
  },
  {
    "id": 7302,
    "name": "devices",
    "operations": "R",
    "instancetype": "single",
    "mandatory": true,
    "type": "string",
    "range": "",
    "units": "",
    "description": "devices type should be \"736421b0-e6a2-11e4-9715-7535251c3282\""
  },
  [ . . . ]
}
```

This reflects the LwM2M specification for a waste event. So, if a device was to be configured to be registered in the broker as a waste sensor, it would need to have defined every attribute in the “7300” object specification.

When the device registration is received by the broker, a message scan is performed and it is discovered what the device actually represents and the resources that it is able to retrieve from it.

For example, if it is wanted to register a waste sensor, there will be the need of creating a LwM2M client that contains an object id “7300”. The URI of this instance would be “/7300/x”, being x the instance id of the device.

This client would need to define the resources that are within the waste event object specification. To read a resource, the server needs to access the “/7300/x/y”, where y is a resource id. So the URI “/7300/x/7301” corresponds to the name of the device.

One big problem with this approach, due to the state of the art, is that whenever a new attribute needs to be added to a client, it is absolutely required that the object specification is also updated to support it. There is no way for the server to discover the attributes that are being sent to him without this context. This breaks the whole interoperability concept because, for example, if you want to start using different sensors that support different kinds of resources (or even the same resource specified in some other format), the server is not able to know what changed and what is the URI that it is receiving. There are some efforts being made (involving machine learning) in order to fix this issue but they are still in a premature state and not ready to be deployed in production environments like ours.

3.4.5 COMMUNICATION FLOW

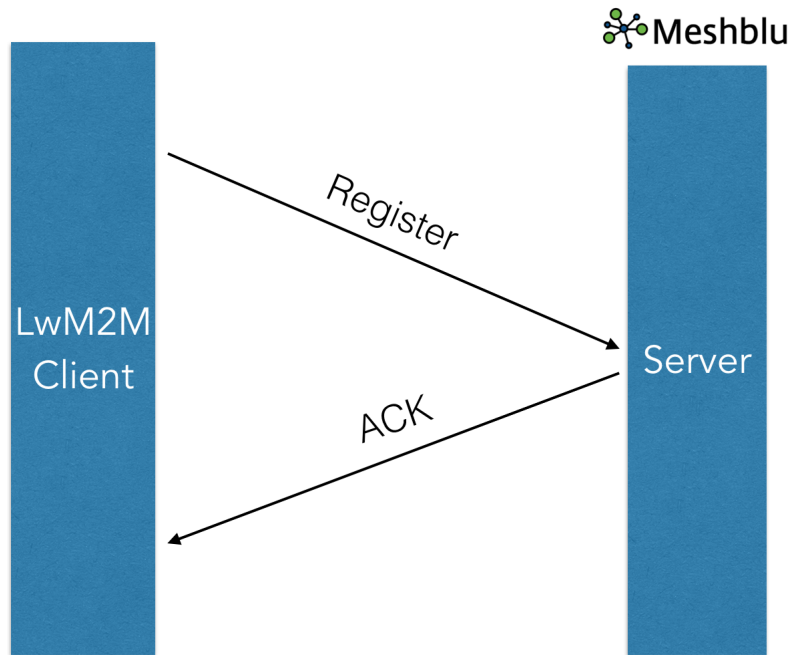


Figure 3.28: LwM2M registration flow

The first step is having the client send a register command to the server, which in Citibrain’s case is a Meshblu message broker instance. This message contains valuable client information like its address, the port from which the client is communicating (this will be useful), LwM2M’s object id and instance id, client lifetime, LwM2M version, among other data.

Receiving this message, the server gets the client’s object id and scans the object specification for it. By doing so, the server is “discovering” what the client actually is and resources that it is able to provide. After this discovery, we are able to generate the same JSON-serialized content that would be stored in an HTTP device register request. This enables us to register a LwM2M device exactly in the same way that we are currently doing, meaning by this that every device’s would be accessible through meshblu’s northbound API, independently of how this device was created.

When the device is created, Meshblu generates its required authentication and identification tokens for that device. This is the point where a new problem arises. There is the need to process how will Meshblu authenticate the client and where will we store these credentials.

Every time there is a change to be made in a device, the broker needs the credentials to validate the device that it is trying to update. These are necessary for every interaction after the registration. CoAP and LwM2M offers servers the ability to write values in the clients resources. For this situation, this feature becomes very interesting.

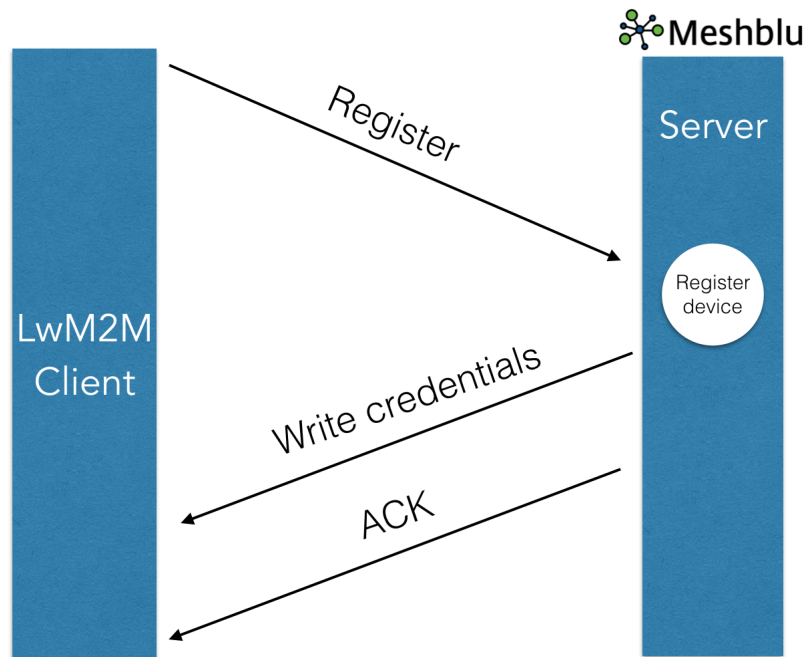


Figure 3.29: LwM2M registration flow saving the Meshblu’s credentials

To do this there is a need to define new resources in the object specification so that the server knows where to write the credentials in the client, during the registration process, and where to read from when the client’s credentials are needed to perform certain operations.

```

{
  "name": "waste_event",
  "id": 7300,
  "instancetype": "multiple",
  "mandatory": false,
  "description": "Description: Citibrain's waste message spec",
  "resourcedefs": [
    ...
  ]
}
{
  "id": 7308,
  "name": "meshblu_auth_uuid",
  "operations": "RW",
  "instancetype": "single",
  "mandatory": false,
  "type": "string",
  "range": "",
  "units": ""
}

```

```

    "description":"meshblu authentication uuid"
  },
  {
    "id":7309,
    "name":"meshblu_auth_token",
    "operations":"RW",
    "instancetype":"single",
    "mandatory":false,
    "type":"string",
    "range":"",
    "units":"",
    "description":"meshblu authentication token"
  }
]
}

```

Now that the broker's authentication fields are supported by the client, the server is able to read and write them at the appropriate times.

The next step is to make the server observe the desired client resources so that, whenever a client updates a certain attribute, the server is notified and maintains its device registry updated.

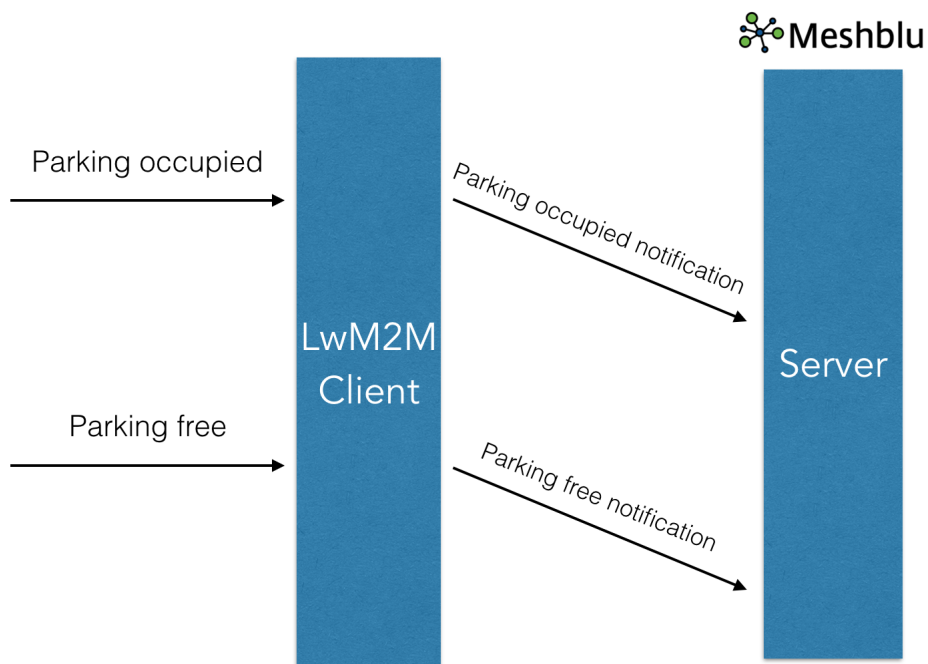


Figure 3.30: LwM2M object observation

For example, if our LwM2M client has a parking sensor monitoring a certain parking spot, the server doesn't need to query the client in order to know if the parking spot is empty or not. Whenever the parking situation changes, our broker will get notified and proceed to forward this information to it northbound (APIs used by applications, IFTTT engines, and other components).

So, if the resources of the clients are observed when they register into Meshblu, no more major operations are necessary to keep the object up-to-date, other than handling the observe notifications.

The process of handling the notifications includes being able to read the authentication resources that were written by the server in the registration process so that the device is identified against the existing Meshblu architecture.

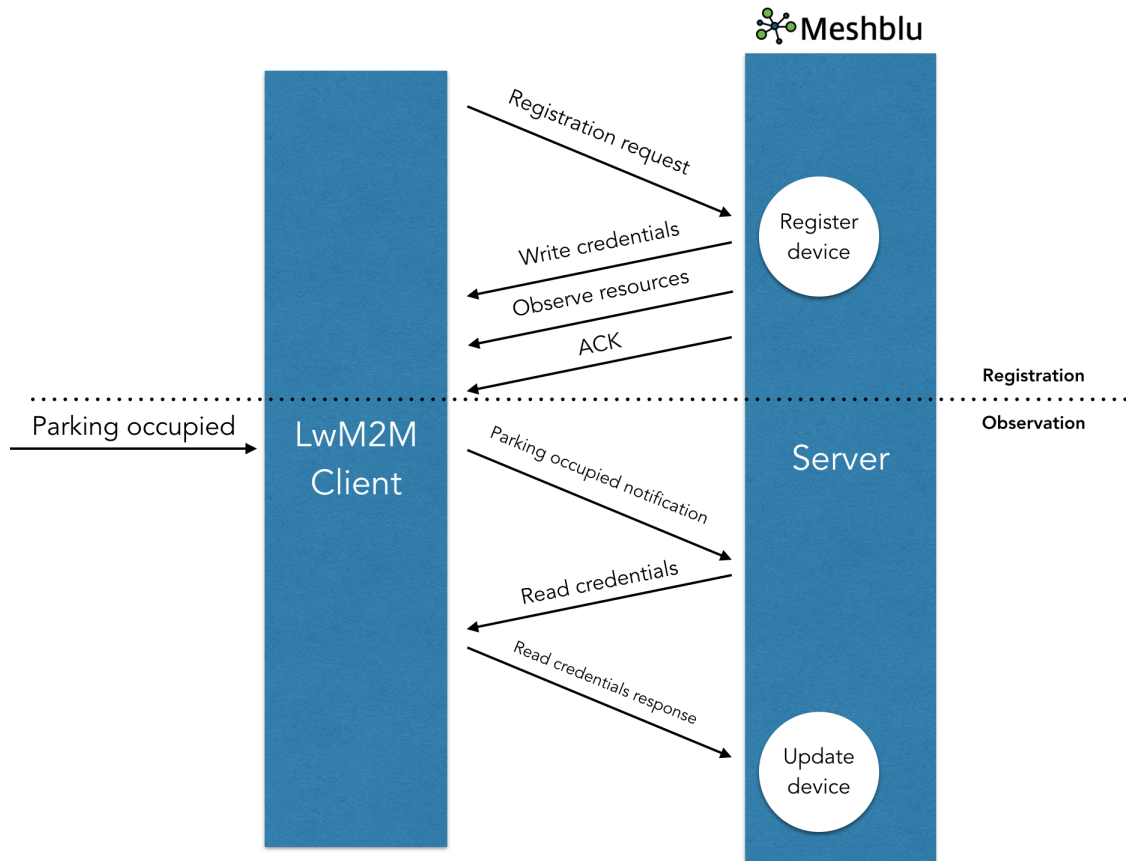


Figure 3.31: LwM2M complete message flow example for parking sensor

Very briefly, the interaction begins with the device’s registration. After the device is registered in the server, Meshblu emits two authentication fields which are then written to a client resource. Next, Meshblu will observe the resources that are interesting to follow. This completes the registration process.

When one of the observed resources changes its value, the server will receive a notification. The notification handler will then proceed to read the mandatory credentials from the client, in order to authenticate the device, and only then will it update the device registry with the new value.

3.4.6 TESTS

The tests performed covered a range of three LwM2M operations. They are device registration, that happens when it is wanted for a device to be recognized by Meshblu, device observation, which is

done right after the registration and the resource update that is executed every time the server receives a value change notification.

3.4.6.1 DEVICE REGISTRATION

In this section, the main goal is to demonstrate the process of registering a device into the Meshblu message broker using the LwM2M protocol.

The present example will show a Citibrain water leakage sensor being created and registered in the server.

First, the object specification for the sensor is shown bellow.

```
{
  "name": "Citibrain Water Leakage",
  "id": 7100,
  "instancetype": "multiple",
  "mandatory": false,
  "description": "Description: Citibrain's water leakage message spec",
  "resourcedefs": [
    {
      "id": 7101,
      "name": "name",
      "operations": "R",
      "instancetype": "single",
      "mandatory": true,
      "type": "string",
      "range": "",
      "units": "",
      "description": "The name of the signal"
    },
    {
      "id": 7102,
      "name": "devices",
      "operations": "R",
      "instancetype": "single",
      "mandatory": true,
      "type": "string",
      "range": "",
      "units": "",
      "description": "devices should be \"WATER_TARGET_UUID\""
    },
    {
      "id": 7103,
      "name": "device_type",
      "operations": "R",
```

```

    "instancetype": "single",
    "mandatory": true,
    "type": "string",
    "range": "",
    "units": "",
    "description": "The type of the device that's sending the message"
  },
  {
    "id": 7104,
    "name": "water_flow",
    "operations": "R",
    "instancetype": "single",
    "mandatory": true,
    "type": "integer",
    "range": "",
    "units": "",
    "description": "Water flow measurement of the sensor"
  },
  {
    "id": 7105,
    "name": "uuid",
    "operations": "R",
    "instancetype": "single",
    "mandatory": true,
    "type": "string",
    "range": "",
    "units": "",
    "description": "Meshblu device uuid"
  },
  {
    "id": 7106,
    "name": "token",
    "operations": "R",
    "instancetype": "single",
    "mandatory": true,
    "type": "string",
    "range": "",
    "units": "",
    "description": "Meshblu device auth tokens"
  }
]
}

```

According to it, the LwM2M client needs to have an object with the id “7100” in order to be identified as capable of giving water leakage measurements. Following this logic, a client was created

with that same object id and the instance id “1”. This results in the “/7100/1” object URI.

```

ivosilva@aluno-7180 ~/Documents/ubiwhere/dissertation/lwm2m-node-lib master bin/iotagent-lwm2m-client.js
LWM2M-Client> create /7100/1

Object:
-----
ObjectType: 7100
ObjectId: 1
ObjectUri: /7100/1
LWM2M-Client> connect localhost 5684 test /

Connecting to the server. This may take a while.

LWM2M-Client> Client Port:
65091
time=2016-05-26T13:40:23.505Z | lvl=INFO | corr=n/a | trans=n/a | op=LWM2MLib.COAPRouter | msg=Starting COAP Server
on port [65091]
REQUEST ARRIVED!
time=2016-05-26T13:40:23.509Z | lvl=ERROR | corr=n/a | trans=n/a | op=LWM2MLib.COAPRouter | msg=An error occurred cr
eating COAP listener: {"code":"EADDRINUSE","errno":"EADDRINUSE","syscall":"bind","address":"0.0.0.0","port":65091}
time=2016-05-26T13:40:23.521Z | lvl=INFO | corr=n/a | trans=n/a | op=LWM2MLib.COAPRouter | msg=Starting COAP Server
on port [65091]
REQUEST ARRIVED!
time=2016-05-26T13:40:23.522Z | lvl=INFO | corr=n/a | trans=n/a | op=LWM2MLib.COAPRouter | msg=COAP Server started s
uccessfully

```

Figure 3.32: LwM2M client creation and registration.

The Meshblu server receives the registration message and, with the help of the object specification, builds the same structured JSON data as what is stored via an HTTP registration. After saving this data in the meshblu platform, the authentication and identification credentials are generated. These will be needed for every operation that the device requests to the server.

```

ivosilva@aluno-7180 ~/Desktop/meshblu master sudo node server.js --http --coap

MM MM hh bb lll
MMM MMM eee sss hh bb lll uu uu
MM MM MM ee e s hhhhhh bbbbbb lll uu uu
MM MM eeeee sss hh hh bb bb lll uu uu
MM MM eeeee s hh hh bbbbbb lll uuuu u
sss
Meshblu (formerly skynet.im) development environment loaded...

Starting CoAP...started
done.
Starting HTTP/HTTPS... done.
CoAP listening at coap://localhost:5683
HTTP listening at 0.0.0.0:80
LwM2M listening at coap://localhost:5684
201
Generated meshblu uuid: 67c93e9b-1fa4-48cd-b0bd-018427a34683
Generated meshblu token: c1125bc814b675c50cfa2b11e125f22723a19259

```

Figure 3.33: Meshblu registering the device and printing the generated device UUID and authentication token.

Right after the registration operation happens, the device’s credentials are written into the device. This is a big “bonus” that LwM2M offers: being able, from the server, to write into the client’s resources. These values will not be written in any resource chosen by chance, but in the ones "programmed" in the object specification according to the object id.

```

Connected:
-----
Device location: rd/1
LWM2M-Client>
Value written:
-----
-> ObjectType: 7100
-> ObjectId: 1
-> ResourceId: 7106
-> Written value: c1125bc814b675c50cfa2b11e125f22723a19259
LWM2M-Client>
Value written:
-----
-> ObjectType: 7100
-> ObjectId: 1
-> ResourceId: 7105
-> Written value: 67c93e9b-1fa4-48cd-b0bd-018427a34683
LWM2M-Client>

```

After the device's registration into the server, meshblu's authentication credentials are written into the pre-defined resources of the client

Figure 3.34: Client's Meshblu credentials being written in appropriate resources.

In the end of this process, the client is finally able to communicate and update its values in the server.

By checking all the defined resources for the clients, we can confirm that the server's writing operation was executed successfully.

```

LWM2M-Client> get /7100/1

Object:
-----
ObjectType: 7100
ObjectId: 1
ObjectUri: /7100/1

Attributes:
-> 7105: 67c93e9b-1fa4-48cd-b0bd-018427a34683
-> 7106: c1125bc814b675c50cfa2b11e125f22723a19259

LWM2M-Client>

```

Figure 3.35: Client's resource value list.

For the final test, an HTTP request was performed to list all the devices stored in the Meshblu broker. For this operation, the previously generated device UUID and authentication token are going to be used. This test is able to demonstrate the interoperability and correct implementation of the LwM2M registration flow.

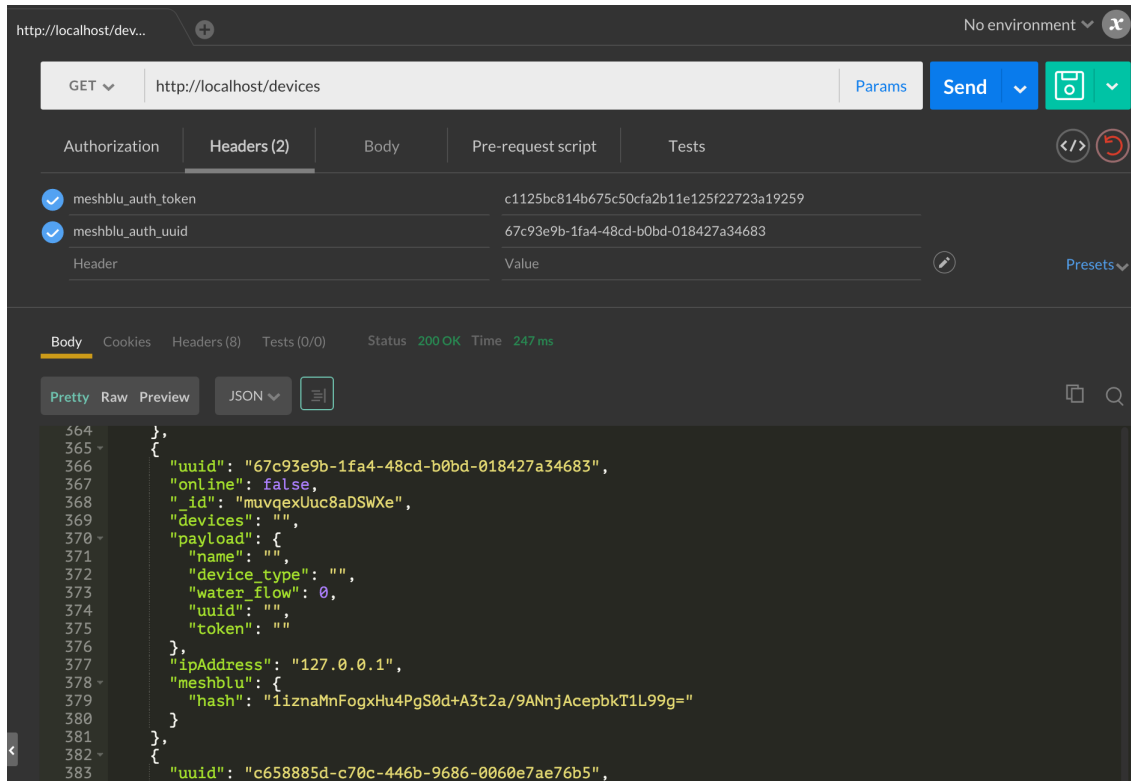


Figure 3.36: Meshblu's HTTP registered clients list method.

It is visible by the content of the response that the LwM2M device registration was performed correctly. Otherwise, it would not even validate the request and would return an "Unauthorized" message.

Not only that but the LwM2M device that was registered is displayed in the response and contains all the attributes as described in the object specification for a water leakage sensor: "name", "devices", "device_type" and "water_flow".

Some of these attributes are inside the "payload" JSON dictionary in order to mimic the current format of the messages being used in the Citibrain platform in order to integrate this implementation seamlessly.

3.4.6.2 DEVICE OBSERVATION

After the registration process is completed, to follow the flow defined in section 6.5, the server will observe the desired resources of the client. For our use case, the resources worth following are every one that is listed in the object specification for that type of object with the exception of the Meshblu's attributed UUID and authentication token, which are constant.

In this particular situation, the server will observe the "name", "device_type" and "water_flow" which correspond to the URIs "/7100/1/7101", "/7100/1/7103" and "/7100/1/7104" respectively.


```

ivosilva@aluno-7180 ~/Desktop/meshblu master sudo node server.js --http --coap
Password:

MM  MM      hh   bb   lll
MMM MMM  eee  sss  hh   bb   lll uu  uu
MM MM MM ee  e s  hhhhhh bbbbbb lll uu  uu
MM  MM eeeee  sss  hh   hh  bb   bb  lll uu  uu
MM  MM eeeee  s  hh   hh  bbbbbb lll  uuuu u
      sss
Meshblu (formerly skynet.im) development environment loaded...

Starting CoAP...started
done.
Starting HTTP/HTTPS... done.
CoAP listening at coap://localhost:5683
HTTP listening at 0.0.0.0:80
LwM2M listening at coap://localhost:5684
201
Generated meshblu uuid: 927f8e35-a38b-455e-a394-10ca8ffe1eef
Generated meshblu token: 8acc15d2e17e34da842ee700cbb8a2bb2fed22af

Observer established over resource [/7100/1/7104]

Observer established over resource [/7100/1/7101]

Observer established over resource [/7100/1/7103]

```

Figure 3.37: Meshblu establishing an observer connection with the valuable client's resources.

From this point on, for every change to those resources that is made in the client, the server will receive one notification.

3.4.6.3 RESOURCE UPDATE

However, Meshblu does not accept updates without validating the source of them. This validation is done through the created credentials in the registration.

In the LwM2M case, these credentials are stored in client's resources that can be discovered through the object specification. This means that for every update that the server gets, there is the need to identify and authenticate the device performing the notification before committing the changes.

In order to test this, a value was set in the "water_flow" resource to simulate a sensor reading, for example.

```

LWM2M-Client> set /7100/1 7104 3

Object:
-----
ObjectType: 7100
ObjectId: 1
ObjectUri: /7100/1

Attributes:
  -> 7104: 3
  -> 7105: 30fd3ca1-6676-4e9d-9bbf-ffbca054b445
  -> 7106: 8f6ab289701144da2e0ce4f6dced84f32881429f

```

Figure 3.38: A LwM2M client updating its "water_flow" resource.

In figure 3.38 it is understandable that the resource "/7100/1/7104" was updated in the client. Being this resource an observed one, this triggered an notification in the Meshblu server.

```

ivosilva@aluno-7180 ~/Desktop/meshblu master sudo node server.js --http --coap
Password:

MM   MM           hh   bb   lll
MMM  MMM   eee   sss  hh   bb   lll uu  uu
MM MM MM ee  e s   hhhhhh bbbbbb lll uu  uu
MM   MM eeeee  sss  hh   hh  bb   bb lll uu  uu
MM   MM eeeee   s  hh   hh  bbbbbb lll  uuuu u
      sss
Meshblu (formerly skynet.im) development environment loaded...

Starting CoAP...started
done.
Starting HTTP/HTTPS... done.
CoAP listening at coap://localhost:5683
HTTP listening at 0.0.0.0:80
LwM2M listening at coap://localhost:5684
201
Generated meshblu uuid: 30fd3ca1-6676-4e9d-9bbf-ffbca054b445
Generated meshblu token: 8f6ab289701144da2e0ce4f6dced84f32881429f

Observer established over resource [/7100/1/7103]

Observer established over resource [/7100/1/7101]

Observer established over resource [/7100/1/7104]

Got new value on resource /7100/1/7104 in device [1]: 3

Read uuid from client: 30fd3ca1-6676-4e9d-9bbf-ffbca054b445
Read token from client: 8f6ab289701144da2e0ce4f6dced84f32881429f
Device update successfull

```

Figure 3.39: Meshblu’s output for a device registration and following update.

The above image expresses the server’s "reaction" on the event of receiving the notification from the client:

1. The updated value and the resource URI are received;
2. Server performs two read operations to fetch the UUID and authentication token from the client’s resources;
3. Proceeds to update the device in its internal registry.

To further prove the viability of this implementation, one is able to check the device’s records through other protocol, say HTTP for simplicity sake.

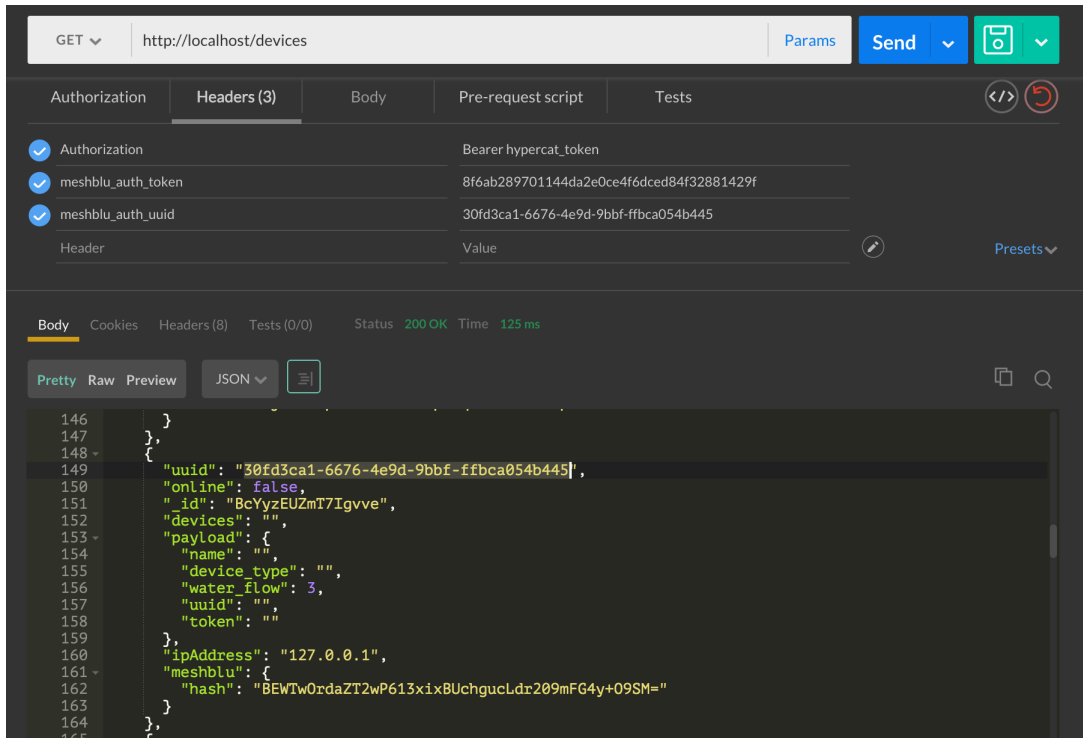


Figure 3.40: HTTP list devices request to Meshblu.

The registered device shows in the Meshblu device list and has the "water_flow" value that the client just updated. With this implementation, now Citibrain is able to support LwM2M devices in its Southbound.

3.4.7 BENCHMARKS

Benchmarking the LwM2M implementation was rather difficult because there is no comparison to be made whatsoever. The Meshblu registration via LwM2M is not just a regular operation as it implies the actual registration and writing the authentication and identification values into the server. When a client is updated, that implies the notification to the server, the reading of the credentials and only then the actual submission of the updates.

However, it was decided to compare it in the best way possible to the HTTP version of the same operations which is what the platform is actually performing right now.

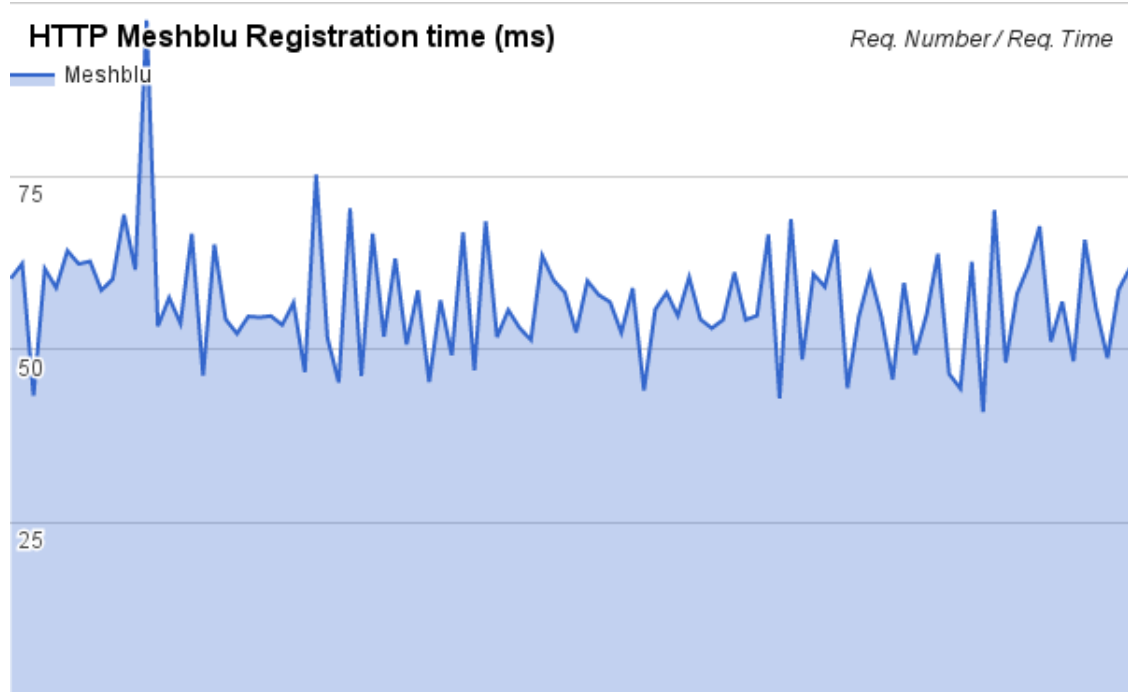


Figure 3.41: Time elapsed registering devices into Meshblu via HTTP

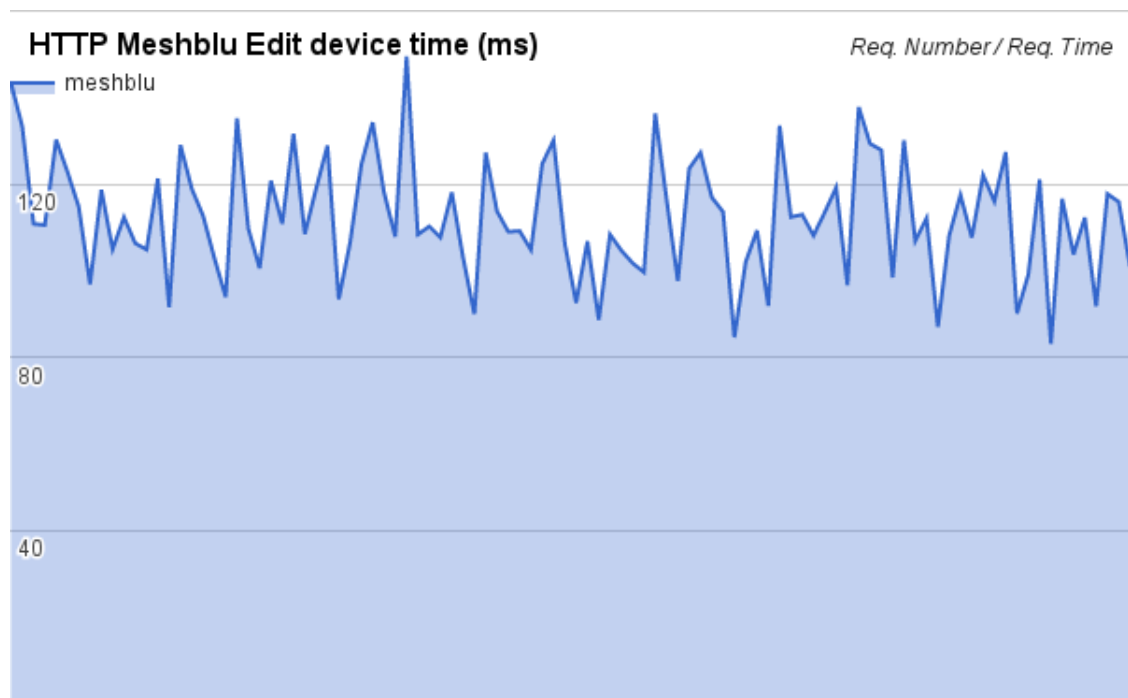


Figure 3.42: Time elapsed updating devices into Meshblu via HTTP

Above, figure 3.41 and 3.42 illustrate the time spent registering and editing a device through the HTTP endpoints provided by Meshblu. It is to be reminded that all the needed credentials and tokens need to be placed in the request's headers so that the server is able to authenticate the device requesting the update. In the registration process, the credentials are returned and it is completely the responsibility of the client to store them for further requests.

It is easily observable that an update operation takes roughly double the time of a registration request, on average. A big part of this time is, most likely, spent on authenticating and retrieving the desired device to update, while on the registration operation there is no need as the device is non-existent at that point in time.

Below, the actual LwM2M benchmarks are shown.

One hundred registrations were performed by a LwM2M client to Meshblu and the time interval was registered. Take into account that the time started counting in the moment that the device initiated the request until the moment that the last server credential (UUID or authentication token) was written into the appropriate resources of the client.

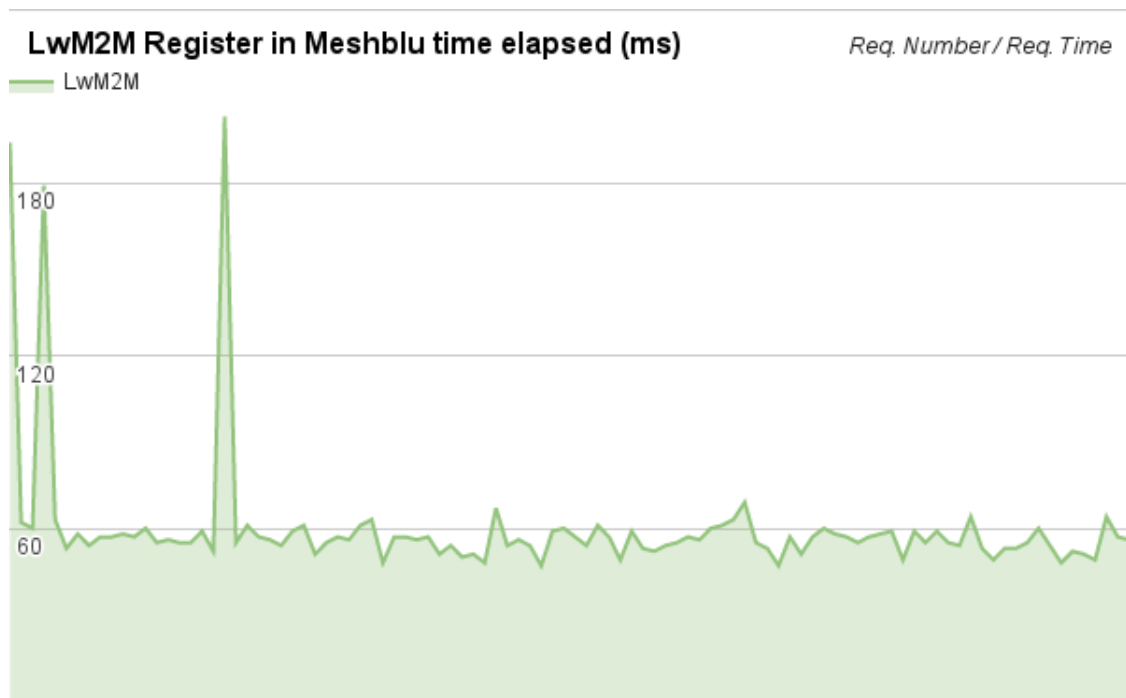


Figure 3.43: Time elapsed registering devices into Meshblu via LwM2M

The benchmark shows that the registration time is, on average, roughly the same as the HTTP implementation (60ms). This implementation of LwM2M is based on CoAP which is a lighter protocol when compared to HTTP. This can justify why, even performing two more operations (writing of the credentials), the time elapsed does not increase.

On the update operation, again one hundred client updates were performed and the time was taken from the moment when the server received the device update notification until the moment when the changes were actually committed. Mind that the server's reaction for receiving a client update is to read that same client's credentials, which means that between the notification reception and the submission of the updates, two read operations are performed to the client.

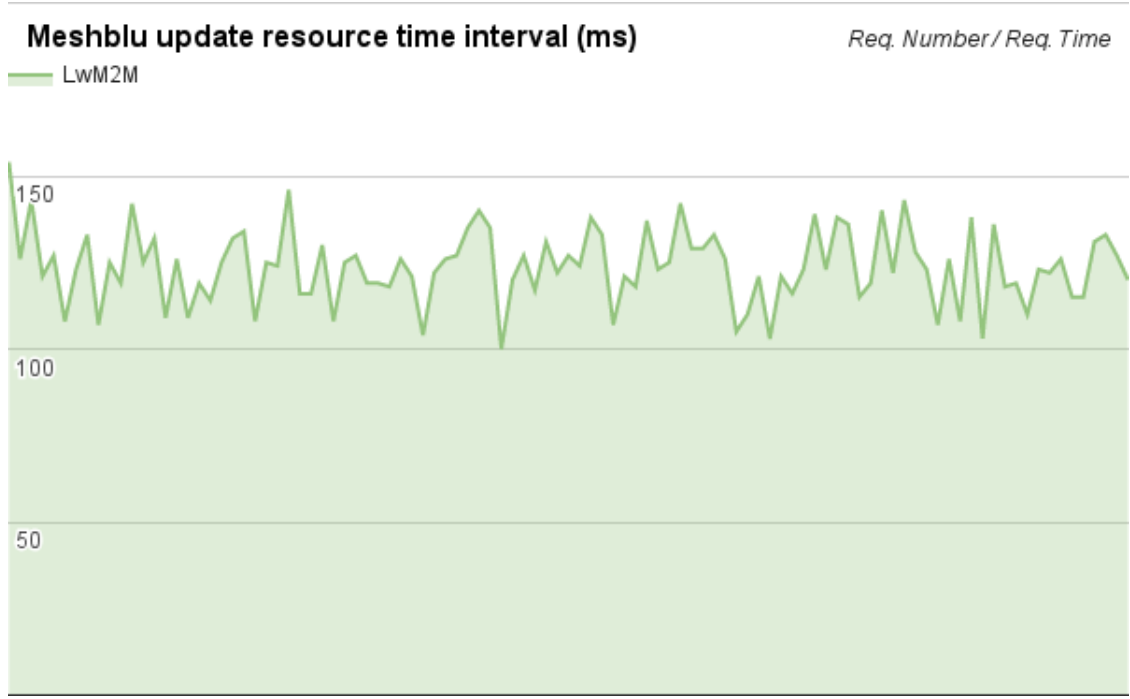


Figure 3.44: Time elapsed updating devices into Meshblu via LwM2M

The information in the graph above shows that the average time elapsed on an update operation is very similar to the one spent on the HTTP implementation, although in this case it was not possible to track the time immediately since the notification is sent from the device but only from the point where the server received the said notification. To be realistic, that time needs to be taken into account when comparing to other implementations.

OPEN-SOURCE CONTRIBUTIONS

4.1 LWM2M

While developing our solution and using other libraries for testing purposes, I had the opportunity to contribute to the official Lwm2m Node.js implementation.

The problem I and other developers were facing was that, if a server was subscribed to client's resource, when the client updated its resource value, the server would get notified and get the updated value. The issue here was that the server had no information about which client device changed some property.

In other words, the server knew that someone had just updated a resource but didn't know who.

My pull request aimed at solving this problem and it was something that the community was also asking for, as there were other developers helping my pull request get accepted into the main repository.

The accepted pull request can be found at:

- <https://github.com/telefonicaid/lwm2m-node-lib/pull/99>.

The previous pull request that generated somewhat of a discussion are at:

- <https://github.com/telefonicaid/lwm2m-node-lib/pull/94>;

- <https://github.com/telefonicaid/lwm2m-node-lib/pull/97>.

4.2 HYPERCAT

During the dissertation period, the Hypercat format underwent a few revisions. The official python repositories were not being updated so I took the chance to do it myself.

The pull requests can be found at:

- <https://github.com/HyperCatIoT/python-tools/pull/1>;

- <https://github.com/HyperCatIoT/python-tools/pull/2>.

4.3 FIWARE

For the Fiware NGSI implementation I fixed, tweaked and fully documented a Django REST NGSI 10 serializer. I also created a demo Django application that can be found in the repository to give an example on how to use the library.

This library was used in the NGSI 10 implementation on Citibrain but it was also extensively documented in order to be made available to the community in general but specifically at Smart City Hackathons that will be supported by Ubiwhere. An example is "Hack-a-City" which, on the 27th and 28th of May of 2016, occurred in Amersfoort, Utrecht, Porto, Santander, Olinda and Recife. It is essentially "a hackathon that aims to test big data and promotes its use to develop solutions that will have an impact in the city" [31].

The library can be found at:

- <https://github.com/Ubiwhere/django-rest-ngsi>.

CONCLUSION

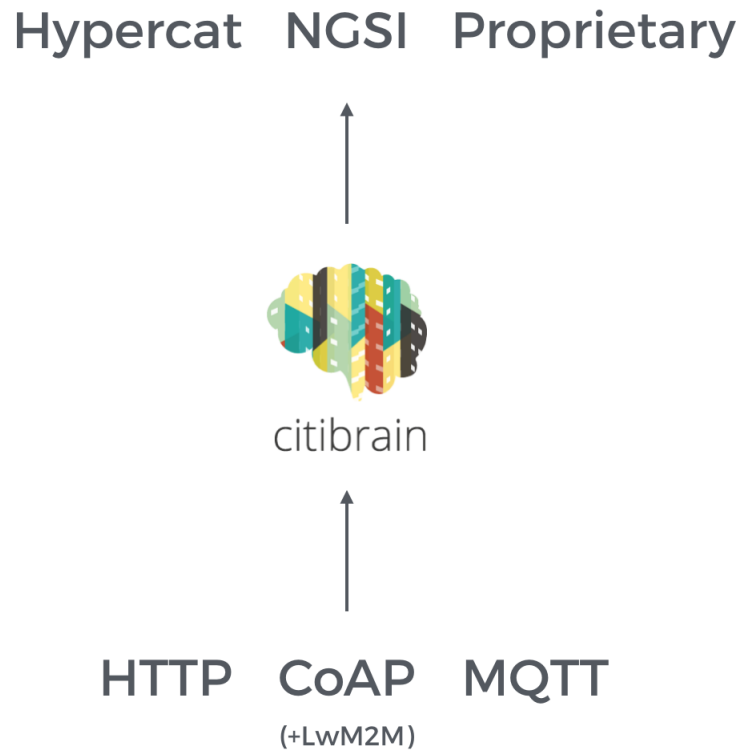


Figure 5.1: Overview of the Citibrain platform

In this very premature era for the Internet of Things, Smart Cities are a concept that is very well thought out but not well executed. The execution lacks due to the state of the art and interoperability issues. One true standard that every implementation follows does not exist and this fact imposes limits in the communication between different ecosystems and platforms.

At Ubiwhere, the decision was to play it safe and be smart. If a true Smart City platform is to be built and it is not known which IoT standard and protocols will persevere, then an effort needs

to be made in order to be prepared for whatever the outcome may be. It is not viable to implement a platform with more features, different add-ons and that implies a lot of development effort before paving the way for interoperability. Otherwise, if a new standard is pushed and adopted by the industry, Ubiwhere is left with a useless great peace of software. It can truly be tremendously brilliant but, all of a sudden, it becomes obsolete.

My master's dissertation was the first step into making the Citibrain platform available for different communication channels and supporting various information formats. The work done opens the door for the platform to be understood by Hypercat and Fiware applications. In other words, Citibrain can be used to disseminate information in the Fiware's supported format, in a similar way to what is happening in Porto and Santander, making it compatible with devices and applications that already are consuming and producing the said information.

Hypercat is becoming widely popular among companies in the United Kingdom, so that becomes a market where Ubiwhere is able to test the product.

On the other end of the platform, on its Southbound, Citibrain is now able to support LwM2M devices which is a communication protocol built specifically with IoT devices in mind, being a lightweight (hence the name) alternative to heavier protocols such as HTTP which was used in every communication in the ecosystem.

5.1 FUTURE WORK

There is still a lot that can be done to future-proof the platform. MQTT is also a very interesting communication protocol to support in the lower end of the system. It is already supported by Meshblu but has various limitations so it would become a promising investigation to dive into that subject.

It can also be interesting to experiment with Ponte by Eclipse. Their architecture provides HTTP, CoAP and MQTT servers that communicate with storage engines and publish subscribe brokers. Ponte can be a good alternative to the Meshblu broker being used right now in Citibrain. It brings to the table the same protocol support with the benefit of having better MQTT specification coverage. Along with that, it is also implemented with Node.js so adding LwM2M support should not be very different from the Meshblu's implementation.

APPENDIX A

6.1 CITIBRAIN'S HYPERCAT "/CAT" METHOD RESPONSE

```
{
  "catalogue-metadata": [
    {
      "val": "application/vnd.hypercat.catalogue+json",
      "rel": "urn:X-hypercat:rels:isContentType"
    },
    {
      "val": "Citibrain Hypercat Catalogue",
      "rel": "urn:X-hypercat:rels:hasDescription:en"
    }
  ],
  "items": [
    {
      "href": "/cat/parking/assets/",
      "item-metadata": [
        {
          "val": "application/parking_assets",
          "rel": "urn:X-hypercat:rels:isContentType"
        },
        {
          "val": "Parking Assets",
          "rel": "urn:X-hypercat:rels:hasDescription:en"
        }
      ]
    },
    {
      "href": "/cat/parking/events/",
      "item-metadata": [
```

```

    {
      "val": "application/parking_events",
      "rel": "urn:X-hypercat:rels:isContentType"
    },
    {
      "val": "Parking Events",
      "rel": "urn:X-hypercat:rels:hasDescription:en"
    }
  ]
},
{
  "href": "/cat/waste/assets/",
  "item-metadata": [
    {
      "val": "application/waste_assets",
      "rel": "urn:X-hypercat:rels:isContentType"
    },
    {
      "val": "Waste Assets",
      "rel": "urn:X-hypercat:rels:hasDescription:en"
    }
  ]
},
{
  "href": "/cat/waste/events/",
  "item-metadata": [
    {
      "val": "application/waste_events",
      "rel": "urn:X-hypercat:rels:isContentType"
    },
    {
      "val": "Waste Events",
      "rel": "urn:X-hypercat:rels:hasDescription:en"
    }
  ]
},
{
  "href": "/cat/environment/assets/",
  "item-metadata": [
    {
      "val": "application/environment_assets",
      "rel": "urn:X-hypercat:rels:isContentType"
    },
    {
      "val": "Environment Assets",

```

```

        "rel": "urn:X-hypercat:rels:hasDescription:en"
    }
]
},
{
    "href": "/cat/environment/events/",
    "item-metadata": [
        {
            "val": "application/environment_events",
            "rel": "urn:X-hypercat:rels:isContentType"
        },
        {
            "val": "Environment Events",
            "rel": "urn:X-hypercat:rels:hasDescription:en"
        }
    ]
},
{
    "href": "/cat/traffic/assets/",
    "item-metadata": [
        {
            "val": "application/traffic_assets",
            "rel": "urn:X-hypercat:rels:isContentType"
        },
        {
            "val": "Traffic Assets",
            "rel": "urn:X-hypercat:rels:hasDescription:en"
        }
    ]
},
{
    "href": "/cat/traffic/events/",
    "item-metadata": [
        {
            "val": "application/traffic_events",
            "rel": "urn:X-hypercat:rels:isContentType"
        },
        {
            "val": "Traffic Events",
            "rel": "urn:X-hypercat:rels:hasDescription:en"
        }
    ]
}
]
}
}

```


APPENDIX B

7.1 CITIBRAIN'S HYPERCAT "/CAT/PARKING/ASSETS"
REQUEST RESPONSE

```
{
  "items": [
    {
      "i-object-metadata": [
        {
          "val": "application/asset",
          "rel": "urn:X-hypercat:rels:isContentType"
        },
        {
          "val": "30-0001",
          "rel": "urn:X-hypercat:rels:hasDescription:en"
        },
        {
          "val": "",
          "rel": "additional_fields"
        },
        {
          "val": "30-0001",
          "rel": "uuid"
        },
        {
          "val": "True",
          "rel": "is_active"
        },
        {
          "val": "N/A",
```

```

        "rel": "last_activity"
    },
    {
        "val": "-8.59954833984",
        "rel": "longitude"
    },
    {
        "val": "40.7510375977",
        "rel": "latitude"
    },
    {
        "val": "sensor",
        "rel": "type"
    },
    {
        "val": "",
        "rel": "parent_asset"
    },
    {
        "val": "sensor1",
        "rel": "name"
    }
    ],
    "href": "30-0001"
},


...


],
"item-metadata": [
    {
        "val": "application/vnd.hypercat.catalogue+json",
        "rel": "urn:X-hypercat:rels:isContentType"
    },
    {
        "val": "Citibrain Parking Assets Hypercat Catalogue",
        "rel": "urn:X-hypercat:rels:hasDescription:en"
    }
]
}

```


APPENDIX C

8.1 LWM2M WASTE EVENT SPECIFICATION

```
{
  "name": "waste_event",
  "id": 7300,
  "instancetype": "multiple",
  "mandatory": false,
  "description": "Description: Citibrain's waste message spec",
  "resourcedefs": [
    {
      "id": 7301,
      "name": "name",
      "operations": "R",
      "instancetype": "single",
      "mandatory": true,
      "type": "string",
      "range": "",
      "units": "",
      "description": "The name of the signal"
    },
    {
      "id": 7302,
      "name": "devices",
      "operations": "R",
      "instancetype": "single",
      "mandatory": true,
      "type": "string",
      "range": "",
      "units": "",
      "description": "devices type should be \"736421b0-e6a2-11e4-9715-7535251c3282\""
    }
  ]
}
```

```

},
{
  "id": 7303,
  "name": "device_type",
  "operations": "R",
  "instancetype": "single",
  "mandatory": true,
  "type": "string",
  "range": "",
  "units": "",
  "description": "The type of the device that's sending the message"
},
{
  "id": 7304,
  "name": "status",
  "operations": "R",
  "instancetype": "single",
  "mandatory": true,
  "type": "integer",
  "range": "",
  "units": "",
  "description": "Status of the sensor"
},
{
  "id": 7305,
  "name": "distance",
  "operations": "R",
  "instancetype": "single",
  "mandatory": true,
  "type": "integer",
  "range": "",
  "units": "",
  "description": "Distance to full"
},
{
  "id": 7306,
  "name": "temperature",
  "operations": "R",
  "instancetype": "single",
  "mandatory": true,
  "type": "integer",
  "range": "",
  "units": "",
  "description": "Temperature of the sensor"
},

```

```
{
  "id": 7307,
  "name": "movement_detected",
  "operations": "R",
  "instancetype": "single",
  "mandatory": true,
  "type": "integer",
  "range": "0-1",
  "units": "",
  "description": "Movement detected at the sensor"
}
]
}
```


BIBLIOGRAPHY

- [1] F. Group, “The internet of things in the cloud”, no. December, 2013. DOI: 10.1201/b13090.
- [2] *What you need to know about iot*. [Online]. Available: <http://www.business.att.com/content/whitepaper/what-you-need-to-know-about-IoT.pdf>.
- [3] *Loon for all – project loon – google*. [Online]. Available: <http://www.google.com/loon/>.
- [4] *Google x confirms the rumors: it really did try to design a space elevator | fast company | business + innovation*. [Online]. Available: <http://www.fastcompany.com/3029138/world-changing-ideas/google-x-confirms-the-rumors-it-really-did-try-to-design-a-space-elevat>.
- [5] *In-depth: top 10 internet of things companies to watch*. [Online]. Available: <http://www.rcrwireless.com/20151130/internet-of-things/in-depth-top-10-internet-of-things-companies-to-watch>.
- [6] *The internet of things*. [Online]. Available: http://www.cisco.com/c/dam/en%7B%5C_%7Dus/solutions/trends/iot/docs/iot-aag.pdf.
- [7] *Core framework | allseen alliance*. [Online]. Available: <https://allseenalliance.org/framework/documentation/learn/core>.
- [8] *Oma lwm2m - wikipedia, the free encyclopedia*. [Online]. Available: https://en.wikipedia.org/wiki/OMA%7B%5C_%7DLWM2M.
- [9] T. Specification, “Etsi ts 102 690”, vol. 1, pp. 1–279, 2013.
- [10] *Iot.eclipse.org — standards*. [Online]. Available: <http://iot.eclipse.org/standards>.
- [11] *Fiware sites index*. [Online]. Available: <http://mediafi.org/about-fi-content/fiware/>.
- [12] *Telefonica, orange, engineering and atos join forces to push common standards for smart cities based on the fiware platform*. [Online]. Available: http://atos.net/en-us/home/we-are/news/press-release/2015/pr-2015%7B%5C_%7D03%7B%5C_%7D03%7B%5C_%7D01.html.
- [13] *Porto, a city that has become a real-time guide » fiware*. [Online]. Available: <https://www.fiware.org/2015/11/20/porto-a-city-that-has-become-a-real-time-guide/>.
- [14] *Porto, a city that has become a real-time guide » fiware*. [Online]. Available: <https://www.fiware.org/2015/11/20/porto-a-city-that-has-become-a-real-time-guide/>.
- [15] *Santander: the smartest smart city*. [Online]. Available: <http://www.governing.com/topics/urban/gov-santander-spain-smart-city.html>.
- [16] *Array of things*. [Online]. Available: <https://arrayofthings.github.io/>.

- [17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, *Hypertext transfer protocol – http/1.1*, RFC 2068 (Proposed Standard), Obsoleted by RFC 2616, Internet Engineering Task Force, Jan. 1997. [Online]. Available: <http://www.ietf.org/rfc/rfc2068.txt>.
- [18] *Constrained application protocol - wikipedia, the free encyclopedia*. [Online]. Available: https://en.wikipedia.org/wiki/Constrained%7B%5C_%7DApplication%7B%5C_%7DProtocol.
- [19] Z. Shelby, K. Hartke, and C. Bormann, *The constrained application protocol (coap)*, RFC 7252 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: <http://www.ietf.org/rfc/rfc7252.txt>.
- [20] T. Levä, O. Mazhelis, and H. Suomi, “Comparing the cost-efficiency of coap and http in web of things applications”, *Decision Support Systems*, vol. 63, no. September 2013, pp. 23–38, 2014, ISSN: 01679236. DOI: 10.1016/j.dss.2013.09.009.
- [21] *Mqtt*. [Online]. Available: <http://mqtt.org/>.
- [22] P. Saint-Andre, *Extensible messaging and presence protocol (xmpp): core*, RFC 6120 (Proposed Standard), Internet Engineering Task Force, Mar. 2011. [Online]. Available: <http://www.ietf.org/rfc/rfc6120.txt>.
- [23] *Android wear*. [Online]. Available: <https://www.android.com/wear/>.
- [24] *Android auto*. [Online]. Available: <https://www.android.com/auto/>.
- [25] *Welcome to meshblu · meshblu*. [Online]. Available: <https://meshblu.readme.io/>.
- [26] *Ponte - bringing things to rest developers*. [Online]. Available: <http://www.eclipse.org/ponte/>.
- [27] J. Eriksson, S. Wengbrand, F. Handledare, and J. Juni, “A case study”, 2015. DOI: 10.1177/0961000605057849.
- [28] *Boosting the development of innovative m2m and iot applications*. [Online]. Available: <http://www.open-mtc.org/index.html%7B%5C#%7Dopenmtc>.
- [29] *Rabbitmq - compatibility and conformance*. [Online]. Available: <https://www.rabbitmq.com/specification.html>.
- [30] *Leshan oma lwm2m*. [Online]. Available: <https://github.com/eclipse/leshan>.
- [31] *Hackacity*. [Online]. Available: <https://www.hackacity.eu/>.