Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

Ilan Emanuel Moreira Pegoraro

NSCL na Plataforma ETSI M2M

NSCL in the ETSI M2M Platform

Universidade de Aveiro
Departamento de Eletrónica,
Telecomunicações e Informática
2014

Ilan Emanuel Moreira Pegoraro

NSCL na Plataforma ETSI M2M

NSCL in the ETSI M2M Platform

"*The greatest challenge to any thinker is stating the problem in a way that will allow a solution*"

— Bertrand Russell

Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

Ilan Emanuel Moreira Pegoraro

NSCL na Plataforma ETSI M2M

NSCL in the ETSI M2M Platform

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 do Doutor João Paulo Silva Barraca, Professor assistente convidado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Diogo Nuno Pereira Gomes, Professor auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho aos meus pais e irmão pelo incansável apoio.

o júri / the jury

presidente / president

Prof. Doutora Susana Isabel Barreto de Miranda Sargento
professora associada da Universidade de Aveiro

vogais / examiners committee

Mestre Ricardo Azevedo Guerra Raposo Pereira
product manager, PTIns

Prof. Doutor João Paulo Silva Barraca
professor assistente da Universidade de Aveiro

agradecimentos /
acknowledgements

Gostava de agradecer em primeiro lugar ao meu irmão e aos meus pais pelo apoio incondicional que me deram durante a realização deste trabalho, e a todos os meus amigos e colegas que de forma direta ou indireta me apoiaram durante a realização deste ano de trabalho.
Queria agradecer ao meu orientador, o Professor Doutor João Paulo Barraca e ao coorientador o Professor Doutor Diogo Gomes, pela oportunidade de realizar a dissertação no grupo do ATNoG, e pela orientação durante todo o curso desta dissertação e pelo conhecimento que me ajudaram ao obter.
Em último, mas não menos importante, gostaria de agradecer a todos os elementos e amigos do grupo ATNoG pelos bons momentos e por um bom ambiente de trabalho. Um muito obrigado a todos.

Palavras Chave

Maquina-a-Maquina, Internet das Coisas, Computação Ubiqua, Inteligência Ambiental, ETSI, NSCL, Plataformas Horizontais.

Resumo

A evolução dos dispositivos do dia a dia em dispositivos inteligentes capazes de reagir ao ambiente que os rodeia está a permitir a criação de novas aplicações que visam revolucionar a industria. Atualmente tem-se dado muita atenção a estandardização da Internet das Coisas e comunicações máquina-a-máquina, com o objetivo de construir uma fundação interoperável que permitirá o crescimento futuro da Internet, onde os dispositivos irão comunicar sem, ou com mínima, intervenção humana. Nesta dissertação é apresentado em primeiro lugar requisitos como heterogeneidade, escalabilidade, endereçamento e a primeira abordagem feita pelo standard M2M do ETSI. Consequentemente, é a apresentada a visão e a arquitetura e o trabalho realizado nesta área. Por fim é apresentada a implementação da componente de rede realizada nesta dissertação juntamente com os respetivos testes.

Keywords

Machine-to-Machine, Internet of Things, Ubiquitous Computing, Ambient Intelligence, ETSI, NSCL, Horizontal platforms.

Abstract

The evolution of every day gadgets into smart-devices able to react to their surrounding environment is enabling the development of novel applications aiming revolutionize the industry related to this technology. Currently much attention has been given to standardizing IoT and M2M in order to build an interoperable foundation that will enable the growth of the future Internet, where devices will communicate without, or at least minimizing, human intervention. In this dissertation is presented in first place issues such as: heterogeneity, scalability, addressing and the first approach taken by the ETSI M2M standard. Subsequently, is presented the ETSI M2M vision and high-level architecture together with current work in this area. Finally an implementation of the network middleware is going to be presented along with further testing.

Contents

List of Figures

List of Tables

Acronyms

3GPP	3rd Generation Partnership Project
6LoWPAN	IPv6 over Low Power Wireless Personal Area Network
AAA	Authorization, Authentication and Accounting
AmI	Ambient Intelligence
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CA	Certification Authority
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful environments
CPU	Central Processing Unit
CRL	Certificate Revocation List
DA	Device Application
DARPA	Defense Advanced Research Projects Agency
DCN	Data Collection Network
DSCL	Device Service Capability Layer
DSN	Distributed Sensor Network
EAP	Extensible Authentication Protocol
EC	European Commission
ETSI	European Telecommunications Standard Institute
GA	Gateway Application
GBA	Generic Bootstrapping Architecture
GIP	Gateway Interworking Proxy
GPS	Global Positioning System
GRAR	Gateway Reachability, Addressing and Repository
GSCL	Gateway Service Capability Layer
GSec	Gateway Security
GSM	Global System for Mobile Communications
HMAC	Hash-based Message Authentication Codes
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IoT	Internet of Things
IPC	Inter-Process Communication
IP	Internet Protocol
IPU	Interworking Proxy Unit
IPv6	Internet Protocol Version 6
ITU-T	International Telecommunication Union - Telecommunication
JMS	Java Messaging Service
JSON	JavaScript Object Notation
Kmc	M2M Connection Key
Kmr	M2M Root Key
LWM2M	LightweightM2M
M2M	Machine to Machine
MAC	Message Authentication Codes
MAS	M2M Authentication Server
MCU	Micro Controller Unit

MQTT	Messaging Queuing Telemetry Transport
MSBF	M2M Service Bootstrap Function
NAE	Network Application Enablement
NA	Network Application
NAT	Network Address Translation
NCS	Network Communication Selection
NFC	Near Field Communication
NGC	Network Generic Communication
NIP	Network Interworking Proxy
NRAR	Network Reachability, Addressing and Repository
NREM	Network Remote Entity Management
NSCL	Network Service Capability Layer
NSec	Network Security
OA	Organizing Agent
OCSP	Online Certificate Status Protocol
OMA	Open Mobile Alliance
ONG	Object Network Gateway
P2P	Peer-to-Peer
PANA	Protocol for Carrying Authentication for Network Access
POJO	Plain Old Java Object
PSK	Pre-Shared Key
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RPC	Remote Procedure Call
SAP	Service Access Point
SA	Sensing Agent
SASL	Simple Authentication and Security Layer
SCADA	Supervisory Control and Data Acquisition
SCL	Service Capability Layer
SC	Service Capabilities
SDO	Standards Developing Organization
SOA	Service-Oriented Architecture
SP	Service Provider
SQL	Structured Query Language
SRD	Short Ranged Devices
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TC	Technical Committee
TETRA	Terrestrial Trunked Radio
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WSDL	Web Services Description Language
WSN	Wireless Sensor Network
WWW	World Wide Web
XML	Extensible Markup Language

CHAPTER 1

Introduction

The Internet is once again evolving, we're entering a new area of computing that many call Internet of Things (IoT) and M2M. Today ours cellphones, tablets and computers are already connected to the Internet allowing us to exchange information between them. Soon a whole new range of devices will also be connected to the Internet and able to exchange information. This vision has been known by various names like ubiquitous computing or Ambient Intelligence (Aml), in which technological gadgets (from now on referred to as objects or things) are integrated within our lives, cooperating with other things and helping human beings carrying out their everyday tasks. In this paradigm the objects are completely connected (through wireless or wired connections), constantly available, embedded with a Micro Controller Unit (MCU) and with the ability to transfer data over the Internet, with the sole purpose of communicating information and making it easily available. Although, very similar to the idea of the IoT, previous attempts in this area were tightly-coupled to the manufacturers, i.e., communication between devices was isolated within the manufacturer's domain, fragmenting the market and preventing interoperability and the sharing of information. Consequently, a new platform that would support many different types of devices and allow their integration, became a necessity.

The recent advances in sensing and actuation technologies and wireless communications has given M2Ms an impulse for attention. The essence of M2M lies in the way the "Things" can establish a communication channels, and bidirectionally exchange information without human intervention. It is expected that this new network will be able to support billions of smart objects, producing massive amounts of data that will be intelligently processed into real actions.

Moreover, given the ramifications these platforms will have within our society, advancements has been made in the last years to provide a standardization for M2M systems. Furthermore, the world's most prominent Standards Developing Organizations (SDOs) have started to come together to stop the fragmentation in this area, thus a new global standard for M2M systems, called oneM2M, was created.

1.1 MOTIVATION

A growth in M2M communications is expected in the future due to recent advancements in technology. The miniaturization of electronic components enables the integration of more of those components into "pocket devices" offering more features per device. Together with the evolution of the broad address space offered by Internet Protocol Version 6 (IPv6), is allowing direct connectivity between all devices and consequently their integration into the public network, the Internet. Once all every day objects are integrated within this future Internet, the possibilities for novel applications will be endless, applications ranging from human health related (eHealth) to domestic and industrial automation and smart power grids, will bring many advantages to both network carriers and our society itself. Although to achieve such a future, a consensus much be reached to cope with the heterogeneity of networks and devices, interoperability between multiple vendors, and an easy deployment of applications.

In order to provide the aforementioned characteristics, European Telecommunications Standard Institute (ETSI) has devoted resources in the standardization of a service layer to bridge the gap between the devices and the networks. In this standard, ETSI defines the resources' data structure, security mechanism, and the messages and interfaces that allows communication between entities in the M2M ecosystem.

1.2 OBJECTIVES

This dissertation covers the study of the ETSI M2M standard, more specifically the study of the network domain. Additionally, an implementation of the network middleware was also achieved. Given the complexity of the standard, it isn't the goal to implement all the capabilities defined in it, but give priority to the most critical capabilities in order to have a working NSCL at the end of this dissertation work. Moreover, work done, besides connectivity management, it also focuses on the authorization and authentication of devices attached to the network, as well as forwarding the information produced, thus allowing application deployed on top of this platform could leverage this rich source of information. This component also requires to be scalable.

1.3 STRUCTURE

The remainder of this document has the following structure: Chapter 2 starts by looking at the state of the art, it analyzes the concepts like Wireless Sensor Network (WSN), IoT and M2M. The end of this chapter is devoted to the introduction of the firsts solutions in this area.

Chapter 3 is devoted to the introduction and analysis of the standard proposed by ETSI for M2M systems. The chapter begins with an overview of the organization and the vision that led to the creation of the working group that drives the development of the M2M standard, following with the study of the architecture, and ends with a presentation of the current projects that leverage this standard.

Chapter 4 follows with the study of the components that belong in the network domain. The chapter introduces the network domain framework, followed by a description of each service capability

in detail.

Chapter 5 starts with an explanation of the objectives that would led to the development of a working NSCL. Additionally, an explanation of the architecture that was conceptualized in the context of this dissertation, along with an explanation of how each component are linked together to form the ETSI NSCL.

Finally, Chapter 6 presents test cases made to the NSCL to see how it performs under pressure, it also presents the results taken and gives an analysis to those same results. While analyzing the results some comments on how to optimize the performance of the NSCL is also presented.

CHAPTER 2

State of the art

2.1 Evolution to smart objects

Technological advances in wireless communications, Radio-Frequency Identification (RFID), Near Field Communication (NFC), Global Positioning System (GPS), sensors and actuators had changed the way IoT is perceived by giving birth to a whole new range of applications. IoT has grown from a technology that offered machines an easy way to talk to each other to a revolution that may come to change the world we live in.

The widespread coverage of cellular networks and rapid transmission rates, along with low cost of deployments, will further drive rapid growth in the number of devices that will be connected to the IoT. Moreover, is not just a matter of connecting devices to the network, they need to be adapted to be useful in this new revolutionary world. One commonly used word related with IoT is the term smart. For an object to be considered smart it must incorporate a processing unit (normally a processor or MCU), be context aware (the ability to sense the environment around it), be equipped with network capabilities and be uniquely addressable, thus becoming an autonomous physical object. In contrast to conventional objects, this new kind of devices are also combined with application code that, besides of sensing of their environment, they can act on their own, intercommunicate with each other and even interact with people. This new and exotic devices are the building blocks of the IoT, and will eventually, integrate within our lives to develop an environment that is sensitive and responsive to our needs, helping us carry out our daily tasks, thus improving our quality of life, just as was the vision of Weiser [1] which was named pervasive computing at that time - “The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it”.

Currently, when one speaks of the IoT, many areas comes to mind like: energy management, medical and health care systems, home automation, transportation and even entertainment. This clearly states its prosperous future.

2.2 WIRELESS SENSOR NETWORKS

WSNs [2] are networks composed of spatially disperse and dedicated sensors that capture and monitor specific variables of the environment, like temperature, pressure, humidity and sound, and transfer that sample data (obtained from the sensor) through the network, to a centralized point. Each node (also known as mote) is equipped with one or more sensors in addition to a MCU and a wireless transceiver and a energy source (typically a battery). Given the flexibility offered by wireless networks, these devices can be placed almost in any location, obviously within range of one another.

The development of WSN was primarily motivated by military applications, back in 1980 when the United States Defense Advanced Research Projects Agency (DARPA) started a project named Distributed Sensor Network (DSN) [3]. Nevertheless, given the challenges it imposes in the implementation of a distributed sensor network, the project started to migrate into the academia, seeking the expertise of universities. Eventually, the WSN made its way into the corporate world. Today it is commonly used in other areas like: the studying of animal natural habitat, record of traffic patterns, agriculture greenhouses and climate control, and is slowly evolving into what's known today as IoT and M2M [4].

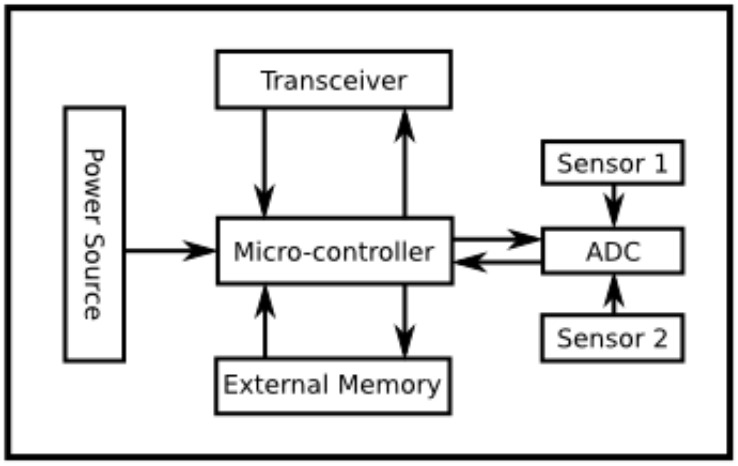

Figure 2.1: Mote's architecture.

Figure 2.1 depicts the common architecture of a mote. A mote, as was mentioned above, is a WSN's node that is composed with sensory functionality, the ability to perform computations and communicate with other nodes in the network, and has a power source. In addition, motes can also incorporate actuators in order to interact with the environment around it. As consequence of their limited power source and physical dimensions, motes are usually constrained, i.e., not as powerful as other devices.

2.3 INTERNET OF THINGS

2.3.1 CONCEPT AND ORIGINS

The phenomenon behind IoT is not new, even though lately it has been giving too much attention, the concept goes back, at least, a decade where it was known as ubiquitous computing or ambient

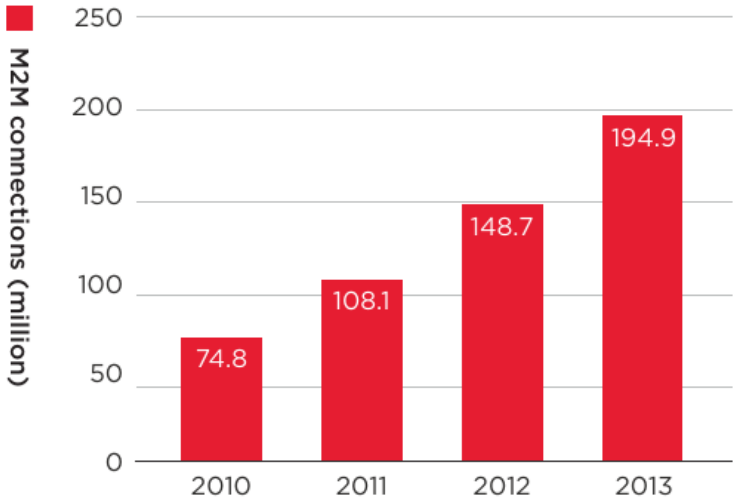

Figure 2.2: M2M global connections [6].

intelligence, and it has been applied in areas like power stations and water utilities. However this are custom implementations, specific to the task they were programmed to do, without any sharing of common functionalities or standardization. Consequently they were isolated and evidently not connected to the Internet, or other public network for that matter. IoT obviously brings new functionality and not just a new name, it represents the vision of a network of devices that collect data from the world and share it through the Internet, mainly without human intervention, and where it can be later processed and used by others, with the goal of enriching it into valuable information or knowledge. In addition, within this new paradigm, and in contrast to previous approaches, IoT will leverage M2M communications, where information will flow from machines to machines, reducing human intervention, with the intent of having access to the information directly from devices, since human intervention would only delay the process.

The IoT is starting to infiltrate our society with wearable devices, televisions (all consumer electronics) and even cars. Is expected that the number of devices, that will be part of this movement, will raise in the coming years (see Figure 2.2), and also is expected a growth in data that is generated by these devices, for instance a Boeing jet generates up to 10 Terabytes of information per engine every 30 minutes of flight, given that there are, roughly, 28.537 commercial flights in the United States of America, the amount of information generated could easily scale up to petabytes scale[1] [5]. Consequently, organizations are starting to see opportunity for Big Data application in the IoT, since is needed to extract meaningful information from this massive amounts of data [6].

Moreover, given the ubiquity of the devices, mobile operators are reporting an increase in the demand for M2M connectivity using mobile networks, therefore they are offering M2M services across a wide range of vertical industries to satisfy the market demands. In Figure 2.3 is shown the percentage of mobile operators offering M2M services.

[1] A Petabyte is 10^5 bytes of digital information.

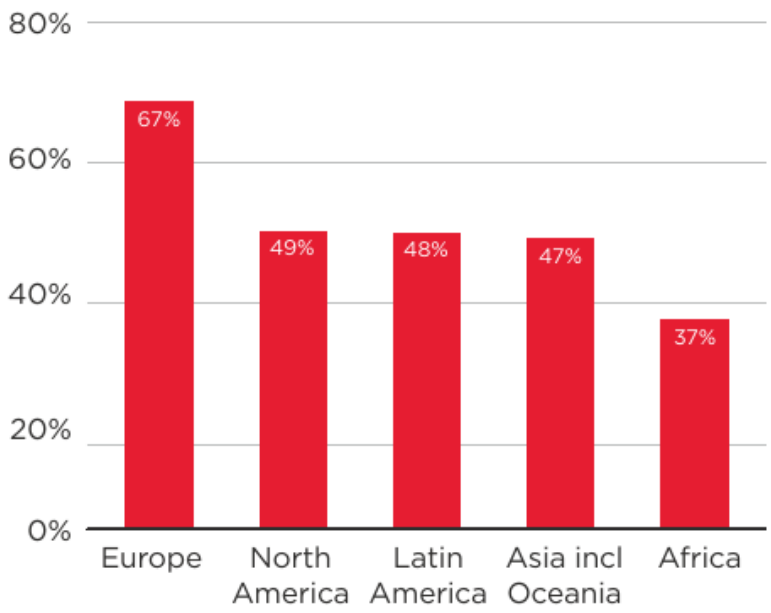

Figure 2.3: Mobile operators offering M2M services [6].

2.3.2 MACHINE-TO-MACHINE

M2M, just like IoT, is not a new concept, communication between machines without human intervention has been happening for a while now in applications like telemetry, industrial engineering and Supervisory Control and Data Acquisition (SCADA) systems. However, only recently M2M has expanded beyond one-to-one communications into a network of things, therefore its integration with IoT. This evolutionary step towards IoT is enabled by M2M, because the "things" are able to communicate autonomously, give meaning to the information and take automated actions. By taking human intervention out of the equation, information is gathered quickly and efficiently so that actions can be taken in real time [7]. Thus, lately deployment of M2M networks has been receiving much attention because it could influence both industrial and domestic areas [8].

In order for M2M to accommodate the IoT's demand and its large scale adoption, global deployment of the IPv6 is needed to support the extremely large unique address space of the devices (both sensors and actuators) [9]. Providing an Internet Protocol (IP) address to each device, they become Internet-aware and allows for end-to-end communication without the need to relay on gateways. In addition to the large address space, the protocol offers other advantages, namely, the IPv6 provides security mechanism for end-to-end communications and an improved header compression. Furthermore and giving the constraint nature of devices in M2M networks, the Internet Engineering Task Force (IETF) has created a specialized working group for the development of IPv6 over Low Power Wireless Personal Area Network (6LoWPAN) standard. This standard provides an adaptation layer for the 802.15.4[2] data link layer to integrate well with the IP stack. 6LoWPAN offers a compressed and compatible IPv6 header optimized for constraint devices [10].

Summing up, M2M is the key enabler for the future Internet, i.e., in the sense that M2M is the backbone that drives the IoT capabilities, so that it provides manifold advantages – e.g. incremental

[2]The Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 is a standard that describes the physical layer and media access control for low-rate wireless communications. An example network that uses this standard is ZigBee.

network traffic and revenues for the carrier, new applications for developers and systems integration opportunities, efficiency and productivity gains for the end-user. Hence it is expected that the number of devices will grow rapidly in the coming years, more specifically by 2020 there will up to 50 billion devices connected to the Internet, compared to previous years which was at 50 million in 2008 and 200 million in 2014 [8], this growth is what has motivated the world SDOs to standardize M2M communications.

2.3.3 USE CASES

The ever shrinking hardware and ubiquitous connectivity is supporting the various scenarios where IoT is expected to flourish. One common scenario is related to location aware devices, for instance smart-phones, smart watches, or more specifically the Google Glasses. These devices use their embedded sensors to report back to the system their location, velocity or motion, in order to obtain services that may be of interest to the device owner, services ranges from applications that receives publicity if the person is near a shop, to sport applications that can measure certain variables about the person's workout. While the described scenarios are focused on the end-user, IoT is also expected to have an impact in industrial applications like smart metering, smart grid, and fleet management. This section is devoted to introducing some use cases applicable to IoT.

HEALTH CARE

One category where IoT can make a difference is in health care [11]. Patients with diseases like cancer, heart or respiratory disease, that need constant monitoring and cannot afford to travel to the doctor's office, should not put their life at risk. For that reason, many start-up companies are starting to develop devices and applications to address this problem. By using such products, patient's health is constantly updated in real time to their doctor or caretaker, therefore an alarm could be triggered in an emergency, and the patient is not putting their well being at risk.

Another scenario related to health care is ambient assisted living [11]. In the case of a person that is taking care of an elderly relative and cannot afford a nursing home. Naturally it cannot abandon his/her job to take care of a relative, still their relative needs constant monitoring. Consequently, IoT can be exploited to create applications that can monitor the heath and activity of the person inside their home, and alert the caretaker and an emergency contact, whenever any issues arises. Afterwards home appliances can be equipped to help identify future needs, for instance, a refrigerator could easily identify when certain products were due for replenishing. The system would send a shopping list to the caretaker or place an order to a home delivery grocery service.

To summarize, these use cases shows a promising future for IoT, and at the same time, improves the quality of life of both the caretaker and the patient.

TRANSPORTATION

Within transportation there are two subareas that could greatly benefit from M2M communications: smart roads and smart cars thus achieving an intelligent transportation system [12]. A smart road is set to be equipped with sensors to measure traffic, which enables the synchronization of traffic lights

in order to reduce road congestions[3]. The same data gathered by these sensors could be sent to the web and be accessible to the public, through services like Google Maps, TomTom Congestion Index[4] or even to cars itself, so that drivers could make informed decisions about traffic situations, i.e.: which roads to take and which roads to avoid. Additionally, the data gathered from these smart roads could deliver other services like an alternative mean of transportation that would result in a faster travel.

Furthermore, the other topic in the realm of transportation is the so called smart cars (or connected cars). A connected car [13], [14] is said to be connected to the Internet and able to communicate between cars (vehicle-to-vehicle) or with roads or GPS satellites (vehicle-to-infrastructure) to transmit and use traffic information. With such technology, navigation system can offer alternatives routes, due to the real time access to this data, in order to avoid the traffic jams or accidents, since they impact they quality of life of the citizens. More importantly, in case of an accident the car could assess the gravity of the accident and send an automatic message with the gravity and location to emergency contacts along the road.

The IoT is also helping cities, airports, and malls, by embedding sensors in their parking lots so that drivers don't waste their time and fuel for searching for a place to park.

AGRICULTURE

Agriculture is a complex and important industry since human beings cannot live without eating. The importance of this sector has allowed IoT to set roots in it, by deploying a sensors and actuators in the field (crops) not only with the purpose of gathering data such as: soil humidity, temperature, ambient light conditions, pest infections for future analysis, but also to take automated actions whenever needed. For instance, the irrigation system may be remotely activated when the weather forecast foresees that it will not rain, or in case of rain prediction it can be delayed thus saving costs and water.

One such project was Apollo, which was developed in the cooperation between Portugal Telecom and the Instituto de Telecomunicações of Aveiro, [15]. One of the test case scenarios was based on the deployment of a WSN on a greenhouse, in order to gather information related to: different stages of plant development, optimization of the cultivation process and also the automation of various tasks.

WEARABLES

A smart watch is a computerized watch to enhance the functionality of the traditional watch. With early models trying to go beyond the basic functionality of just keeping track of the time, a smart watch is able function as a portable media player, playing video games and running mobile apps. The device is equipped with many sensors and communication technologies, allowing it, for instance, to remind the person of important events like the next meeting.

Another kind of wearable devices are the activity trackers. An activity tracker is such device that may be continuously capturing heart rate patterns, calories expenditure, and other variables as long as it is attached to the persons' wrist.

[3]An example of such a smart road was constructed in Virginia, United States. https://www.vtti.vt.edu/smart-road/virginia-smart-road.html.

[4]http://www.tomtom.com/en_gb/trafficindex/

SUMMARY

This is just the beginning, we are just merely scratching the surface of the possibilities allowed by IoT, more and more use cases are arising, taking it a bit further, therefore infiltrating every aspect of our society. Although it seems limitless the value offered by IoT, more attention should be giving towards the education and standardization for the IoT, if the goal is to achieve a smart world. In [16] are shown 50 more use cases where is expected a revolution.

2.3.4 FIRST SOLUTIONS (VERTICALITY)

With the appearance of WSN, development of applications that would interact with such networks of devices was common. Usually, with these simple cases, many applications were embedded with the logic of talking directly to the hardware, using proprietary protocols, instead of relying on a layer of abstraction. Repeatedly, applications were burden with specific protocols and data formats, in order to take advantage of the data available in the network, therefore forsaking important aspects like interoperability, re-usability, scalability and easy maintainability. In addition, since the manufacturer controls the whole system, the end user is dependent on the manufacturer for improvements and upgrades. To this end, the vertical model makes it harder for the end user to manage the system, when it provides a variety of tasks (multiple manufacturers), because each individual part of the system has its own gateway (for accessing the network) and service operations, therefore the user must be familiar with both manufacturers.

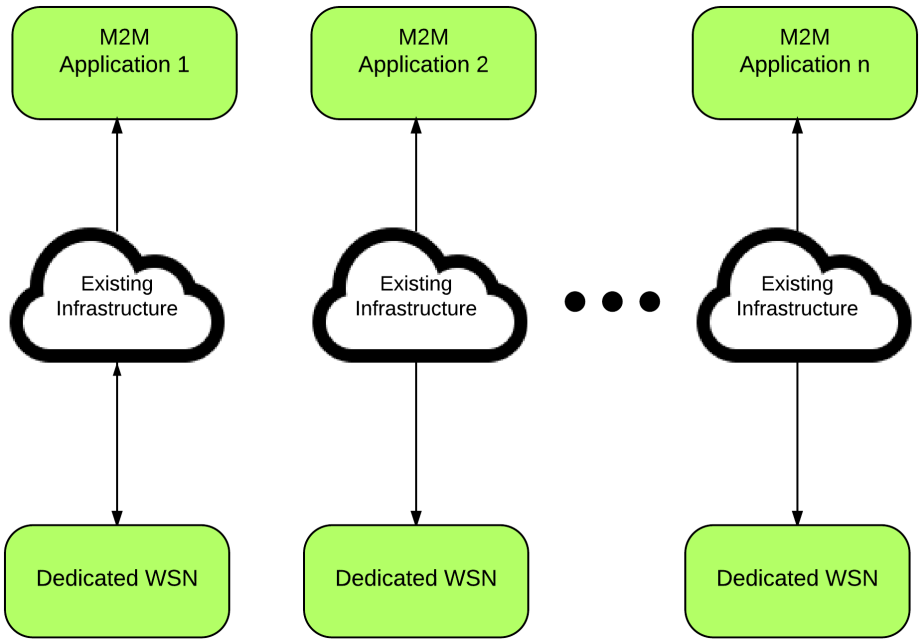

Figure 2.4: Example deployment of vertical solution.

Figure 2.4 shows an example deployment of a vertical solution, as it can be seen, each vendor has to implement their own infrastructure, in order to have connectivity with the network of devices.

To conclude, the simplicity of this approach may be tempting at first sight, although for complex systems, like IoT, it will not suffice. More research is needed to satisfy the requirements and growth of IoT and M2M. The following section will introduce an evolutionary approach, horizontality.

2.3.5 EVOLUTION (HORIZONTALITY)

While the applicability of vertical solutions have survived this far, given the complexity and heterogeneity demanded by next generation of services (see section 2.3.3), has forced organizations, like the ETSI, to search for a long-term solution to the problems faced by vertical solutions, like isolation of the network of devices and the lack of scalability, i.e., strong coupling between the network and application layer. In contrast, horizontal architectures try to mitigate these problems by dividing into two separate groups, one where the devices lie and the other being solely for the network communication, while, at the same time, having the providers working on a common framework. By decoupling the applications from the network specificities, the system becomes more manageable, i.e., easy to maintain and to scale, since there's no need to keep implementing network logic to every new sensor or actuator that is added. Equally important, this approach has the advantage for new innovators to focus on the real task, i.e., assuming that lower level functionality is already in place instead of reinventing the wheel. What's more to this architecture is the ability of re-using already working building blocks.

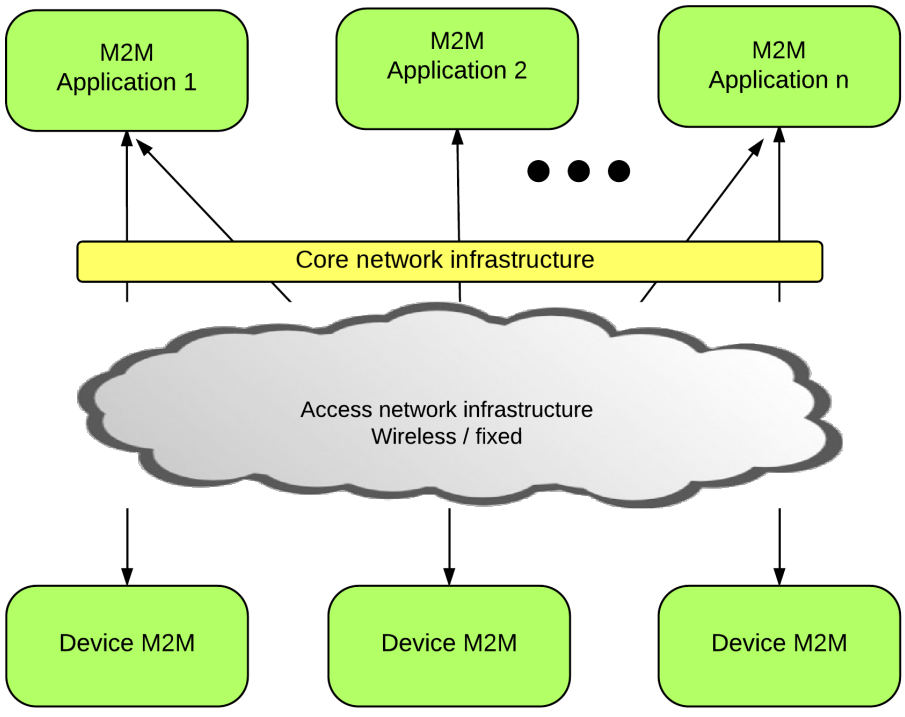

Figure 2.5: Example deployment of horizontal solution.

In Figure 2.5 is represented graphically the deployment of a horizontal solutions. As was already mention in this section, and what is shown in the figure, the network infrastructure along with common functionalities are implemented only once, and are share by the vendor's applications running on top of this infrastructure, which is evidently the opposite of the vertical solution previously addressed.

Summing up, IoT requires the realization of an agreed upon architectural reference model, based on open protocols and services that favors both interoperability and re-use, to ensure a successful deployment of IoT across the heterogeneity of domains, in order to allow rapid proliferation of new applications and businesses.

2.3.6 RELATED WORK

This section presents a description and brief analysis of some of the most relevant projects in this area. The platforms presented try to mitigate the issues that emerged with previous vertical solutions. Those are heterogeneity of devices - different devices may eventually require different communication protocols; scalability - the platform must support a vast number of devices concurrently, without a notable decrease in performance; a standard model for storing data coming from the devices; and most importantly to be easily configurable - that is the user can tweak device and the system itself with ease.

The following sections will present some platforms that meet the requirements defined above.

IRISNET

IrisNet (Internet-scale Resource-intensive Sensor Network Services) was the first project to provide services on top of a WSN [17]. It envisions a global system with many heterogeneous devices, namely sensors but also PC-class nodes, that forms the sensing network, that is widely distributed and connected to the Internet and whose data is later consumed by third party applications, in order to meet the users demands.

IrisNet is composed of a two-tier architecture (Figure 2.6), one tier is for accessing and retrieving the information directly from the sensors in a unified and generic way, which the authors named: Sensing Agents (SAs). The goal of these SAs is to hide the specificities of the different sensors from the application developers. The other tier is meant for data storage produced by these devices. This distributed storage is called Organizing Agent (OA). The following paragraph will introduce each of tier and their purpose within IrisNet.

OA tier. As mentioned in the above paragraph, an OA is responsible for the storage and organization of the incoming sensing data. In addition, multiples OAs are grouped together to create a service that is able to answer queries relevant to a particular use case. Typically an OA runs in a terminal and is only able to participate in one service, although a terminal is able to run multiple instances of different OAs in order to provide fault tolerance and load-balancing. Within each OA the information is stored in the Extensible Markup Language (XML) data format since, in the authors' view, it is the best way to hierarchically organize information. Moreover each service author must provide its own database schema in order to identify the portions of the database that belongs to the service. Since the OAs are disperse in various terminals, a subset of the database is also distributed among the OAs. Given the distributed nature of the database, IrisNet provides fast access and correctness to user queries, by caching mechanism and routing the queries to the appropriate nodes. Evidently, the usage of XML, as the data source, allows service developers to use XPATH, as the mechanism to access the information and to apply filters.

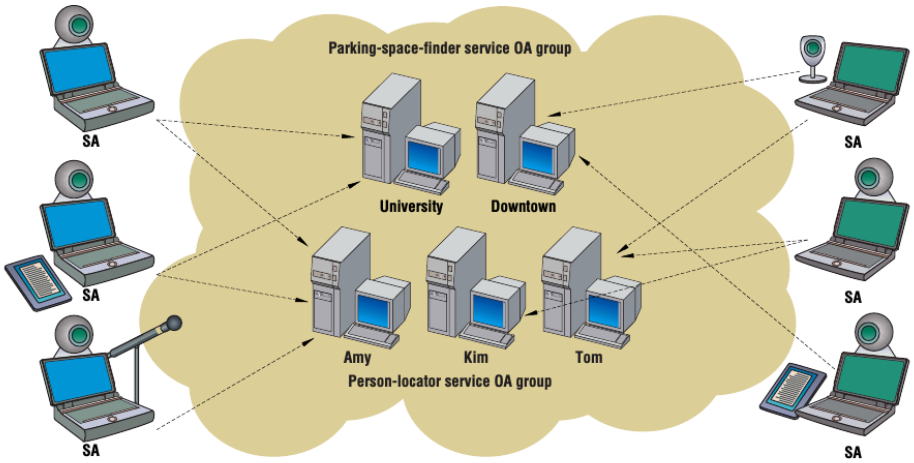

Figure 2.6: IrisNet two-tier architecture [17].

SA tier. In this tier, each SA is able to capture data from various sensors, and also provides an execution environment where application code, known as *senselets*, can run and operate over the sensor readings before sending them to the OA. The SA can run as many senselets as the number of services that have registered to access its sensor feed. In addition, since a single SA can run multiples senselets, it provides a mechanism to protect itself and other senselets from malicious code, by running them in separate processes, or by running them inside a virtual machine, which in this case may also limit resource usage. Lastly, IrisNet distinguishes senselets in trusted and untrusted. This distinction allows the system to enforce privacy by sending raw data feed to the trusted senselets, meanwhile the untrusted counterpart receives a filtered data feed.

Prototype services. The authors refers to three prototype projects that leverage IrisNet. The first prototype service tells a driver the nearest parking lot with an available parking slot. The second is a network monitoring tool that collects data from host and allows users to query this data efficiently and remotely. Lastly, the third project allows to monitor coast shores and search for anomalies.

HOURGLASS

The Hourglass[5] project [18] is based on a peer-to-peer architecture in which humans users, intelligent agents and computers interacts with WSNs. The project aims to provide a scalable and robust Data Collection Network (DCN), which is an infrastructure that not only enables integration between heterogeneous WSN, but also addresses problems like network failures, discoverability, addressability, routing, querying and delivery of sensor sample data. In addition it is also capable of handling intermittent communications with the usage of buffers. These buffers store information during connection outage, that should be sent after connection is resumed. Roughly speaking, the Hourglass infrastructure enables large numbers of applications to extract data, from geographically-diverse sensor networks, that are connected to the Internet. The path that ensures an application receives the data it is interested in is called "circuit". A circuit besides of being a virtual pipe between the data producer

[5]Project's web site: http://www.eecs.harvard.edu/~syrah/hourglass/index.shtml

(mostly sensors) and a data consumer (the application), which enables data to travel along the network, may also include programs that may alter the data. These programs are named intermediary services. Therefore applications doesn't need to be burden with the creation of the data flows, that is the DCN's responsibility.

Hourglass classifies these services into individual Service Providers (SPs), which can be thought of as administrative domains. The structure of each SP is composed of the following two elements: a registry - which aids service discovery, and a circuit manager - whose responsibility is the setup and management of circuits. Figure 2.7 presents an example deployment of Hourglass.

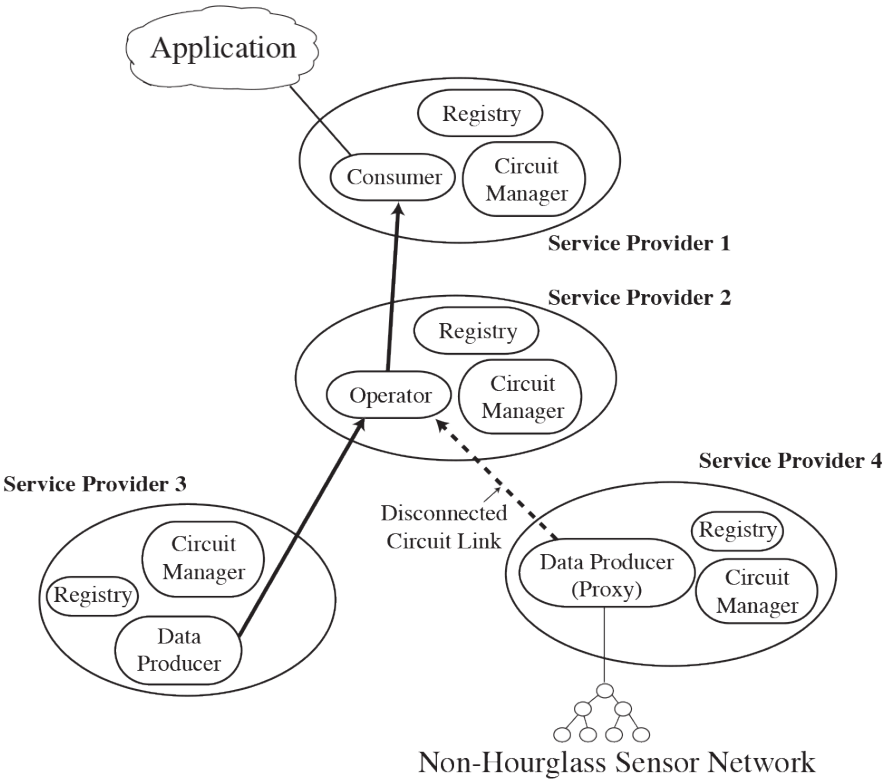

Figure 2.7: Hourglass example architecture [18].

In conclusion, Hourglass presented an architecture that can coupe with the heterogeneity of connections and intermittent communications, but as stated in [19], problems with scalability may arise, given the fact that the state of each circuit has to be maintained in all of the nodes of the circuit and also because circuits are created upon request of the application that wishes to use the service. This poses no problems with long lived streams, but with short lived streams may not be appropriate.

WEBDUST

The WebDust [20] project is a Peer-to-Peer (P2P) platform for managing and monitoring various WSN of heterogeneous devices, using a web-based interface that allows the administration and visualization of network state. The software architecture of WebDust promotes the implementation of customized applications. The architecture defines 3 domains: a) *peer-to-peer* network - applications that run it desktop computers, b) *nano-peers* - which are composed of the sensors and the wireless transport medium, c) *gateway peers* - are the nodes that have access to the WSN and allow the

interaction between the actual peer-to-peer and the nano-peers. Moreover, and in order to achieve the distributed nature desired by the authors, WebDust uses an event driven model, i.e. actions are triggered as a response to the various kinds of events that take place on the system.

The fundamental part of the architecture are the so called *Gateway Peers*. These devices, besides their sensing and monitoring capabilities, are able to abstract the existence of the WSN from the top layers and consequently achieving better scalability. Giving the abstraction between the overlay computers and the WSNs, the gateway peers are responsible for routing the data from (and to) the devices. In addition, the gateway peers stores the data retrieved from the WSNs in a relational database.

Each peer, of the P2P network, is organized in a two layer architecture: the first layer is called the inner layer, while the second is called the outer layer. The inner layer is characterized for maintaining information related to the sensor network, the device's specifications and the results of the queries already performed. Consequently, it is within this layer that all the functionalities used by the higher level services are defined, and to that end the inner layer, sometimes is also referred to as the core. On the other hand, in the peers' outer layer are executed the high level services leveraging functionalities of the inner layer.

Furthermore, the WebDust architecture still offers more relevant services that are worth mentioning, such as: the aggregation of data from a set of sensors, a buffering service that offers a temporary storage to store sending data while on connection failures, therefore data is not lost when such an event occurs and can be sent when the connection is reestablished, finally a monitoring service is also available to measure the network's overall performance.

Figure 2.8 depicts a overview of the WebDust architecture.

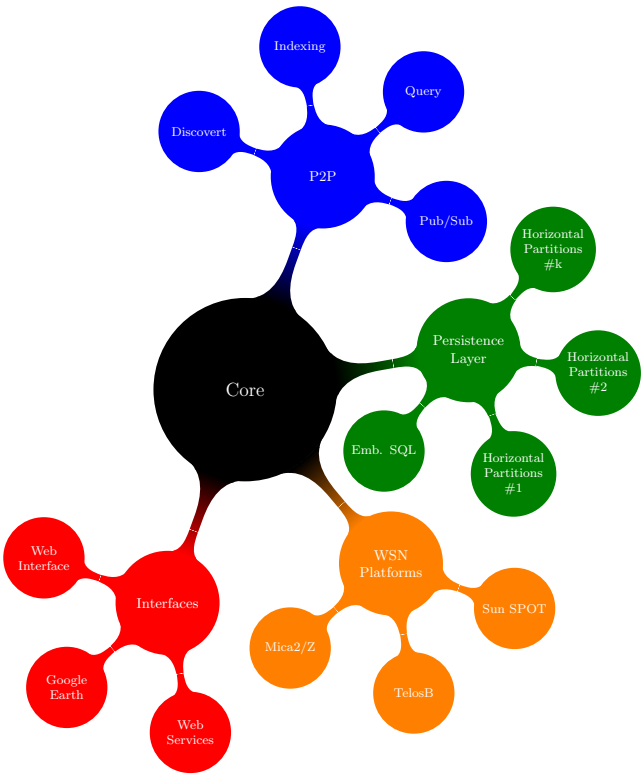

Figure 2.8: WebDust's architecture overview [20].

E-SENSE

The e-SENSE project [4] supported by the European Framework Programme (FP6) whose goal is to bridge the gap between Europe, North-America and Asia in the area of WSN platforms. During the project two years course, it contributed to standardization efforts to the Zigbee Alliance and IEEE 802.15.4, in order to produce quality specifications that later would become world known standards. Additionally, the project achieved its goal of creating a platform for the integration of WSNs worldwide.

e-SENSE offers a flexible architecture, not only offers deployment of WSNs over a ZigBee network, but also to other networks that correspond to the applications needs. Moreover, the architecture was conceived from the beginning with the intent of allowing the integration in smart environments, i.e., WSN in common objects such as cars, buildings and even in clothing.

Figure 2.9 depicts the architecture envisioned by the authors and is composed of 4 sub-systems that can communicate with others through the use of the Service Access Points (SAPs). One important characteristic of this architecture is that the communication between the sub-systems is not strict, thus any sub-system is able to communicate with any sub-system of the architecture directly.

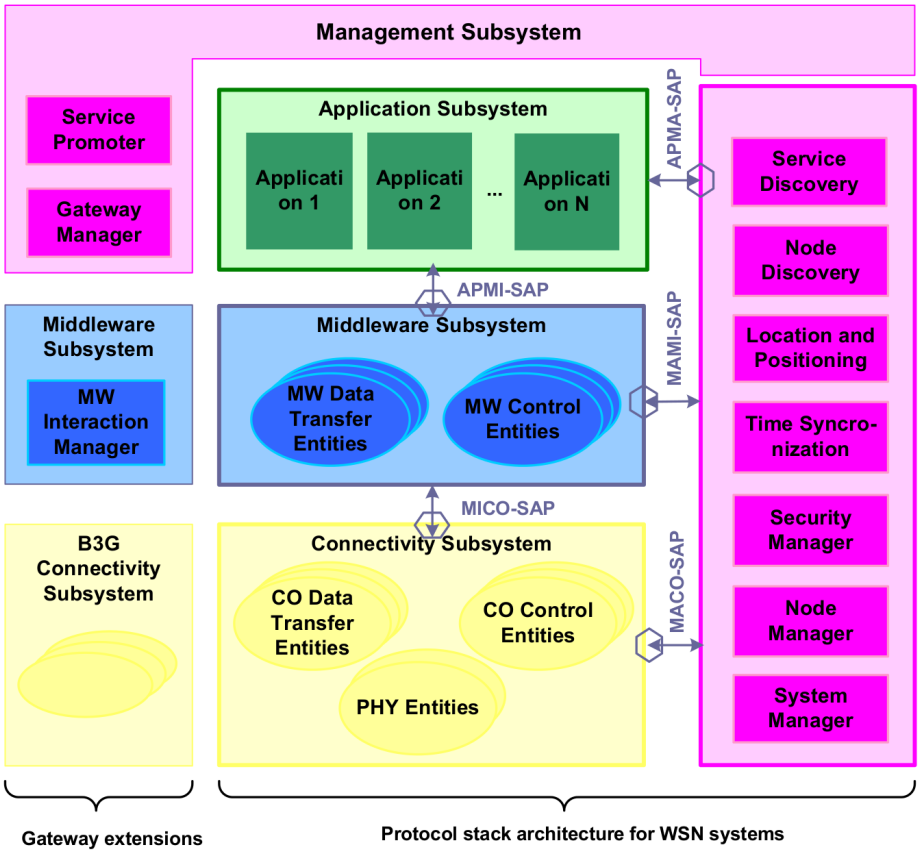

Figure 2.9: e-SENSE protocol stack architecture [4].

Connectivity (CO) sub-system. The CO subsystem is responsible for offering the functionality of network management, for the network that connects to the WSN nodes; data exchange between the nodes in the network and access control mechanism of the communication medium.

Middleware (MI) sub-system. The next sub-system is the MI whose purpose is to allow distributed processing of information originating from sensors in the WSN. In addition, within the MI data may flow in two ways, node oriented and data oriented. While in the former a WSN node sends the information to other nodes through the use of a network address, in the later data is transmitted through a publish/subscribe mechanism.

Management sub-system. Offers the functionality of configuring the CO and MI sub-system, to that end, it is used the SAPs in order to initialize and configure the sub-systems. Beyond the initialization and configuration functionalities, the management sub-system is also responsible for maintaining a database with the other sub-system information, the discovery of other nodes and services within the network, time synchronization, node and system management to ensure that both operate under normal circumstances.

Application sub-system. Lastly, the application sub-system is composed by the client applications. This applications leverage MI sub-system's functionalities with the purpose of receiving or sending data from and to the WSN.

e-SENSE promises a good architecture for the integration of WSN and provides functionalities such as service and node discovery, quality of service, system and node management. In addition, the architecture tries to integrate efficiently with mobile communication systems like IP Multimedia Subsystem (IMS).

ECLIPSE IOT

The Eclipse foundation has created a working group dedicated to help the advancements towards a unified and simple IoT, by reusing common standards that are already known in IoT, for instance, Messaging Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Open Mobile Alliance (OMA) LightweightM2M (LWM2M) and ETSI M2M, and to promote the usage of these standards. The foundation believes that the complexity associated with the IoT is given by the large number of communication protocols used in today's industries, ranging from obtaining the data out of the sensors and delivering it to a server in the Cloud, device management protocol that allows for remote firmware upgrades.

They propose an Open Source implementation, that uses the protocols mentioned above, to encourage their adoption and to improve the quality of these implementations.

Their contribution to IoT is based on the development of an Integrated Development Environment (IDE), built on top of the Eclipse[6] that ease the development of applications for M2M and IoT. According to [21], the following tools are important to support IoT:

- Embedded Development - to assist in the embedded development for the Lua language.
- Simulation - giving the complexity of M2M infrastructure, it must be able to allow simulation of M2M devices and of the communication scenarios.
- Server Development - to allow the discovery of the capabilities exposed by M2M servers as well as to configure them (for instance: the NSCL).

[6]Eclipse here refers to the already existent IDE use for development of JAVA programs

CHAPTER 3

M2M STANDARDIZATION IN EUROPE

This chapter is devoted to briefly look at the ETSI organization, and specifically to the work done with the specification made by the M2M Technical Committee (TC). Section 3.1 will present the vision and mission of the organization and the creation of the M2M TC, while in section 3.2 presents an introduction to the standard, and thus to the high-level architecture exposed by it. Finally, in section 3.3 shows the known real world implementations of the standard.

3.1 M2M IN ETSI

ETSI is a non-profit, independent, standardizing organization in the telecommunications industry, officially recognized by the European Commission (EC), develops standards by using and providing state-of-the-art methodology and processes. It was founded in 1988 and since then it has published many standards for key global technologies such as Global System for Mobile Communications (GSM), Terrestrial Trunked Radio (TETRA), Short Ranged Devices (SRD), and Internet technologies among many others. Even though it resides in Europe, it has worldwide projections for its standards.

In a complex industry, like the telecommunications industry, a need for standardization is a must-have characteristic, if the goal is to achieve interoperability, safety, and quality between the various vendors involved, i.e., a solid and consistent foundation. As of 2014 [22] there are 34 TC actively working in the areas of eHealh, mobile, satellite, security, M2M, smart cards among others.

Organizations like ETSI are constantly at the forefront of any new technological area, consequently, they are starting to realized that M2M systems are compromised because the lack of standardization (individual vertical implementations), efficiency, scalability, security, and the complexity involved in developing and deploying M2M applications. Consequently in late 2008, and after analyzing the importance and the impact the technology will have in the future, ETSI created a new TC for developing standards in the area of M2M communications. The aim was to provide a end-to-end ecosystem for M2M, while having a close relationship with ETSI's Next Generation Networks and also with 3rd

Generation Partnership Project (3GPP) for mobile communications, therefore putting the organization at the center of M2M.

In addition, ETSI goal besides the standardization effort, is to keep a close relationship with companies interested in M2M systems. Soon after (October 2010), ETSI began organizing a set of workshops and presentations to demystify M2M to newcomers, and also for testing different implementations that were starting to emerge.

Since July 2012, the world's most prominent SDOs came together and launched a new global organization called the oneM2M Partnership Project, to avoid creation of competing M2M standards [23]. OneM2M is working to unify the current M2M community by enabling the federation and interoperability of M2M systems, across multiple networks and topologies. They're starting to work in cooperation to avoid fragmenting the ecosystem with overlapping standards. In [23] is stated that there are approximately 270 partners and members actively collaborating in the project.

3.2 M2M STANDARD OVERVIEW

The ETSI M2M vision is based on a horizontal service platform, that favors the reuse of already proven standards, while being technology agnostic. This section provides an overview of the components being standardize, including the importance of the standardization efforts around M2M and the advantages it will bring to the ecosystem.

3.2.1 ETSI'S HIGH-LEVEL ARCHITECTURE

The architecture identifies a set of functional entities, which are divided in two domains: the Device and Gateway domain and the Network domain. There are also a set of reference points (which are explained in detail in section 3.2.7) that provides the communication capabilities between the two domains. In each of the domains, these functionalities are exposed over a service layer, named as Service Capability Layer (SCL), "to provide the functionalities for the management of interactions between entities (i.e. applications) involving communication across networks without requiring human intervention". The aforementioned architecture is intended to facilitate the implementation of vertical solutions, that will leverage this horizontal platform. Within the architecture information is represented as resources, which are shared memory blocks that applications can used for data exchange or events, throughout the M2M network using of the reference points. In addition, information is organized in a hierarchical resource-based structure, where each resource is uniquely identified by a Uniform Resource Identifier (URI), and is accessible following a RESTful model. Resources contain the data itself, and also contain metadata (in ETSI's nomenclature is defined as attributes) that provides information like the access and modified times, access rights (policies), and other relevant information. Besides the more obvious attributes such as creation and modification time, the standard also provides a mechanism for resources to be easily discoverable, i.e., they can incorporate a special kind of attribute, called the *SearchStrings*, that allows the SCLs to apply certain pattern matching techniques, in order to filter and obtain the relevant resources. Another important characteristic of the standard are the group resources. A group is a special resource that can contain other resources, so that regular operations can be apply to the group, thus applying to all resources that belong to that group. The current version of

the standard only allow static grouping, although for a future revision is expected the expansion of the concept for supporting dynamic grouping, where resources are attached and detached after group's creation. Lastly, the ETSI TC is relying on a previously defined standard for device management, OMA Device Management [24], [25]. By reusing, this already proven standard the committee was able to integrate the device management features within the M2M standard, for the remote management of wireless devices in the ecosystem.

By analyzing Figure 3.1 [26] one can clearly observe the separation of the two domains the device/gateway domain and the network domain. At the bottom of the figure is represented the device/gateway domain, which is defined by the devices that interact with the real world, either sensor or actuator devices, and is composed of the following elements:

- **M2M Device**: Any device capable of running M2M applications, from now on known as Device Applications (DAs), and thus using the service capabilities can have access to the network domain either: a) independently - using an embedded communication module that allows it to talk directly to the service capabilities of the network domain, b) through a gateway - available for constraint devices not capable of communicating directly with the network domain, and thus uses the M2M area network for communications with the gateway, which mediates the communication to the network domain.
- **M2M Area Network**: Any kind of network that provides both physical and MAC layer connectivity to the devices. Example of such networks include, but not limited to: Zigbee, Bluetooth or IEEE 802.15.
- **M2M Gateway**: Special device capable of running M2M SCL that allows the communication between an M2M device and the network domain. In addition, the gateway support the installation of multiple M2M applications to extend its functionalities. Usually these devices are equipped with a communication modules (such as GSM/GPRS) and multiple M2M are network communication modules to allow it to communicate with constraint devices.

At the upper level is located the network domain which is characterized by the high-level services that besides of turning data into meaningful information, is responsible for presenting that information with a nice user interface (e.g. by drawing charts) to the user, such user interfaces are known as dashboards. Additionally, the network domain is also responsible for connecting both domains, hiding network communications details (such as IP address and ports) and easing the integration of applications running in separate domains. Finally, the network domain is composed by the following components:

- **Access Network**: A network which allows the M2M device/gateway domain to communicate with the Core Network. Include networks such as xDSL, satellite, W-LAN, and WiMAX.
- **Core Network**: Provide features related to IP connectivity, service, and network control functions.
- **M2M Service Capabilities**: Provide M2M functions that are exposed to Network Applications (NAs) through a set of open interfaces.
- **Network Management Functions**: Include all the functions related to the management of the access and core networks.

- **M2M Management Functions**: Include all the function related to the management of applications and SCL in the network domain.

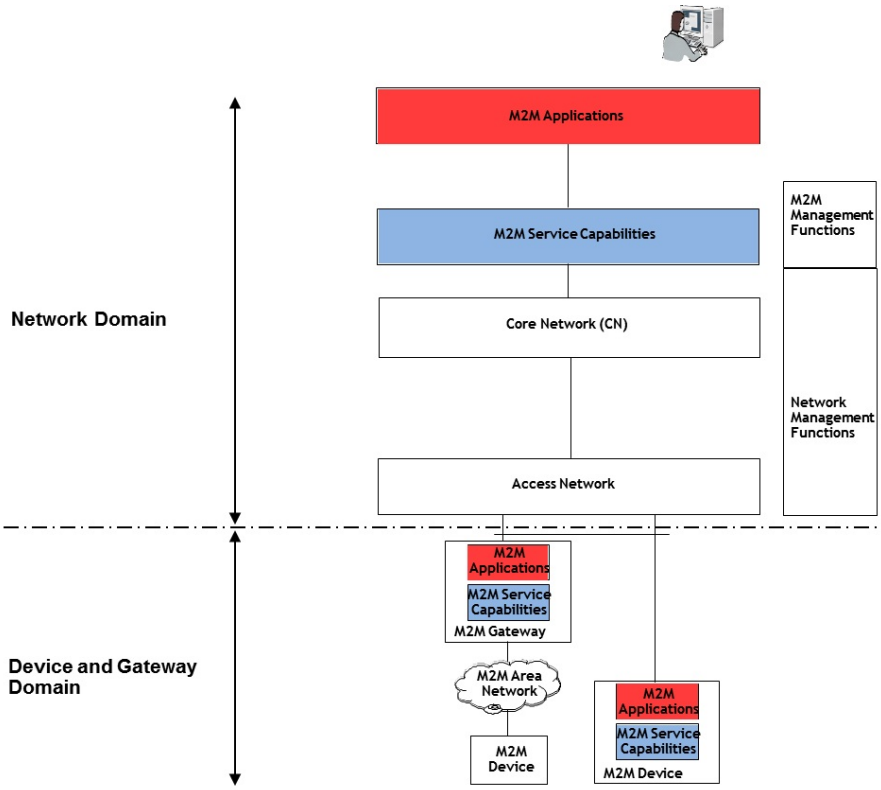

Figure 3.1: ETSI M2M high-level architecture [26].

3.2.2 HORIZONTALITY

The standard provides an horizontal architecture, referred to as Service Capabilities (SC), as a substitute for vendor specific vertical architectures, where the service layer ought to be shared by multiple applications. The transition to such an architecture was the next step because many vendors are already implementing M2M on their own and although leveraging the Internet for their services, the downside is that the deployment uses proprietary protocols, data formats, security mechanism, in order to offer the application service. This vertical setup creates what is called a "silo" [27]. While communication works as it is expected within the same silo, because of the tight coupling between the devices, the network and the back-end applications, but when attempting to integrate two or more services from different vendors then the incompatibilities and unexpected behavior arises.

In contrast, the ETSI SC aims to provide a standardized framework that offers common functionalities, which was previously implemented by all vendors individually, which enables rapid development of M2M applications and breaks the barriers that have slowed down the evolution of M2M. Furthermore, the components of the SC provides features related to: connectivity - by hiding network transport and protocol specificities; device management - for easy configuration management of the device, software/firmware upgrades and performance management, since devices are widely disperse on the

field; data control and arbitration - this function allows for the storage of data and definition of the data structure used for the message exchange, and also could provide more elaborate functionalities such as subscription/notification and data aggregation; finally, security and privacy is also addressed by the SC, for instance is responsible for providing necessary credentials (such as cryptographic keys) to ensure communication between the device and the SC is encrypted, the device's location and billing functions.

3.2.3 SCALABILITY

In computer science if often used the term scalability. Any system is said to be scalable, if it has the ability to accommodate to the demand the market ask. Scalability is also associated to the addition of new hardware to cope with the growth. Given the horizontal architecture of the standard, scalability is achieved by means of the distributing the load throughout the M2M nodes, i.e., when new devices are attached to the system, the only affected domain is the device/gateway domain, therefore one could easily add more gateways to support the load, if necessary. On the other hand, in the network domain, modifications to the core network does not affect the gateway.

Another important point to note about scalability, is that the reference point mId allows for caching mechanism, since it is exposed through the Hypertext Transfer Protocol (HTTP) protocol, and there are already working proxies in the Internet that cache many HTTP requests. This was not a coincidence, when drafting the specification. M2M TC chose to go with a resource-based structure and the HTTP as the communication protocol, to take advantage of the infrastructure already in-place, in this case, the proxies would store results of the most common used methods, improving the response time of each query, since the operation does not need to take place again.

3.2.4 ADDRESSING

As previously said, in the IoT world, every "thing" (for instance a device) is addressable through a unique identifier. To this end each resource, in the hierarchy, is addressable by the URI, which along with the hierarchy path it also specifies the protocol to be used in the communication. For instance, a application identifier could be *http://nscl.domain.com/**sclBase**/applications/**temperatureSensor***, where the **sclBase** and **temperatureSensor** are both user configurable and describe a common name for the SCL and for the application, respectively. The M2M TC has come up with many identifications:

- **App-ID** - Uniquely identifies an application (either DA, GA or NA) that is registered with a SCL.
- **SCL-ID** - Use when a SCL is registered on another SCL, for instance, a Gateway Service Capability Layer (GSCL) registers on a NSCL.
- **M2M-Node-ID** - It identifies any device in the system.
- **M2M-Connection-ID** - Identifier of a valid connection between a M2M Device/Gateway and the NSCL.
- **MSBF-ID** - The service provider's M2M Service Bootstrap Function (MSBF) unique identifier, which is statically assigned.

The list above shows some of the available identifiers in the ETSI standard [26].

3.2.5 INTERWORKING WITH LEGACY DEVICES

The aforementioned architecture is not meant to disrupt already working systems, in fact, ETSI's goal is to allow integration of legacy devices in the ETSI M2M network. As a result, the approach was to define an Interworking Proxy Unit (IPU), which is an application for non-compliant area networks, that aid devices non capable of running a SCL, translating communications between the two networks. For example, given the existence of a device that uses a Zigbee network to communicate, the vendor can write a Gateway Interworking Proxy (GIP) (i.e. a IPU that runs in the gateway domain) that registers the device in the GSCL, creates containers, adds subscriptions or publishes data just like any other ETSI compliant device, but through a Zigbee network.

Furthermore, IPU is not specific to the device and gateway domain, in the network domain an IPU can be defined in order to communicate over non standard protocol, so that the NSCL can be extended to work outside the scope covered by the ETSI standard. For example, an IPU, which in the network domain is called Network Interworking Proxy (NIP), can be developed to obtain data from a legacy application that uses some legacy Inter-Process Communications (IPCs) mechanisms, or could sent data through a new protocol such as MQTT, which is a new lightweight messaging protocol designed for M2M applications.

3.2.6 SECURITY

Any system connected to the Internet is a potential target for attacks. It is fact that many companies and sites have been victim of malicious activities once they're connected to the Internet, and IoT is not going to be an exception, given the many areas with increasing associate value, the smart grid for example. Taking a closer look at the inner workings of the IoT, some reasons for attacks come to mind, like for instance:

- New technologies introduce new vulnerabilities.
- The number of connected devices is growing dramatically.
- Always available (24/7).
- Devices run without supervision in the field.

Given the security demands, ETSI took the initiative to include security features in the standard from the beginning, because delaying it for a later time would create inconsistencies between implementations. As a result, two types of security mechanism were included, one for authentication and access to the infrastructure and the other for data authorization. The M2M Service bootstrap and Service Connection is the procedure that allows, either a device running a Device Service Capability Layer (DSCL), or a gateway running a GSCL, to authenticate and retrieve parameters from the NSCL. Within this two procedures the NSCL provides the requester with a M2M Root Key (Kmr) and the M2M Connection Key (Kmc) which are used to perform mutual authentication of the mId endpoint (this is explained in section 5.3). The second type of security is concerned with resource's authorization, i.e., a device can only have access to resources which hold certain permissions. Permissions, in M2M

standard are configured through a special type of resource called *AccessRights*. These resources are stored within the resource structure and can be reference by other resources through configuration of the attribute *accessRightId*. Permissions are ensured by the use of the CREATE, DELETE, RETRIEVE, READ, WRITE AND DISCOVER flags.

3.2.7 M2M FRAMEWORK

M2M SCs are the foundations of the ETSI M2M work, by providing a standard Application Programming Interface (API) which exposes the capabilities of the domain where is applicable, together with the reference points - which constitute the communication channels, protocols and messages to be used by the entities of the M2M system, form the framework that will share the components of the network, such as: radio links, network resources and edge gateways across all M2M applications, while at the same time remaining technology agnostic, in the event of supporting any network technology. Additionally and given its future usage, the framework has been designed with extensibility in mind, meaning that not only all the capabilities are not mandatory but also it is expected the addition of new capabilities with future revision of the standard.

A capability is a software module that provides a specific service. The following list enumerates the capabilities currently defined the ETSI M2M standard [28], an important note is that each capability can be instantiated by the SCL running in the device (DSCL), gateway (GSCL) or network (NSCL) domain, therefore the "x" in each capability can be replaced by D, G, N for the device, gateway or network respectively.

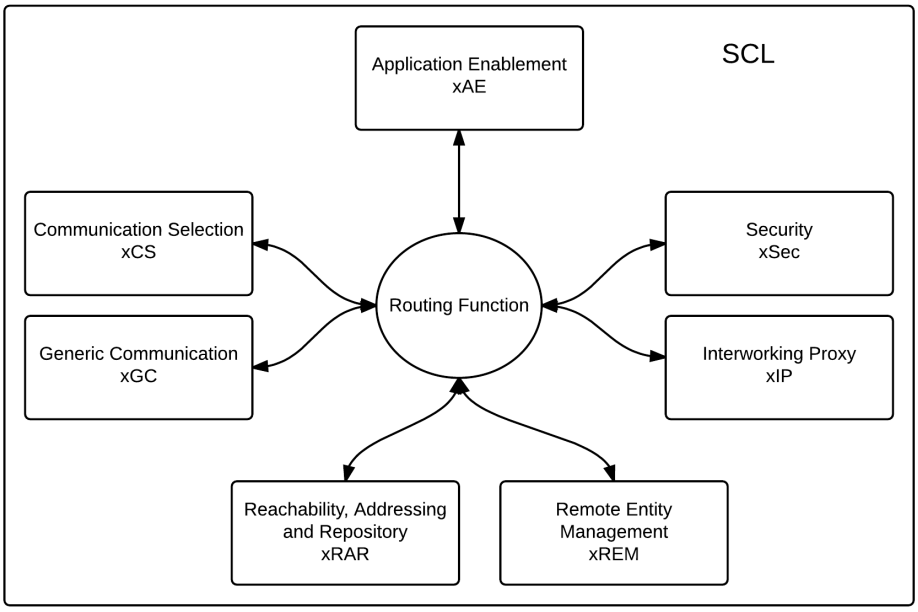

Figure 3.2: SCL reference architecture, adapted [26].

- **Application Enablement (xAE)** provides the interface for applications to use other capabilities.

- **Generic Communication (xGC)** responsible for managing secure transport and session establishment and teardown.
- **Reachability, Addressing and Repository (xRAR)** provides persistence for the state of applications of other entities of the system.
- **Remote Entity Management (xREM)** provides the functionality to device management, including software and firmware upgrades.
- **SECurity (xSEC)** offers bootstrapping, authentication, authorization, key management and secure storage.
- **Interworking proxy (xIP)** allows a non ETSI-compliant device to interact with the rest of the M2M system.

Beyond the specification of the three SCLs, the standard provides communication methods between the domains, or even inside the same domain, through the use of the reference points. Consequently, a reference point can be seen as an entry point to the SCL (an API), thus exposing functionalities of the domain through an open interface, and as an API it reinforces the actions that can be done, hiding any specific technology details. The operations that can be used for communication are define in [28].

Additionally, since the standard only specifies a guideline to the reference points, it is up to the developer to choose the method by which it exposes the reference point, i.e.: it can use the local method call inside the same process address space, or it could choose an IPC mechanism, such as RESTful web services, leading to a more distributed architecture.

These reference points are enumerated as follows:

- **dIa** Device Application Interface - It is used by DAs and Gateway Applications (GAs) to communicate with the local SCL, which in this case refers to the DSCL or GSCL respectively.
- **mIa** Network Application Interface - It is used by the NAs to communicate with their local SCL, the NSCL (Figure 3.3).
- **mId** SCL Interface - It is used for mutual communication between SCLs, i.e., the GSCL uses it to communicate with the NSCL, and vice-versa. mId uses core network connectivity functions as an underlying transport.
- **mIm** NSCL interface - It is used for inter-domain communication between NSCLs of different service providers (network domains), therefore mIm extends the reachability of services offered over mId reference point (Figure 3.3).

To finalize, the primitives exposed by the reference points mentioned above, are related to allow resource manipulation with the goal of data exchange by the entities in the M2M system. Although certain procedures need to take place before the operations are allowed by the SCL in question, this procedures are: SCL and application registration. The former procedure allows for a DSCL or GSCL to register to the NSCL, while the later allows an application (GA, DA, NA) to register to its local SCL.

Figure 3.3 depicts the various uses of the reference points and which M2M's entities uses them.

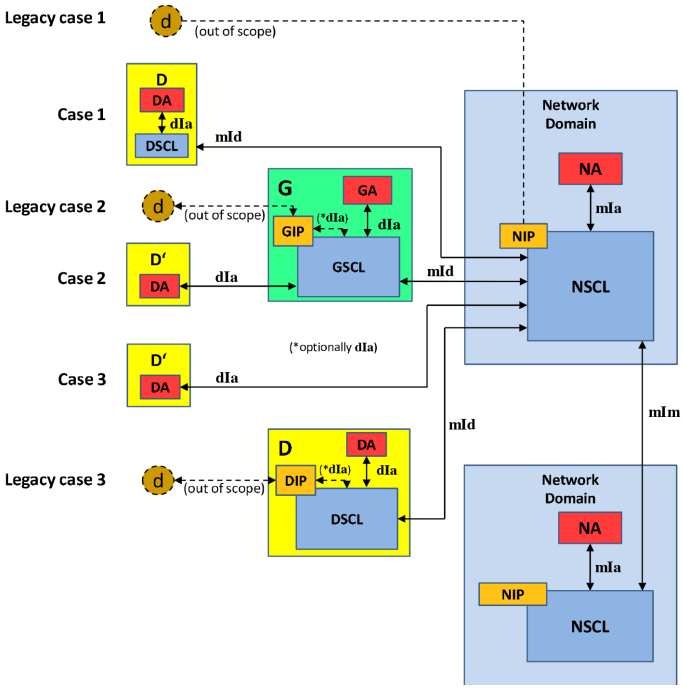

Figure 3.3: Mapping of reference points [26].

3.2.8 M2M APPLICATIONS

In a M2M environment there could be up to 3 types of applications. A M2M application is classified by the domain where is running, i.e., a DA is known for running in the device domain. GA and NA, are known for running in the gateway and network domain, respectively. M2M applications can encapsulate any business logic for a specific task, the standard only mandates the usage of the respective reference point to leverage the functionalities provided by their local SCL. With the existence of M2M applications, the SCLs in each domain, are more than just simple data pipes, given the added flexibility of programmability it can enrich the data that is provided by the devices. For instance, applications in the network domain include back-end applications that collects and analyzes smart sensor data and display it on its dashboard.

Although it was already mentioned, it is important to note that any application must register itself with the domain's SCL, in order to be able to obtain the information they may eventually need, otherwise the SCL will deny access.

3.3 KNOWN IMPLEMENTATIONS

The following sections will present the projects that have implemented the standard, which are: Cocoon from Actility, OpenMTC from Fraunhoufer and OM2M from the Eclipse foundation. Each section presents a description of the project, its architecture and the new features it brings to the world of M2M.

3.3.1 COCOON

Cocoon [29] is an open source project from Actility that aims to provide a compatible system with the ETSI standard. Development of Cocoon began in 2011, and since then the company has been organizing a series of video tutorials and documentation to ease the initiation of new application developers into the world of IoT, and therefore to the Cocoon project itself. Actility vision is to enable "mass market M2M deployments with millions of gateways accessing hundreds of applications" and help hardware manufacturers that want to build ETSI compliant gateways to have a working software that could run on their devices. Actility gave the name of Object Network Gateway (ONG) to its gateway implementation [26], [28], which provides native capabilities including a RESTful interface, data containers, data, logging, subscriptions and notifications. ONG is mostly written in the Java programming language, Figure 3.4 depicts the ONG's environment. Along with the ONG's implementation, the company has been working on an application store, called ThingPark Store[1], from where in the future the company expects to make profits.

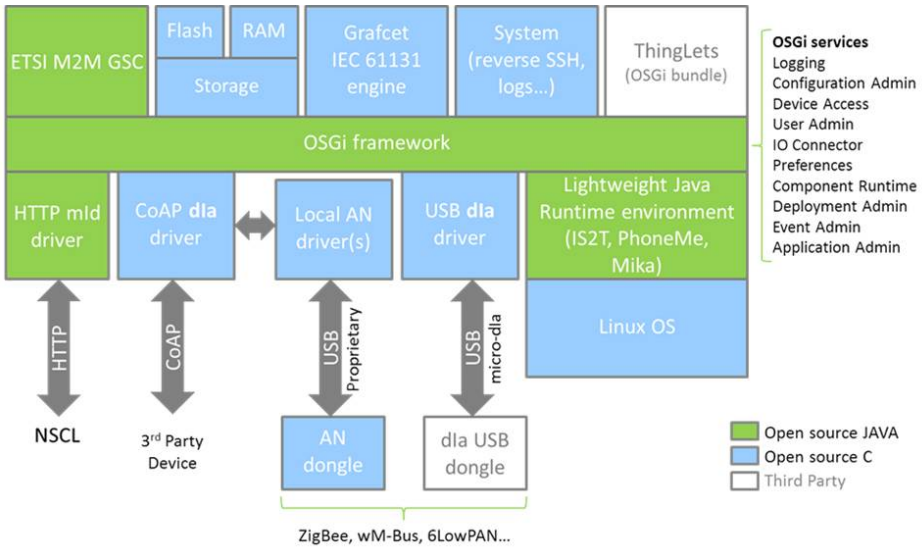

Figure 3.4: Cocoon ONG environment [30].

3.3.2 OPENMTC

OpenMTC[2] is being developed in cooperation between Fraunhofer FOKUS and Technische Universität Berlin [31], as a platform that supports the horizontal layer that drives M2M type of communications. Although the project is not open source, they offer the source code or the binary under a license fee.

One of the goals of the project is to provide a compatible middleware for M2M communications, so that vertical applications like health care, smart cities, and the like, could be deployed on top of this middleware. OpenMTC provides a compatible platform with the ETSI standard, by providing both a gateway and network SCLs, i.e.: GSCL and NSCL. Furthermore, the project has been expanding its roots, is starting to include support for transport technologies like Zigbee and Bluetooth, and is also

[1]Store website: http://www.thingpark.com/en
[2]http://www.open-mtc.org/

adding support for the integration of platforms like Arduino, Android and the Raspberry Pi, within its middleware. To that end, OpenMTC offers a DSCL that runs on top of an Android smart-phone, and turns the device into a smart-thing, using the sensors embedded in the phone to obtain data from the environment.

An important topic within the group is to integrate OpenMTC with the existent OpenIMS project (which is an implementation of the IMS framework), in fact, OpenIMS application will leverage the data that comes from devices in the IoT, once the integration is completed.

According to [32], the project is still under active development, with a recent release in April 2014 and with a future release in November 2014. Figure 3.5 shows the project's roadmap.

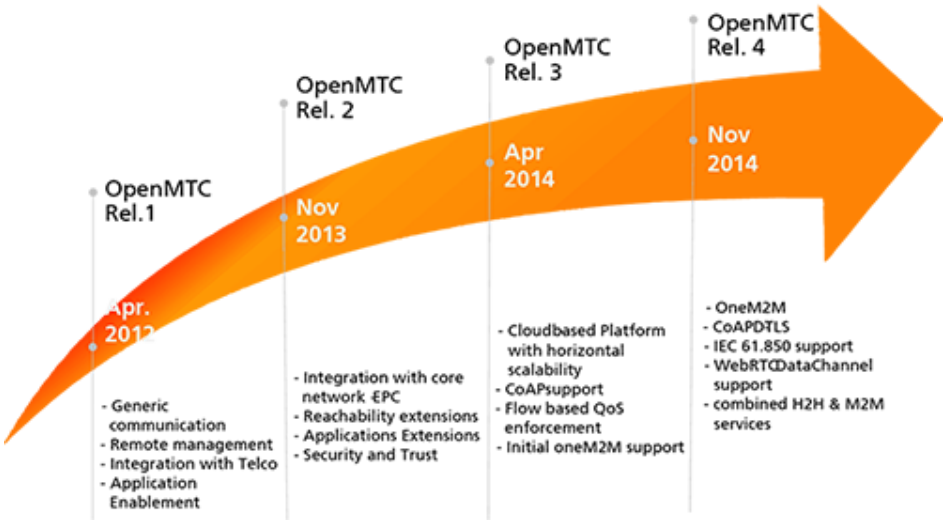

Figure 3.5: OpenMTC roadmap [32].

Figure 3.6 depicts the high level architecture of the OpenMTC middleware. Starting at the top of the diagram the FOKUS broker provides the necessary service platform for the interaction between the vertical applications and the NSCL, i.e.: provides the necessary API that enables the applications to access the OpenMTC Core. At the NSCL level integration with the IMS is provided by the use of a NIP, that mediate all communications from and to the NSCL. Communication between the two domains is facilitated by the EPC, which is an abstraction of the underlying access network used to communicate with the GSCL. Finally, at the lowest level is the GSCL which provides all the ETSI capabilities, and have access to sensors and actuators using any of the supported technologies.

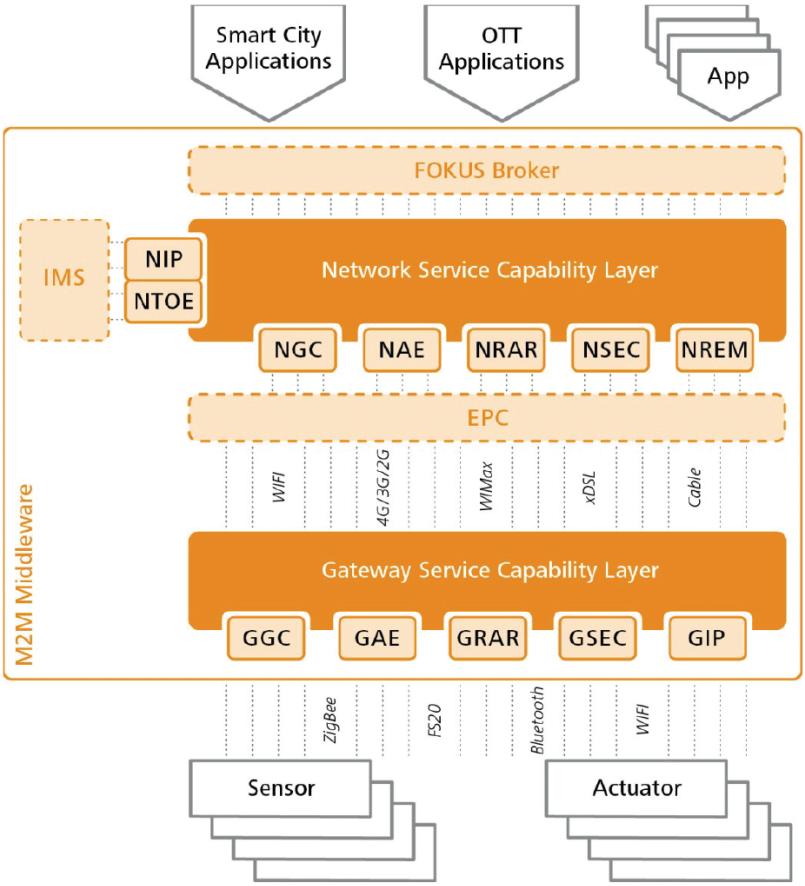

Figure 3.6: OpenMTC platform architecture [31].

3.3.3 ECLIPSE OM2M

The Eclipse foundation has also started their own ETSI M2M compliant implementation, which they called OM2M [33], and which is available as open source software[3]. The service platform follows the same RESTful approach with open interfaces and proposes a modular architecture within a highly distributed environment. Their implementation of the SCL is composed of a plug-in architecture, which allows it to be adapted to the domain where it will run, for instance, it can run in the network, device or gateway domain using the specific plug-ins. The only plug-in that must be deployed in each SCL is the CORE plug-in, this plug-in provides the mechanisms to handle the incoming RESTful requests, so that specific communication plug-ins can be added to the SCL, currently they're offering plug-ins that support both HTTP and CoAP protocols. A plug-in, within this architecture can be easily started, stopped, updated or even uninstalled, without restarting the SCL. In addition, it provides a service registry that automatically detects new services, in order to extend the SCL functionalities.

[3]Download link: http://wiki.eclipse.org/OM2M/Download

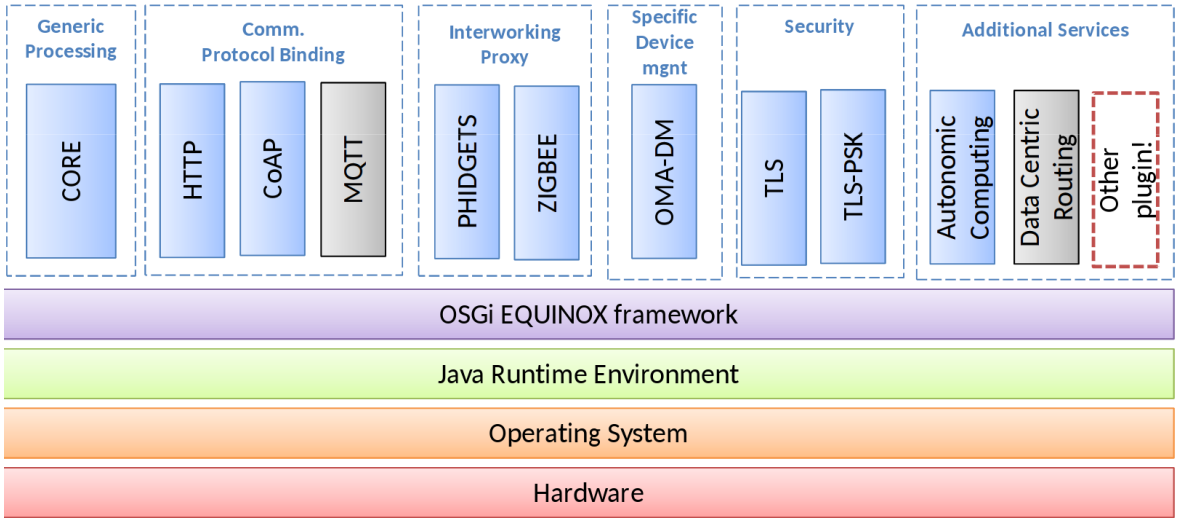

Figure 3.7: OM2M building blocks [33].

CHAPTER 4

Network Service Capability Layer

Given the focus of this dissertation, this chapter is devoted to the demystification of the network middleware, also known as the NSCL. The NSCL is the most important component of the M2M system because without it many of the advantages offered by such system would be lost, or at least much more difficult to achieve. With that said, the NSCL primarily function is to act as a proxy for applications running in different domains, that is, it hides all the details of network communication (addresses, ports) and simply forwards the message to the appropriate GSCL, in ETSI M2M standard this procedures is known as retargetting. For example having two authenticated and registered GSCLs in the NSCL, a sensor in the first GSCL could send data to another device, say an actuator, in the second GSCL by simply using the address (**App-ID**) published by the NSCL, without either application knowing the type of communication involved. Beyond the retargetting mechanism offered, the NSCL offers yet another important functionality: by exporting its vast storage capabilities in the form of resources, a more constrained GSCL (or device) could leverage this functionality in order to offer storage capabilities to the applications local to that SCL, benefiting from an additional database for data storage.

Figure 4.1 depicts a generic structure of the NSCL as shown in [26]. At the center is located the routing function, whose name originates from the fact that it routes requests to the corresponding capability. Within the NSCL a request could originate either internally - a capability is executing functionalities offered by other capability, or externally - the request originates from the outside of the NSCL, typically through one of the reference points.

The NAs are shown outside of the NSCL to present a possible distributed nature of the system. Deploying applications outside of the NSCL offers a more robust and reliable architecture given the isolation between the core and the NAs. As a result, it is harder for a malicious NA to try to bypass security mechanisms imposed by the former. As a consequence of the distributed architecture, there is an overhead associated with the communications, but given the speed of today's network technologies, the overhead is less noticeable and worth the benefits.

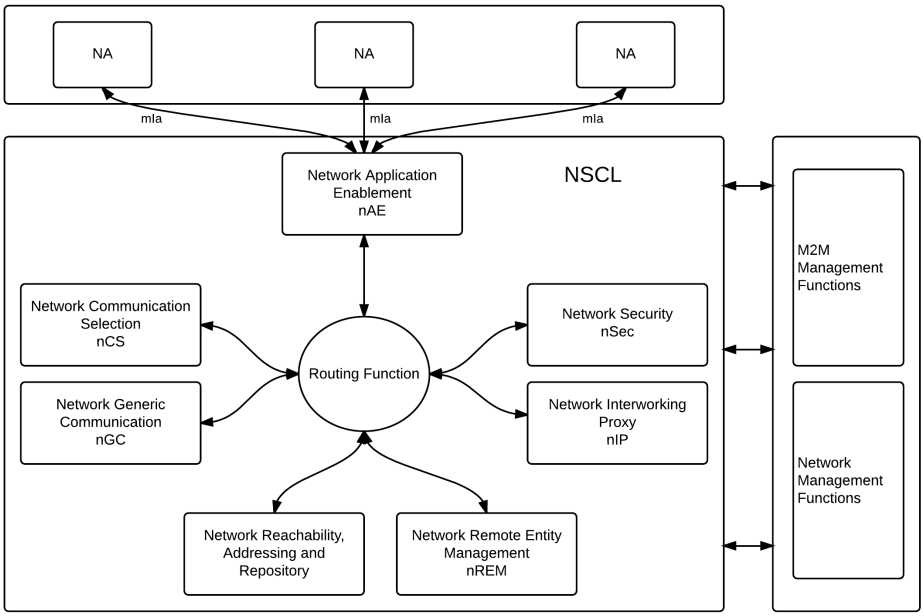

Figure 4.1: NSCL reference architecture, adapted [26].

4.1 NSCL's functionalities

This section will take a closer look at the capabilities provided by the NSCL. Even though many of the functionalities, described bellow, are provided by others SCLs in other domains, and the reader can position himself/herself in the other domain and the concepts still apply, in the following sections focus will be given to the NSCL with all its particularities.

The capabilities besides providing the necessary functions they all have essential things in common, they perform the necessary syntax checking when new requests arrive, they validate the application has access rights to manipulate the resource in question and they use the routing function to perform necessary routing to other capabilities, or to lower level core components.

The following sections are dedicated to describe, in detail, the capabilities and components that the NSCL shall comply with, as it is stated in the standard [26].

Network application enablement (NAE)

Network Application Enablement (NAE) is the capability that allows the applications, through the mIa reference point, to communicate with the NSCL. NAE allows application to perform registration, accessing resources, and if available by the NSCL, generate accounting and charging records. Naturally, since this is the point of contact between the applications and the NSCL, it does the necessary routing to the corresponding capability, based on the received request, for example, the case of a request that stores application information that is routed to the Network Reachability, Addressing and Repository (NRAR) capability.

NETWORK COMMUNICATION SELECTION (NCS)

Network Communication Selection (NCS) capability in the NSCL provides the functionality of network selection based on policies, whenever a M2M device or gateway can be reached through several networks. The capability can leverage network redundancy in order to maintain connectivity when the selected network has a communication failure.

NETWORK GENERIC COMMUNICATION (NGC)

Network Generic Communication (NGC) is the capability whose responsibility is to assist in the communication between SCLs, therefore it encompasses the mId reference point. It's divided in primitives that can be mapped to XML or JSON (as it is defined in Annex B.1 of [28]) and the transport protocol. The transport protocol can either be HTTP, if the device is capable of running it, or the CoAP when the device is less capable. Although the standard promotes bindings to these two protocols, other bindings can be used instead if desired, as long as the protocol is a suitable substitution (offers the same functionality). It is also within this capability, that lies the support for the following communication methods:

- Synchronous - is a client-server model where the issuer sends a request and it shall receive the reply directly in the same interaction.
- Semi-synchronous - is used when it may take the receiver longer than expected to answer a request, in this case the receiver answers directly with a message that communicates the acceptance of the request and that it will be processed. The issuer has to be actively asking the receiver if the request is ready. This type of communication is also know as short polling.
- Server-Server - just like in the previous case the receiver is not able to answer right away, it only send a confirmation that the request was accepted, but in contrast to the previous scenario where polling was necessary, the result is sent asynchronously by the receiver, using the HTTP POST method (in ETSI's nomenclature is called a notify). However, in this scenario the issuer needs to be server capable, in order to receive the reply at a later time.

NGC is responsible for the management of session initialization and tear-down, to report errors that may arise in communications over mId, as well as the security mechanism associated with it.

NETWORK REACHABILITY, ADDRESSING AND REPOSITORY (NRAR)

As mentioned previously, the ETSI M2M standard mandates that information be stored in a tree-like structure and be accessible in the form of resources, which are nothing more than shared memory buffers that holds data. Given the nature of ETSI's structure, the NRAR capability is responsible for exposing the information as uniquely addressable resources, each resource can be either applications, devices, SCLs or information related to other resources (such as access rights or subscriptions). Having a centralized storage allows for devices with limited storage capabilities and likely to face energy shortages, to not break the connectivity of the system when other applications are trying to obtain those resources.

The NRAR capability exchanges resource information through a RESTful model, therefore resources are manipulated with the CRUD verbs of the web: *Create*, *Retrieve*, *Update* and *Delete*. Additionally,

it is also specified two new additional methods: *Notify* - used for the reporting of notification of a resource that was previously subscribed-to, and *Execute* - for the execution of management operations represented by the resource.

This capability offers the following features: mapping between the M2M addresses (application or device names) and IP address; device and application information storage, including, but not limited to, device next wake-up time, since it is needed in case of asynchronous communications; SCLs information storage; allows the creation of group resources; container creation, usage and management which allows for application to share data; access right management in order to allow fine-grained authorizations for individual applications; subscription and notification management which allows for applications to register and be notified when a resource is modified. Through this last mechanism, applications within the platform are able to receive asynchronous events notifications from resources of devices and gateways, as long as there is an active subscription to those resources.

Finally, request from applications are forwarded from the NAE to the NRAR, since the former serves as a single point of contact for the NAs.

NETWORK REMOTE ENTITY MANAGEMENT (NREM)

The ETSI M2M standard defines the mechanism for remote entity management of devices and gateways [28]. Each device and gateway may have resources within the NSCL to perform the management operations, these operations include firmware updates for stability and security updates, firmware rollback in the event of firmware instability. The resources contain the OMA-DM and TR-069 data models which holds management information and provides the necessary functions for remote management.

The capability offers the following functionality for device life cycle management:

- **Application life cycle management** - remote installation/removal of application in the device, including updates of the same applications, for stability purposes or otherwise.
- **Configuration management** - remote configuration of devices parameters, for further enhancing devices' operations.
- **Firmware management** - remote upgrade/rollback of device's firmware. The firmware just like any other piece of software is vulnerable to bugs, the functionality allows for upgrade of the firmware, which includes the operating system modules, service capabilities.
- **Fault and performance management** - allows to capture device's system status that indicates the overall performance of the device. This feature allows administrators to monitors the device's health remotely in order to avoid future system interruption, for instance detecting anomalies with the battery, if its battery powered, or other parameters that could compromise its proper operation.

Moreover, since the NSCL will operate on devices and gateways, all these operations will be triggered over the mId reference point.

NETWORK SECURITY (NSEC)

Network Security (NSec) provides the NSCL with a bootstrap function, establishment of service connection, secure storage of sensitive key material, and integrity validation.

An M2M bootstrap is the procedure used to derive cryptographic key materials (deriving the Kmr), to secure communications between the device/gateway and the NSCL and also to provision other necessary attributes, such as: an M2M-Node-ID, SCL-ID, and a list of one or more NSCL identifiers. The Kmr specifically is not used to encrypt the whole communication, on the contrary, it is used for mutual authentication and to create a key hierarchy in order to establish the secure communication channel between the two entities. On the other hand, the service connection uses the material, previously generated, and creates the session, where future communication will be based on. In addition, the standard mandates the usage of a secure environment - which is an environment that employs cryptography to secure the data stored, and also provides mechanisms to control access to the information. In the network domain this secure environment is called M2M Authentication Server (MAS).

Currently the standard specifies various methods for bootstrapping a device/gateway [26].

- Access Network Assisted
  - Generic Bootstrapping Architecture (GBA) based M2M bootstrap [34].
  - Extensible Authentication Protocol (EAP) [35] based using SIM credentials.
  - EAP based on Network Access Authentication.
- Access Network Independent
  - EAP over Protocol for Carrying Authentication for Network Access (PANA) [36]
    - EAP-IBAKE [37] over PANA
    - EAP-Transport Layer Security (TLS) over EAP/PANA
  - TLS over Transmission Control Protocol (TCP) [38], [39]

To summarize, the capability NSec offers the functionality for the realization of key materials, such as Kmr and Kmc keys, and authorization of devices and gateways; perform mutual authentication of the channel over mId reference point; provides a secure environment where sensitive information like the device/gateway's key hierarchy, its own certificates and Certification Authority (CA)[1] are stored, in order to be less susceptible to attacks that intent to obtain such materials; and finally may provide functionalities to verify each message arriving over the mId reference point.

M2M MANAGEMENT FUNCTIONS

This group of functions exists only within the network domain, and therefore cannot be converted into a equivalent in the other domains. Consists of all the functions that are required to manage the M2M SCL in the Network Domain. Moreover, is within this component that the MSBF and MAS belong, they provide the NSCL with the functionality of permanently bootstrapped M2M security credentials (such as the Kmr and Kmc) of the devices and gateways, and to store and authenticate the devices and gateways, respectively. Having a dedicated safe environment for the storage and authentication of the credentials (the MAS), reduces the chances of compromising the system, and at the same time, allows the service to be implemented using already defined standards in this area, for instance a Authorization, Authentication and Accounting (AAA) server such as DIAMETER [40].

[1]The Certificate authority is explained in detail in section A.3.2

NETWORK INTERWORKING PROXY (NIP)

As stated in the previous section, the standard provides integration with older devices or with non-compliant protocols and applications, that is the purpose of the NIP capability, i.e., to enable the NSCL to communicate with non ETSI compliant devices or gateways and vice-versa. Furthermore with a NIP the device in question is able to take advantage of the NSCL's capabilities and integrate with the rest of the system, for instance by pushing data to the NRAR capability.

Even though the usefulness of this capability, is not mandatory to be implemented by an NSCL.

4.2 CHAPTER SUMMARY

This chapter addressed every component of the SCL in the network domain, including the capabilities specified in the current version of the ETSI standard, as well as the resource-based structure and what methods are allowed to access these resources. The work done by the ETSI TC in standardizing M2M is a step forward in the creation of an horizontal platform that will serve the market's demands. As time progresses and new use cases arises, the standard is open to adapt to new scenarios, by improving the already existing service capabilities or by adding new ones.

CHAPTER 5

NSCL Implementation

This chapter presents the objectives and the approach taken, in Section 5, details on how each capability is implemented, in Section 5.2, a description of the bootstrap and service connection procedures used in this dissertation, in Section 5.3, followed by the subscription engine for the publish-subscribe mechanism, in Section 5.4. Finally, the chapter ends with the description of the implementation of both GA and NA, in Section 5.5.1.

5.1 APPROACH

The objective of this dissertation is to have a working implementation of the network middleware of ETSI standard (NSCL). Given the complexity and time needed to develop the NSCL from scratch, this work is based on a previous dissertation that implemented the GSCL component [41], and base future work on this base.

This adaptation is possible due to the similarities between different SCLs. Starting with the persistence layer, the resource structure is the same between all the SCLs, therefore it is possible to reuse the GSCL database model, with minimal modification. The NRAR capability was adapted from its counterpart in the GSCL (the Gateway Reachability, Addressing and Repository (GRAR)), maintaining the same architecture but modifying the components when necessary.

So in favor of not reinventing the wheel and have similar result to what is already available, the first step was to adapt the rest of GSCL to the context of the network domain, refactoring the code, so that it is in the context of the network domain, and the removal of specificities of the device/gateway domain, therefore the following list is an enumeration of the modifications made to the previous architecture.

- **Remove the GApps and GIP components** - these was the functionality needed for the support of gateway applications and gateway interworking proxies. The approach used was built-in in the GSCL, i.e., application were a part of the binary and run inside the GSCL application. In the network domain, applications, or interworking proxies, run more disperse, i.e., distributed in various computational nodes, using the mIa reference point for communication with the NSCL.

- **Add support for sub-containers** - Support for application and SCL sub-container was added. Sub-container support was only available after the second review of the standard [26].
- **Remove the dIa reference point, and introduction of mIa** - Since the dIa is the reference point for application to talk with the GSCL, instead support for the mIa reference point was implemented, which is explained in detail in section 5.2.3.
- **Subscription and notification** - Modifications to the subscription manager were necessary, given that previously it relied on local method call invocation for notification and in this context is not applicable.

In addition to the adaption of the NSCL, it was required to maintain a working copy of the GSCL. Continued work on the GSCL was necessary to develop the bootstrap and service connection procedures, and to tested it against the NSCL. Also, the development of a GA to access a device to collect information about the environment, and send it to the NSCL, from there an NA will have access to this information, the development of this NA is out of the scope of this dissertation.

5.2 ARCHITECTURE

The NSCL base architecture follows closely the standard's recommendation for the division of the different functionalities, despite this recommendation, developers have enough freedom to chose other ways to organize their implementation. With that said, the standard recommends that functionalities are exported through the following capabilities: NAE, NGC, NRAR, NSec, Network Remote Entity Management (NREM), NCS and NIP. The explanation of these capabilities were addressed in the previous chapter, in Section 4.1.

Figure 5.1 depicts a package diagram that shows the dependency between the all the capabilities and the persistence layer. At the top of the diagram is located the service layer. In this layer lies the capabilities that allow external entities (like NAs) to access core functionalities, these capabilities are: NIP, NGC and NAE. As the name implies, the service layer offers applications, and other SCLs, access to the NSCL as a set of services. The diagram clearly shows a dependency with the business layer, which relates to the use of the functionalities exposed by the NRAR and NSec capabilities. Lastly the NRAR capability depends on the data layer, which provides persistence for the resources. All the capabilities are implemented within the same address space, i.e., the same process, therefore the communication method used, for the communication between them, is by local method call. In contrast to the other capabilities, the NAE and NGC capabilities uses an IPC mechanism for the communication. The IPC is needed since the both capabilities are divided in two components, one which is located inside the NSCL and functions like a server, i.e., that it waits for incoming requests to process, and it was implemented to use the Advanced Message Queuing Protocol (AMQP) protocol. While the other runs inside an application server and offers a RESTful interface.

The following paragraphs will introduce the most relevant packages in the architecture, starting at the bottom.

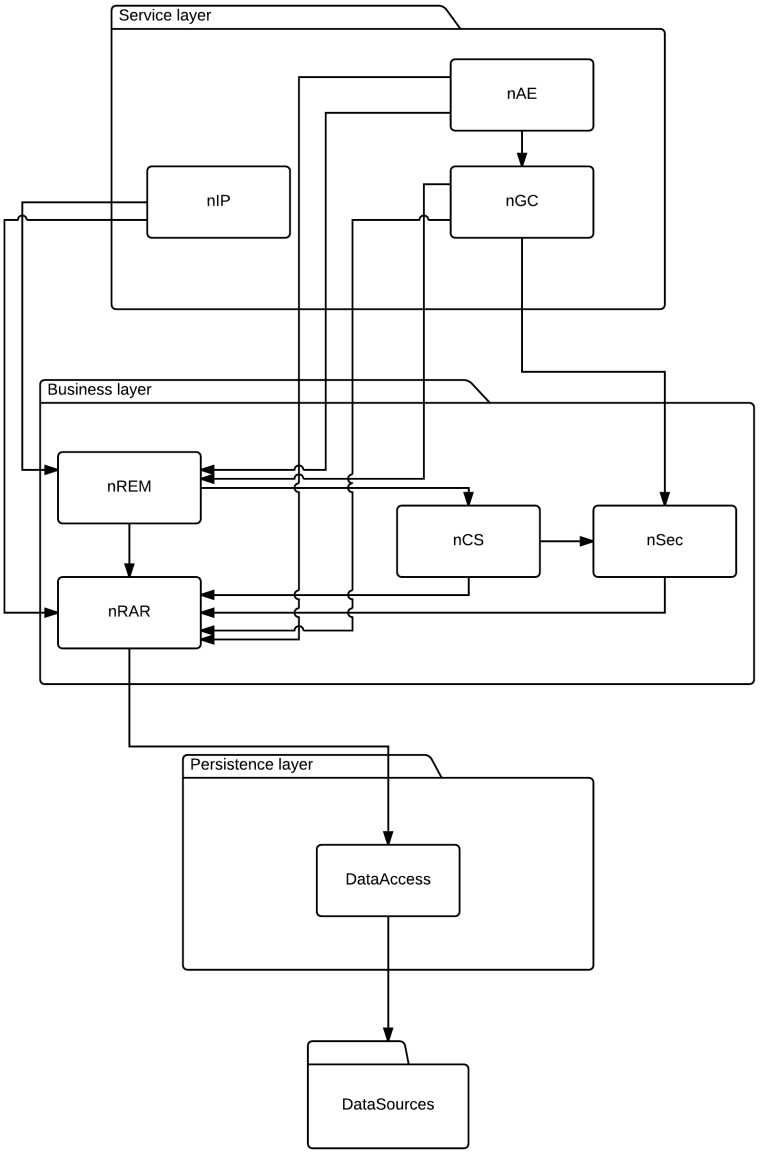

Figure 5.1: NSCL's architecture.

5.2.1 PERSISTENCE LAYER

In this layer lies the dataAccess package, which offers the persistence needed to maintain the ETSI resource structure (See Annex B) and its current state. It provides an APIs for the CRUD methods used by the upper level capabilities, like the NRAR capability. The storage of the resource's state is necessary, since the NSCL must have access to, not only to the most current state, but also to all the values previously pushed into the NSCL. The classes that implements the already mentioned functionalities are: the *DBMediator*, *ResourceDAO* and *AttributesDAO*. While the two later classes are provides a fine-grained interface to access the database, i.e., the classes provides methods to manage resources, and its attributes by using the Structured Query Language (SQL), and interacting directly with the database management system, in addition it provides the needed mapping between the raw data and the ETSI Java objects. The former class, on the other hand provides a "nicer" (high-level) API, in fact, this class abstracts the lower level details of the other two classes (i.e. merges the resources

and attributes in to one class), in order for the NRAR capability to not use the cumbersome methods. Inner details of the package are shown in Figure 5.2.

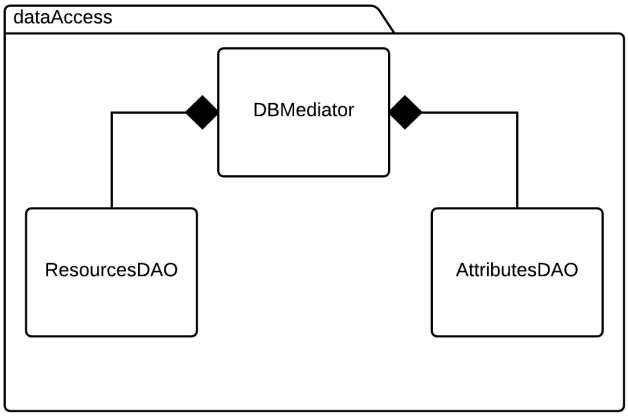

Figure 5.2: Inner representation of the dataAccess package.

5.2.2 BUSINESS LAYER

The following section will present the modules that covers the business logic of the M2M ecosystem. The routing function, is not a component so to speak, but is spread over the components of this layer.

NETWORK REACHABILITY, ADDRESSING AND REPOSITORY

The NRAR capability is implemented within the *nRAR* package, whose responsibility is to manage the internal data structure that is exposed by other capabilities, like NAE and NGC in a resource-based manner. The dependency on the persistence layer comes naturally, since a record of all the information sent by an application, or another SCL, needs to be kept, along with the all the information associated with the entity, like addressing and reachability information. So this package, by using the lower level methods of the persistence layer, offers an API to access the data as a resource. Moreover the inner structure of the package is divided into 4 classes that are mapped to the 4 verbs of the RESTful architecture: Create, Retrieve, Update and Delete (CRUD). The classes are as follows: *RepositoryCreator*, *RepositoryRetriever*, *RepositoryUpdater* and *RepositoryDeleter*, which are mapped to create, retrieve, update and delete, respectively. In addition, the capability also offers the functionality to notify about modifications, for the subscribed-to resources. All of the CRUD classes, apart from the RepositoryRetriever class, have a dependency on the *NotificationEngine* class, in order for the notification engine to know when to process the notification, i.e., once a resources has been created, modified or deleted, it triggers an notification that is sent to the entity that made the subscription. The notification engine uses the *Registry* class to keep track of all the entities' addresses and their mapping to the subscribed-to resources. Figure 5.3 depicts the internal structure of the *nRAR* package.

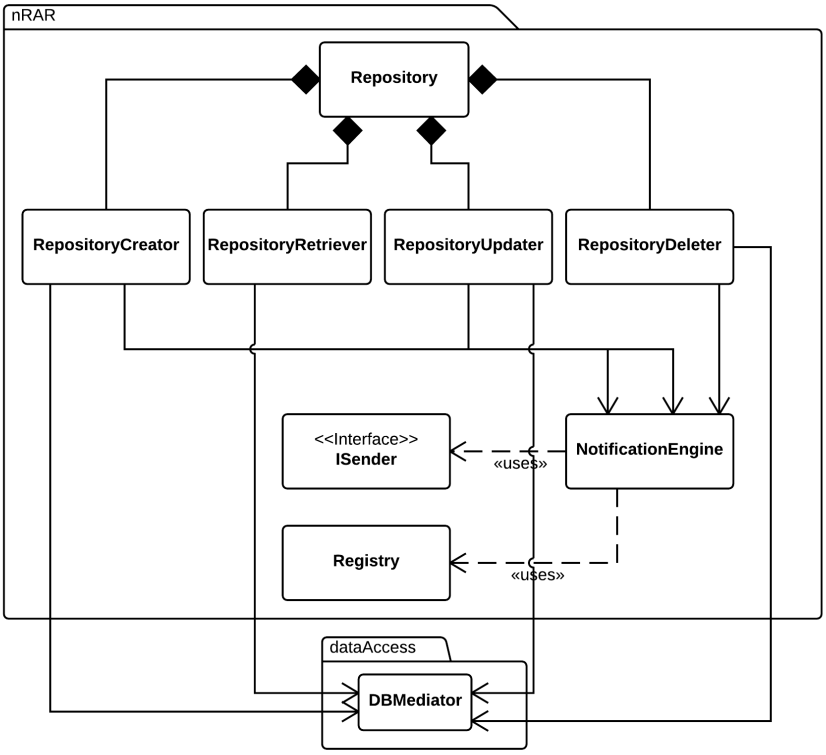

Figure 5.3: Network Reachability, Addressing and Repository components.

NETWORK SECURITY

As was described in the previous chapter, in section 4.1, this capability is responsible for the bootstrap and service connection procedures, thus generating the necessary keys for securing communications with the GSCL over mId reference point. Therefore, in order to provide the mentioned functionality, the following Figure 5.4 shows the components inside the *nSec* package. The package is composed by the class that implements these two procedures, which name is *MSBF*. The class *MSBF* implements the interface, which was called *IMAS*, and provides the mechanisms to access a MAS. The choice to use the already mentioned interface, is to decouple the MAS from the rest of the NSCL, therefore the instantiation of the MAS could either be local or remote, depending on the implementation of the interface chosen. In this case, it was chosen to be a remote instantiation, and therefore a class to mediate the communication between the NSCL and the MAS was developed, which received the name of *MASQ*. This class uses a message queue to pass the message between the two peers, and uses the request-response model, mention earlier, to achieve a bidirectional communication channel. So the purpose of the *MSBF*, class is to offer the methods to achieve the bootstrap and service connection procedures, upon a request from a device or gateway, these two methods are explained in detail later in this chapter (section 5.3).

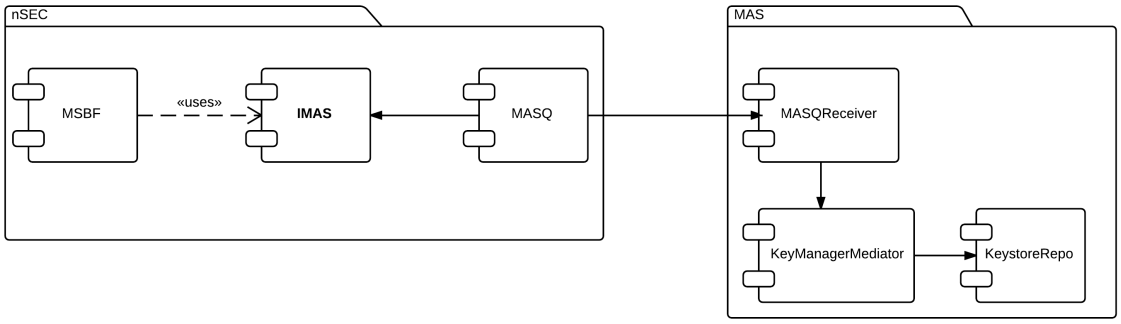

Figure 5.4: Network Security components.

GATEWAY SECURITY

Although this capability does not belong in network domain, and therefore is not part of the architecture mentioned above, is it important to mentioned it, because development on the GSCL was made in order to test the NSCL bootstrap implementation. Modifications made to the GSCL are part of the Gateway Security (GSec) capability, in order to bootstrap using TLS-over-TCP.

Figure 5.5 depicts the components used to perform both procedures. The architecture was not changed from what was previously designed, i.e. the components and the dependencies were maintained and only the new mechanism for bootstrap was implemented . With that said, the gSEC package is composed of 4 classes which are: ServiceBootstrap, ConnectionProcedures, M2MKeyManager and M2MKeyRepository, regarding the former 2 classes are responsible for implementing the bootstrap and service connection procedures, respectively. The other 2 classes are responsible for offering the secure environment for the gateway in order to store sensitive information.

The class ServiceBootstrap is responsible for bootstrapping the Root key (Kmr), along with other information sent by the NSCL like its point of contact or the SCL-ID, using public key certificates (using the mechanism of TLS-overTCP). Once this information is obtained, the dependency on the *M2MKeyManager* is for the secure storage of this information. Finally, the dependency on the *gRAR* package is necessary to register parameters obtained from the bootstrap procedure, that do not need to be securely stored.

On the other hand, the *ConnectionProcedure* uses the *M2MKeyManager* to obtain the Kmr, in order to perform the necessary functions to derive the connection key (Kmc) and authenticate the mId reference point.

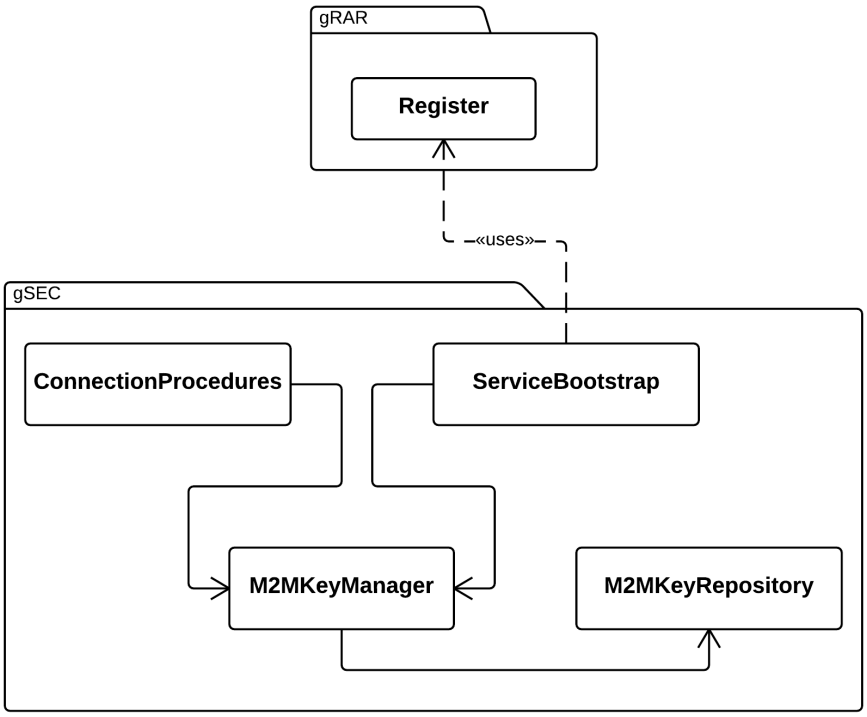

Figure 5.5: Gateway Security components.

5.2.3 SERVICE LAYER

The following section will present the modules that provides the mechanism to expose functionalities to the outside of the NSCL, for example to M2M applications in the network domain or other SCLs, hence the grouping of these functionalities in the service layer package.

NETWORK APPLICATION ENABLEMENT

As explained in section 3.2.7, the NSCL should provide the mIa reference point for external communication between applications and the NSCL's core functionalities. This reference point is openly exposed over a private trusted network, which means that the applications developers are trusted to behave accordingly and not to misuse the API. The mIa reference point is available for NA developers to take advantage of the NSCL's features. The following list enumerates all the functionalities exported through this API:

- Allow application registration.
- Allow the creation of groups.
- Manipulation of containers and sub-containers.
- Provides data storage containers for applications to push, and retrieve, data.
- Management of resource permissions

- Management of subscriptions.

In addition, this capability must ensure that incoming request has the proper resource's syntax, and that the requesting entity has the right permissions, before the routing of the incoming request to the appropriate capability is made.

The interface is exported as a set of RESTful web services, which provides CRUD methods to manage the resources found in the NRAR capability. The service are bound to HTTP protocol and supports two data formats: XML and JavaScript Object Notation (JSON). The web services, which were implemented in J2EE, are exposed through the Glassfish application server[1]. Since the core NSCL and the mIa reference point are running in two different context (one is a standalone Java process, while the other is running inside the application server), a IPC method is necessary to provide communication between the two modules. For that reason, a message broker was chosen.

Regarding the internal organization of the NAE's components, the package is divided in two main classes: the *Handler* and the *Router* classes.

The Handler class provides the mechanism to perform validation checks on the permissions to the resources and the syntax validation of the incoming request, which is delegated to one of the descendant classes: CreatesHandler, RetrieveHandler, UpdateHanlder and DeletesHandler, depending on its request type, i.e., if it is a Create, Retrieve, Update and Delete request.

Afterwards the request is handed to the Router class, if the request has passed all validation checks. Within this class, the request is routed to the appropriate router class, that maps the request type, therefore the existence of the 4 classes CreateRouter, RetrieveRouter, UpdateRouter and DeleteRouter; which in turn, the request is then routed to the NRAR capability, where the operation may be performed.

Lastly, there is the interface *IRefenrecePointmIa* and the proxy class *MQRefPointmIa*, that represents the realization of the mIa reference point using a message broker, and whose responsibility is to register new requesting entities (NAs) and to forward the request to the Handler class. The existence of the interface *IRefenrecePointmIa* is to allow future realizations of the mIa reference point using other technologies, therefore the NSCL is not bound to the message broker implementation.

Figure 5.6 presets the class diagram of the *nAE* package.

[1]Although Glassfish was primarily used for development, the JBoss application server was also tested, and was compatible as a drop-in replacement.

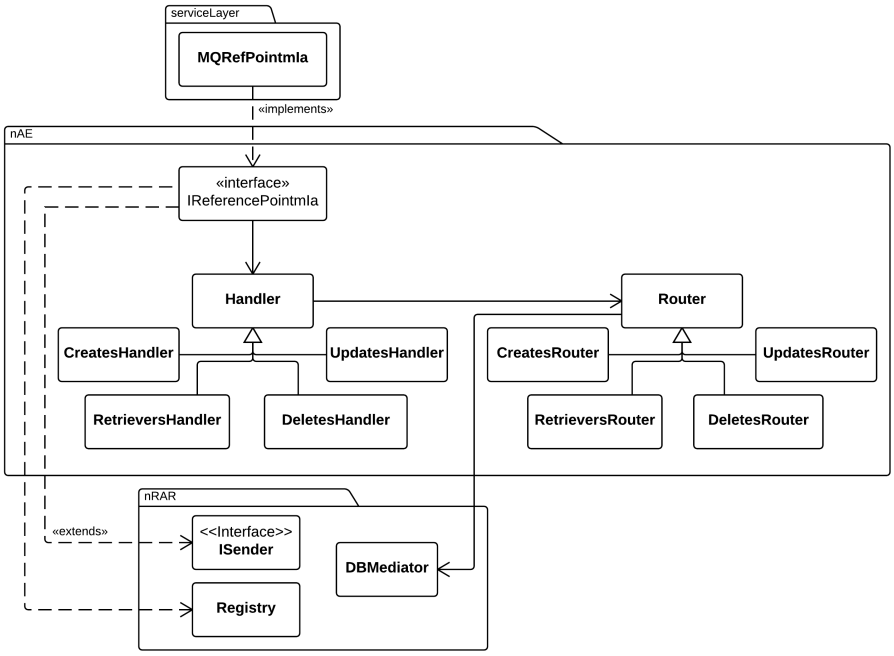

Figure 5.6: Network Application Enablement components.

NETWORK GENERIC COMMUNICATION

The reference point mId, that is the functionality that allows the NSCL to communicate with the device/gateway domain, is implemented in this package and can be perceived as an extension of the NAE capability, since it provides the same functionalities plus the security (including integrity checks, authorization and authentication), profiling of the communication channel and remote management related functionalities - as a result a dependency on the NSec and NREM is present.

The mId reference point, just like the mIa, is exported as RESTful web service model. In contrast to the later, which runs in a trusted network where no cryptographic operation were applied before transport, the former does apply cryptographic operations, basically it uses TLS with Pre-Shared Key (PSK), to ensure that communications over the mId reference point must be secure. Also the same abstraction, just like the one done in the nAE package, to support various realizations with other technologies of the mId reference points. About the internal structure of the package, this package has a dependency on the nAE package, to offer the same functionalities, and to avoid duplicating the code. Beyond the usage of nAE package, the nSec package is also a dependency that is used for authentication and for key negotiations and session establishment.

Lastly, a dependency on the nREM package for the remote management of devices and gateways, allows the NSCL to apply firmware upgrades and management of devices' configurations remotely.

This implementation allows for mutual authentication of the SCL, registration and access to the resources just like the mIa.

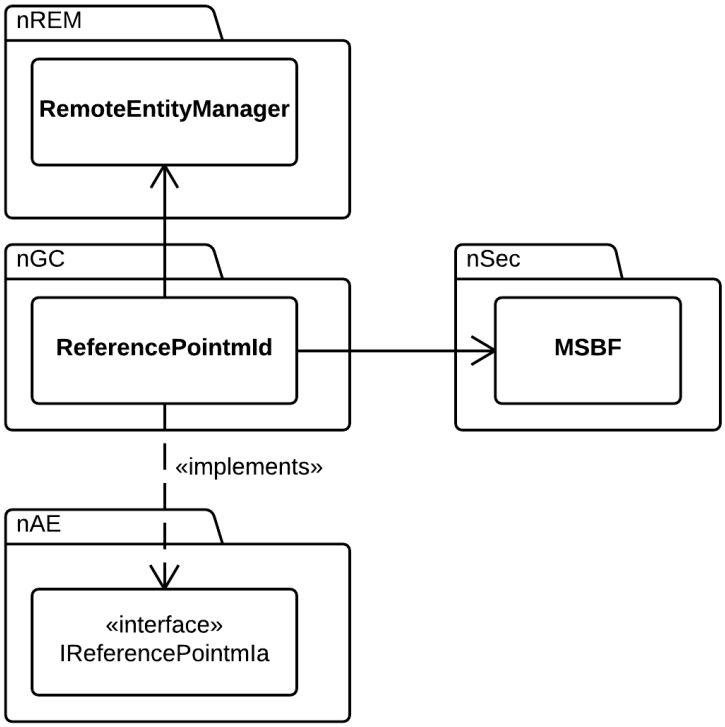

Figure 5.7: Network Generic Communication components.

NETWORK INTERWORKING PROXY

Lastly in the service layer is the *nIP* package, that allows the communication between legacy devices (non compliant with the ETSI M2M standard) and the NSCL. Therefore, this package extends the functionalities of the NSCL, allowing it to interact with other ad-hoc networks, such as Zigbee or Bluetooth, with the goal of obtaining the information they provide by mapping their requests into ETSI requests and vice-versa.

To that end, this package has a dependency on the *nREM* package, in order to integrate management functionalities of the device; the *nRAR*, for the data persistence and retrieval. These functions are exposed from an instantiation of the *nAE* package. To note that the *nIP* package, for being decoupled from the NSCL, serves only as a reference implementation, and if desired, may not be used. However, this package is composed of the following classes: the *Receiver* and the *Translator*.

The *Receiver* class performs the needed reading from the legacy network, that the NIP is being written to, i.e.: for each legacy device that is incorporated into the M2M network, a Receiver class must be instantiated. The network can be either a serial based, like Zigbee or it can be something completely different like accessing a file on a file system. After the operation has taken place the data will flow towards the *Translator* class.

Furthermore, the *Translator* class provides the function of accessing the NSCL through the instantiation of the NAE, which is called the *nIP*, and does the necessary mapping between the resources in ETSI and request for the legacy network. This class therefore has a dependency on the *nIP* package, which in turn, the remote part, i.e.: the one running inside the NSCL, depends on the

nRAR and *nREM* packages.

On the other hand, and to allow the NSCL to send commands to the legacy device, the *Translator* implements the interface *ISender*, and offers the communication channel to support these operations.

The internal structure of the *nIP* package is shown in Figure 5.8, which describes what has been previously stated through a package diagram.

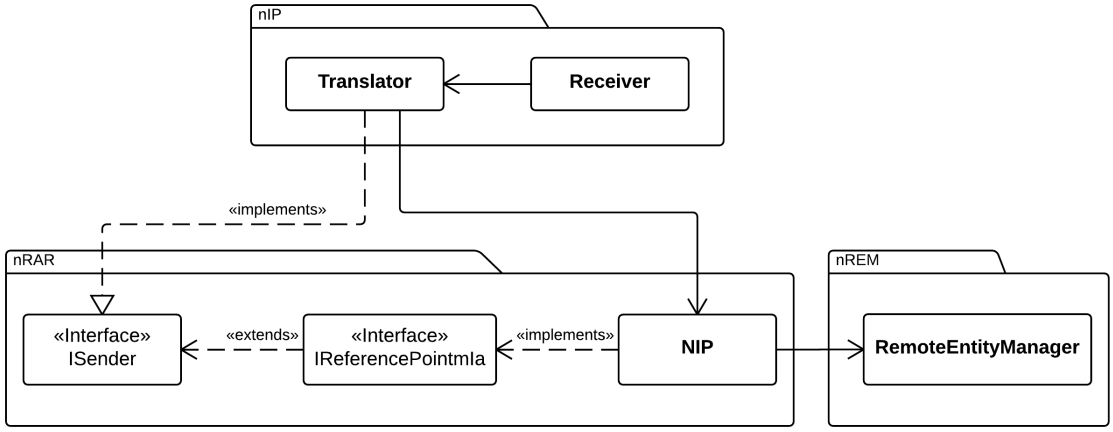

Figure 5.8: Internal structure of the *nIP* package.

5.3 BOOTSTRAP AND SERVICE CONNECTION

This section presents an introduction to service bootstrap and service connection methods from [26], and how they were implemented.

The method chosen was the access network independent method **TLS-over-TCP**. In this implementation, the MSBF enforces the authentication of devices and secure communications by the leverage of TLS with client and server digital certificates[2]. The authentication of the M2M nodes is enforced by the presentation of the servers' certificate, by the MSBF, and a request to the client, to present his own certificate, when the connection is started. By using digital certificates for node authentication, it allows both entities to verified the authenticity of each other, using their trusted CA to verify the certificate they receive. The certificate presented by the MSBF must be signed with the CA trusted by the device, in order for the communication to proceed, the same applies for the certificate presented by the device, i.e., the certificate must be signed by the MSBF's CA. If the certificate's signature is not recognized by the node, or in the case of the MSBF the device does not present the certificate, the procedure is aborted.

Figure 5.9 presents the components implemented in this dissertation, needed for the bootstrap and service connection procedures. At the left side is where a gateway running a GSCL, uses the mId reference point to initiate the bootstrap procedure, and after it has finished successfully it initiates the service connection procedure. As it can be seen in Figure 5.9, the communication between the

[2]Using the X.509 structure standard for digital certificates

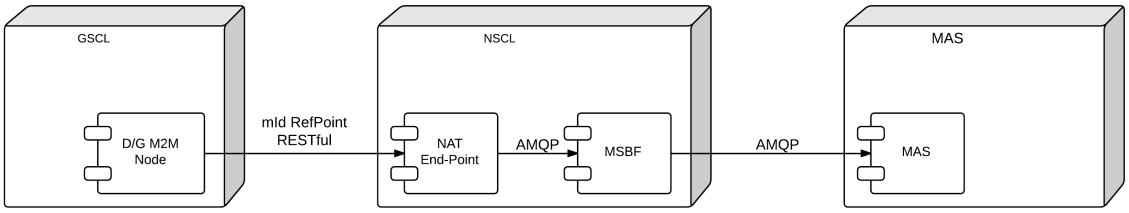

Figure 5.9: High-level view of the bootstrap components.

gateway and the NSCL is over a RESTful channel, where the MSBF exposes a virtual resource called *bootstrapParamSet*. In the NSCL side, the end-point receives the request, which is then routed to the MSBF over a AMQP queue (using the mechanism of the request-response model explained in Section A.2.2), which will then process the request and if the request is valid a Kmr key will be generated and the MSBF will then send two copies, one to the gateway and the other to the MAS and the bootstrap procedure will successfully be finished. After the device has been bootstrapped, i.e. has a valid Kmr key, it will do a similar process to establish a connection. The service connection procedure is initiated by the gateway when it accesses the virtual resource *connectionParamSet*. The MSBF will then process the request, validating the received Kmr against the one stored in the MAS, and if its correct then it will generate the Kmc key and send one copy to the gateway and the other to the MAS. Once this two procedures are completed, the GSCL must register itself with the NSCL and secure communications over mId. The MAS is the secure environment in the network domain, its primary function is to securely store the mapping between a node-ID and keys provided by MSBF, and perform related cryptographic operations.

5.3.1 M2M BOOTSTRAPPING PROCEDURE

The bootstrap procedure in this implementation is exported as a web service, as was mentioned in the above paragraph. The web service is deployed on the application server and its configured to use TLS and to force the client authentications. Moreover, the application server is configured with a dedicated CA to validate the incoming certificates.

The end-point is located within the network middleware in a special resource called bootstrapParamSet, for instance and assuming the NSCL is running under this name the resource is available at: https://nscl.example.org:8080/bootstrapParamSet. Before any processing is made, the gateway/device starts a TLS session with the NSCL. An important note is that the NSCL will act as a Network Address Translation (NAT), will only forward messages to and from the MSBF and the gateway/device and will not process any message. Figure 5.10 depicts the sequence of events carried out between the gateway/device and the network middleware, the black dot in the NSCL represents the routing to the MSBF.

First the client starts the conversation by sending the *client hello* message to the MSBF, in response the MSBF answers with its own certificate and asks the client to provide its certificate. Once the client receives the certificate it will proceed with the verification of authenticity, if the verification does not succeed the connection is aborted, otherwise the process will continue and the client will send its own certificate. Then, after receiving the certificate the MSBF will validate it and decides if the entity is authentic, and only after the verification has succeeded the root key (Kmr) will be generated.

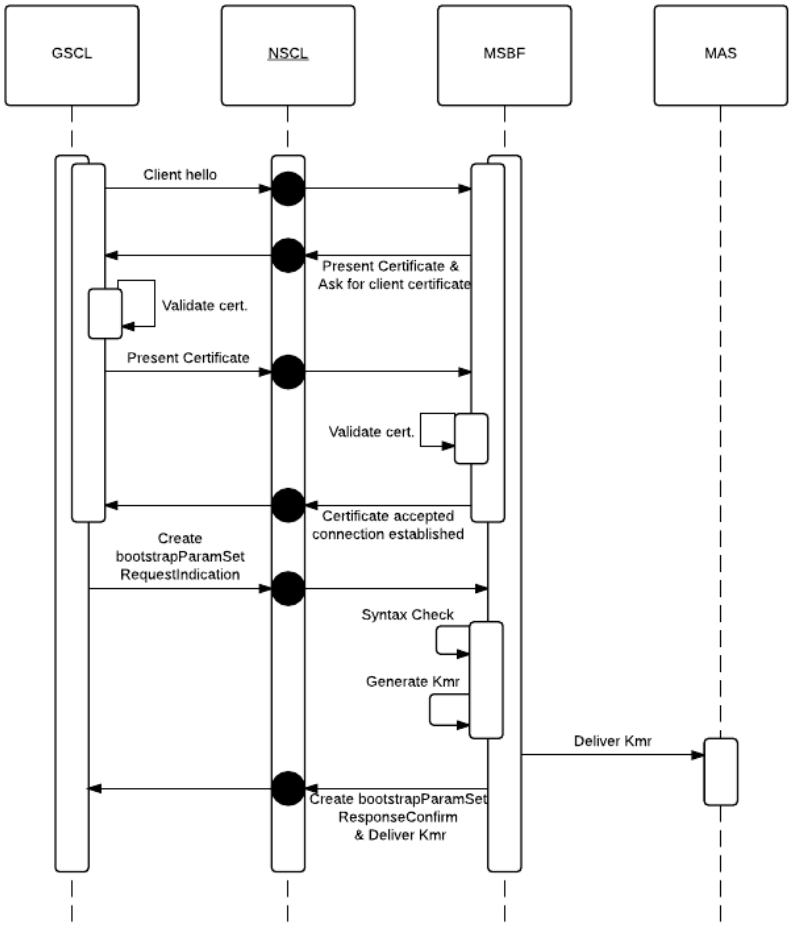

Figure 5.10: Bootstrap procedure sequence diagram.

Once generated the root key will be securely stored in the network middleware secure environment (MAS), and a copy will be sent to the client, along with other sensitive material needed, finishing the bootstrap procedure.

5.3.2 M2M SERVICE CONNECTION PROCEDURE

Just like the bootstrap, the service connection procedure is also exported as a virtual resource through the web service. The end point is located in the network middleware at the resource *connectionParamSet*. For instance given a NSCL running at the *nscl.example.org* domain, the end point is available through the URI: https://nscl.example.org:8080/connectionParamSet. Unlike the bootstrap procedure that used TLS using certificates so that the two entities could to authenticate each other, the service connection uses the specialized TLS-PSK. In this protocol both entities must have a shared secret in order to secure the communication channel, in this implementation that secret is the previously established Kmr key.

Figure 5.11 presents a sequence diagram about the service connection procedure. First the GSCL

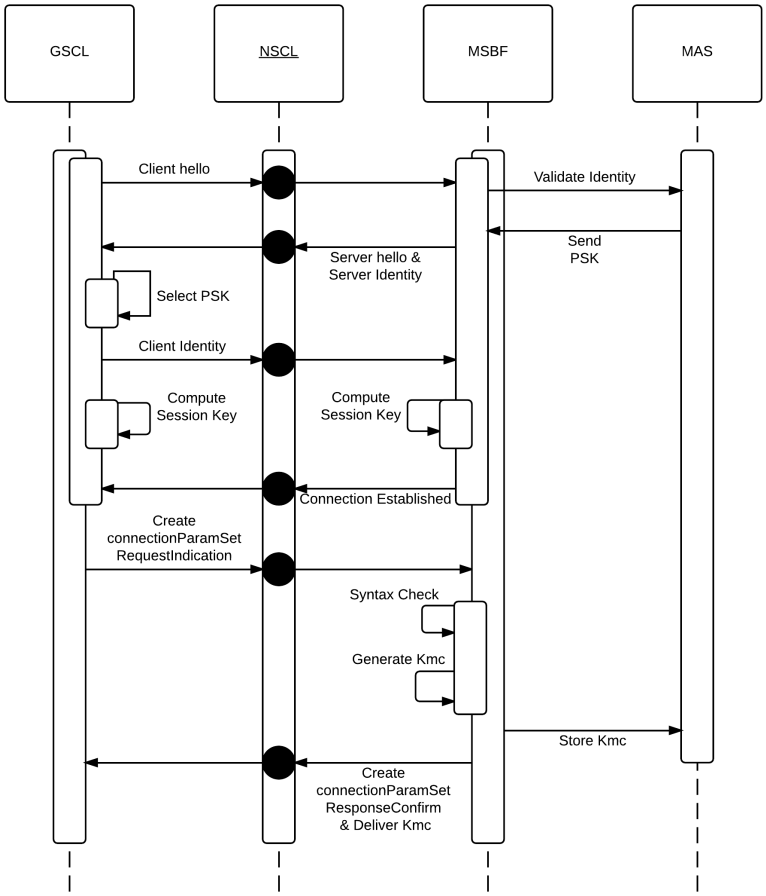

Figure 5.11: Service Connection procedure sequence diagram.

starts the communication by sending the *client hello* specifying the cipher suite about to use for the communication, i.e.: TLS-PSK. The MSBF chooses the cipher suite to use and answers with the *server hello* message plus the server's identity. Then the GSCL receives the message and sends its identity back to the MSBF and uses the server's identity to obtain the corresponding Kmr. Once both entities are synchronized with each other's identities, the secure channel is established and the service connection conversation start to take place.

At this point the channel is secure and the GSCL starts by creating the *connectionParamSetRequestIndication* object and sends it towards the MSBF. Then the MSBF will do a syntax check and derive the Kmc key based on the root key (Kmr), once the key has been derive successfully the MSBF will proceed to store a copy in the secure environment (MAS) and send another copy to the GSCL. After the GSCL has received the key, the service connection procedure is finished and communications with the network middleware can take place over a secure channel.

5.4 SUBSCRIPTIONS

The NSCL provides a subscription methods to NA, in order to receive notifications when the resources, which were subscribed to, were modified. The notification engine relies on a message broker (ActiveMQ) to store, temporarily, the message on a queue. The broker is configured to function as publish/subscribe, which means that the message goes to all the subscribers who are interested - so one or more subscribers will receive a copy of the message. What's more, is that the broker is also persistent, meaning that if the application is not ready to accept the notification at a given time, the broker will be keep until the application is able to process the notification.

After the message has been delivered to the queue, the NA is able to retrieve the message in two different ways: asynchronously, by configuring a callback method which will be call when a notification arrives; or synchronously, by actively polling the message broker for new messages.

Figure 5.12 depicts a sequence diagram, on the right side is shown when an NA wants to subscribe to a resource. First of all the NA will register itself against the NSCL, once registered the NSCL will create a queue, in the message broker, on behalf of the NA, where all the notification will be store for later retrieval by the application.

Then an NA can subscribe to one, or more resources, in order to receive notifications, once the subscription has been done, the NSCL shall notify the NA through the message broker, by sending a notification message to the destination queue. The message broker will then, asynchronously send the incoming message to the application.

On the left side is shown how the GA pushes data to a container, first the GA collects the data from the sensors, subsequently it pushes the data to a container, which in turn the GSCL will push the data to the NSCL.

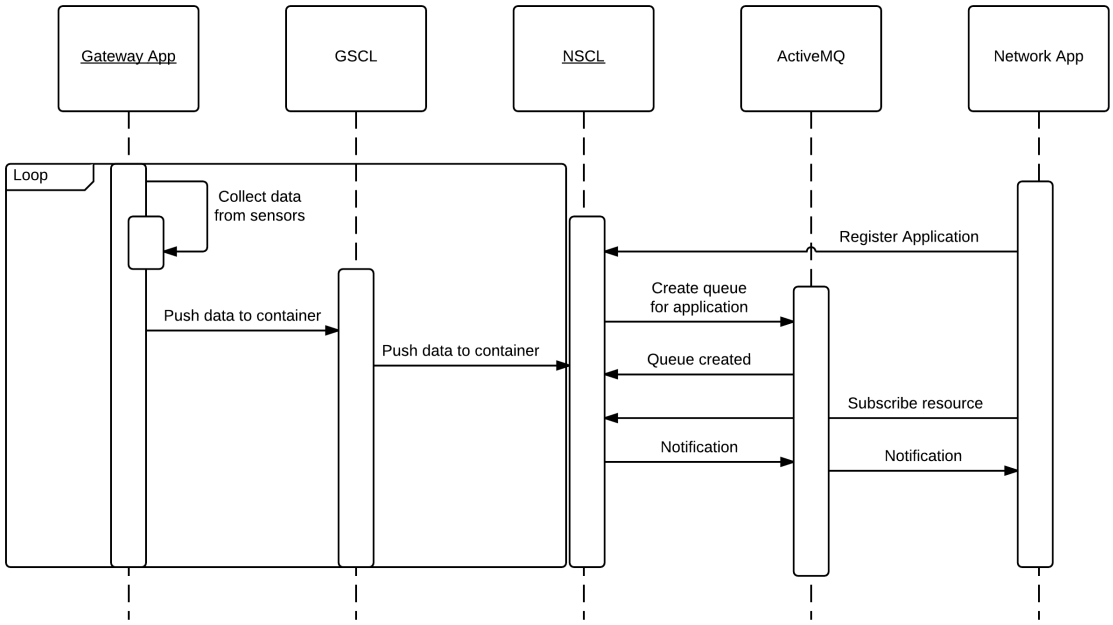

Figure 5.12: Sequence diagram for the notification use case.

5.5 M2M APPLICATIONS

This section covers two of the M2M applications that were develop in the course of the work of this dissertation. The first application developed was in the gateway domain, while the second in the network domain.

5.5.1 GATEWAY APPLICATION

A GA was developed to interact with a device so that some environment variables could be captured. The device is composed of 6 sensors: 2 temperatures analog sensors, 1 temperature and humidity digital sensor, 1 accelerometer, one luminosity sensor and one proximity sensor. In addition the GA is also a GIP, since the device is considered to be a legacy device, and uses a Zigbee area network.

The GA has the following objective, which were all met:

- **Bootstrap** - The GA should be able to bootstrap the device. Although in this specific case is not a M2M bootstrap, with cryptographic key materials, because the device is very constraint and cryptographic communication were not viable, the bootstrap is used to configure some device's variables like the how frequent it should take samples of the environment and send it to the GA.
- **Zigbee area network** - Since the device uses a Zigbee transceiver, the GA must be able to communicate with the device using the Zigbee. Therefore a Zigbee communication module was develop, and CoAP is used, as the transport protocol, over serial communication channel.
- **Notifications** - The GA is programmed to receive asynchronous notifications from the device. Notifications are not processed beyond the removal of the CoAP header, i.e., the payload of the message is not analyze locally.
- **Integration with the NSCL** - Once the notification is processed, the next step is to be sent the payload up to the NSCL, so that it can be analyze properly.

In Figure 5.12 is presented a sequence diagram which shows the procedures explained above.

5.5.2 ADMINISTRATION NA

An NA for administration of local access rights was developed. This NA runs with administration rights, i.e., it has the ability to access and modify any resource that resides in the NSCL's resource tree, and/or create new ones. The purpose of this NA is to allow that the Service Provider has an entry point to the resource tree, and to temporarily allow a user access to some resource.

It is an extension of the mIa reference point, in that it offers a new resource to manage other resources in the tree. As opposed to other resource access in the mIa reference point, this new resource has restricted access only to administrators. By using public key certificates and HTTP basic authentication mechanism, the web service is exposed securely and it ensures both a secure communication channel and proper user authentication. The resource name is called *accessManagement*, and accepts a JSON payload, that has reference to the access right resource that is to be modified,

along with a list of key pair values that specifies if the new permission is to be added or removed and the user name.

CHAPTER 6

Evalution of the solution

Analyzing the impact that the network middleware has on the performance of the ecosystem, since it is the NSCL that offers all the services and data that M2M applications, will eventually come to use. The section is devoted to the evaluation and analysis of the common procedures that applications will come to use, in order to have an idea of the delay imposed by the NSCL and of possible optimization paths.

This chapter is divided in two main tests scenarios, the first test case scenario is devoted to testing the NSCL functionalities and Central Processing Unit (CPU) used, while the second test case scenario is devoted to testing the GSCL implementation of the bootstrap with the NSCL, and to test the GA develop to obtain sensor readings.

6.1 NA and NSCL communication

6.1.1 Deployment scenario

Figure 6.1 illustrates the deployment scenario, where the same computer is running both the NSCL and the NA, and whose data flow goes through the mIa reference point, bidirectionally.

Both software components (namely the NSCL and the testing NA) were deployed on a personal laptop computer, whose specifications are described in the following Table 6.1, including the operating system, the Java virtual machine version, Glassfish application server, and the ActiveMQ message broker.

Characteristic	Description
Processor:	Intel Core-i5 3210M @ 2.5 GHz - 3.10 GHz
Memory (RAM):	6GB DDR3 @ 1600 MHz
Operating System:	Arch Linux - Kernel 3.17 (64-bit)
Java Virtual Machine:	Oracle Java 1.7.0_75
Glassfish Application Server:	4.1
ActiveMQ Broker:	5.9.1

Table 6.1: NSCL host specification

The scenario was setup to test the time it takes the NSCL to respond to the most common operations available, and also to analyze the behavior, and how well it deals with bursts of requests.

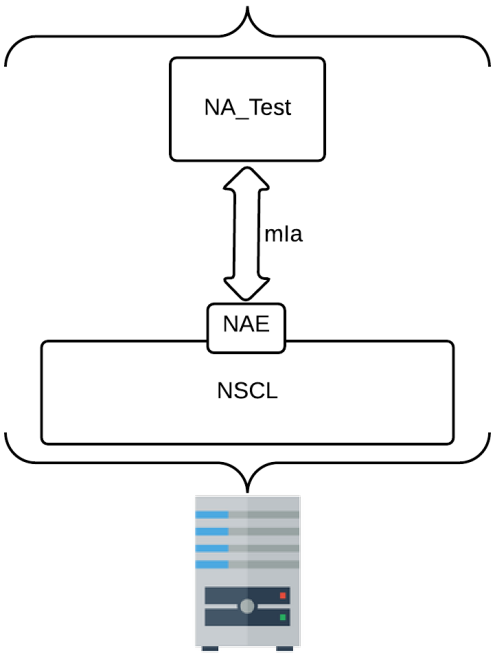

Figure 6.1: Test case 1: NA communication with the NSCL.

The scenario begins with a simple deployment of the NSCL, and a NA that emulates reading data from a sensor and sends it towards the NSCL. This is not the purpose of a NA, on the contrary a NA relates to applications that take data out of the NSCL, perform some accounting operations and display it on charts, like a dashboard. But for this test case scenario the NA will act like a device in order to push date onto the NSCL. The sensor data (the payload) is structured in JSON data format in order to minimize the payload length, since an XML format is much more verbose. To note that communication is running over the mIa reference point, and no cryptographic operations were made.

6.1.2 EXPERIMENT & ANALYSIS

In table 6.3 is shown the values measured when the NA performed the various operations. For each operation 6 samples were taken to obtain an average time and the corresponding standard deviation. Taking in consideration the *Time* column, the time measured encompasses the time of the IPC mechanism, i.e., the time it takes to travel from the message broker to the NSCL, plus the time to process the request and the time to return to the message broker and get consumed by the NA. The next column (*ETSI Overhead*) shows the overhead imposed by the ETSI's XML request. Finally, the last column (*Payload Length*) shows the data payload length of each request, when applicable. Taking a closer look to the 3 first operations one can see that the time taken to finish the operations is a bit high, this is due to the fact that the NSCL uses a method called Marshaller to transform XML documents into their corresponding Plain Old Java Object (POJO) and vice-versa. These operations are complex (including string parsing and type conversions) and thus add delays to the operation

that are quite noticeable, a simple marshalling operation may take, approximately, up to 453 ms. For these tests, two NAs were developed, one was written in Java and the other in Bash. The most expensive operation involves the registration of an application, which takes approximately 1636 ms, followed by the creation of a container 998 ms, and pushing data to a container 588 ms. Lastly, the notification engine time was measured and it takes 198 ms from the moment a resource is modified until the notification is sent to the corresponding queue, since this operation does not involves many marshalling/unmarshalling it takes less time to be done.

Operation	**Time Average (ms)**	σ	**ETSI Overhead (B)**	**Payload Length (B)**
Register NA/NIP (Java)	1636	24.0	596	N/A
Register NA/NIP (Bash)	1373	18.6	596	N/A
Create Container (Java)	998	8.8	621	N/A
Create Container (Bash)	873	10.8	621	N/A
Push data - package 1 (Java)	539	15.5	751	173
Push data - package 1 (Bash)	466	14.2	751	173
Push data - package 2 (Java)	566	9.5	749	170
Push data - package 2 (Bash)	561	8.3	749	170
Push data - package 3 (Java)	502	4.2	748	169
Push data - package 3 (Bash)	588	11.7	748	169
Notification fired	198	4.1	456	169

Table 6.2: NSCL host specification

In order to have a better understanding of the performance of the NSCL, data was collected about the CPU usage percentage and the allocated memory.

Figure 6.2 shows the CPU percentage used by just the NSCL. As can be observed in the graph, the NSCL consumes more resources at the start of the process due to the initialization of the database and the resource structure, and also the initialization of the reference points. After the NSCL has been initialized and the firsts requests start to arrive, the NSCL barely raises up to 10 % of CPU usage. When the the NA start sending messages in burst mode, the NSCL raises up to 25.35 %, and with a response time of 955 ms.

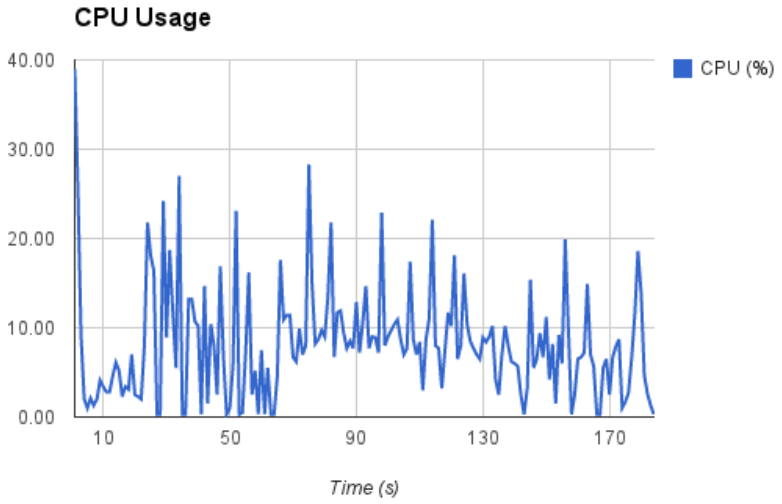

Figure 6.2: NSCL CPU usage

Regarding the memory consumption analysis, Figure 6.3 shows the memory usage of the NSCL for this test case. Initially, the memory consumption rises exponentially given the initialization of the resource structure, plus all the necessary reference points and working threads. Around the 10 seconds the NSCL starts receiving packets at a normal rate, and the memory keeps rising normally and reaches 30 MB. Then, approximately around 30 seconds the NSCL receives packets in burst mode, which makes it reach a peak of 54 MB, but given the different sizes in the incoming data the memory usage fluctuates between 46 MB and 50 MB.

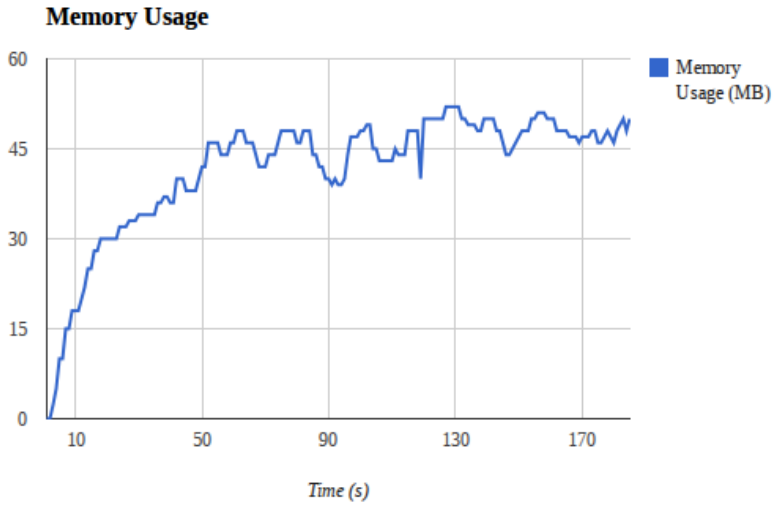

Figure 6.3: NSCL CPU usage

6.2 Complete M2M scenario (DA, GSCL, NSCL)

6.2.1 Deployment scenario

In the second case test scenario, the deployment setup was still using the same laptop computer, Table 6.1, and encompasses the interaction between a device with the GSCL, and consequently the GSCL with the NSCL. Furthermore, the device is running a simple application that perform a basic bootstrap, where the device obtain settings like the time interval between successive readings; and performs readings from the various embedded sensors. The gateway is simulated and there is no real gateway device, hence the GSCL was running in the same computer as the NSCL, which performs the M2M bootstrap and service connection procedures. Figure 6.4 illustrates the scenario just described, where there's no real gateway device and its running beside the NSCL, within the same environment.

In this scenario the end goal was to test the interaction between the device with the gateway (GSCL) and the gateway (GSCL) with the NSCL, and measure the response time it takes each component to perform a particular operation. Therefore, there are 3 components in play in this test scenario: the NSCL, the GSCL and a real device. As mentioned previously, both the NSCL and the GSCL were running in the same computer. Regarding the device, its characteristics are as follows: an 8-bit AVR

CPU with a maximum frequency of 20 MHz and 32 KBytes of flash memory. As can be seen from the characteristics, the device is constraint and not capable of running a DSCL. To that end, the use of the GSCL was necessary in order to allow the device to be connected to the M2M system, even though it is possible to connect a device to a NSCL as long as it is running a DSCL.

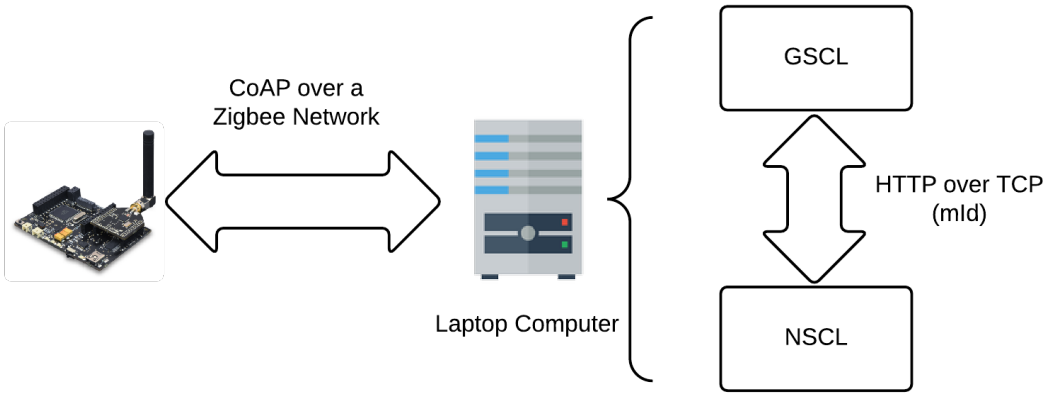

Figure 6.4: Test scenario with a legacy device

6.2.2 EXPERIMENT & ANALYSIS

Operation	Time Average (ms)	σ	Request (B)	Response (B)
Bootstrap device	1494	81.5	23	31
Sending 5 readings	2857	56.0	35	7
Sending analogue	94.5	3.1	15	7
GSCL bootstrap	2019	60.0	102	53
GSCL Service Connection	535	41.5	96	47
GA pushes data to a container in the NSCL	607	5.7	751	46

Table 6.3: NSCL host specification

In the second and third task, are executed consecutive readings from the 5 sensors and are sent towards the GSCL, in order to test the communication medium and how fast it reaches the GSCL. Once the data has reached the GSCL, a GA is going to route the data towards the NSCL, where it can be later processed.

In the case of network bootstrap and service connection between the GSCL and the NSCL, the time needed for the procedures to take place is 2019 ms and 535 ms, respectively. The bootstrap procedure takes longer to accomplished, given the fact that there are both asymmetric and symmetric cryptography operations involved, thus the time is higher. On the other hand, service connection procedure is less expensive given that, even though all key material is in place to perform the cryptographic operations, since there is no open implementation of TLS-PSK in Java, the communication is not encrypted. An important note is that both cases were carefully designed to minimize the usage of the marshalling and unmarshalling (XML to POJO and vice-versa) operations, thus the time is not as high as it was in the previous case.

Regarding the operations on the device, the device bootstrap takes approximately 1494 ms which is mainly due to the fact that it is the first communication in the Zigbee network and the parsing of the message by the device. The time it takes for the data to travel in the Zigbee network, after reading the 5 sensors, is approximately 2857 ms, while reading from the analogue sensor takes approximately 94.5 ms.

Finally, the test where the GA sends data towards the NSCL which takes approximately 607 ms, which is not surprising given that the technologies used in the implementation of the mId reference point, are the same as for the mIa reference point, i.e., HTTP/CoAP through a message broker.

CHAPTER 7

Conclusion

The importance of having interoperable M2M systems for the future Internet has led many SDOs, including ETSI, to produce a standard for such systems, since it is the key to a successful M2M market. This document started with an analysis of the first system architectures that interacted with WSNs, it also presented the main disadvantages that limited their world wide adoption. Then, it proceed with the introduction of an evolutionary system architecture (horizontal architecture). Presented a comparison to its predecessor and showed the many advantages of having an horizontal architecture, more specifically, it presented the ETSI proposal for the system architecture, where vertical M2M applications would leverage this platform in order to provide end user services. By establishing a common framework where manufacturers and network operators will participate, ETSI drove M2M to a framework where the ecosystem will innovate rapidly and have widespread adoption, while at the same time it will lower the costs. Finally, an overview of the previous work with WSN was presented with their evaluation, together with the current projects, around the world, that offered a middleware that implemented the ETSI M2M standard.

Followed by the design of the work done in this dissertation, which mainly focused on the implementation of the network middleware as specified in the ETSI M2M standard. The end goal was to have a working ETSI NSCL middleware, with support for integration with ETSI GSCL, leveraging the bootstrap and service connection procedures to secure communications. The architecture was conceived to allow its extensibility in the addition of capabilities introduced by future revisions of the M2M standard. The extensibility is allowed because of the clear separation between the interface and the implementation of the current capabilities, this allowed for easy integration of new feature, or the replacement of deprecated capabilities. In addition, it also permits extensibility of external entities, through the use of NAs, or when necessary through the use of NIPs service capabilities.

This middleware was tested to prove its functional state and also to search for possible bottlenecks in the current implementation. The test scenarios allowed to stress the NSCL and analyzed its behavior, while under heavy loads. This analysis revealed at least one possible optimization that could be applied to the NSCL, in order to offer better response times. Finally, a legacy device was attached to the system in order to prove the functional state of the integration between a GSCL and the NSCL.

7.1 FUTURE WORK

While the current implementation does cover all the proposed objectives, and provides a successfully tested and working middleware, there is still room for improvement. First improving communication with the NSCL by adding support for MQTT. The second improvement is the synchronization between the GSCL and the NSCL by using *announce* resources and implementing the retargeting mechanism, in order for the NSCL to route request to the appropriate GSCL or DSCL. The third improvement is in securing the mId reference point, even though the bootstrap and service connection procedure does generate the session keys successfully, communications over the mId reference point are not secure because the lack of a library that implements TLS-PSK, i.e., both parties (namely the NSCL and the device or GSCL) securely exchange session keys but future communications over this reference point will not be secure. Another improvement relates to the usage of cache mechanisms offered by proxies in the Internet, the both reference points mIa and mId, could take advantage of this feature so that future requests could be resolved rapidly without accessing the NSCL.

Appendices

APPENDIX A

TECHNOLOGICAL BACKGROUND

In the following sections will be introduced the technologies used in the development of this dissertation.

A.1 SERVICE-ORIENTED ARCHITECTURE

Service-Oriented Architecture (SOA) is a paradigm that focuses on the development of services that provides a higher level of abstraction from a functional standpoint. A service is a formal contract which defines the interface bound to the one or more implementations, in order to satisfy a business function. Furthermore the service can be self-contained, autonomous and loosely coupled, i.e., it must not depend on other service in order to work - in other words *atomic*. Or it can be a composite service, that is, it may depend on one, or more, services to achieve the required functionality. The great benefits of SOA are the flexibility and agility it brings to the development of a project and to adaptability for future changes. SOA is based in the following principles: abstraction, autonomy, reusability, discoverability, formal contract, loose coupling and statelessness.

RESTful. Representational State Transfer (REST) is an architecture, based on a client-server model, which allows applications to rely on a loosely coupled set of services that can be shared and reused. In REST everything is viewed as a resource, which is any document or object that can be stored on a computer and is identified by an URI and is available as a stream of bytes. This architecture when applied to the web service technology (the so called RESTful web services) has the following advantages to older Remote Procedure Call (RPC) technologies.

RPC	**RESTful**
Use two methods (GET and POST) for all requests	Variety of method for each kind of request (GET, POST, DELETE, etc)
Message send within an envelope	Usage of nouns as part of their URIs
One or two URI for all the methods (described in a Web Services Description Language (WSDL))	Multiples URI, one for each resource
The document body used to describe errors	Uses HTTP error codes to describe errors
Describe metadata in the envelope (message body)	Describe metadata using HTTP headers.

Table A.1: Comparison between RPC-style and RESTful-style

A.2 COMMUNICATION PROTOCOLS

This section presents the various communication protocols used in the development of this dissertation. First it compares the two standard protocols for a RESTful architecture: HTTP and CoAP. Second it introduces the AMQP used for messaging.

a.2.1 HTTP AND COAP

The HTTP is an application protocol standardized by IETF in 1999 for distributed, collaborative, hypermedia information systems. The HTTP protocol is used, largely, by the World Wide Web (WWW) as means of transferring data across the network. The protocol uses one of the basic methods of communication between computers, the request-response model wherein the client sends a request to a server and waits for the response. In addition the protocol offers a reliable transport and flow control, so that the sender cannot overflow the receiver by sending too much information too fast, because it uses TCP for transport.

CoAP was design in March 2010, by the IETF Constrained RESTful environments (CoRE) working group as the protocol for RESTful web service in resource-constraint devices, with the goal of keeping the protocol overhead as small as possible and limit the use of fragmentation. The syntax of CoAP is very similar to HTTP, since it was based on it, in fact the similarities with HTTP makes the conversion between these two protocols very easy. Some HTTP functionalities were re-design, taking into account the low processing power and energy of embedded devices, and at the same time, new functionalities have been added in order to make the protocol ideal for IoT and M2M communications.

The most significant change between CoAP and HTTP, is that the former relies in the User Datagram Protocol (UDP) as the transport protocol, whereas the later relies on TCP, as mentioned above. The TCP flow control mechanism was found not appropriate and as having a high overhead. In addition TCP does not have multicast support, which was added to CoAP, resulting in a simplistic implementation with low overhead, which is necessary for M2M communications.

CoAP has two layers: a transaction layer and a request/response layer. The former is responsible for the handling of single message exchange between the endpoint, while the later is where the REST based communication occurs. This approach allows the protocol to provide reliability mechanisms even without the use of TCP as transport protocol.

Figure A.1 shows a stack comparison between the two protocols, at the left side is the stack use by HTTP, while at the right side is the stack used by CoAP.

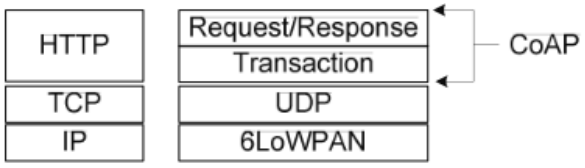

Figure A.1: Comparison between HTTP and CoAP [42]

In this dissertation both protocols were used, HTTP is used in the network domain to support communication between the applications and the NSCL, while CoAP is used in the device/gateway domain to support the communication between the GA and the device.

a.2.2 ADVANCED MESSAGE QUEUING PROTOCOL

AMQP is an open standard application layer protocol for message-oriented middleware. It is a binary protocol that defines the format of the data sent across the network. It is design to support many messaging and communication patterns and with interoperability between vendors, in contrast to previous standardization like Java Messaging Service (JMS) that only provides compatibility at the API level. As a result any implementation that conforms to AMQP's data format can interoperate.

AMQP protocol features the following functionality: message orientation and message delivery, queuing, routing (including point-to-point and publish-and-subscribe), reliability, authentication and security through the use of Simple Authentication and Security Layer (SASL) and/or TLS. The standard mandates the usage of a reliable transport protocol, such as TCP.

In the work of this dissertation, the message broker is used for the publish/subscribe model, which is provided natively, and is also used as the IPC mechanism for a bidirectionally communication pipe to exchange data between two entities. Since the publish/subscribe model is asynchronous, i.e., the recipient has a dedicated queue where it awaits for incoming request, and cannot respond back to the sender. With the usage of JMS it was possible to work around this problem by creating a temporary queue on each request. By having this temporary queue, the sender can wait for a response on this temporary queue, while the recipient has a way to put the response. This method allows for synchronous communication over an asynchronous channel, and also allows the request-response model needed for bidirectionally exchange data, just like method calls.

A.3 CRYPTOGRAPHY AND AUTHENTICATION

Cryptography is within the field of computer security that covers the mechanism that protects information from unauthorized users. Cryptography besides of being related to data confidentiality, is also related to data integrity and authentication.

This section is divided in: Section A.3.1 will introduce the notion of data encryption, Sections A.3.2 will introduce the notion of a cryptographic key and its types.

a.3.1 ENCRYPTION

Encryption is the process by which a message, called plaintext, is transformed into an encoded or unintelligible text called ciphertext, that only the intended party will be able to decipher and understand. Encryption is used when confidential, or sensitive, data must travel in insecure communication channels without being intercepted by unauthorized entities. Encryption is also used to provide user's privacy. A given encryption algorithm uses a value, named key, to create the ciphertext. A key is a parameter that determines the functional output of the algorithm.

The strength of a ciphertext depends in the algorithm used and also in the key chosen. If the key is short/weak, an attacker could crack the encryption and have access to the information, whereas if the key is strong, it takes longer to crack the encryption thus making the attempt unfeasible.

Decryption is the inverse process of encryption and transforms a ciphertext into the original message.

a.3.2 CRYPTOGRAPHIC KEYS

A cryptographic key is a parameter used in an encryption algorithm that determines its functional output. There are many types of cryptographic keys, but in this section will be presented only three types of keys: Symmetric keys, Symmetric key wrapping and Asymmetric keys.

SYMMETRIC-KEY ALGORITHMS

Symmetric-key, or PSK, algorithms, as the name implies, uses the same common value (the key) to encrypt and decrypt the information. Since there is only one key to achieve the encryption and decryption, any party with access to that key will have access to the information. Symmetric encryption can be used in two main scenarios: for an individual to protect sensitive data, in this scenario the key must remain secret; or to establish a secure communication channel between two or more entities that wishes to exchange information, in this scenario the key must be provision to the involved parties before any conversation takes place. The drawback of key exchange can be mitigated by using a key exchanged protocol, like the Diffie-Hellman key exchange protocol or through an already established secure channel. Another disadvantage of this type of cryptography is related to key management, as the number of peers increases, it also increases the number of keys that need to be exchanged.

Given the nature of symmetric-key algorithms, is computationally less expensive to encrypt/decrypt a piece of information with symmetric-key algorithms, than it is with the alternative Asymmetric-key algorithms.

Implementations of symmetric-key algorithms are based on two types of ciphers: stream ciphers and block ciphers, the first one encrypts the message byte-by-byte, while the later uses a predefined number of bits, that forms a block, to perform the encryption. When the text is not enough to match the block's size, padding may be used. Actual implementations of symmetric-key encryption include: Twofish, AES, Blowfish and RC4.

ASYMMETRIC-KEY ALGORITHMS

In contrast to Symmetric-key algorithms (refer to Section A.3.2), Asymmetric-key (or public key) algorithms need two keys to achieve encryption and decryption procedures. The asymmetric key pairs used in these kind of algorithms allows for a key to be publicly known used for encryption or verification of a digital signature, and a private key that only the owner of the asymmetric key pair should posses (e.g. a device) used for decryption or the creation of a digital signature. Therefore, since the two keys perform different operations, there is no need of an exchange key protocol like in the case of symmetric cryptography.

Other uses of public key cryptography is for digitally signed information, for that, the public key is used for decrypting the ciphertext and the private key is used for encrypting it. In this specific scenario, the goal is not to hide the information but to assure that they owner of the key pair was, in fact, the one who encrypted it, through the assumption of the mathematical correlation between the public and private key pair.

In comparison with their counterpart, public key cryptography offers better security at the cost of being computationally more expensive, given the use of more complex mathematical problems like: integer factorization, discrete logarithm and elliptic curve relationships. The strength in this kind of encryption is obtained from the fact that it is computationally unfeasible to crack, a properly

generated, private key from its corresponding public key.

An example algorithm that implements asymmetric cryptography is known as RSA. It was first published in 1977 by Ronald Rivest, Adi Shammir and Leonard Adleman, and later on implemented. Another example is the ElGamal algorithm publish in 1985 by its author Taher ElGamal.

KEY WRAPPING

Key Wrapping is a class of symmetric encryption algorithms designed to encapsulate cryptographic key material. By encapsulating a key, its possible to protect the key while transmitting it over a insecure communication channels.

PUBLIC KEY CERTIFICATES

A certificate is an electronic document with a predefined structure[1] that holds the identity of a given entity: the name of a person or organization, the address, the e-mail, the entity's public key (from a previously generated asymmetric key pair) and a digital signature that binds all this information together. The signature is made by a CA, whose role is to validate the information so that every person or entity that trust it can verify, and also trust, the certificate. A certificate also have a limited validity time, which can be configure within the certificate itself or through a Certificate Revocation List (CRL) issued by the CA. The later can be used if the private key is compromised before the validity time has expired. Therefore a certificate is said to be valid if the validity time has not been exceed or it hasn't been revoked through the latest CRL issued by the CA.

An alternative method to validate certificates was design later on by the IETF and called Online Certificate Status Protocol (OCSP). The protocol carries less information than the CRLs, making it suitable to be used efficiently in network environments to obtain certificate's revocation status.

a.3.3 HASH-BASED MESSAGE AUTHENTICATION CODES

A Hash-based Message Authentication Codes (HMAC) is a robust method of generating a Message Authentication Codes (MAC), since it is bind with a hash function. The hash function used is configurable and can be any of the existing algorithms, e.g. MD5, SHA-1. To generate a HMAC the hash function is applied twice, the first time it processes the secret key and the message (sometimes referred as *inner* hash), while the second time it processes the secret with the inner hash (sometimes referred as *outer* hash).

HMAC was first proposed by Bellare, Canetti and Krawczyk in 1996. Uses of HMAC can be found in many popular software implementations like IPSec, SSH[2], and TLS.

[1]The certificate's structure is standardized by International Telecommunication Union - Telecommunication (ITU-T) by the name of X.509

[2]SSH is both a protocol and set of programs that provides secure shell sessions over a network, usually the Internet.

a.3.4 TRANSPORT LAYER SECURITY

TLS is a cryptographic protocol that provides communications security over insecure channels. Its predecessor was the Secure Socket Layer (SSL) that was develop by Netscape as to provide secure communications for the web (HTTP). SSL was standardized by the IETF with a new name, TLS, and it is now used as a replacement for communications over the Internet. The core functionalities of both protocols is to prevent eavesdropping, tampering and message forgery.

The TLS protocol uses public key certificates to authenticate the server, and optionally, the client in a conversation. The first step is the so called Handshake process, in which the server presents its certificate, along with other values, to the client and this one validates it, optionally, if the server is configured, it asks the client to presents its own certificate before continuing with the process. If both certificates are considered valid, then the handshake is finished with the creation of a set of symmetric session keys. The symmetric session keys can now be used to secure the channel.

APPENDIX B

ETSI RESOURCE STRUCTURE

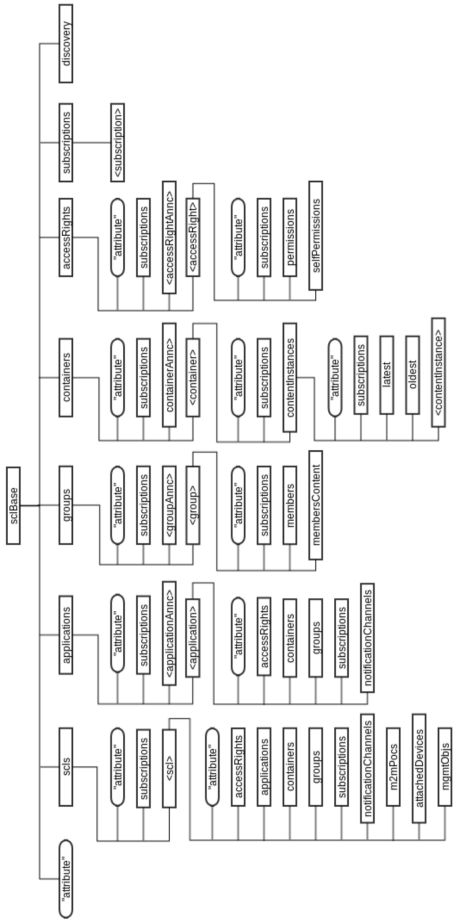

Figure B.1: ETSI resource structure

References

[1] N. Streitz. (). Smart future initiative, [Online]. Available: http://www.smart-future.net/10.html (visited on 09/30/2014).

[2] Q. Wang and I. Balasingham, "Wireless sensor networks-an introduction", *Chapter*, no. 187857, 2010. [Online]. Available: http://cdn.intechweb.org/pdfs/12464.pdf.

[3] S. Labs, "The Evolution of Wireless Sensor Networks", pp. 1–5,

[4] A. Gluhak and W. Schott, "A WSN System Architecture to Capture Context Information for beyond 3G Communication Systems", *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*, pp. 49–54, 2007. DOI: 10.1109/ISSNIP.2007.4496818.

[5] S. Higginbotham, *Sensor Networks Top Social Networks for Big Data*, 2010. [Online]. Available: https://gigaom.com/2010/09/13/sensor-networks-top-social-networks-for-big-data-2/ (visited on 10/20/2014).

[6] G. Intelligence, "From concept to delivery: the M2M market today", *White Paper, Feb*, no. February, pp. 1–21, 2014.

[7] M. Chen, J. Wan, S. González, X. Liao, and V. Leung, "A survey of recent developments in home M2M networks", vol. 16, no. 1, pp. 98–114, 2013.

[8] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey", *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010, ISSN: 13891286. DOI: 10.1016/j.comnet.2010.05.010.

[9] A. Jara, L. Ladid, and A. Skarmeta, "The Internet of Everything through IPv6: An Analysis of Challenges, Solutions and Opportunities", *J. Wirel. Mob. Netw. Ubiquitous*, pp. 97–118, 2013. [Online]. Available: http://ipv6forum.com/iot/images/jowua-v4n3-6.pdf.

[10] N. Kushalnagar, G. Montenegro, and C. Schumacher, *RFC 4919 - IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*, 2007.

[11] A. Jara, M. Zamora, and A. Skarmeta, "An Ambient Assisted Living Platform to Integrate Biometric Sensors to Detect Respiratory Failures for Patients with Serious Breathing Problems", *Ambient Assisted Living*, 2011.

[12] S. Mendoza, *IoT Leading to Intelligent Transportation*, 2014. [Online]. Available: http://www.iotworld.com/author.asp?section%5C_id=3194%5C&doc%5C_id=563228.

[13] D. Bernstein, N. Vidovic, and S. Modi, "A Cloud PAAS for High Scale, Function, and Velocity Mobile Applications - With Reference Application as the Fully Connected Car", *2010 Fifth International Conference on Systems and Networks Communications*, pp. 117–123, Aug. 2010. DOI: 10.1109/ICSNC.2010.24.

[14] C. V. T. Association. (). The connected vehicle trade association (cvta), [Online]. Available: http://www.connectedvehicle.org/ (visited on 09/30/2014).

[15] A. T. ATNoG and N. G. .-.-. I. de Telecomunicações. (). The apollo project, [Online]. Available: http://atnog.av.it.pt/projects/apollo (visited on 09/24/2014).

[16] L. C. Distribuidas. (). Use cases for the internet of thing, [Online]. Available: http://www.libelium.com/top_50_iot_sensor_applications_ranking/ (visited on 09/20/2014).

[17] P. B. Gibbons, "IrisNet : An Architecture for Internet-Scale Sensing",

[18] J. Shneidman, P. Pietzuch, and J. Ledlie, "Hourglass: An infrastructure for connecting sensor networks and applications", 2004. [Online]. Available: http://eecs.harvard.edu/~syrah/hourglass/papers/tr2104.pdf.

[19] R. Gold, A. Gluhak, N. Bui, A. Waller, C. Sorge, F. Montagut, V. Stirbu, H. Karvonen, V. Tsiatsis, S. Pennington, Z. Shelby, J. Vercher, M. Johansson, J. Bohli, T. Bauge, S. Esfandiyari, S. Haller, E. Kovacs, R. Egan, F. Carrez, and M. Presser, "Sensei d3.1: State of The Art – Sensor Frameworks and Future", 2008.

[20] I. Chatzigiannakis and C. Koninis, "A peer-to-peer framework for globally-available sensor networks and its application in building management", 2009.

[21] E. Foundation. (). Eclipse openiot working group, [Online]. Available: http://iot.eclipse.org/ecosystem.html (visited on 09/20/2014).

[22] ETSI. (). Technologies standardized by etsi, [Online]. Available: http://www.etsi.org/index.php/technologies-clusters/technologies/ (visited on 09/18/2014).

[23] J. Swetina and G. Lu, "Toward a standardized common m2m service layer platform: Introduction to onem2m", *Wireless ...*, no. June, pp. 20–26, 2014.

[24] O. M. Alliance. (). Oma device management, [Online]. Available: http://openmobilealliance.org/about-oma (visited on 10/20/2014).

[25] ——, (). Oma device management releases, [Online]. Available: http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases (visited on 10/20/2014).

[26] ETSI TS 102 690 V2.1.1 (2013-10), "Machine-to-Machine communications (M2M); Functional architecture", vol. 1, 2013.

[27] D. Boswarthick, O. Elloumi, and O. Hersent, *M2M Communications: A Systems Approach*. Wiley, 2012, ISBN: 9781119994756. [Online]. Available: http://books.google.pt/books?id=7Xdz3ryx0TIC.

[28] ETSI, "Machine-to-Machine communications (M2M), mIa, dIa and mId interfaces", vol. 1, pp. 1–538, 2012.

[29] A. SA, "Cocoon ® Project User Guide : ONG tutorial", 2011.

[30] Actility. (). Cocoon overview, [Online]. Available: http://cocoon.actility.com/documentation/ongv2/overview (visited on 09/20/2014).

[31] M. Corici, H. Coskun, A. Elmangoush, A. Kurniawan, T. Mao, T. Magedanz, and S. Wahle, "OpenMTC: Prototyping Machine Type communication in carrier grade operator networks", *2012 IEEE Globecom Workshops*, pp. 1735–1740, Dec. 2012. DOI: 10.1109/GLOCOMW.2012.6477847.

[32] F. FOKUS. (). Openmtc releases roadmap, [Online]. Available: http://www.open-mtc.org/openmtc_platform/releases/index.html (visited on 09/20/2014).

[33] M. B. Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira, "OM2M: Extensible ETSI-compliant M2M Service Platform with Self-configuration Capability", *Procedia Computer Science*, vol. 32, pp. 1079–1086, 2014, ISSN: 18770509. DOI: 10.1016/j.procs.2014.05.536.

[34] ETSI Technical Specification, "Digital cellular telecommunications system; Universal Mobile Telecommunications System; LTE; Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA)", 2012.

[35] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, *RFC 3748 - Extensible Authentication Protocol (EAP)*.

[36] F. D, O. Y, P. B, T. H, and Y. A, *RFC 5191 - Protocol for Carrying Authentication for Network Access (PANA)*.

[37] V. Cakulev, G. Sundaram, and I. Broustis, *RFC 6539 - IBAKE: Identity-Based Authenticated Key Exchange*.

[38] T. Dierks and E. Rescorla, *RFC 5246 - The Transport Layer Security (TLS) Protocol*.

[39] P. E. H. Tschofenig, *RFC 4279 - Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*.

[40] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, *RFC 6733 - Diameter Base Protocol*.

[41] T. M. G. Lam, "Machine-to-Machine (M2M) in Ubiquitous Computing", 2013.

[42] W. Colitti, K. Steenhaut, and N. D. Caro, "Integrating wireless sensor networks with the web", pp. 2–6, 2011.