FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mobile Healthcare on a M2M Mobile Gateway

Ricardo Jorge Travanca Morgado

WORKING VERSION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Doctor Ana Cristina Costa Aguiar

June 15, 2014

Abstract

Here goes the abstract written in English.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed vehicula lorem commodo dui. Fusce mollis feugiat elit. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec eu quam. Aenean consectetuer odio quis nisi. Fusce molestie metus sed neque. Praesent nulla. Donec quis urna. Pellentesque hendrerit vulputate nunc. Donec id eros et leo ullamcorper placerat. Curabitur aliquam tellus et diam.

Ut tortor. Morbi eget elit. Maecenas nec risus. Sed ultricies. Sed scelerisque libero faucibus sem. Nullam molestie leo quis tellus. Donec ipsum. Nulla lobortis purus pharetra turpis. Nulla laoreet, arcu nec hendrerit vulputate, tortor elit eleifend turpis, et aliquam leo metus in dolor. Praesent sed nulla. Mauris ac augue. Cras ac orci. Etiam sed urna eget nulla sodales venenatis. Donec faucibus ante eget dui. Nam magna. Suspendisse sollicitudin est et mi.

Fusce sed ipsum vel velit imperdiet dictum. Sed nisi purus, dapibus ut, iaculis ac, placerat id, purus. Integer aliquet elementum libero. Phasellus facilisis leo eget elit. Nullam nisi magna, ornare at, aliquet et, porta id, odio. Sed volutpat tellus consectetuer ligula. Phasellus turpis augue, malesuada et, placerat fringilla, ornare nec, eros. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vivamus ornare quam nec sem mattis vulputate. Nullam porta, diam nec porta mollis, orci leo condimentum sapien, quis venenatis mi dolor a metus. Nullam mollis. Aenean metus massa, pellentesque sit amet, sagittis eget, tincidunt in, arcu. Vestibulum porta laoreet tortor. Nullam mollis elit nec justo. In nulla ligula, pellentesque sit amet, consequat sed, faucibus id, velit. Fusce purus. Quisque sagittis urna at quam. Ut eu lacus. Maecenas tortor nibh, ultricies nec, vestibulum varius, egestas id, sapien.

Donec hendrerit. Vivamus suscipit egestas nibh. In ornare leo ut massa. Donec nisi nisl, dignissim quis, faucibus a, bibendum ac, diam. Nam adipiscing hendrerit mi. Morbi ac nulla. Nullam id est ac nisi consectetuer commodo. Pellentesque aliquam massa sit amet tellus. Vivamus sodales aliquam leo.

Acknowledgements

Aliquam id dui. Nulla facilisi. Nullam ligula nunc, viverra a, iaculis at, faucibus quis, sapien. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Curabitur magna ligula, ornare luctus, aliquam non, aliquet at, tortor. Donec iaculis nulla sed eros. Sed felis. Nam lobortis libero. Pellentesque odio. Suspendisse potenti. Morbi imperdiet rhoncus magna. Morbi vestibulum interdum turpis. Pellentesque varius. Morbi nulla urna, euismod in, molestie ac, placerat in, orci.

Ut convallis. Suspendisse luctus pharetra sem. Sed sit amet mi in diam luctus suscipit. Nulla facilisi. Integer commodo, turpis et semper auctor, nisl ligula vestibulum erat, sed tempor lacus nibh at turpis. Quisque vestibulum pulvinar justo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nam sed tellus vel tortor hendrerit pulvinar. Phasellus eleifend, augue at mattis tincidunt, lorem lorem sodales arcu, id volutpat risus est id neque. Phasellus egestas ante. Nam porttitor justo sit amet urna. Suspendisse ligula nunc, mollis ac, elementum non, venenatis ut, mauris. Mauris augue risus, tempus scelerisque, rutrum quis, hendrerit at, nunc. Nulla posuere porta orci. Nulla dui.

Fusce gravida placerat sem. Aenean ipsum diam, pharetra vitae, ornare et, semper sit amet, nibh. Nam id tellus. Etiam ultrices. Praesent gravida. Aliquam nec sapien. Morbi sagittis vulputate dolor. Donec sapien lorem, laoreet egestas, pellentesque euismod, porta at, sapien. Integer vitae lacus id dui convallis blandit. Mauris non sem. Integer in velit eget lorem scelerisque vehicula. Etiam tincidunt turpis ac nunc. Pellentesque a justo. Mauris faucibus quam id eros. Cras pharetra. Fusce rutrum vulputate lorem. Cras pretium magna in nisl. Integer ornare dui non pede.

Ricardo Morgado

"I work on the motto that if something's not impossible, there must be a way of doing it."

Sir Nicholas Winton

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

CoAP	Constrained Application Protocol
HTTP	Hypertext Transfer Protocol
M2M	Machine to Machine

Chapter 1

Introduction

In a society irreversibly marked by the everyday use of technology, new ways to automatically share this data and automate systems and decisions are constantly emerging. Machine-to-Machine (M2M) solutions are becoming increasingly popular, for the great scalability they provide, and because they generally make people's lives easier. An easily understandable example of M2M system is GPS (Global Positioning Satellites) [6, 7]. There is no human interaction, and yet by gathering and processing data from satellites, it is able to compute the user's current position. The ease of use of these systems grant them the massive adoption they hold in today's society.

The concept of the Internet of Things (IoT), explained in [8] by Atzori, Iera and Morabito, states that in the near future, everyday objects as simple as food packages, paper documents or even furniture will have the ability to be connected to each other and to the Internet, providing services that can hopefully ease people's lives. M2M systems can act as a possible enabler of this IoT future, by providing the means for the objects to interact with each other and enable the creation or services that rely on their information.

This dissertation is going to study how the smartphone can fit into this M2M system, in an health-care scenario, aiming to allow the user to effortlessly make blood pressure or weight measurements and store them in the Cloud. This will allow the user, for example, to conveniently search his medical measurements history. The use cases and scenario in study are detailed in Section 3.1.

The context of the project in which this dissertation is inserted, is explained in Section 1.1, followed by the motivation for studying the subject at hand, in Section 1.2. The objectives this dissertation aims to achieve are stated in Section 1.3, before a brief explanation of the document's structure, in Section 1.4.

1.1 Context of Project

This dissertation is integrated in a joint project between the *Instituto de Telecomunicações (IT Porto)* and *Portugal Telecomunicações Inovação (PT Inovação)*. It involves the creation of a mobile M2M Gateway using the ETSI M2M standard (explained in greater detail in Section 2.3)

as an Android application. This gateway will aggregate sensor data (either internal or external) and manage the communication with the M2M network domain, as a proxy for the sensors. The Gateway was designed to be multi-protocol, and originally implemented with MQTT (Message Queue Telemetry Transport [9]). HTTP [10] has since then been used to implement the message exchanges with the server, while CoAP (Constrained Application Protocol [11, 12]) support is on hold and will be added later. AMQP [13] remains a possibility for later addition, although as we will see in the next Chapter, they will not fit the M2M system. All these protocols are described in Section 2.5.

Back when the project started in May 2013, it focused in the architectural design and implementation of the M2M Gateway, with MQTT. The aim was to develop it as modularly as possible, to ease the addition of protocols and sensors. At the beginning of this dissertation (September 2013), version 0 of the Gateway had been released with the MQTT protocol working (sending data to *PT*'s Broker), as well as the Gateways base architecture, which had the ability to gather the internal sensors data, store it and then send in defined intervals or when the buffer got full. It also featured the modularity which was desired, having each significant module running in its own thread, to improve parallel processing and thus overall performance.

1.2 Motivation

Nowadays, the ascension of technology is a given fact, as evident by the well-known Moore's Law [14], which has been fairly accurate since the paper was published back in 1965. As today's trends focus on the ubiquity of smartphones, there is a massification of applications that aim to ease people's lives. Despite this, there are not many applications focused on M2M systems, let alone mobile M2M scenarios, which inspired this dissertation's theme.

M2M communications, which will be further explained in Section 2.3, have an important role in interconnecting sensors and services, providing the means to allow data to be seamlessly stored and then available to applications authorized to access it. As an example, cities are starting to explore the possibility of using M2M Smart Grids [15, 16], that allow meter measurements to be automatically uploaded to the service provider, either it is water, electricity or gas. There are also other scenarios in smart homes, where M2M communication can be used to autonomously optimize and manage energy consumption of any applicable appliance that runs on electricity [17].

These and many other scenarios are becoming a reality every day, but the lack of mobile solutions for the M2M architecture limits its possibilities. Evaluating an health-care scenario, it should be possible for someone who suddenly felt bad to make a blood pressure measurement, with a friend's or pharmacy's sphygmomanometer, and upload for his doctor's assessment, storing it in the service for future reference in his medical history.

These needs provided the proper motivation for the study of this subject, since it will only be possible if the M2M modules are mobile. Smartphones are then an excellent choice, since they are permanently around the user, possess the necessary processing, battery, and connectivity

power needed for a Gateway, while retaining the mobility necessary to fulfill a new wider range of scenarios.

Having a smartphone working as an M2M Gateway has its advantages, but since users usually do their best to save battery power, the smartphone in this dissertation's scenario is being considered constrained, so that the application working as the M2M Gateway uses the absolute minimum resources possible, to fulfill this requirement.

1.3 Objectives

The goal of this dissertation's project is to create a mobile M2M system composed by a Gateway and a Network Application that will work together in a health-care scenario. Both will be built on a modular structure that can be facilitate the addition of new protocols, sensors and functionalities. The communication between them and the server will be executed with a supported protocol, following the ETSI M2M communications standard.

The M2M Gateway will gather and process the sensor information and send it to the Network Application through the NSCL. On the other hand, the Network Application will be able to send commands to the Gateway, triggering sensor searches and starting/stopping the readings. The M2M concepts will be further explained in Section 2.3.1 and the Use Cases will be addressed in Section 3.1.

Given the constrained nature of smartphones, there is a special interest in saving its scarce resources, and so one of the objectives is also to determine a way to save the battery drain and network usage on 3G/Wi-Fi networks, by using a local bypass in certain situation.

1.4 Structure

This document is structured in 5 Chapters. After this introduction to the subject of this dissertation, Chapter 2 is focused on the bibliographic revision focusing on picturing the current state of the art of the topics related to this dissertation. On Chapter 3 the complete scenario in study is explained, explaining the importance of each part to the project's goal. Then, in Chapter 4, the future tasks and the work plan for the upcoming semester is detailed, followed by Chapter 5 which concludes this document with some final remarks about the dissertation.

Chapter 2

State of the Art

In this Chapter there is going to be a closer study of M2M communications, explaining its importance in the evolution of information technology. This Chapter aims to help the reader fully understand the technologies involved in the development of this dissertation's project.

First off, a useful explanation of the Publish Subscribe paradigm on which the M2M communication is based is explained in Section 2.1, followed by an overview of the OSI Layers on which every networking communication relies, on Section 2.2.

First, in Section 2.3, there is going to be a wider approach into machine-to-machine communications, followed by the ETSI proposed general architecture in Section 2.3.1. Then a a deeper look will be taken into the role of a Gateway in the general M2M architecture (2.3.1.2), followed by the role of a Network Application (2.3.1.3), followed by the most common marshalling techniques such as XML and JSON in Section 2.4.

This will pave the way for a deeper look into the more commonly discussed protocols to provide M2M communications, namely HTTP, CoAP, MQTT and AMQP, respectively in Sections 2.5.1, 2.5.2, 2.5.3 and 2.5.4.

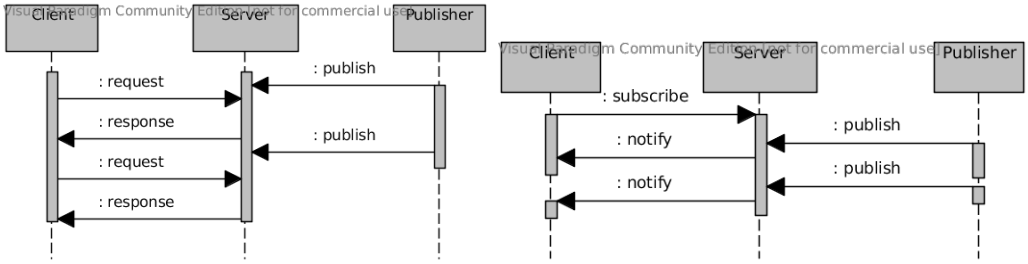
Then, in Sections 2.7.1 and 2.7.2, the IEEE 11073 standard and the Continua Alliance protocol will be detailed, followed by an application revision of the systems currently using these technologies in health-care scenarios.

Last but not least, the Section 2.8 will resume this Chapter's study.

2.1 Publish-Subscribe Paradigm

As wireless grew, by 2002, the author of [18] stated that: "Wireless has experienced explosive growth in recent years, and 'push' will be the predominant wireless service delivery paradigm of the future". Push is another term for the publish subscribe paradigm that basically refers to systems that have the architecture necessary for users to subscribe to a topic, message, publisher, etc., and receive all the subsequent updates about it. The usual web paradigm needs a request if information is desired (polling), but this is inefficient as it overloads the servers with unnecessary requests, slowing down the response times.

Please note Figure 2.1, for the exact same interaction. In 2.1a the client is polling the server for the publisher's content, having the need to re-poll whenever there is new content. On the other hand, in Figure 2.1a, the client only has the need to subscribe once, to receive notifications for the following publications, until the subscription is canceled.



(a) Request-Response model (b) Publish-Subscribe model

Figure 2.1: Model comparison

Publishers and subscribers do not need to be actively participating in the interaction at the same time, or even know about each other. They can both produce and consume events in an asynchronous way (i.e. not online at the same time), which allows for very flexible scenarios.

2.2 Networking Layers - OSI Model

The Open Systems Interconnection, or OSI model (ISO/IEC 7498-1 [19]), was created by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) in order to create an abstraction model where each layer had a specific objective of providing some communication characteristics for the layer above, and together create a functional and reliable system. The OSI model Layers are shown in Figure 2.2 and explained below.

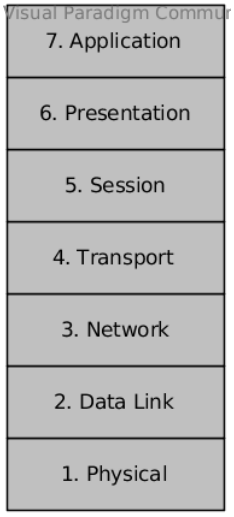


Figure 2.2: OSI Layers

Physical Layer

The physical layer is the lowest layer in the model, and assures that the information reaches its destination properly. This means encoding it in its binary form, and making sure that the destination knows how to decode it afterwards. It is also the layer responsible for knowing its connection to the medium, and how each specific connector or pin works, while also determining the appropriate transmission technique (if analog or digital), and how many volts/db should be used to ensure a safe transmission [19, 20].

Data Link

This layer ensures error-free data frames exchange between nodes over the physical layer. It the connection between nodes, while performing all the actions to ensure that frames are being successfully transmitted: Traffic control ensures that data is only transmitted when frame buffers are available; Sequencing ensures that the frames are transmitted and received sequentially; Frame acknowledgment ensures that the data frames are correctly received by providing and expecting acknowledgments. If error occur, non-acknowledged frames are re-transmitted; Delimiting provides the creation and recognition of the frames boundaries; Error checking provides integrity checks on every frame received.
The Data Link also manages which node has access to the physical medium at any time [19, 20].

Network Layer

First, the Network Layer provides a communication sub-network (subnet) that nodes can connect to, where each node has an unique address. This layer is responsible for handling the exchange of frames (datagrams) between nodes, by choosing the best route for it, and deciphering the physical location of each logical address or name. This layer may also fragments the datagrams, which will be reassembled at the destination, and it is also responsible for handling the traffic in the router, when its buffers fill up. This layer does not guarantee reliable delivery of datagrams [19, 20].

Transport Layer

The Transport Layer assures quality of service to the layers above, guaranteeing error-free deliveries with no loss or duplication. This allows the upper layers to focus on other aspects of the communication, while knowing that the message will be properly received at the destination, behaving as an end-to-end layer (does not know how it connects to the destination, only that the messages reach it). This layer is responsible for providing message segmentation, where messages sent from above layers are split into smaller units and sent to the network layer while it handles the reliability of deliveries with acknowledgments control [19, 20].

Session Layer

This layer provides session (virtual connection between points of contact) control. This means managing the creation and termination of connections between two applications processes on different machines. It also provides the associated session support such as security, name recognition, logging, etc.[19, 20].

Presentation Layer

This layer is responsible for how the data is shown on the Application Layer. It translates the data from the format used by the Application Layer to an easier format for the transport, and then re-translate it into an Application Layer format at the receiving end. This layer is also responsible for data compression and encryption, reducing the size of the message to send, and providing security for sensitive information such as passwords [19, 20].

Application Layer

This layer is the closest to the end user, meaning that users or application processes can access it, with the quality of service guarantees provided from all the layers below. The applications running on top of this layer fall out of the OSI scope, but most protocols fall in this category, such as the those studied in Section 2.5 [19, 20].

2.3 M2M Communications

> "*The exponential growth of wireless communication devices and the ubiquity of wireless communication networks have recently led to the emergence of wireless machine-to-machine (M2M) communications as the most promising solution for revolutionizing the future "intelligent" pervasive applications*", [7]

As technology advances, machine-to-machine (M2M), which can also be called machine-type communications (MTC), will continue to increasingly replace the traditional human-to-machine (H2M) operations [17]. By definition, *M2M communications* is a term used to refer to data communications without or with limited human intervention amongst various terminal devices such as computers, embedded processors, smart sensors/actuators and mobile devices, etc. [21]. Since M2M, as opposed to H2M, does not need user interaction to provide their service, the devices communicating with each other have some degree of decision making, and thus the services provided can be at some extent considered intelligent, as the quote above, from [7], suggests.

The journey towards having a wide range of devices using M2M to communicate each other converges into the Internet of Things (IoT) paradigm, which in essence states that if devices (tags, sensors, actuators, mobile phones, etc.) can be uniquely addressed, they will be able to interact with each other and cooperate to reach common goals [8].

This raising interest in M2M communications arises due to the impact possibilities for both domestic (e.g. domotics, assisted living, e-health, etc.) or working (e.g. automation, industrial manufacturing, logistics, business/process management, etc.) fields [8]. In mobile scenarios M2M communications can allow for resource optimization, optimizing the usage of energy, network resources, computational cost, etc., allowing for more efficient and cheaper solutions for the consumers [3, 7]. Some interesting statistics are shown in [17], stating that the number of M2M-enabled devices (terminals) is increasing exponentially, forecasted to grow from 50 million in 2008 to well over 200 million in 2014, and up to 50 billion by 2020.

As the growing interest in M2M communications increased, so did the need to standardize it, so that the roughly 50 billion devices to exist in 2020 can have the ability to communicate with

each other. The standardization and currently accepted M2M architecture are going to be detailed in the next Section.

2.3.1 ETSI M2M Standard

In 2005 3GPP started with the standardization of M2M in the Global System for Mobile (GSM) and the Universal Mobile Telecommunications Systems (UMTS). In 2007 the technical report (TR) 22.868 (release 8) [22] completed the study on facilitating Machine-to-Machine communications in 3GPP systems, after which the 3GPP working group for M2M standardization was organized.

In January 2009 the European Telecommunications Standards Institute (ETSI), which is the independent and non-profit standardization organization in the telecommunications industry, picked up where 3GPP left of and continued the standardizing process of M2M, defining entities and functions to provide efficient end-to-end information delivery. ETSI published the Technical Specification (TS) 102 689 [23] with the M2M service requirements, and TS 102 690 [1] with the functional architecture for M2M.

The system architecture is based on current Network and Applications domain standards, and it is extended with M2M Applications, and with the Service Capability Layer (SCL) as a key functionality in the M2M service platform. This layer provides the different elements in the M2M architecture with a consistent resource tree, allowing information to be shared between them. The elements of that resource tree are called Service Capabilities (SCs), which is where the information is kept.

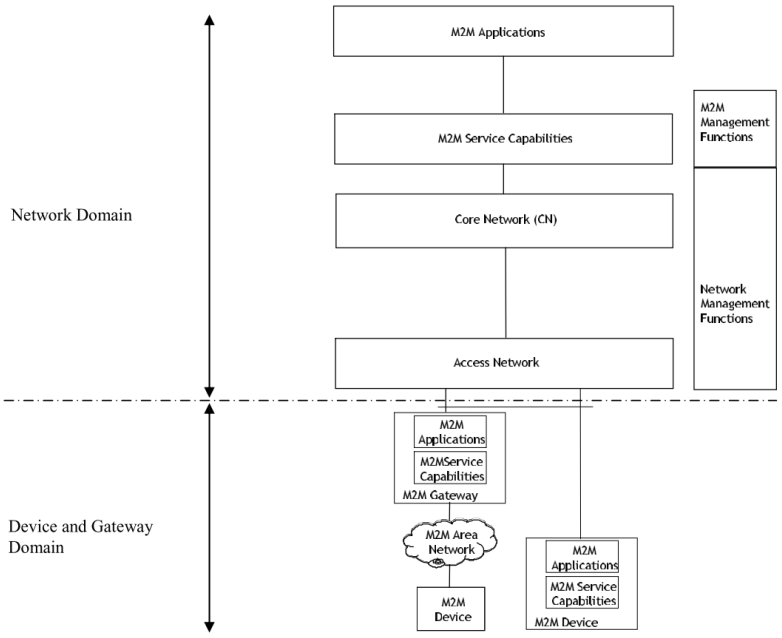


Figure 2.3: Machine-to-Machine high level architecture. From [1].

Figure 2.3 presents the high level view of the ETSI M2M system architecture within the scope of [23, 1]. Two domains are depicted: The M2M Device domain and the Network and Application domain.

The elements present in the Device and Gateway domain are:

- **M2M Device**: Device that runs M2M Applications using M2M Service Capabilities (SCs). Devices can connect to the Network Domain in two ways:

  Direct Connectivity: Directly connects to the Network Domain, performing all the procedures such as registration, authentication, authorization, management and provisioning with the Network Domain.

  Gateway as a Network Proxy: Using the M2M Area Network, the device connects to a M2M Gateway that acts as a proxy, handling all the procedures mentioned above. Devices can connect to the Network Domains via multiple M2M Gateways.

- **M2M Area Network**: Provides connectivity between the M2M Devices and M2M Gateways.

- **M2M Gateway**: The Gateway runs M2M Application(s) using M2M SCs. It acts as a proxy between the M2M Devices and the Network Domain, and it may provide a legacy service to other devices that are hidden from the Network Domain.

The elements present in the Network Domain are:

- **Access Network**: Allows the M2M Device and Gateway Domain to communicate with the Core Network.

- **M2M Core**: The M2M Core is composed by the Core Network (CN) and the M2M Service Capabilities (SCs):

  Core Network: provides IP connectivity, interconnections, and roaming capabilities within the M2M core.

  M2M Service Capabilities: The SCs provide M2M functions that are to be shared by different Applications, exposing those functions through a set of open interfaces. It simplifies and optimizes application development and deployment through hiding network specificities.

- **M2M Applications**: Applications that run the service logic and use M2M Service Capabilities accessible via an open interface.

- **Network Management Functions**: Consists of all the functions required to manage the Access and Core networks. These include Provisioning, Supervision, Fault Management, etc.

- **M2M Management Functions**: Consists of all the functions required to manage M2M Service Capabilities in the Network Domain. The management of the M2M Devices and Gateways uses a specific M2M Service Capability, such as the M2M Service Bootstrap Function (MSBF) or the M2M Authentication Server (MAS).

See Figure 2.4. The interface between a M2M application in the M2M Device Domain and the M2M Service Capability (SC) in the Network and Application Domain is termed mIa; the interface mId is between a M2M device or M2M gateway and the M2M SC in the Network and Application Domain; and the dIa interface is between a M2M device or M2M gateway and the M2M at the same device.

Representational State Transfer, or just REST [24], is the dominant approach in client-server communications, that allows for stateless communication (meaning it does not need to be synchronized nor have an open session), cacheable resources and resource operations. The latter allows the use of CRUD verbs (Create, Retrieve, Update, Delete), which can largely reduce implementation efforts. Since REST is widely adopted and can be easily applied to M2M communications, RESTful protocols (which allow for stateless communications) are encouraged [25, 1].

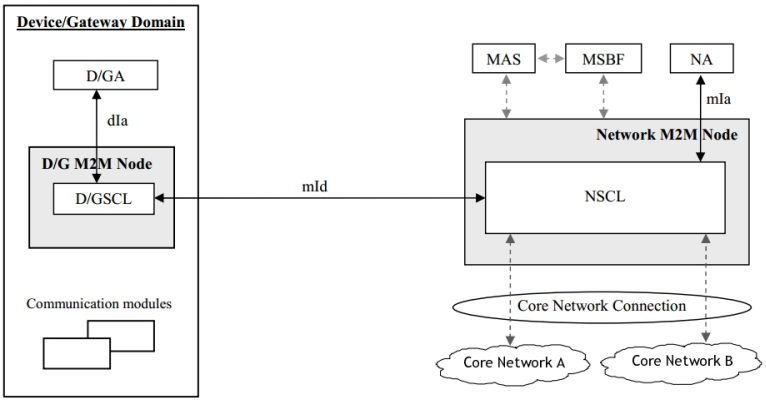


Figure 2.4: M2M Service Capabilities functional architecture framework. From [1].

As mentioned above in the M2M Management Functions of the Network Domain, one of its responsibilities is the handling of the Bootstrap procedures. These procedures provide a way to securely connect the different entities using certificates and private keys. Every entity has a pre-provided certificate-key pair that allows them to receive a session key, which is then used to encrypt the communications with TLS (HTTP) or DTLS (CoAP). The protocol specificities will be discussed in Sections 2.5.1 and 2.5.2 for HTTP and CoAP respectively.

The next Sections will focus on understanding the resource structure provided in the standard.

2.3.1.1 Resource Structure

The ETSI resource structure is shown in Figure 2.5 and illustrates the different kinds of resources and how they are connected to each other. The structure diversity is based on alternating arrays and single resources. This Figure illustrates how this standard is very open to interpretation, since it can have a varying tree size for every resource.

Analyzing the Figure it in greater detail, and knowing that the resources surrounded by '<' and '>' are arrays, it becomes easy to recognize how connected every resource is, since for instance, inside the <*scls*> there is a *scl* resource (or more since *scls* is an array), and that inside the *scl*

resource there is quite a number of arrays, that have the exact same structure as the ones below <*sclBase*>, on the left.

Different M2M elements have varying resource trees, so the next Sections will show what changes between a Gateway and a Network Application.

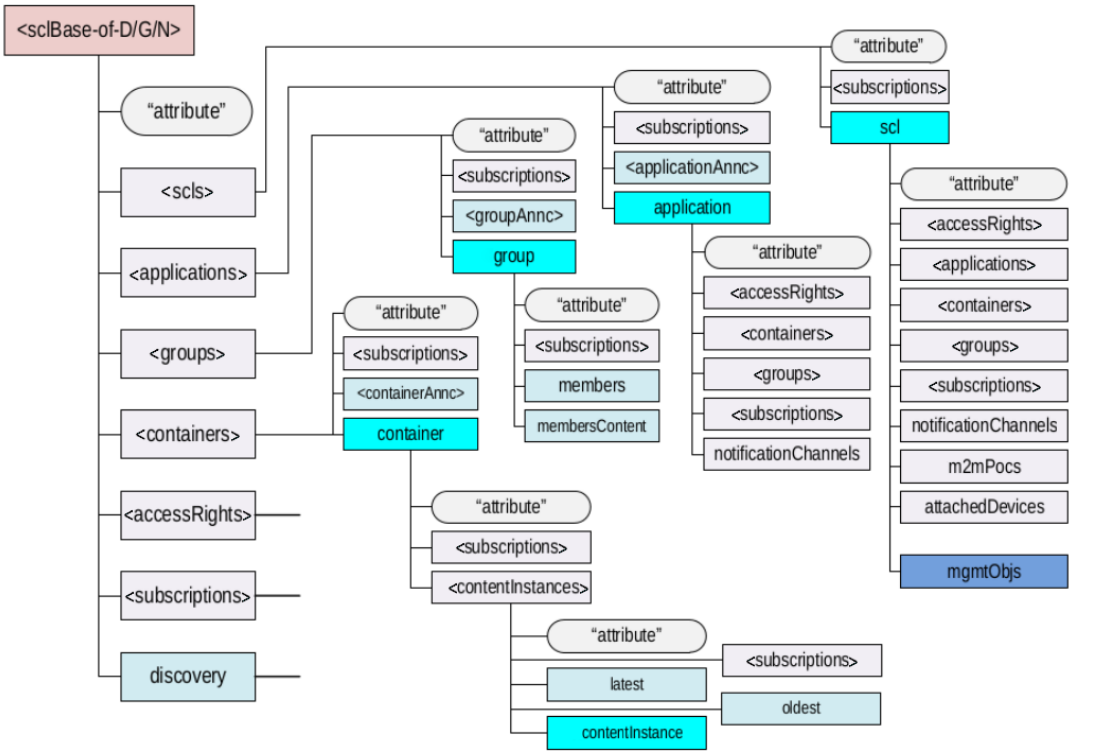


Figure 2.5: Base SCL on ETSI's standard. Adapted from [2]

2.3.1.2 Gateway

The Gateway entity communicates through the dIa interface with network capable sensors, and communicates with the NSCL using the mId interface. The latter also effectively connects the Gateway and sensors M2M domain to the application's domain, as shown earlier in Figure 2.4 [1, 26].

As seen in the previous Section, each entity may have a resource tree quite large, and for the purpose of this study there was no need to implement the totality of resources. So, this simplification led to Figure 2.6, which represents the important resources for this dissertation's study.

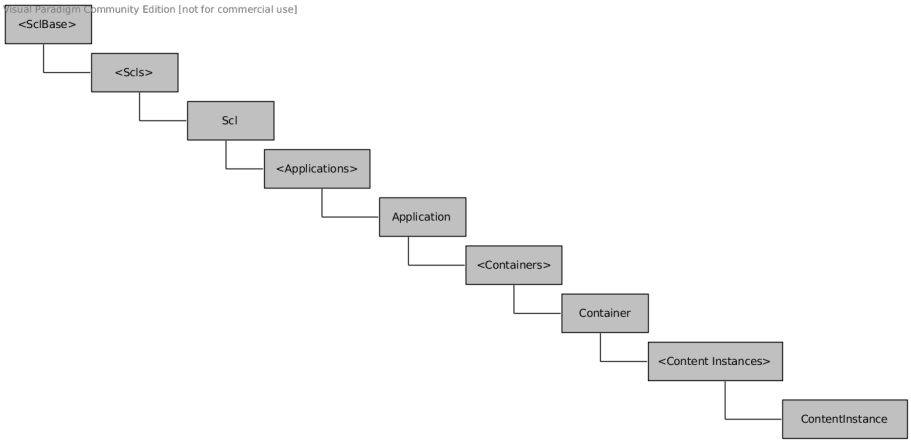


Figure 2.6: Simplified Gateway ETSI resource tree.

How these resources are mapped into actual devices and concepts will be discussed in Section 3.5.1.

For the Gateway to fully interact with the NSCL using this structure, it has to perform a series of actions:

- Register the *scl* resource (with a specific *'sclId'*) under the */sclBase/scls* target on the NSCL host.
- Receive the OK response for that registration.
- Receive a subscription from the NSCL on the *scl* resource just created.
- Register the *application* resource (one or more) under */sclBase/scls/'sclId'/applications*.
- Receive the OK response for each *application* resource registered.
- Receive a subscription from the NSCL on the (one or more) *application* resource registered.
- Register a certain *container* resource belonging to the *application 'appId'* under */sclBase/scls/'sclId'/applications/'appId'/containers*.
- Receive the OK response for that registration.
- Receive a subscription from the NSCL on the *container* resource just registered.
- Register a *contentInstance* resource belonging to the *container 'containerId'* with under */sclBase/scls/'sclId'/applications/'appId'/containers/'containerId'/contentInstances*.
- Receive the OK response for that registration.

Being based in the publish-subscribe paradigm, subscriptions are needed to advance to the next step. These actions illustrate the path taken to create the registration for one specific resource. The last resource (*contentInstance*) is carries the information regarding the registered resource (for example heart-rate measurements) encoded in Base64 [27].

2.3.1.3 Network Application

On the other hand, the Network Application entity communicates through the mIa interface with the NSCL (as shown in Figure 2.4), that in turn may provide a different communication interface apart from ETSI, to reach a wider range of Applications that do not need to know the ETSI communication specificities [1, 26].

Just as in the previous Section there were some simplifications to the resource tree, that resulted in Figure 2.7's representation, which will be the base structure used in this dissertation's study.

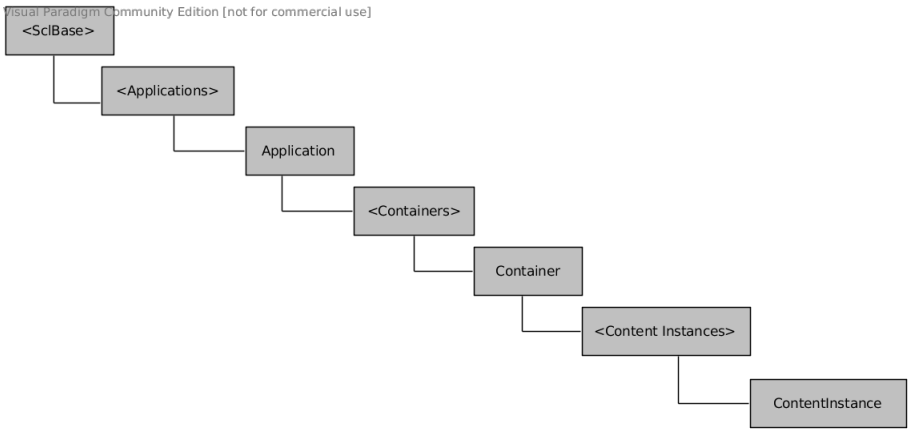


Figure 2.7: Simplified Network Application ETSI resource tree.

How these resources are mapped into tangible concepts will be discussed in Section 3.5.1.

For the Network Application to fully interact with the NSCL using this structure, it has to perform a series of actions:

- Register the *application* resource (with a specific *'appId'*) under the */sclBase/applications* target on the NSCL host.
- Receive the OK response for that registration.
- Receive a subscription from the NSCL on the *application* resource just created.
- Register a certain *container* resource under */sclBase/applications/'appId'/containers*.
- Receive the OK response for that registration.
- Receive a subscription from the NSCL on the *container* resource just registered.
- Register a *contentInstance* resource belonging to the *container 'containerId'* with under */sclBase/applications/'appId'/containers/'containerId'/contentInstances*.
- Receive the OK response for that registration.

Being based in the publish-subscribe paradigm, subscriptions are needed to advance to the next step. These actions illustrate the path taken to create the registration for one specific resource. The last resource (*contentInstance*) is carries the information regarding the registered resource (for example heart-rate measurements) encoded in Base64 [27].

2.3.1.4 Mobile Considerations

Given the importance of being transparent to the user about the use of their hardware, the most important resources of a mobile M2M system are data transmission and energy efficiency, reliability and security [3, 7], so that the impact of the running the service does not affect the normal use of the device. It is also critical that only someone with permissions over the smartphone can trigger its functionalities, so that personal data is not misused.

Using the smartphone to implement a M2M Gateway stresses the issues just mentioned. Users tend to be self-aware of the battery power of their phones, as they need it to be able to communicate, so a M2M Gateway needs to be considered constrained, keeping the footprint as low as possible, and work unnoticed by the user.

2.4 Marshalling

Marshalling, or Serializing, is the nomenclature given to the process of encapsulating data for storage or transport, so that it can be easily processed by different systems. Since marshalling techniques exist in most programming languages available, it allows for interoperability of services by using standards readable by both machines and humans [28].

The most common types of marshalling languages are XML and JSON, and both will be explained in the next Sections.

2.4.1 XML

The eXtensible Markup Language, commonly known as XML, was first released in a World Wide Web (W3C) draft in 1996 [29] as a derivation from the Standard Generalized Markup Language (SGML), which had been standardized in the International Organization for Standardization (ISO) in ISO 8879 [30]. XML 1.0 was originally submitted for standardization on an RFC (Request for Comments) in 1998 [31] and has since become one of the most widely used markup languages. It is, for instance being used in Android OS for detailing the layout specifications and the permissions needed to run the application.

An XML example is below:

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
```

```
  </phoneNumbers>
  <gender>
    <type>male</type>
  </gender>
</person>
```

After parsing, the variables would be:

```
<person firstName="John" lastName="Smith" age="25">
  <address streetAddress="21 2nd Street" city="New York"/>
  <phoneNumbers>
     <phoneNumber type="home" number="212 555-1234"/>
     <phoneNumber type="fax"  number="646 555-4567"/>
  </phoneNumbers>
  <gender type="male">
</person>
```

2.4.2 JSON

JavaScript Object Notation (JSON), was originally specified in [32] as a less verbose alternative to XML, featuring pairs of attribute-value, and it is currently in use in the many server-application communication systems. It is used primarily to transmit data between a server and web application, as an alternative to XML. Its official and most widely used MIME type is "application/json".

A JSON schema example from [33] is below. As intended it is very readable, while maintaining the ability to be parsed automatically.

```
{
    "title": "Example Schema",
    "type": "object",
    "properties": {
        "firstName": {
            "type": "string"
        },
        "lastName": {
            "type": "string"
        },
        "age": {
            "description": "Age in years",
            "type": "integer",
            "minimum": 0
        }
    },
    "required": ["firstName", "lastName"]
}
```

2.5 M2M Protocols

Given the ascension of M2M demand, protocols started to shift their focus onto M2M scenarios. In this Section, protocols that can be applied to these communications will be studied.

Section 2.5.1 will focus on explaining the most commonly used web protocol, HTTP. Then, in Section 2.5.2, CoAP will be studied, followed by MQTT and AMQP in Sections 2.5.3 and 2.5.4 respectively. Although all these protocols can be used in M2M scenarios, only HTTP and CoAP are supported by the ETSI standard, because of their stateless and REST approach.

2.5.1 HTTP

Hypertext Transfer Protocol, or HTTP [10, 34], is the most well known protocol currently available. Nearly every website is transmitted over HTTP or its secure version HTTPS, which adds TLS (Transport Layer Security [35]) to encapsulate the HTTP packets.

HTTP started to be used in the early 1990's World-Wide Web global information initiative only being proposed as standard in 1997 [36], receiving that status in 1999 with version 1.1 [10]. It can use TCP [37] or UDP [38] as Transport Layer protocols, but TCP is much more common because of the reliability it provides.

It relies heavily on Uniform Resource Identifiers (URI) [39] to uniquely identify names of web resources, to be able to execute methods on them. HTTP provides a variety of request methods (also known as verbs) to allow for versatile communication between clients and servers.

GET	This verb requests the server for a representation of the specified resource (i.e. a Download)
POST	This verb sends some information from the client to the server (i.e. submitted online form, login, etc.)
PUT	This verb requests the server to store the sent entity at the destination URI.
DELETE	This verb requests the deletion of the specified resource.
HEAD	This verb executes a request similar to a GET, but without receiving the body of the response. This is useful to retrieve meta-information.
TRACE	This verb is returned unchanged by the server, so that a client can see if any changes occurred during transit.
OPTIONS	This verb returns the verbs supported by the server for that specific URI.
CONNECT	This verb is used to request a connection to a TCP/IP tunnel, which is useful for creating SSL/TLS encrypted communications when the client is behind an unencrypted proxy.
PATCH	This verb is used to partially modify a resource.

The normal HTTP packet, is illustrated in Figure 2.8.

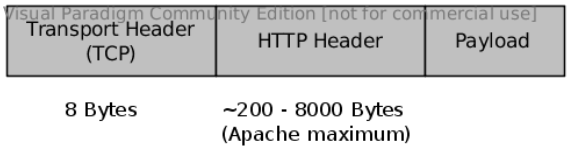


Figure 2.8: HTTP packet size

2.5.2 CoAP

The Constrained Application Protocol (CoAP) [11] is a lightweight protocol designed for constrained devices (with low memory, processing power, etc) and constrained networks (e.g low power, lossy), and specially fulfilling M2M requirements. It is currently a final IETF draft (version 18), and awaiting the RFC standardization in the near future.

Its simple interface and applicability was demonstrated in the 2013 ETSI plug-tests where it was shown that in an event with eight companies with several different CoAP implementations of clients and servers, the interoperability rate was 94.1% [40].

CoAP derives from Representational State Transfer (REST), explained in Section 2.3.1, focused for the use in constrained networks and nodes in M2M applications [3]. To identify resources Uniform Resource Identifier (URI) [41] are used, just like in HTTP. CoAP is also easily translated to HTTP for integration with the Web while accomplishing specialized requirements, such as multicast support, built-in resource discovery, block-wise transfer, observation, and simplicity for constrained environments [42]. This allows for compatibility with existing systems.

Being RESTful in is nature, CoAP allows four HTTP verbs to be used: GET, POST, PUT and DELETE. But unlike HTTP, where transactions are always client-initiated, and GET operations must be repeatedly performed (polling), CoAP has an asynchronous approach to support pushing information from servers to clients: observation (see Figure 2.9). In constrained environments, the polling wastes to many resources, and CoAP addresses this with a special GET request, where a client can indicate its interest in further updates from a resource by specifying the *OBSERVE* option. If the server accepts this option, the client becomes an observer of this resource and receives an asynchronous notification message each time it changes. Every notification is identical in structure to the response to the initial GET request [42]. This model is an attempt to achieve stateless publish-subscribe communication, ideal for devices with fewer resources.

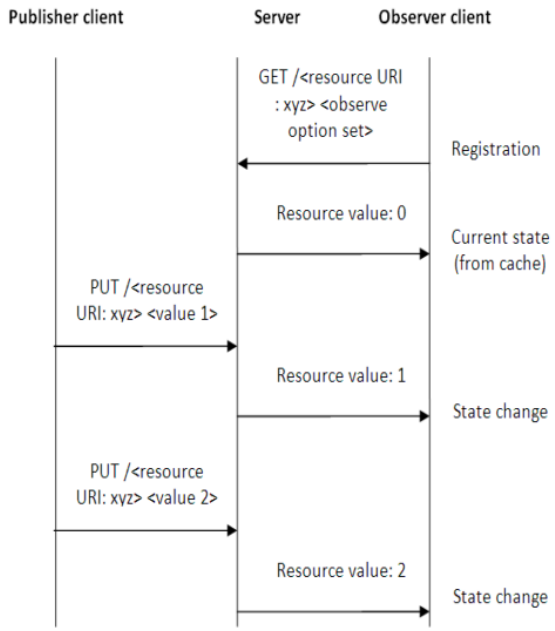


Figure 2.9: CoAP resource-observe. From [3].

The clients do not need to maintain state, i.e., the client can be stateless. Furthermore, in order to implement CoAP in constrained small devices (memory available, computational and power consumption restrictions), the transport protocol is User Datagram Protocol (UDP) [43] and the protocol overhead in the header can be up to 4 bytes. Reliability can be implemented by an optional stop-and-wait protocol, and security by the use of Internet Protocol Security (IPsec) [44] or Datagram Transport Layer Security (DTLS) [45]. CoAP also supports TCP [46] connections, which add reliability of delivery, but increases the overhead of each message, which is not ideal for constrained situations.

The CoAP packet is illustrated in Figure 2.10.

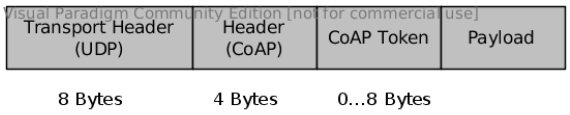


Figure 2.10: CoAP packet size

2.5.3 MQTT

MQ Telemetry Transport (MQTT) [9] is a lightweight broker-based publish/subscribe messaging protocol designed to be open, simple, lightweight and easy to implement.

These characteristics make it ideal for use in constrained environments just like a smartphone. It is a protocol developed by IBM to address the issue of reliable M2M communications [9].

It is Based on publish/subscribe paradigm associated with topics. It is connection oriented, used over TCP and features 3 quality of service (QoS) levels for assuring delivery (no retransmission, re-transmit once and re-transmit until received). Can be used for PUSH applications, to save the constrained device to have to answer to requests, saving messages, and thus processing power and battery.

The underlying TCP connection causes MQTT to have a bigger overhead than other protocols such as CoAP, which in evidenced in the protocol comparison tests done in [3].

The CoAP packet is illustrated in Figure 2.11.

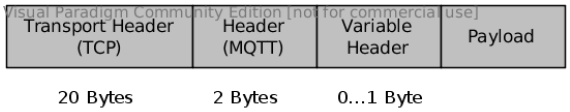


Figure 2.11: MQTT packet size

2.5.4 AMQP

The Advances Message Queue Protocol (AMQP) [47] is an open standard application layer asynchronous protocol for message oriented middleware. It uses the publish-subscribe model defined earlier in Section 2.1

AMQP uses brokers (servers) to receive messages from publishers, and route them to consumers. Figure 2.12 gives a high level perspective of the protocol. [4] has a simple explanation of the functionality of the protocol: "Messages are published to exchanges, which are often compared to post offices or mailboxes. Exchanges then distribute message copies to queues using rules called bindings. Then AMQP brokers either deliver messages to consumers subscribed to queues, or consumers fetch/pull messages from queues on demand".

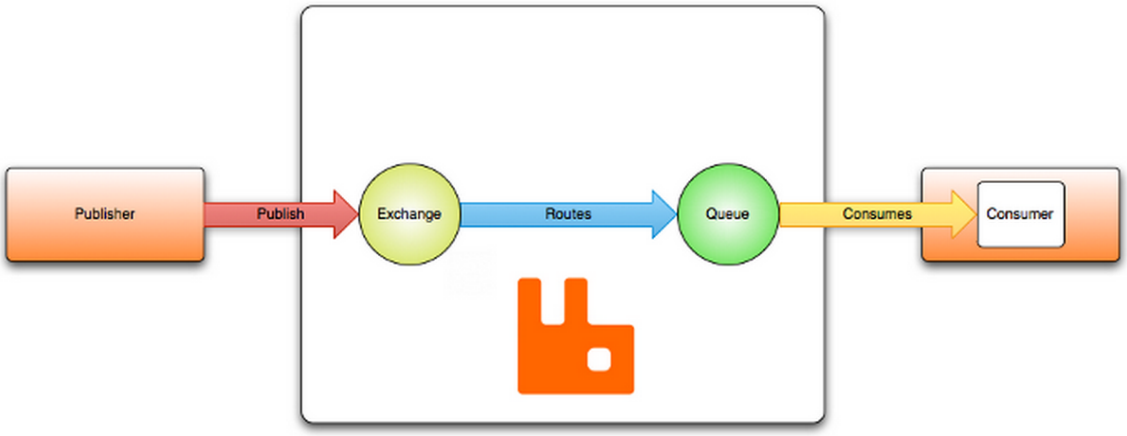


Figure 2.12: AMQP functionality overview. From [4]

In terms of security, AMQP implements SASL [48] and TLS [49]. Figure 2.11 illustrates the AMQP packet size.

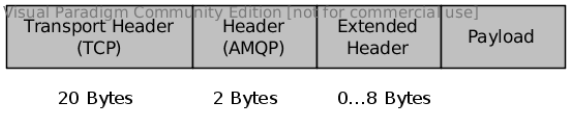


Figure 2.13: AMQP packet size

2.6 Security Mechanisms

2.6.1 X.509 Certificates

[50]

2.6.2 RSA Criptography

[51].

2.7 E-Health Technologies

In this Section, the current health-care standards are going to be presented, as the scenario of this dissertation's project is focused on this subject.

2.7.1 ISO/IEEE 11073

The ISO/IEEE 11073 is the family of standards for medical devices that was originally called IEEE 1073, or just X73 [52]. It arose in 1982 with the need for easy to use (plug-and-play) medical grade equipment for operating rooms or bedside monitoring in intensive care units (ICU), aiming for real-time and efficient exchange of data [53]. In the past decade, the IEEE 11073 has focused on developing Personal Health Devices (PHD) standards, standardizing functions for each of the OSI layers mentioned in Section 2.2.

Figure 2.14 shows the IEEE 11073 Framework as of 2012, where several device specializations can be depicted. The main goal of all these standardizations is to facilitate the development of equipment to monitor people's well-being inside or outside of hospitals and provide interoperability on medical grade equipments. These standards are not available for public scrutiny, causing details on them to be scarce.

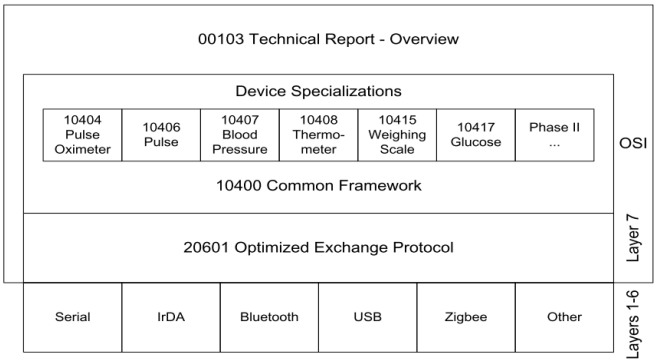


Figure 2.14: IEEE 11073 Framework. From [5].

On Android OS version 4.0 (API 14), a Bluetooth Health API was introduced, facilitating the integration with devices following the IEEE 11073 specifications [54].

2.7.2 Continua Alliance

The Continua Health Alliance [55] is a profile of standards built on top of the IEEE 11073 family. Its goal is to provide the application layers with semantic interoperability, to further allow communication between devices and services. Figure 2.15 shows how the protocol builds around IEEE 11073 to allow for greater interoperability with a wider range of devices [5].

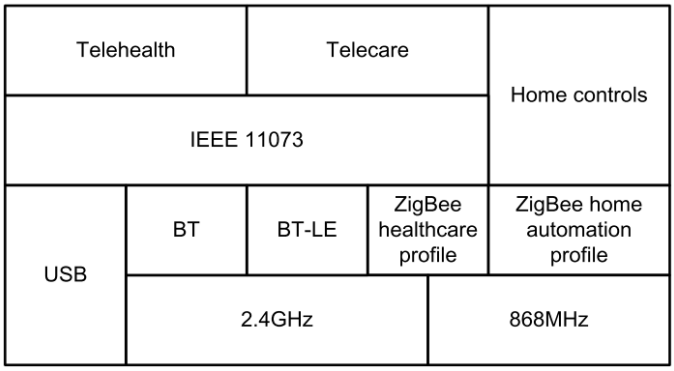


Figure 2.15: Continua Alliance profile of standards. From [5].

Currently the Continua Alliance has a set of guidelines for manufacturers to follow, to provide them with the proper certification, ensuring the compatibility of all Continua certified devices. The details of these guidelines and of the resulting communication protocol are proprietary, and thus not available for deeper discussion in this state-of-the-art.

2.8 Conclusion

The ETSI M2M standard appears to have an in-depth approach on the concept, and aims to thoroughly standardize every aspect of the communication between devices and applications. It

does so without ever demanding that certain resource should represent any specific application or device, leaving it to the interpretation of anyone who wishes to implement it.

The project will be built upon the ETSI standard, and to properly communicate and build a system, a Gateway and a Network Application will be built, thus implementing the mId and mIa interfaces. This will have to be done either by HTTP or CoAP, since ETSI does not support MQTT nor AMQP.

The connection to the sensors may require some implementation of either the IEEE11073 or the Continua Alliance protocol, as most medical devices uses them. The case may be that the sensor chosen uses a proprietary protocol.

Chapter 3

Mobile M2M System

This Chapter will start by explaining the Use Cases that were important in the concept creation for this dissertation's project, in Section 3.1. Then, the Approach will explain the important functional and non-functional requirements in Section 3.2.

Sections 3.3 and 3.4, will detail the architecture and functionality specificities of both the Mobile M2M Gateway and the Network Application respectively. This will be followed by the Technology specifications for this project, on Section 3.7.

Finally, Section 3.5 will explain the interpretations of the ETSI standard that were necessary for the development of this dissertation's project, followed by the Evaluation metrics in Section 3.8.

3.1 Proposed Use Cases

Imagine a scenario where a user enters the proximity of a supported health-care sensors. The user will be able to use the phone's bluetooth capabilities to search and list nearby supported sensors that are previously paired. After listing, he will be able to choose a sensor to connect to, effectively starting that sensor's measurements.

The readings taken by the sensors will be shown in the application and the user will be able to choose the reading(s) he desires to store online. This storage will be supported by a could based service that will handle the medical information for future consultation by physicians or by the user itself. From the moment the measurements start the user is capable of stopping the sensor or saving a specific measurement at any time.

On the chance the user has the Gateway and the Network Application on the same smartphone, a secure local connection between the two will save the bandwidth referring to the transit of the sensors measurements through the NSCL and back to the Network Application, also saving battery in the process. This aims to save the limited smartphone's resources such as battery and mobile data bandwidth, while being fully transparent to the user.

When the user saves the measurements, the data is stored in a supported cloud health-care application that the user has previously logged in to, building a medical record that a physician or himself can consult later, with his relevant medical data.

The UML 2.0 Use Case diagram for the scenario just described is present in Figure 3.1, explicitly defining the user's possible actions.

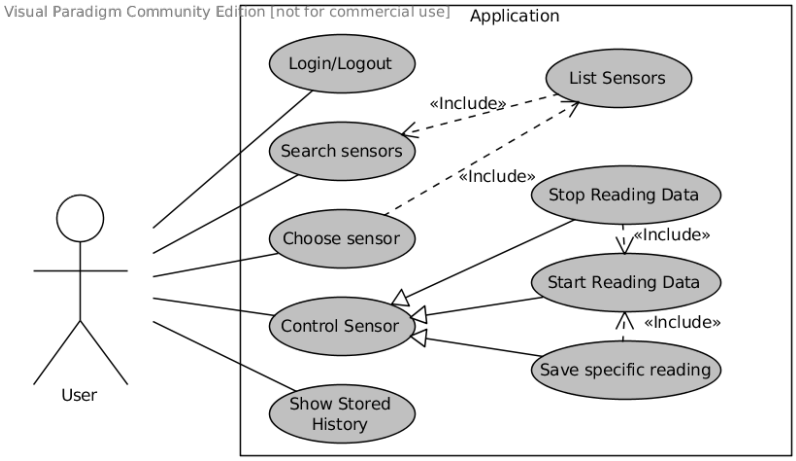


Figure 3.1: UML 2.0 Use Case Diagram for the study scenario

3.2 Approach

The high-level system architecture is shown in Figure 3.2. Here, the M2M Gateway will take advantage of the smartphone's connectivity abilities, connecting to sensors via bluetooth while communicating with the NSCL via Wi-Fi, just as the Network Application. This system aims to allow to user easily control the way he connects to supported health-care devices and sees information relevant to him.

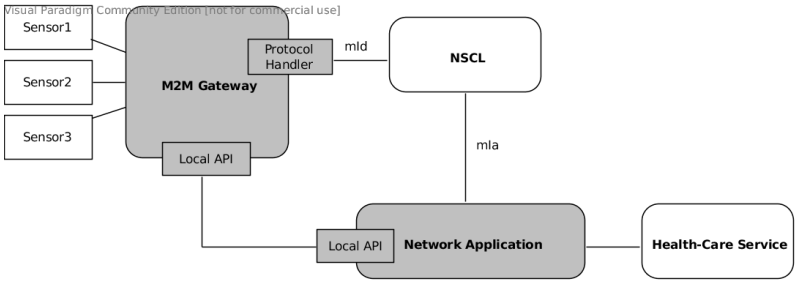


Figure 3.2: Architectural overview of integrated solution

The Gateway and Network Application work as a system to fulfill the Use Cases explained in the previous Section.

To tackle the M2M mobility issue explained in Section 1.2, this dissertation's approach relies on a versatile mobile Gateway, that will have to accommodate the following functional and non-functional requirements.

Functional requirements:

- Support disconnection and save any unsent data for next connection.
- Support a Web Server that allows the reception of requests and subscriptions from the Server.
- Support health-care external sensors:

  Heart-Rate monitor.

  Sphygmomanometer.

  Scale.
- Support API for local access.

Non-functional requirements:

- Support a modular design.
- Multi protocol support:

  HTTP.

  CoAP.

  MQTT.

  AMQP (Low Priority).
- ETSI M2M standard compliant.

3.3 Mobile Gateway

This Section will describe the architecture and functionality of the mobile M2M Gateway. The original Gateway architecture that was present in Version 0 (mentioned in Section 1.1) has suffered a massive overhaul to accommodate all the requirements and to better fit the ETSI standard.

The overall design of the gateway is based on the concept of services and threads to run entirely in the background. Together, they manage the connectivity to the NSCL and to the Sensors, as shown in Figure 3.3. The design's main guideline was to build a Gateway as modular as possible, easing the addition of features, external sensors, protocol implementations so that as the gateway becomes more complex, there is no need to change the whole code to accommodate small changes.

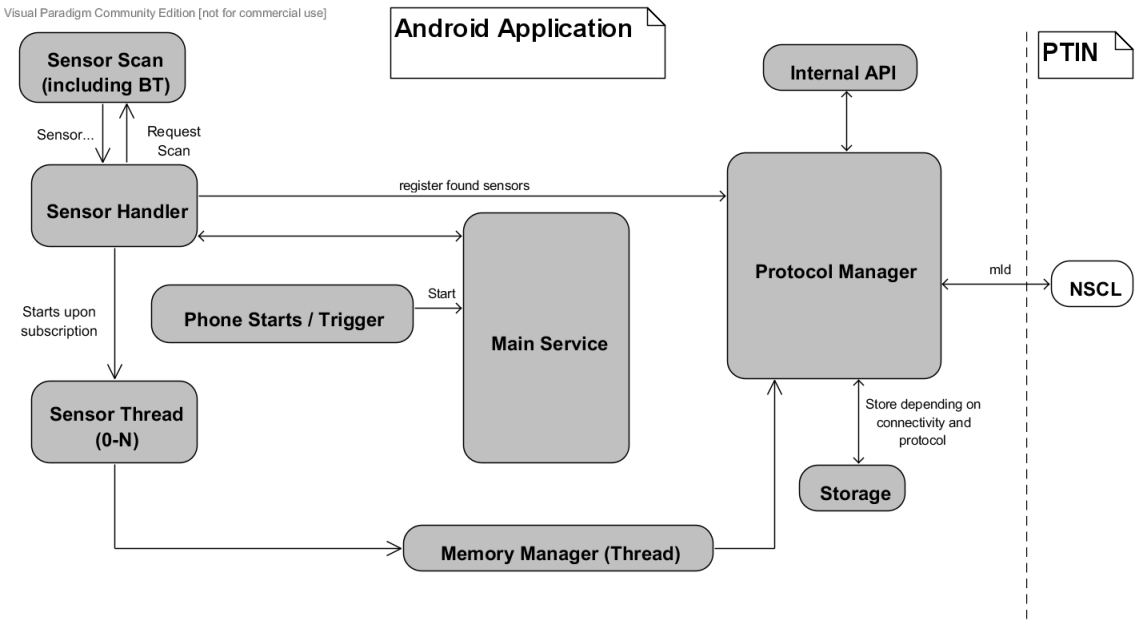


Figure 3.3: Mobile M2M Gateway's Architecture

The gateway application is built with one service and it starts when the phone boots or when a specific Intent (Android inter-process communication mechanism) is received. Upon initialization, it automatically starts the Protocol and Memory Managers as well as the GPS module in separate threads. Having their own threads and handlers is effective in reducing race conditions and delay of processing, as the messages are stored and handled in a first-in first-out (FIFO) pile.

The Protocol Manager is responsible for handling status of the network connectivity, as well as managing the messages to be sent. Upon creation it also starts a local storage file, so that data is not lost when connection is unavailable. This module is also responsible for the marshalling of the sensor specific data, so that it is ready to be sent.

As shown in Figure 3.4, and as the name suggests, the Protocol Manager also handles different Protocols. If the protocol is supported by the ETSI standard, the communication is handled by the GSCL module, and a *Content Instance* resource will be created, as described in Section 2.3.1.2. If however, the protocol is not supported by ETSI, a different route is taken, and it connects directly to the Protocol Manager (in this case MQTT), to execute the connection to a plugin in the NSCL (this is not an mId communication as the image may suggest). As the non ETSI protocols may be synchronous, the Protocol Manager also has a mechanism to reconnect after the internet connection returns after a drop.

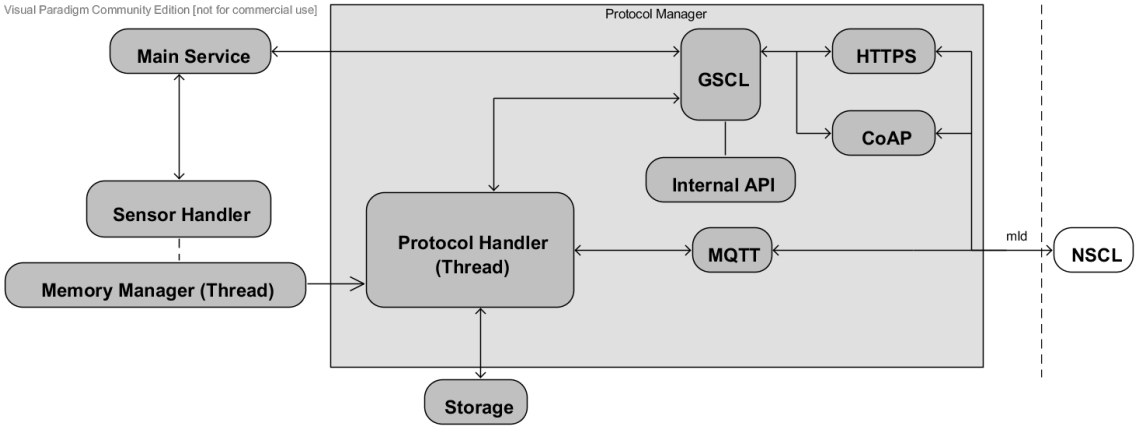


Figure 3.4: Protocol Manager Architecture

At start-up there is still the initialization of the Memory Manager, which has its own thread. The GPS module starts along side the Memory Manager to start collecting location information in predefined intervals. This module's job is to store the sensor's information in buffers, until it is passed on to the Internet Manager to be sent to the NSCL.

After all the necessary configurations are completed, the Gateway registers itself and the available sensors at that moment in the NSCL. It is important to state that the Gateway does not connect to the sensors and no data is collected before there is a specific command to do so. This assures that no resources are wasted in this step.

Every active sensor has remotely configurable parameters:

Reading Granularity : Maximum interval between two sensor readings.

Minimum Transmission Granularity : Maximum interval between two sensor transmissions.

Maximum Buffer Size : Maximum amount of phone space that is occupied storing data from this sensor for later processing.

The data on the buffers stays stored until the transmission granularity is reached or the buffer gets full. Once this data is "flushed", it is sent to the Protocol Manager for marshalling and dispatch. In case there is no connectivity, the marshalled information, ready to send, is stored in the storage file initialized at start up. When connection returns, this storage file is chached into memory, and the messages are dispatched.

The sensor marshalling process starts with the packaging of the sensors data into a JSON object. The basic output, shown below with an example from a Zephyr HxM Bluetooth sensor. The values array is specific to each sensor, where the relevant data can be sent, whereas the remaining information about the sensor and GPS is common between different types of sensors:

```
{
  "values": [
    {
      "timeStamp": "1397491419169",
      "RRinterval": [
        "728"
      ],
      "heartRate": "87"
    }
  ],
  "sensorModel": "HxM™ BT Heart Rate Monitor",
  "sensorSerial": "12:34:56:78:90:AB",
  "GPS_Latitude": 40.6299045,
  "sensorType": "ZEPHYR",
  "GPS_Accuracy": 20,
  "GPS_Longitude": -8.6460066
}
```


This marshalled sensor data is used on the creation of a *Content* resource, which is an attribute of the *Content Instance* resource. The *Content* automatically encodes the payload in Base64 [27]. The *Content Instance* is then sent to the NSCL, as explained in Section 2.3.1.2, using one of the ETSI supported protocols.

The Gateway is also going to support a local API module, that will bypass the communication locally when the Gateway and the Network Application are in the same smartphone. This local API is further explained in Section 3.6.

3.4 Mobile Network Application

The Network Application's design is based on the Gateway, because despite being a different entity in the ETSI standard, the communication does not differ very much. So, the Network Application has the architecture depicted in Figure 3.5.

The Main Service was kept from the Gateway's design, in order to process the Graphical User Interface data without prejudicing the responsiveness of the Application. The Protocol Manager works in the same way as it does for the Gateway (see Figure 3.4), except it does not support non-ETSI protocols. It has the natural changes that the different entities require, and it implements the communication with the NSCL as explained in Section 2.3.1.3.

Just like the Gateway, and as the Approach Section suggested, with Figure 3.2, the Network Application also has a local API, that will allow to fulfill the energy and bandwidth requirements of the Use Cases (as explained in Section 3.1).

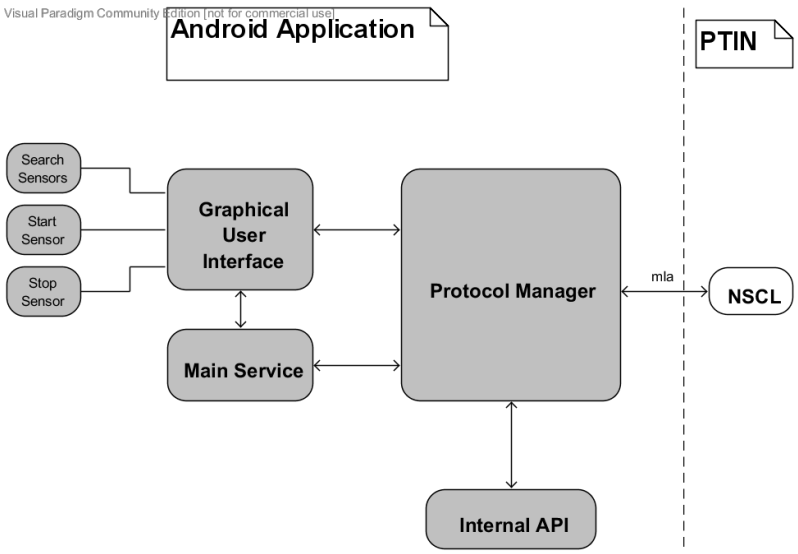


Figure 3.5: Network Application Architecture

3.5 ETSI Interpretation

The ETSI M2M standard is undoubtedly very complete when it comes to the message exchanges between all parties. This includes which attributes should or should not be sent in each message, and what are the procedures in case of failure or success. However, it does not map its resources onto tangible things, leaving it open to interpretation. This Section explains the interpretation made for the purpose of this dissertation's project development.

3.5.1 ETSI mapping

Given the tree-like structure of the data (as shown in Figure 2.5), it made sense that the lower on the tree, the closest the mapping would be of a physical device. The mapping for the M2M Gateway arose from this, associated with the nature of the *ContentInstance* resource. The latter has an attribute that stores data encoded in Base 64, which led to the *Container* resource being mapped as a sensor, such as a bluetooth heart rate monitor.

Given this, the *Application* resource had to somehow generalize the sensors, and as such it made sense that it group the different kinds of sensors the Gateway can connect to. Relating to the above example of a Container as a heart rate sensor, the application it would be associated with would be referring to health care monitoring.

Because of all the different kinds of sensors possible, all handled by the same mobile device, the *Scl* resource was though as the smartphone, being the "father" of all the other resources.

This mapping, for the mobile Gateway is illustrated in Figure 3.6.

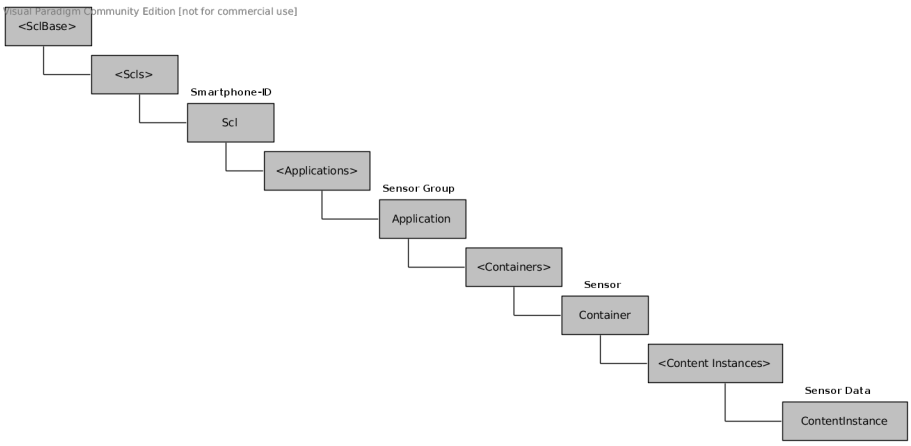


Figure 3.6: Mapped Gateway ETSI resource tree.

Naturally the proof-of-concept Network Application had to have some mapping of its own, and it differs a little bit from the Gateway. The only information the NA has to deliver are the commands that manage the Gateway's actions. Because of this, the *Container* was literally called Actions, and the associated *ContentInstances* carry specific information about the action to be performed.

The *Application* resource in this situation has no particular mapping, as it is only a proof of concept to show that it is possible to control the Gateway remotely. The NA mapping is depicted in Figure 3.7.

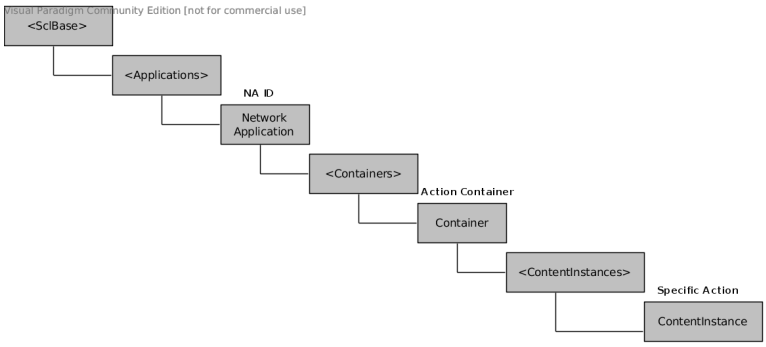


Figure 3.7: Mapped Network Application ETSI resource tree.

3.5.2 Send commands to the Gateway

As shown above in Section 3.4, dedicated to the Network Application, one of the key features of this dissertation's project is the ability to control the Gateway remotely since itself, the Gateway has no User Interface. Using the ETSI standard for M2M, there was the need to somehow map this interaction. Helped by the *PT Inovação's* partners, two approaches were discussed:

First Approach:
In this approach, the data flow would start with the subscription of the resource *Applications*, by the GSCL. This subscription would be limited in the NSCL by Access Rights, providing only

the relevant NA information back to the GSCL. This means that when the NA changes its own resources, the GW will be notified, containing the relevant actuation data.

Second Approach:
In this approach, when the GSCL is in the process of registering the *Container* resources, two different types of resources would be created, one for the GSCL to write in, that would be subscribed by the NACL, and another for the latter to write in, that would be subscribed by the GSCL. The sensor data would be sent in the container in which the GSCL would write in, and the acting commands would be sent by the NACL *Container*, triggering the notification for the intended destination, thus communicating.

Some considerations were taken into account to this decision:

- On the first approach, without the *Access Rights* limitation by the NSCL, the Gateway would have an enormous amount of information to deal with, to subscribe all active *Application* resources, increasing the network traffic with possibly useless information.
- The first approach is considerably more scalable, since as long as the mechanism for subscribing the Network Applications is working, it will work on every case.
- The second approach would have the NACL write data in a Container that did not belong to itself, which could possibly cause *Access Rights* issues in the future.
- The second approach would have the GSCL subscribe its own resource, because it would be changed by the NA.

All things considered the First Approach was chosen, as shown previously in Figure 2.7, where it becomes clear that the commands information is stored within the NA in the Actions *Container*. The second approach had more issues to tackle to begin with, than the other, and was more scalable and easier to implement.

3.6 Local API

Given the use case , the two major reasons for why a local connection is an interesting choice are the previously mentioned savings in bandwidth and battery as a consequence. This is particularly useful in this dissertation's situation, since the NSCL acts as a intermediary between the Gateway and the Network Application, as shown in Figure 3.8. Only the information flow since the subscriptions are already present is shown

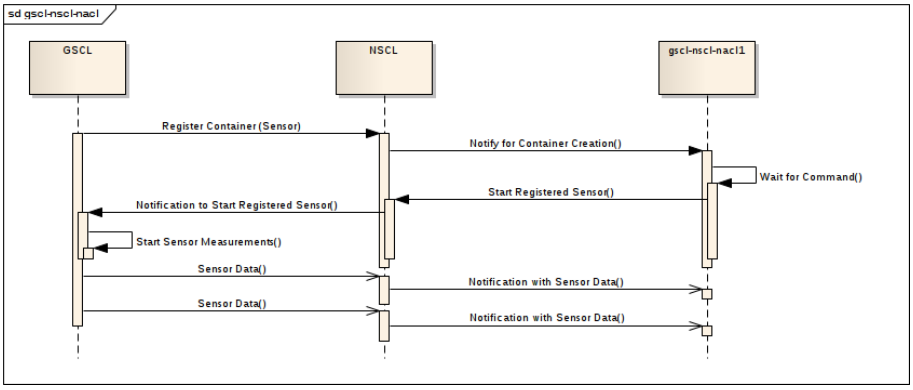


Figure 3.8: NSCL as information forwarder.

If the Gateway and Network Application are both in the same smartphone, it is unpractical to keep communicating through the NSCL, when there could be a much easier local connection. The information would be stored within the Gateway to delivery to the NSCL when the measurement stops.

(In progress.)

3.7 Technologies

First off, both the mobile M2M Gateway and the Network Application are being built as an Android Operating System application. This choice was made given the open source nature of the operating system, which provides easy integration with the external sensors, and because the programming language is in Java, which was already familiar. The wide range of devices currently running Android OS was also an important factor since the result of this dissertation may evolve into a commercially available product.

The main software used in the development was:

- Android Development Kit (SDK) [56].
- IntelliJ Integrated Development Environment (IDE) [57].
- Apache Maven, for the dependency management, and easier integration of the several modules [58].
- Subversion version control [59].

The frameworks used to implement the use cases were:

HTTP Native Apache HTTP implementation.

CoAP Californium CoAP framework [60].

MQTT Paho-mqtt library (Release 0.4.0, MQTT version 3) [61].

Jackson Jackson is the library used to execute the fast [62] marshalling and de-marshalling of ETSI resources.

Also, as communication technologies, there was also need to use the Bluetooth, cellular (2G/3G) and Wi-Fi capabilities to communicate with the NSCL.

3.8 Evaluation

This dissertation is going to be evaluated in according to the goals mentioned in Section 1.3.

To show that the M2M Gateway and proof-of-concept Network Application are fully functional, they must be able to:

GW Have the ability to communicate securely with the NSCL using at least one supported protocol.

GW Have the ability to register itself and its sensors data onto the NSCL.

GW Have the ability to store sensors information for situations where connection is lost, or for battery saving purposes.

GW Have a web server implemented, that allows it to receive and process commands that fulfill the Use Cases explained in Section 3.1.

GW Have the ability to be controlled by both NSCL and local API, with aid of the proof-of-concept application.

NA Have the same abilities as the Gateway (secure connection, functional communication, working webserver), to allow the delivery of commands to the Gateway.

NA Have a graphical user interface that allows the user to fulfill the proposed Use Cases (Section 3.1).

Both Test the proof-of-concept application ability to fulfill the proposed use cases.

Both Gather Network traffic information on the M2M Gateway:

Communicating through the NSCL;

Communicating through the Local API;

Chapter 4

Development

The development of the architectured solution shown in the previous Chapter had its share of obstacles and difficulties, which are always present in this kind of project. This Chapter aims to explain the development of each feature and functionality, what were the main difficulties, and how they were overcome.

First off, the Bootstrap procedures are present in Section 4.1, followed by the ETSI supported protocols implementation and explanation in Section 4.2. Then, in Section 4.3 the Webserver created within the Gateway and Network Application to receive the needed subscriptions is detailed, before the details of the ETSI resource structure implementation in Section 4.4.

Section 4.5 focuses on the integration of the heal-care sensors, paving the way for the explanation of the Gateway communication with the NSCL as well as the functionalities of the commands, in Section 4.6. The Network Application's communication is then detailed in Section 4.7 detailing how different actions are shown in the user interface.

The proposed Local API is detailed in Section 4.8, followed by some Bandwidth measurements in Section 4.9 comparing the communication with and without the use of this Local API.

Finally, in Section 4.10, there will be a discussion on all the development executed and the results obtained.

4.1 Bootstrap

As mentioned in Section 2.3.1 the Bootstrap is the first procedure to be executed, either by the Gateway or by the Network Application (the procedure is the same for both), as it yields the Session Key (which is a Certificate-Key pair) to be used in the secure connection with the NSCL. Certificates and private keys were explained in Section 2.6.1 and 2.6.2 respectively.

In order to correctly execute the Bootstrap, the GW or NA have to have use a pre-shared certificate-key pair, so that they can retrieve the session keys. This pre-shared pair was stored as a project resource, and is compiled with the applications to allow them to execute this procedure. After receiving the NSCL answer, the application

4.2 ETSI Protocol Implementation

The Protocol Manager and its GSCL and NACL sub-modules were built in a very modular way (as explained in Chapter 3), to allow the use of either HTTP or CoAP protocols to communicate with the NSCL.

Initial development efforts focused on CoAP, for the communication with the NSCL, but there were some issues with the Californium CoAP library (suggested by the *PT Inovação* partners because they were also using it) and the use of DTLS, since the framework did not provide a useful API to determine where the configuration and certificate files were located. This forced the import of the framework's source code into the project, to set these options. However, since the NSCL didn't fully support CoAP at the time, this integration was put on hold, and efforts focused on the integration of HTTP.

With HTTP, the first stage was to find a library that could connect over TLS using the certificates retrieved from the NSCL. *PT Inovação* used an Oracle library, that yielded dependency issues in Android, which ultimately led to the use of the native Apache HTTP libraries shipped with the Android Operating System. The choice was fruitful, since the connections were successfully implemented using TLS (v1.2) and also implementing a TLS extension (and ETSI requirement) called SNI. Server Name Indication (or SNI), allows a server (in this case the NSCL) to provide different services over the same IP address, provided that the client specifies its destination in the connection. In this project's case, the NSCL URI was *https://phonegw.nscl.m2m.ptinovacao.pt*, and the SNI "phonegw.nscl.m2m.ptinovacao.pt" was used to differentiate between the different NSCL instances.

On Android there were some issues with this feature implementation, where the well documented Android fragmentation [63] was felt. The Apache features that allow the SNI to be properly implemented in Android were only introduced in API 17 (Android version 4.2), which means that this functionality limits the number of devices that can use the final application.

4.3 Web-server

The web-server exists both in the GW and NA to allow the NSCL (or each other using the local API) to subscribe their resources. These are naturally protocol specific, and the CoAP web-server was never implemented, because the communication development focused on HTTP for the reasons mentioned in the previous Section.

As had happened on the client, there was some research on HTTP lightweight implementations, being NanoHTTPD [64] the most interesting. This library was implemented, but the lack of SSL/TLS support led to its eventual abandon. Since the client was already functional with the Apache libraries, the Server was also implemented using these, with and without support for TLS. Since the NSCL still does not use a secure connection to subscribe to the web-server, the TLS is turned off for now.

Within FEUP there were a few bureaucracy steps[1], as well as configuration challenges[2], to allow the web-server to be accessed from the outside of the FEUP network (from *PT Inovação*, in Aveiro). All this was an unexpected delay that forced some thoughts on how this implementation would scale, since normal users do not know or want to mess with their home router configurations. A solution would be the use of mobile data for this, but assuring that every card had its own external IP address in an age where IPV4 addresses are scarce, would probably force the shift to IPV6, with all the inherent costs.

4.4 ETSI Resource Structure Implementation

Implementing the ETSI resource structure was the first big milestone of this dissertation's project, since it was a requirement for communicating with the NSCL.

The ETSI documents, specifically the TS 102 921 [26], besides the useful information about communication, also provided XML schemas for every resource of the structure. With JAXB (XML marshalling library) it was possible to convert these schemas into Java classes with annotations, for automatic marshalling and de-marshalling. The problem faced was that Android could not understand the annotations because of missing XML libraries. What happens is that although Android uses XML files to build its user interfaces, it does not implement the core XML libraries (javax.xml.*) and prevents them to be compiled with the application, because they are core libraries.

Given this, there were two possible approaches:

- Import the XML libraries under a different package (such as xml.libraries.*) so that they could get compiled along with the application.
- Use a different library and figure out the best way to make use of the provided classes.

The solution chosen was to use the Jackson library, because of its fast performance on Android OS [62] (even when compared with the native Android JSON library), and because both approaches meant editing some or all of the provided 120 classes: The first approach meant changing the imports of every class, while the second meant commenting out the XML annotations and include a few Jackson when needed (only known by trial and error).

The Jackson JSON approach seemed like the best option on the long run, even though it is clear that if there are new changes to those classes, they will have to be implemented by hand, prevent having the same annotation issue as before.

A comprehensive list of the changes executed in the ETSI generated classes can be found in Appendix A.

[1] Acknowledgment of the significant help of Assistant Professor Isidro Ribeiro in requesting the port openings to CICA.

[2] Acknowledgment of the significant help of Paulo Vaz (IT technician), and Carlos Pereira (Doctorate student in FEUP) for the help given with this configuration procedure.

After all the changes were completed and the applications were successfully marshalling and de-marshalling the resources using Jackson, the focus shifted on the communication of the GSCL with the NSCL, but not before integrating a medical sensor to test the data with.

4.5 Sensor Integration

The Gateway's Sensor Handler was built in a modular way, and each sensor connected has its own thread to prevent race condition issues. When the sensors are searched, for each sensor found, a new Container is created, with the sensor name (e.g. ZEPHYR), but these the sensors are not connected until a specific *START* command is received. Only then, does the Gateway connect to the sensor and start collecting data to build and send the ContentInstance resources with. This was done for scalability, because no sensor is going to consume battery unless specifically started.

Originally the sensor chosen was the *Zephyr™ HxM Heart-Rate monitor*, because it has an open Java API for connection and data decoding, with Android examples. This sensor was successfully implemented and used for the tests executed during the development of the communication between the Gateway and the NSCL (explained in the following Section).

Additional health sensors (a medical scale and a sphygmomanometer) integration was planned, but some bureaucracy issues occurred between *PT Inovação* and *Instituto de Telecomunicações* over a Non-Disclosure Agreement. *PT* had signed the NDA with a sensor provider (ForaCARE), for information on their API, and the legal division of *PT* took over three weeks to re-write the agreement between the two entities, so that the NDA would also be extended to *IT*. When the OK finally arrived, the focus was on building the Network Application and achieving an easy way to control the Gateway and its sensors, which left the new devices implementation in stand-by.

4.6 Gateway Communication (GSCL <-> NSCL)

The communication between the Gateway and the NSCL focuses on the Scl resource, where the Gateway is a type of Scl with certain capabilities, hence GSCL.

This communication is shown in Figure 4.1, and explained below.

GSCL->NSCL The first message sent by the Gateway is the POST to create an *Scl* resource (on [nsclURI]/m2m/scls). The *sclId* of the Gateway reference to the device model and serial number, to assure it is unique and there are no two equal IDs.

[NSCL->GSCL] If this smartphone was previously connected to the NSCL and the resource already exists, the NSCL returns the HTTP code 405 (Method not Allowed), specifying that it was a Bad Request. The Gateway then retrieves a list of *Scl*'s registered on the NSCL to generate a unique *sclId* by adding numbers in front, and registers it in a POST request equal to the first, but with a different ID. This is less than ideal, but there still was not an agreement with the *PT Inovação* partners on how to execute the Update of the existing *Scl* resources.

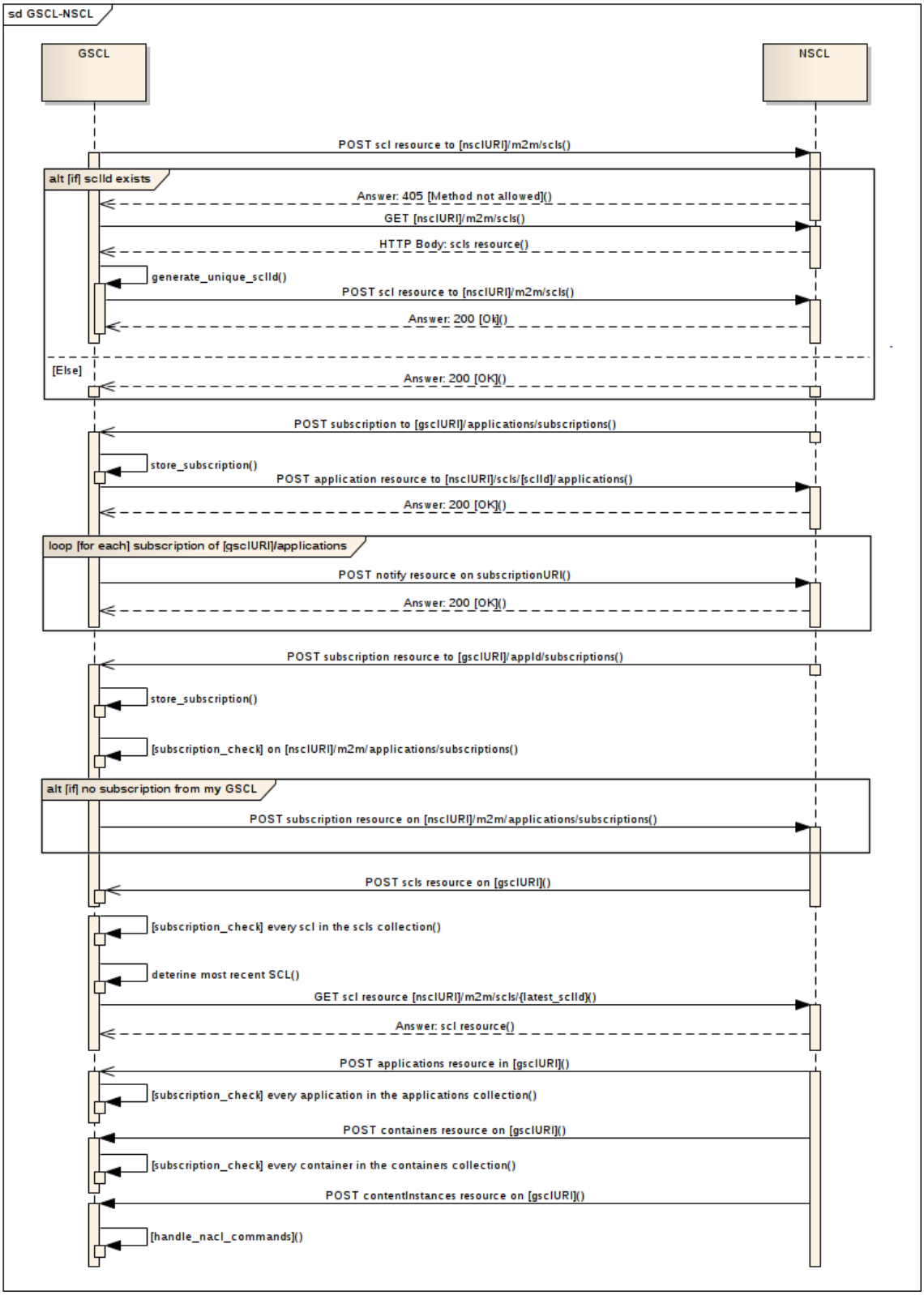


Figure 4.1: Sequence diagram of the communication held between the GSCL and the NSCL.

[NSCL->GSCL] If the Gateway had never connected to the NSCL or the new uniquely generated *sclID* was successfully recognized, the NSCL returns the HTTP code 200 (OK), with the complete *Scl* resource in the message body. This happens because the NSCL is responsible by creating some references that are then sent back to the entity that made the request (GSCL).

NSCL->GSCL After the successful creation of the *Scl* resource, the NSCL uses the "link" attribute to subscribe to the *Applications* collection of said *Scl*, by sending a POST message with the *Subscription* resource to [gsclURI]/applications/subscriptions. This subscription is stored in the memory of the Gateway, in order to later notify the subscribers, and the next step of the Gateway registration is triggered.

GSCL->NSCL After the subscription on the *Applications* collection, the Gateway registers its *Application* resources (one or more) on the NSCL *Applications* collection, by sending a POST message to [nsclURI/scls/[sclId]/applications, with the attributes needed to successfully create the application in the

NSCL->GSCL Creates the *Application* resource locally, and generates the specific NSCL attributes, and returns 200/OK, with the marshalled resource.

GSCL->NSCL Notifies every subscriber of the *Applications* collection sending a POST request with the Notify resource to the specific *subscriptionURI*, which is an attribute of the *Subscription* resource.

NSCL->GSCL Returns the HTTP code 200 (OK) for the Notification of each subscriber.

NSCL->GSCL Subscribes each Application resource registered in the NSCL with a POST request to [gsclURI]/appId/subscriptions.

GSCL<->NSCL Stores the received subscriptions of the *Application* resources and executes a *Subscription Check* procedure (which will be described in Section 4.6.1, with aid of Figure 4.2) to check if the current Scl (with the same URI) has already subscribed to the [nsclURI]/m2m/applications/subscriptions collection, which holds the Network Applications.

GSCL->NSCL If there is no Subscription on the collection, the GSCL subscribes to it, in order to receive updates on the *Container*'s associated with it.

4.6.1 Subscription Check

In progress.

4.7 Network Application Communication (NACL <-> NSCL)

In progress.

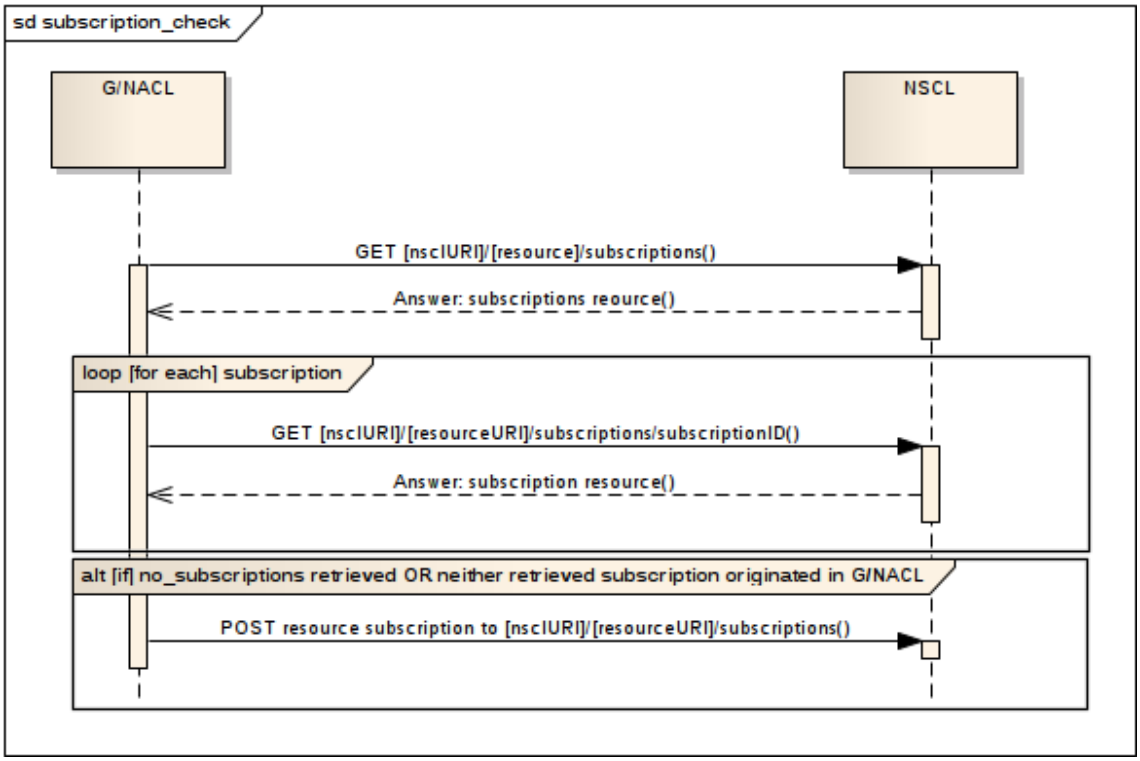


Figure 4.2: Procedure to check if Subscription created by this GSCL is present.

4.8 Local API (GSCL <-> NACL)

In progress.

4.9 Bandwidth Measurements

4.9.1 Communicating through the NSCL

In progress.

4.9.2 Communicating through the Local API

In progress.

4.10 Discussion

In progress.

4.10.1 Main Obstacles

In progress.

Chapter 5

Conclusions and Future Work

In progress.

5.1 Conclusions

In progress.

5.2 Future Work

In progress.

Appendix A

ETSI classes changes

This appendix will document the changes executed on the classes automatically generated from the ETSI standard schemas.

A.1 XML Imports

As mentioned in Section 4.4, all the classes were stripped of the *"javax.xml.*"* imports, as Android OS does not implement them.

A.2 Jackson Annotations

On some cases (specially when there were JSON Arrays), the removal of the XML imports, caused errors during serialization an de-serialization, because the Class variable names were different than they should. To fix this, some Jackson annotations had to be done, most of them to force the variable name into the specified in the documents. The classes that now have Jackson annotations are:

- AnyURIList.java
- APocHandling.java
- Application.java
- Applications.java
- Container.java
- ContentInstanceCollection.java
- ContentTypes.java
- CreateRequestIndication.java
- NamedReferenceCollection.java
- Scl.java

- SearchStrings.java
- Subscription.java
- SubscriptionType.java

A.3 Custom Serializers and De-Serializers

Even with the annotations, some classes could only be fully compliant with the ETSI specifications after they had a custom Jackson serializer and/or de-serializer made for them. Only the class *SearchStrings* needed a custom serializer. The following list enumerates the ones that needed a custom de-serializer:

- Application.java
- Container.java
- ContentTypes.java
- Scl.java
- SearchStrings.java
- Subscription.java

These custom made classes are located in the common module of the project, under "pt.ptinovacao.mtom.common.ma

Bibliography

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

[2] Chonggang Wang. M2M Service Architecture: Delivering M2M Services Over Heterogeneous Networks. *IEEE Communications Quality & Reliability 2012 International Workshop*, 2012.

[3] *Lightweight Internet protocols for web enablement of sensors using constrained gateway devices*, 2013 International Conference on Computing, Networking and Communications (ICNC 2013). IEEE, 2013. URL: http://dx.doi.org/10.1109/ICCNC.2013.6504105.

[4] RabbitMQ. Amqp 0.9.1 model explained. https://www.rabbitmq.com/tutorials/amqp-concepts.html. Online; accessed 11-02-2014.

[5] Malcolm Clarke, Joost de Folter, Charles Palmer, and Vivek Verma. Building point of care health technologies on the IEEE 11073 health device standards. In *2013 IEEE Point-of-Care Healthcare Technologies (PHT)*, pages 117–119. IEEE, January 2013. URL: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6461298, doi:10.1109/PHT.2013.6461298.

[6] Aeronautics and National Research Council Space Engineering Board. *The Global Positioning System:A Shared National Asset*. The National Academies Press, 1995. URL: http://www.nap.edu/openbook.php?record_id=4920.

[7] Rongxing Lu, Xu Li, Xiaohui Liang, Xuemin Shen, and Xiaodong Lin. GRS: the green, reliability, and security of emerging machine to machine communications. *IEEE Communications Magazine*, 49(4):28–35, April 2011. URL: http://dx.doi.org/10.1109/MCOM.2011.5741143, doi:10.1109/MCOM.2011.5741143.

[8] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010. URL: http://dx.doi.org/10.1016/j.comnet.2010.05.010, doi:10.1016/j.comnet.2010.05.010.

[9] IBM. Mqtt v3.1 protocol specification. http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf, 2010. Online; accessed 31-01-2014.

[10] R. Fielding, UC Irvine, J. Gettys, Compaq/W3C, J. Mogul, Compaq, H. Frystyk, W3C/MIT, L. Masinter, Xerox, P. Leach, Microsoft, T. Berners-Lee, and W3C/MIT. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Network Working Group, June 1999. URL: http://tools.ietf.org/pdf/rfc2616.pdf.

[11] Z. Shelby, Sensinode, K. Hartke, C. Bormann, and Universitaet Bremen TZI. Constrained Application Protocol (CoAP) draft-ietf-core-coap-18. IETF Internet Draft, http://tools.ietf.org/pdf/draft-ietf-core-coap-18.pdf, August 2013. Online; accessed 07-02-2014.

[12] C. Bormann, Universitaet Bremen TZI, Ed. Z. Shelby, and Sensinode. Blockwise transfers in CoAP draft-ietf-core-block-14. IETF Internet Draft, http://tools.ietf.org/pdf/draft-ietf-core-block-14.pdf, October 2013. Online; accessed 07-02-2014.

[13] S Vinoski. Advanced Message Queuing Protocol. *Internet Computing, IEEE*, pages 87–89, 2006. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4012603.

[14] GE Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965. URL: http://web.eng.fiu.edu/npala/EEE5425/Gordon_Moore_1965_Article.pdf.

[15] *ETSI TR 102 935 V2.1.1 (2012-09) Machine-to-Machine communications (M2M); Applicability of M2M architecture to Smart Grid Networks; Impact of Smart Grids on M2M platform*, 2012.

[16] *ETSI TR 102 691 V1.1.1 (2010-05) Machine-to-Machine communications (M2M); Smart Metering Use Cases*, 2010.

[17] Min Chen, Jiafu Wan, Sergio Gonzalez, Xiaofei Liao, and Victor C.M. Leung. A Survey of Recent Developments in Home M2M Networks. *IEEE Communications Surveys & Tutorials*, pages 1–17, 2014. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-84888162287&partnerID=tZOtx3y1http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6674156, doi:10.1109/SURV.2013.110113.00249.

[18] Jon Tong-Seng Quah and Guey Long Lim. Push selling—Multicast messages to wireless devices based on the publish/subscribe model. *Electronic Commerce Research and Applications*, 1(3-4):235–246, September 2002. URL: http://www.sciencedirect.com/science/article/pii/S1567422302000194, doi:10.1016/S1567-4223(02)00019-4.

[19] ISO/IEC. Information technology - open systems interconnection - basic reference model: The basic model. International Standard 7498-1, ISO/IEC, 1994.

[20] Microsoft. The osi model's seven layers defined and functions explained. http://support.microsoft.com/kb/103884, 2002. Online; accessed 06-06-2014.

[21] 3GPP. Service requirements for Machine-Type Communications (MTC); stage 1, release 12. Technical Report TS 22.368 V12.3.0, 3GPP, 2013.

[22] 3GPP. Technical Specification Group Services and System Aspects; Study on Facilitating Machine to Machine Communication in 3GPP Systems. Technical Report TR 22.868, 3GPP, 2007.

[23] *ETSI TS 102 689 V2.1.1 (2013-07) Machine-to-Machine communications (M2M); M2M service requirements*, 2013.

[24] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. http://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf, 2002. Online; accessed 07-06-2014.

[25] Guang Lu. *Overview of ETSI M2M Release 1 Stage 3 – API and Resource usage*, 2011.

[26] *ETSI TS 102 921 V2.1.1 (2013-12) Machine-to-Machine communications (M2M); mIa, dIa and mId interfaces*, 2013.

[27] V. Ryan, S. Seligman, R. Lee, and Inc. Sun Microsystems. The Base16, Base32, and Base64 Data Encodings. RFC 4648, Network Working Group, October 2006. URL: http://tools.ietf.org/pdf/rfc4648.pdf.

[28] V. Ryan, S. Seligman, R. Lee, and Inc. Sun Microsystems. Schema for Representing Java(tm) Objects in an LDAP Directory. RFC 2713, Network Working Group, October 1999. URL: http://tools.ietf.org/pdf/rfc2713.pdf.

[29] W3C. Extensible markup language (xml). http://www.w3.org/TR/WD-xml-961114.html, November 1996. Online; accessed 31-01-2014.

[30] ISO 8879. Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML). ISO Standard, 1986.

[31] M. Murata E. Whitehead, UC Irvine. XML Media Types. RFC 2376, Fuji Xerox Info. Systems, July 1998. URL: http://tools.ietf.org/html/rfc2376.

[32] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, JSON.org, July 2006. URL: http://tools.ietf.org/html/rfc4627.

[33] json schema.org. JSON Schema. http://json-schema.org/documentation.html. Online; accessed 10-02-2014.

[34] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994. URL: http://dl.acm.org/citation.cfm?id=179606.179671, doi:10.1145/179606.179671.

[35] T. Dierks, Certicom, and C. Allen. The TLS Protocol, Version 1.0. RFC 2246, Network Working Group, January 1999. URL: http://www.ietf.org/rfc/rfc2246.txt.

[36] R. Fielding, UC Irvine, J. Gettys, J. Mogul, DEC, H. Frystyk, T. Berners-Lee, and MIT/LCS. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068, Network Working Group, January 1997. URL: http://tools.ietf.org/pdf/rfc2068.pdf.

[37] Jon Postel. Transmission Control Protocol. RFC 793, Information Sciences Institute, University of Southern California, September 1981. URL: http://www.ietf.org/rfc/rfc793.txt.

[38] J Postel. User Datagram Protocol. RFC 768, Information Sciences Institute, University of Southern California, August 1980. URL: http://tools.ietf.org/pdf/rfc768.pdf.

[39] T. Berners-Lee, MIT/LCS, R. Fielding, U.C. Irvine, L. Masinter, and Xerox Corporation. Transmission Control Protocol. RFC 2396, Network Working Group, August 1998. URL: http://www.ietf.org/rfc/rfc2396.txt.

[40] ETSI, IPSO Alliance, and OMA. CoAP 3 & OMA Lightweight M2M Plugtest. http://www.etsi.org/news-events/past-events/693-coap-oma-lightweight-m2m, November 2013. Online; accessed 07-02-2014.

[41] N. Shadbolt, T. Berners-Lee, and W. Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1637364, doi:10.1109/MIS.2006.62.

[42] Carsten Bormann, Angelo P. Castellani, and Zach Shelby. CoAP: An Application Protocol for Billions of Tiny Internet Nodes. *IEEE Internet Computing*, 16(2):62–67, March 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6159216, doi:10.1109/MIC.2012.29.

[43] J Postel and ISI. User Datagram Protocol. RFC 768, ISI, August 1980. URL: http://tools.ietf.org/pdf/rfc768.pdf.

[44] Suresh Krishnan and Sheila Frankel. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, IETF, February 2011. URL: http://tools.ietf.org/html/rfc6071.

[45] Eric Rescorla and Nagendra Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, IETF, January 2012. URL: http://tools.ietf.org/search/rfc6347.

[46] Jon Postel. Transmission Control Protocol. RFC 793, Information Sciences Institute, University of Southern California, September 1981. URL: http://www.ietf.org/rfc/rfc793.txt.

[47] OASIS Standard. Oasis advanced message queuing protocol (amqp) version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf. Online; accessed 11-02-2014.

[48] Ed. A. Melnikov, Isode Limited, Ed. K. Zeilenga, and OpenLDAP Foundation. Simple Authentication and Security Layer (SASL). RFC 4422, Network Working Group, June 2006. URL: http://www.ietf.org/rfc/rfc4422.txt.

[49] S. Blake-Wilson, BCI, M. Nystrom, RSA Security, D. Hopwood, Independent Consultant, J. Mikkelsen, Transactionware, T. Wright, and Vodafone. Transport Layer Security (TLS) Extensions. RFC 4366, Network Working Group, June 2006. URL: http://www.ietf.org/rfc/rfc4366.txt.

[50] D. Cooper, S. Santesson, Microsoft, S. Farrell, Trinity College Dublin, S. Boeyen, Entrust, R. Housley, Vigil Security, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, NIST, May 2008. URL: http://tools.ietf.org/pdf/rfc5280.pdf.

[51] B. Kaliski and J. Staddon. PKCS 1: RSA Cryptography Specifications Version 2.0. RFC 2437, RSA Laboratories, October 1998. URL: http://www.ietf.org/rfc/rfc2437.txt.

[52] Jianchu Yao and Steve Warren. Applying the ISO/IEEE 11073 standards to wearable home health monitoring systems. *Journal of clinical monitoring and computing*, 19(6):427–36, December 2005. URL: http://www.ncbi.nlm.nih.gov/pubmed/16437294, doi:10.1007/s10877-005-2033-7.

[53] Malcolm Clarke, Douglas Bogia, Kai Hassing, Lars Steubesand, Tony Chan, and Deepak Ayyagari. Developing a standard for personal health devices based on 11073. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2007(Dim):6175–7, January 2007. URL: http://www.ncbi.nlm.nih.gov/pubmed/18003430, doi:10.1109/IEMBS.2007.4353764.

[54] Google Inc. Android BluetoothHealth API. http://developer.android.com/reference/android/bluetooth/BluetoothHealth.html.

[55] Continua Health Alliance. www.continuaalliance.org/.

[56] Get the Android SDK. http://developer.android.com/sdk/index.html?utm_source=weibolife. Online; accessed 09-06-2014.

[57] Jetbrains IntelliJ IDEA. http://www.jetbrains.com/idea/. Online; accessed 09-06-2014.

[58] Apache maven. http://maven.apache.org/. Online; accessed 09-06-2014.

[59] Apache subversion. http://subversion.apache.org/. Online; accessed 09-06-2014.

[60] Californium (Cf) CoAP framework. http://people.inf.ethz.ch/mkovatsc/californium.php. Online; accessed 09-06-2014.

[61] Eclipse Paho MQTT. http://www.eclipse.org/paho/. Online; accessed 09-06-2014.

[62] Performance Comparison of JSON frameworks for Android OS. https://github.com/martinadamek/json-android-compare. Online; accessed 09-06-2014.

[63] Android Dashboard - Version Statistics. http://subversion.apache.org/. Online; accessed 09-06-2014.

[64] Android Dashboard - Version Statistics. https://github.com/NanoHttpd/nanohttpd. Online; accessed 09-06-2014.