FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mobile Healthcare on a M2M Mobile Gateway

Ricardo Morgado

PREPARAÇÃO DA DISSERTAÇÃO

MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Supervisor: Ana Aguiar

February 16, 2014

Mobile Healthcare on a M2M Mobile Gateway

Ricardo Morgado

MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

February 16, 2014

Contents

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. A quick M2M example is GPS systems [1]. 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.

This dissertation is going to study the use of a smartphone as an enabler of M2M services, 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 3.3.

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 (explained in greater detail in Section 2.1) 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 [2]). CoAP (Constrained Application Protocol [3, 4]) support is currently in development, and AMQP [5] remains a possibility for later addition. All these protocols are described in Section 2.4.

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 [6], 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 2.1, 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 [7, 8], 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 [9].

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

This dissertation's first objective is the conclusion of the mobile M2M Gateway, with the ability to communicate in MQTT and CoAP with *PT*'s Broker, using ETSI's M2M communication standard, explained in Section 2.1.1. The Gateway will also feature the versatile modular architecture explained in greater detail in Section 3.2, which aims to provide the flexibility necessary for adding new sensors or protocols, without the need to re-write the entire application.d

Thereafter, in sequence of the M2M Gateway development, an Application that is works as proof-of-concept in the mobile health-care scenario described in Section 3.3, is also going to be developed.

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 evaluate the performance of the M2M Gateway with each protocol, specifically in terms of battery drain, and network usage on both cellular and Wi-Fi. This could be accomplished by sending the same amount of information though each protocol for the same duration, monitor the behavior of the smartphone and the network. The goal of this study is to make a real scenario comparison between the protocols, that may end up with the suggestion of a smart algorithm to control the communication channels, to both improve battery life and reduce network traffic.

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, in Section 2.1, there is going to be a wider approach into machine-to-machine communications, followed by the ETSI proposed general architecture in Section 2.1.1. Then we will take a deeper look into the role of the mobile M2M Gateways in the general M2M architecture (2.1.2), with focus on the GSCL interface to be developed (2.1.2.1), followed by the most common marshalling techniques such as XML and JSON in Section 2.1.3.

Then, there will be a study of *PT Inovação*'s Smart2M specifics, in Section 2.3, in sequence with an explanation of the publish-subscribe paradigm in Section 2.2. This will pave the way for a deeper look into the more commonly discussed protocols to provide M2M communications, namely CoAP, MQTT and AMQP, respectively in Sections 2.4.1, 2.4.2 and 2.4.3.

Then, in Sections 2.5.1 and 2.5.2, 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 conclusion of this Chapter will be detailed in Section 2.6.

2.1 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*", [1]

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 [9]. 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. [10]. 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 [1], 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 [11].

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 [11]. 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 [1, 12]. Some interesting statistics are shown in [9], 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.

2.1.1 M2M Architecture

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) [13] 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 [14] with the M2M service requirements, and TS 102 690 [15] 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.

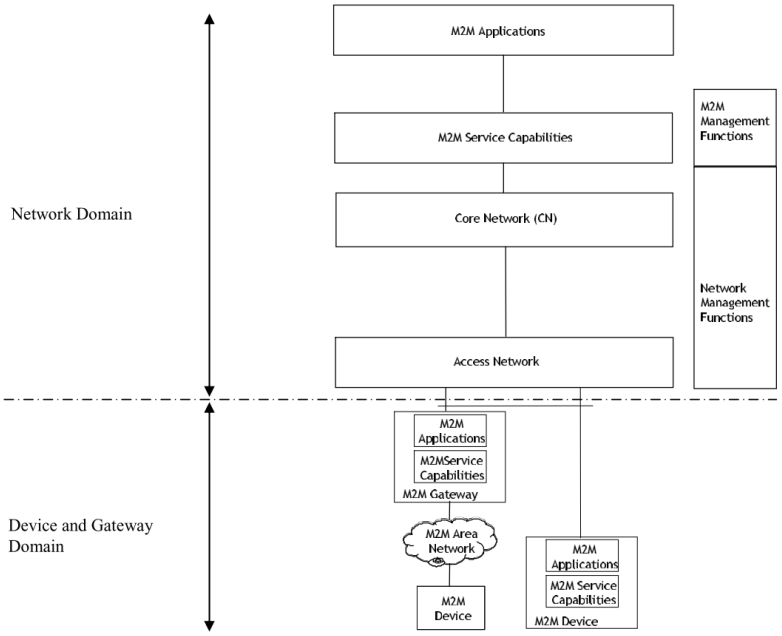


Figure 2.1: Machine-to-Machine high level architecture. From [15].

Figure 2.1 presents the high level view of the ETSI M2M system architecture within the scope of [14, 15]. 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. Area Networks can use technologies such as IEEE 802.15.1, Bluetooth®, ZigBee®, IETF ROLL, ISA100.11a, etc., or local networks such as the PLC, M-BUS and Wireless M-BUS and KNX.

- **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. For this Network xDSL, HFC, satellite, GERAN, UTRAN, eUTRAN, W-LAN and WiMAX can be used.

- **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. This Network may use 3GPP CNs, ETSI TISPAN CN and 3GPP2 CN.

  M2M Service Capabilities: The SCs provide M2M functions that are to be shared by different Applications, exposing thos 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 2.2. 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 (REST) is the dominant approach in client-server communications, that allows for stateless communication, 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 that allow for stateless communications are encouraged [15, 16].

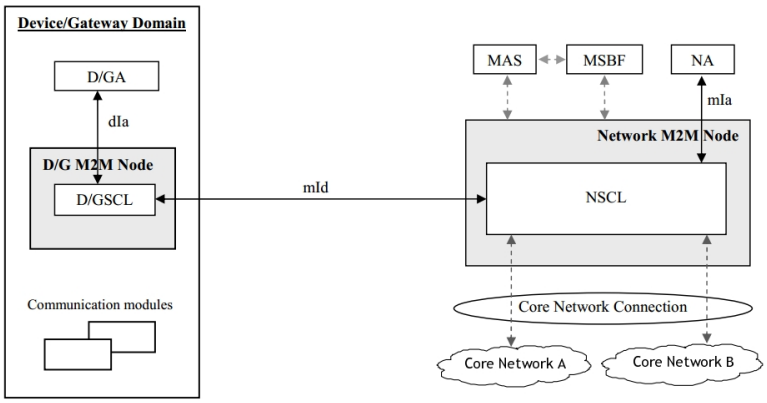


Figure 2.2: M2M Service Capabilities functional architecture framework. From [15].

Next Sections will focus on the role of a mobile M2M Gateway, with focus on the GSCL interface it will implement, because of the importance it holds on this project.

2.1.2 Mobile Gateway

Given the importance of being transparent to the user about the use of their hardware, the most important resources of a mobile M2M Gateway are energy and data transmission efficiency, reliability and security [1, 12], so that the impact of the running Gateway does not affect the normal use of the device. It is also critical that only someone with ownership of the Gateway 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.

2.1.2.1 GSCL

The Gateway Service Capability Layer implements ETSI's Service Capabilities (SCs) allowing the M2M Gateway to communicate with the other entities of the ETSI M2M architecture. The base Resource tree of the SCL in pictured in Figure 2.3.

Given the real possibility that the specification may not need to be entirely implemented for this project's purpose, together with *PT Inovação* the following Resources show in Figure 2.4 were considered critical, because of the mapping intended to them.

This mapping will allow to have one Application entity for each kind of sensor (internal and external), while maintaining another reserved for Gateway control, allowing the reception of commands. The latter will not have devices under Container, but commands instead, allowing for proper remote control of the M2M Gateway.

The SCL concept is, in essence, a marshalling technique. The more common marshalling techniques will be addressed in the next Section.

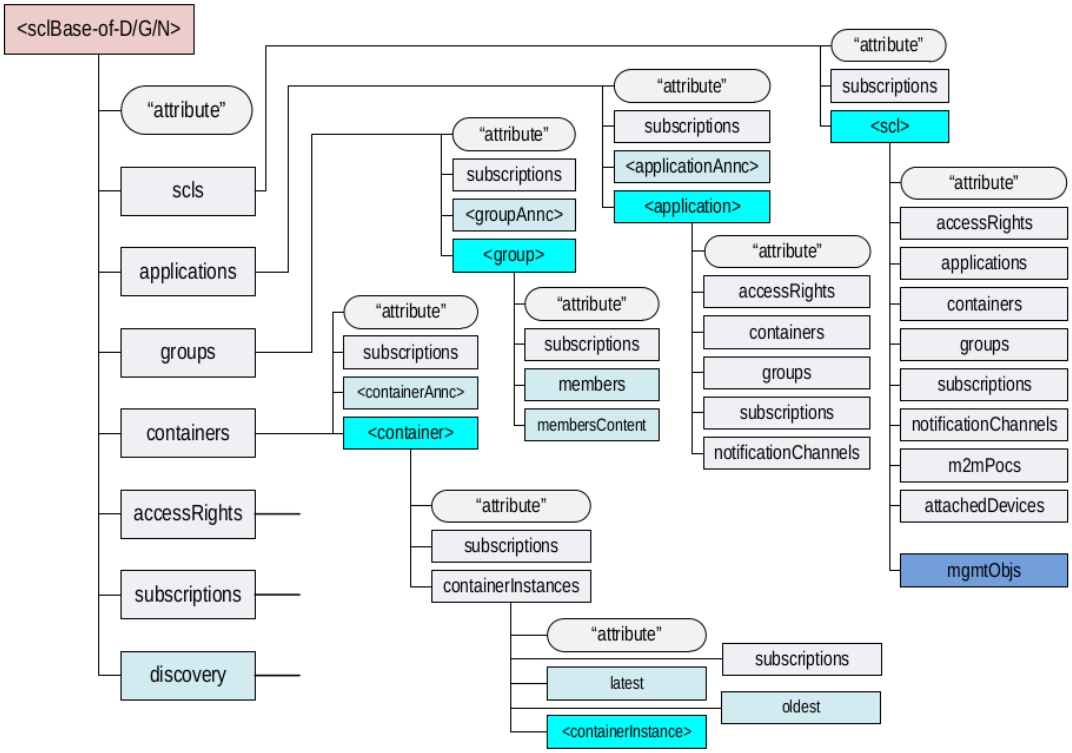


Figure 2.3: Base SCL on ETSI's standard. From [17]

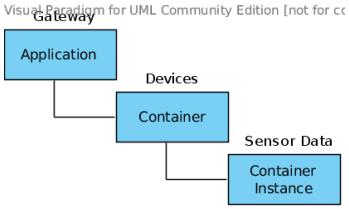


Figure 2.4: GSCL approach to ETSI standard.

2.1.3 Marshalling

Marshalling, or Serializing, is the nomenclature given to the process of packaging data for storage or transport, in order to be easily processed by different systems, which can built with use of any language. This allows for interoperability of services by using standards readable by both machines and humans [18].

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

2.1.3.1 XML

The eXtensible Markup Language, commonly known as XML, was first released in a World Wide Web (W3C) draft in 1996 [19] as a derivation from the Standard Generalized Markup Language (SGML), which had been standardized in the International Organization for Standardization

(ISO) in ISO 8879 [20]. XML 1.0 was originally submitted for standardization on an RFC (Request for Comments) in 1998 [21] 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>
```

2.1.3.2 JSON

JavaScript Object Notation (JSON), was originally specified in [22] 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 [23] is below. As intended it is very readable, while maintaining the ability to be parsed automatically.

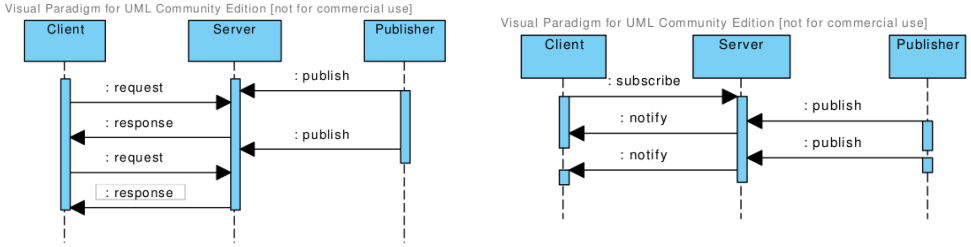
```
{
    "title": "Example Schema",As wireless grew, by 2002, the author of~\cite{pub-sub} st
```

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

2.2 Publish-Subscribe Paradigm

As wireless grew, by 2002, the author of [24] 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 2.5, for the exact same interaction. In 2.5a 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 2.5a, 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 2.5: 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.

2.3 Smart2M Architecture Revision

The Smart2M API is *PT Inovação*'s API to access the data that is stored by the NSCL. It is an M2M Application, and as such has no direct contact with the M2M Gateway, besides through the M2M Service Capabilities and Core Network path (see Figure 2.1).

The details of the architecture are in Appendix A, since they are important to the project, but are not be the focus of this state of the art research.

2.4 M2M Communication Protocols

Given the ascension of M2M demand, protocols started to shift their focus onto M2M scenarios. In this Section, the protocols most commonly used for M2M will be studied.

2.4.1 CoAP

The Constrained Application Protocol (CoAP) [3] 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% [25].

CoAP derives from Representational State Transfer (REST) [26] paradigm, focused for the use in constrained networks and nodes in M2M applications [12]. REST architectures conventionally consist of clients and servers, where the clients perform operations on resources stored on a server by means of request and responses exchanges. There are four types of requests for the clients:

GET Retrieves the content of an existing resource;

POST Creates an new resource;

PUT Changes or updates the content of an existing resource;

DELETE Deletes/removes an existing resource.

To identify resources Uniform Resource Identifier (URI) [27] are used, just like in HTTP [28]. Also CoAP easily translates 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 [29]. This allows for compatibility with existing systems.

In HTTP, transactions are always client-initiated, and the client must perform GET operations again and again (polling) if it wants to stay up-to-date about a resource's status. This pull model becomes expensive in an environment with limited power, and CoAP uses an asynchronous approach to support pushing information from servers to clients: observation (see Figure 2.6). When performing a GET request, 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. Each such notification message is identical in structure to the response to the initial GET request [29]. This model is an attempt to achieve stateless publish-subscribe communication, ideal for devices with fewer resources.

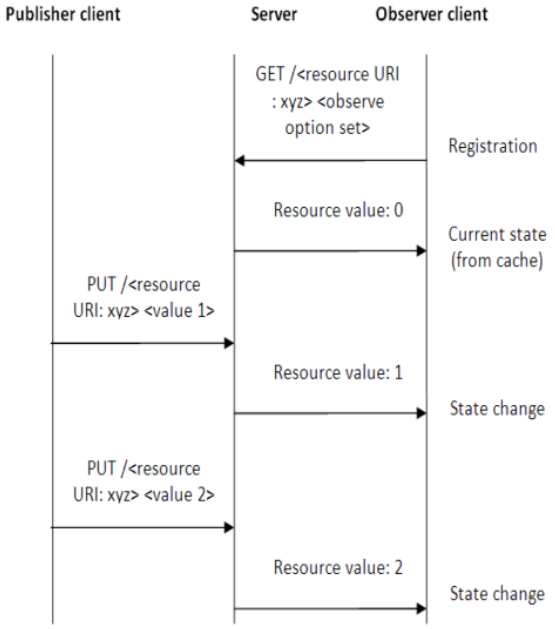


Figure 2.6: CoAP resource-observe. From [12].

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) [30] 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) [31] or Datagram Transport Layer Security (DTLS) [32]. CoAP also supports TCP [33] 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 2.7.

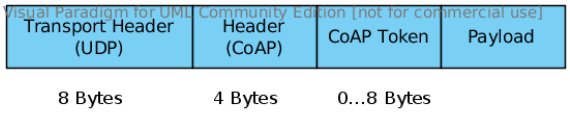


Figure 2.7: CoAP packet size

2.4.2 MQTT

MQ Telemetry Transport (MQTT) [2] 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 [2].

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 [12].

The CoAP packet is illustrated in Figure 2.8.

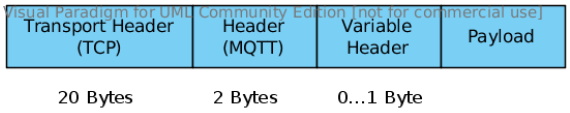


Figure 2.8: MQTT packet size

2.4.3 AMQP

The Advances Message Queue Protocol (AMQP) [34] is an open standard application layer asynchronous protocol for message oriented middleware. It uses the publish-subscribe model defined earlier in Section 2.2

AMQP uses brokers to receive messages from publishers, and route them to consumers. Figure 2.9 gives a high level perspective of the protocol. [35] 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".

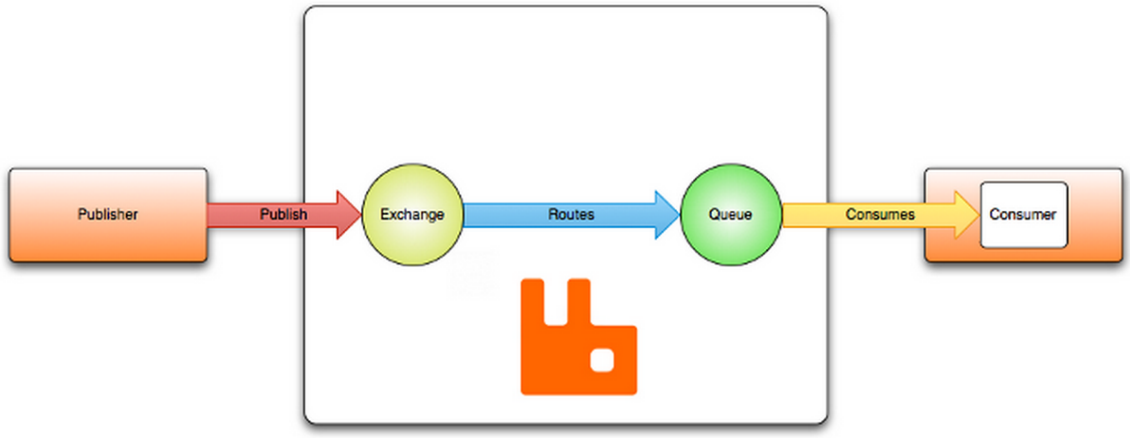


Figure 2.9: AMQP functionality overview. From [35]

In terms of security, AMQP implements SASL [36] and TLS [37]. Figure 2.8 illustrates the AMQP packet size.

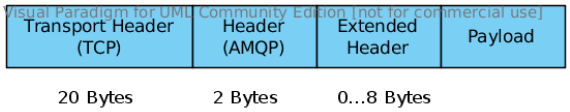


Figure 2.10: AMQP packet size

2.5 E-Health application technologies

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

2.5.1 ISO/IEEE 11073

The ISO/IEEE 11073 is the family of standards for medical devices that was originally called IEEE 1073, or just X73 [38]. 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 [39]. In the past decade, the IEEE 11073 has focused on developing Personal Health Devices (PHD) standards, standardizing functions for each of the Open Systems Interconnection (OSI [40]) layers.

Figure 2.11 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.

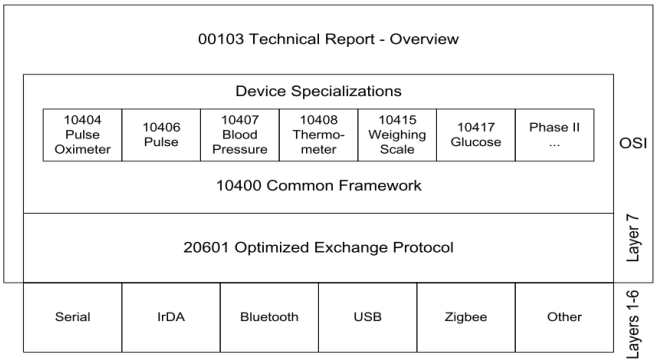


Figure 2.11: IEEE 11073 Framework. From [41].

On Android OS version 4.0 (API 14), a Bluetooth Health API was introduced, facilitating the integration with devices following the IEEE 11073 specifications [42].

2.5.2 Continua Alliance

The Continua Health Alliance [43] 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 2.12 shows how the protocol builds around IEEE 11073 to allow for greater interoperability with a wider range of devices [41].

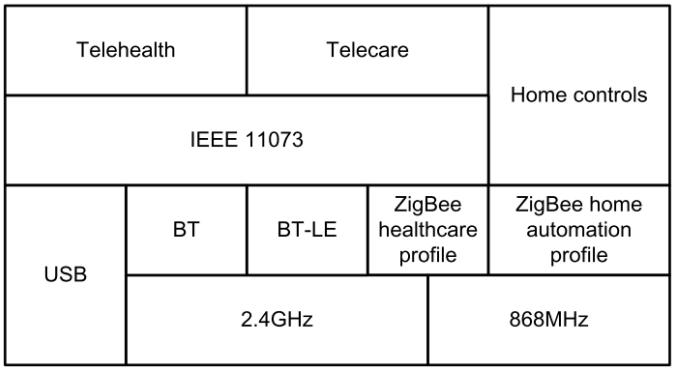


Figure 2.12: Continua Alliance profile of standards. From [41].

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 dissertation.

2.6 Conclusion

Given this Chapter's study, it becomes clear that this dissertation's project will support ETSI's M2M standard by implementing the GSCL for proper mId communication with the NSCL, which

could then be executed by any of the protocols studied. The devices chosen to be integrated in the Gateway will most likely implement a protocol that is based on the IEEE 11073, or even the Continua Alliance guidelines.

Chapter 3

M2M Gateway and Proof-of-Concept Application

This Chapter describes in detail the M2M Gateway and the proof-of-concept application that will demonstrate its potential. In Section 3.1, this dissertation's approach to the issues raised in Section 1.2 is described. Afterward, the mobile M2M Gateway is detailed in Section 3.2. The use cases being considered to study the health-care scenario are disclosed in Section 3.3, along with the sensor procurement which aimed to fulfill its requirements. In Section 3.4, the proof-of-concept application's design is detailed, followed in Section 3.5 by the description of the technologies used in the project. Finally, Section 3.6 states the standards by which the M2M Gateway and proof-of-concept application will be evaluated.

3.1 Approach

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:

  Sphygmomanometer.

  Scale.
- Support API for local access.

Non-functional requirements:

- Support a modular design.

- Multi protocol support:

  CoAP.

  MQTT.

  AMQP (Low Priority).

- ETSI M2M standard compliant.
- Support a modular design.

These requirements will then be demonstrated by the proof-of-concept application, in a health-care scenario that fits the

3.2 M2M Gateway Design

In this section, this dissertation's approach to the problem is explained. First of all the *modus operandi* of the M2M Gateway under development is described. The original architecture that was present in Version 0 of the Gateway (mentioned in section 1.1) has suffered some changes to accommodate all the requirements. This section will reflect the latest approach and decisions made.

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 Broker and the Sensors, as shown in Figure 3.1. 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.

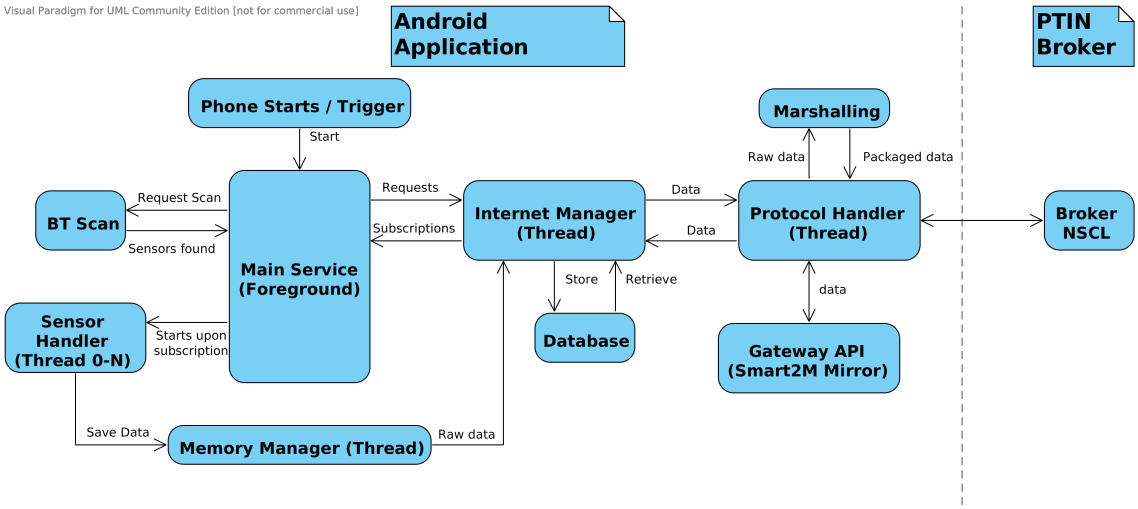


Figure 3.1: Mobile M2M Gateway's Architecture

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 sent to "wake" it up. Upon initialization, it automatically initializes the Internet, Protocol and Memory Managers as well as

GPS and the Marshalling modules 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 Internet Manager is responsible for knowing the status of the network connectivity at all times. Upon creation it also starts the database, so that data is not lost when connection is unavailable. This module will also manage the commands, subscriptions or sensor configurations, so that there is no lost information if a request is made and connection is lost before it is answered.

As for the Protocol Handler, it manages the data to be sent and the messages received. It has default protocol used for the communications, and at start-up it attempts to establish connection via that protocol. If at some point, the phone loses connectivity, when it is regained, it automatically reconnects using the same protocol, and sends order to the Internet Manager to start re-sending the data. Security is always an important issue, and this is handled by the Protocol Manager since, as seen in the previous Chapter, a different security mechanism is required by each protocol. This is also the module responsible by a critical part of the functionality which is the marshalling and de-marshalling, which takes place according to the mId interface that connects the Gateway domain to the NSCL, as seen in Figure 2.2.

At start-up there is still the initialization of the Memory Manager, which besides creating its own thread, triggers the GPS 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 no data is collected before there is a subscription from the NSCL, assuring that no resources are wasted. 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 Internet Manager for dispatch. As mentioned before, a connectivity test is then done determine if the connection has been lost, in which case the data will get stored on the smartphone's internal memory. If there is still connectivity or it has since returned, the data flows to the Protocol Manager for marshalling.

The marshalling process starts with the packaging of the sensors data into a JSON object. The basic output, shown below, has one or more data entries, as there can be multiple sensors' data in each packet. The data object is filled with entries with a timestamp and the sensor reading, which depends on the number of variables of interest, as shown below. It also has a GPS array

with latitude, longitude and accuracy, that only exists once per packet. The structure of the JSON output after this first marshalling operation is as follows[1]:

```
{"data":{
    "sensor_type":"sensor",
    {"timestamp":"12345678","BPM":"89.02"}, -> (1)
    {"timestamp":"12345678","A":"0.123","B":"0.123"}, -> (2)
    {"timestamp":"12345678","Var1":"123",(...),"VarN":"123"}, -> (N)
        }
"gps":  [
    "latitude",
    "longitude",
    "accuracy"
        ]
}
```

The first package is encoded in Base64 and then goes through the second part of the marshalling process involves the creation of the Gateway Service Capability Layer (GSCL) Resource tree, as described in Section 2.1.2.1, with focus on the mId communication.

Then the package is ready to be sent to the Broker, and this can be done with one of the available protocols.

The Gateway is also going to support a local API module, so that the demonstration application can be able to access its functionalities locally, as described in the scenario in Section 3.3. The Smart2M API is described in Section 2.3.

3.3 Use Cases

The overall project design is shown in Figure 3.2. The purpose of this system is to allow the user to take advantage of the smartphone's connectivity abilities to easily use bluetooth capable health-care devices to store information related to himself.

[1] Inside brackets is the number of variables each sensor has

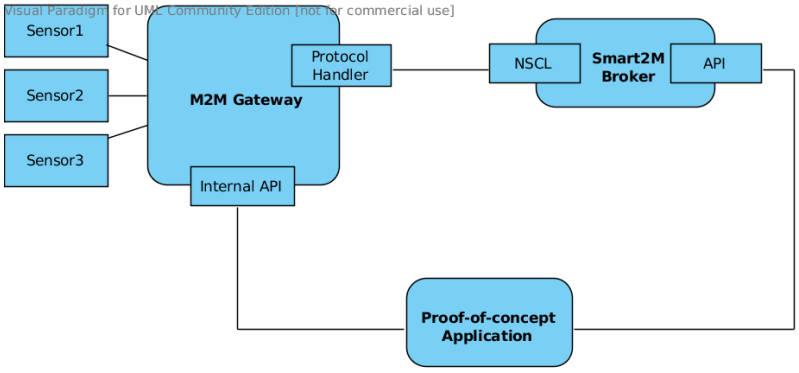


Figure 3.2: Architectural overview of integrated solution

In a scenario of proximity to supported devices, the user will be able, through the application, to use the phone's bluetooth capabilities to search and to connect to supported devices. If the device is not yet paired, the user will be prompted to do so, as it is a requirement to establish the connection.

Once connected, the user will be able to start reading data from that device, and those readings will be shown to him within the application. This will allow the user to choose the reading(s) he desires to store online, in a Smart2M health-care application that provides this service. As soon as the recordings start, the user will be allowed to stop them.

After logging in to the Smart2M API, a token will be stored to allow for local connection to the Gateway. The token will only be used if the proof-of-concept application is in the same smartphone as the M2M Gateway, allowing for a bilateral access capability that aims to save the limited smartphone's resources such as battery and mobile data bandwidth, while being fully transparent to the user.

All the above mechanisms allow the user's data to be stored in Smart2M applications the user logged in to, providing him with the capability to search for any information associated with himself.

The UML Use Case diagram for the scenario just described is present in Figure 3.3, explicitly defining the users possible actions.

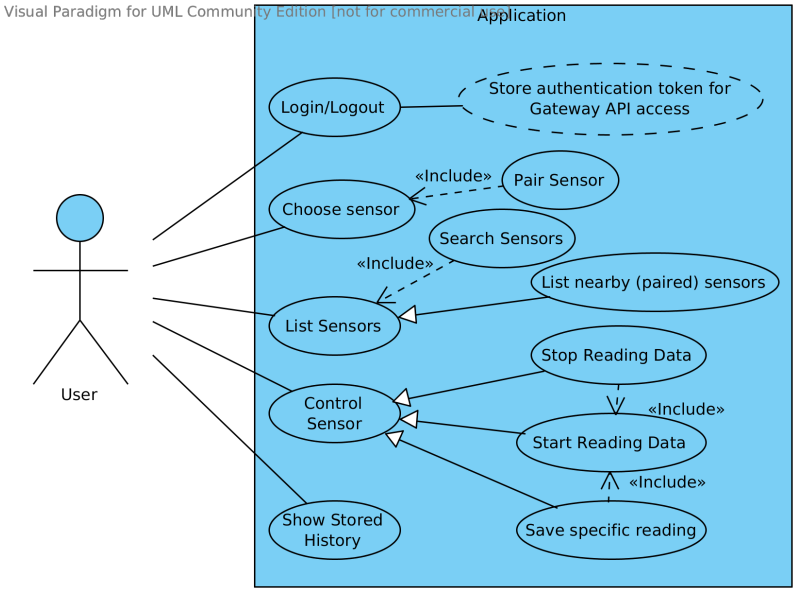


Figure 3.3: Use Case UML 2.0 Diagram for the study scenario

In Section 3.4, the architecture of the Proof-of-Concept Application is going to be detailed, as well as the sequence diagrams that take place between all the modules present in this scenario.

3.3.1 Sensor Procurement

One of the strengths of the M2M Gateway is its modular design, which aims to ease the support of new internal and external sensors and protocols. Given this and since the development is focused on the health care scenario described in Section 3.3, there was the need to procure medical sensors that could be integrated in the Gateway. This integration is necessary so that the sensors can be controlled by the Gateway, and thus remotely by the proof-of-concept application. Discussion of the use-case scenario led to the choice of 2 sensors: a sphygmomanometer and a scale, both bluetooth capable.

The first approach was to search for sensors certified by the Continua Alliance (detailed in Section 2.5.2), and from its extensive list, the ones more suited to our project were selected for price and availability inquiry[2]. After some further research and with the surprisingly scarce information about the protocol and sale points, it was confirmed that this is in fact a proprietary protocol.

Recently the *ForaCare P20 Sphygmomanometer*[3] was chosen, and despite the proprietary protocol, there is a non disclosure agreement in place that provides access to all the necessary details. As for scales, the *Withings WS-30* is a strong possibility since is already available at *Instituto de Telecomunicações*, and supports the project's requirements of Bluetooth connections. A decision has not yet been made since the Android compatibility is still being studied. There should be a decision on this matter in mid February.

[2]The final list is available in Appendix B.

[3]More information in http://www.foracare.com/Blood-Pressure-P20.html.

3.4 Proof-of-concept Application

The demonstration application is still in early development, since it needs a functional Gateway to operate, but the general architecture is detailed in Figure 3.4.

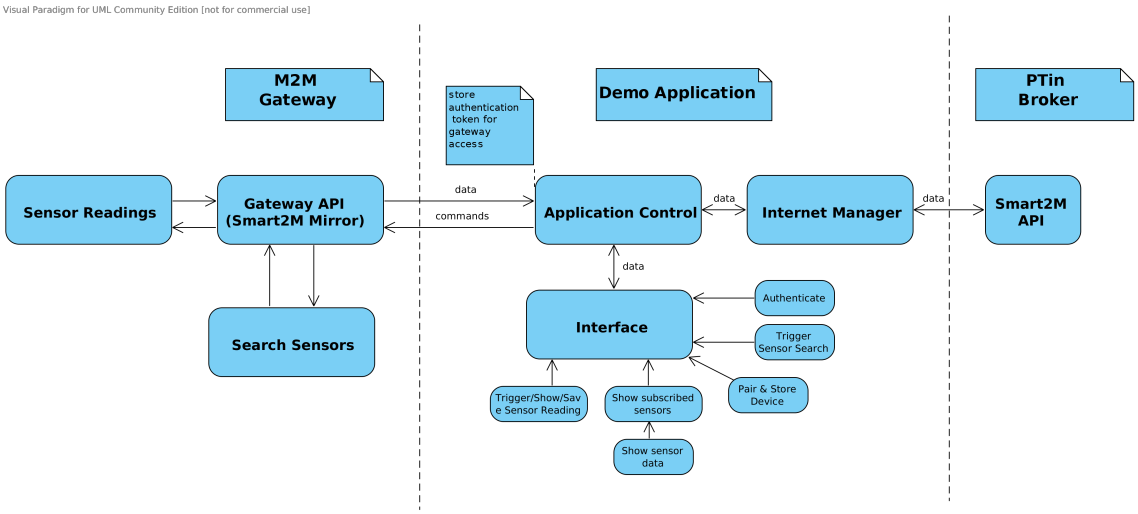


Figure 3.4: Demonstration Application Architecture

The development so far has been focused on detailing the message exchanges in UML 2.0, allowing for easier iterations of the concept, before coding is involved. The in-depth message diagrams are available in Appendix C, along with an explanation of their meaning.

3.5 Technologies

First off, the mobile M2M Gateway is being built as an Android Operating System application. This choice was made given the open source nature of the operating system, which provides the easy integration with the external sensors, and because the programming language is in Java, which was already known. The wide range of devices currently running Android OS was also an important factor since if this dissertation yields a good M2M Gateway, there may be interest in further development on existing services.

For this to be accomplished, one needs to have the Android Software Development Kit (SDK)[4] configured. The configuration currently in use takes advantage of the Android Development Tools (ADT)[5] for Eclipse[6], which is an Integrated Development Environment, to use the Android SDK and compile the code. This allows for easier development and testing. As mentioned in Section 3.2, the application will take advantage of the phone's connectivity, namely Bluetooth, cellular (2G/3G) capabilities, and Wi-Fi to communicate with the Broker's NSCL module. This communication is going to be executed using MQTT and CoAP, which have been described in

[4] Available in http://developer.android.com/sdk/index.html?utm_source=weibolife
[5] Available in http://developer.android.com/sdk/installing/installing-adt.html
[6] Available in http://www.eclipse.org/downloads/

Sections 2.4.2 and 2.4.1 respectively. Base64 [44] is going to be used to encode the payload to be sent.

The MQTT communications are done with aid of the paho-mqtt library (Release 0.4.0)[7], which implements the protocol's version 3. This is a very versatile library that provides control over several option for the connection.

The CoAP implementation is being implemented with Californium CoAP framework[8].

The Demonstration Application, it will also be fully developed as a native Android OS application, to take full advantage of the situations where the Gateway and Application are running in the same device.

The sensors chosen in the previous Section may[9] the connection protocol to the devices, proprietary or not will need to be used, depending on the device.

3.6 Evaluation

The work executed during this dissertation is going to be evaluated according to its goals, enumerated in Section 1.3.

To show that the M2M Gateway and proof-of-concept application are fully functional, the following tests will be performed:

- Test communication ability of the mobile M2M Gateway on CoAP and MQTT, connecting to the NSCL.
- Test subscriptions/commands sent to the Gateway, checking if the results are as expected (e.g. Search Sensors).
- Test M2M Gateway network robustness, storing data when the connection breaks for later re-send.
- M2M Gateway ability to be controlled by both NSCL and local API, with aid of the proof-of-concept application.
- Test the proof-of-concept application ability to fulfill the proposed use cases.
- Gather protocol comparison information on the M2M Gateway:

  Battery drain test;

  Network traffic test;

[7] Available in http://www.eclipse.org/paho/
[8] Available in http://people.inf.ethz.ch/mkovatsc/californium.php
[9] The final choice has not been done yet

Chapter 4

Planning

In this Chapter the work to be done during this semester, regarding this dissertation is explained. First off all, the thorough list of upcoming tasks is shown in Section 4.1. Then, the scaling of those tasks, with the accompanying Gantt chart is present in Section 4.2. Then, to finish this Chapter, the Conclusions are present in Section 4.3.

4.1 Tasks

Below, are the future necessary tasks of this project:
For the mobile M2M Gateway:

- Complete CoAP implementation;
- Complete ETSI data-structure implementation;
- Add support for chosen external sensors;
- Complete Database implementation;
- Implement Smart2M mirror API;
- Implement specific Use Case requirements;

For the Demonstration Application:

- Implement basic application architecture;
- Implement communication with the Smart2M broker;
- Implement communication with the Gateway API;
- Implement specific Use Case requirements;

For the M2M protocol comparison, with the same set up:

- Test each protocol's battery drain;
- Test each protocol's network traffic;

- Analyze test results.

To successfully terminate this dissertation, there's also some other tasks needed:

- Develop dissertation's website;
- Update website with project's progress;
- Write Master's Thesis;
- Deliver document version for revision;
- Deliver final document, for jury appreciation;

4.2 Scaling

The first priority of the project is to complete the CoAP implementation and the ETSI data-structure, as they are requirements for the proper communication with the Broker. CoAP is estimated to take one to two weeks opposed to four weeks for the ETSI data-structured, that has a greater complexity. In the first two weeks the base website is also going to be started, and it will continuously be updated for showcasing the progresses in the development throughout the project.

Then, after having the base communication established, the Gateway's database is going to be strengthened, followed by the implementation of the external sensors chosen to fulfill the e-Health Use Cases. This implementation should take two weeks.

After this is completed, it is time to start implementing the base of the Demonstration Application, which should take around 2/3 weeks. Afterward, the development focuses on achieving communication between the Demonstration Application and the M2M Gateway, from both the local API and the Smart2M API. This development is going to be in parallel with the implementation of the local API on the Gateway, given the knowledge needed for the tasks falls under the same scope. It should however, take about 4 weeks, given the complexity of all the tasks involved.

Only after the communication between the Demonstration Application and the M2M Gateway is solid, can the Use Case requirements be implemented and tested. This should take around two weeks.

At this point, the "ecosystem" between the M2M Gateway and the Demo Application should be stable enough to run a few performance tests with each protocol, to establish a comparison between battery drain and bandwidth used. The beginning of these tests should coincide with the start of the write process of the Master's Thesis. The Revised Version should be delivered on 2014/06/16 and the final version on 2014/06/30.

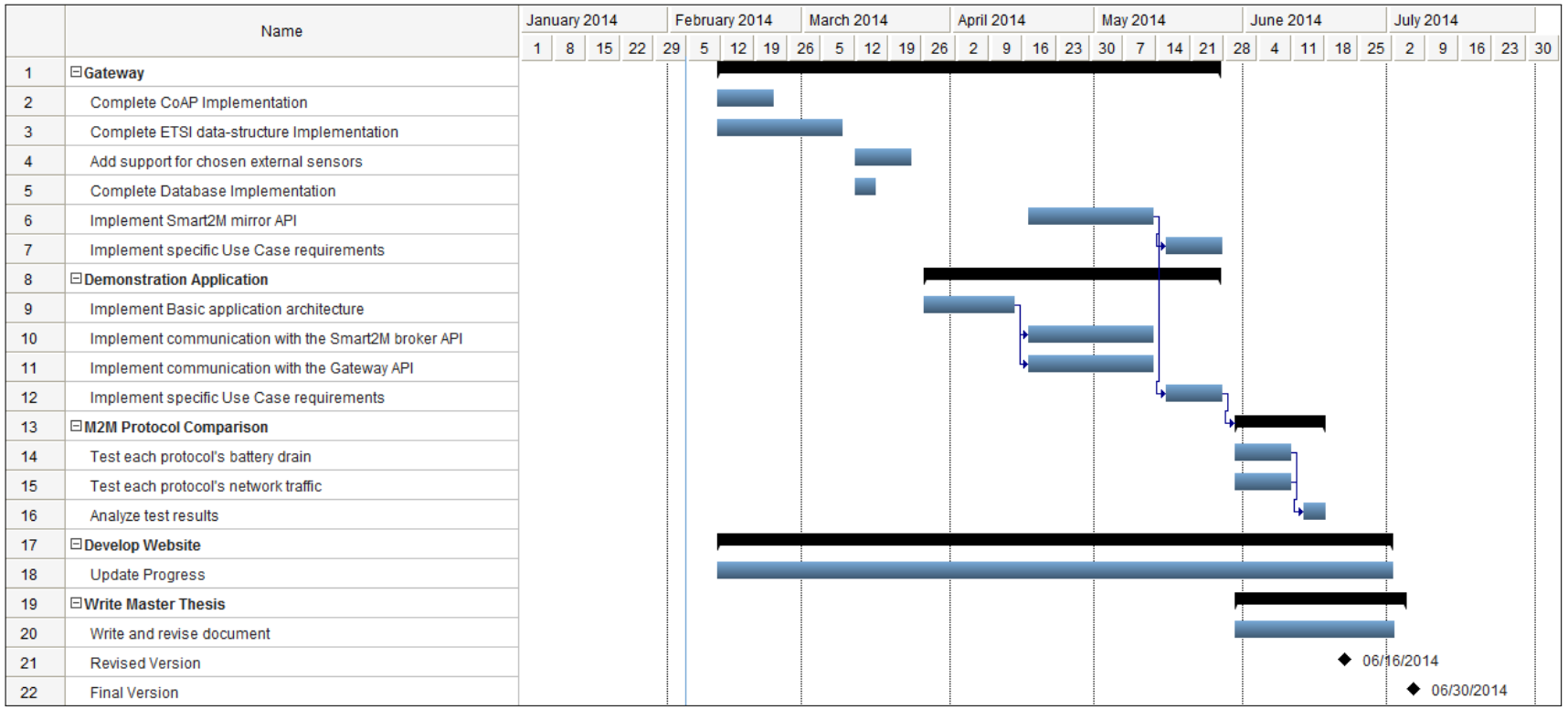


Figure 4.1: Project's Gantt Chart

4.3 Conclusion

Even though there is a high work load associated with this project, there is a solid work plan that aims to allow the project to be completed with success.

Chapter 5

Conclusion

This report as a whole introduces the preparation to the dissertation to be carried on in the next Semester. In Chapter 1 the dissertation's project context is introduced and the motivation and objectives are defined. Then, in Chapter 2 the Bibliographic Revision is carried out, introducing the concepts and technologies needed to understand this dissertation's project. This is followed, in Chapter 3 by the Approach and Methodology taken to tackle the challenges, and by next semester's work-plan in Chapter 4.

Given this, there's a few important notes to take about this project:

- It is an innovative project, which involves the creation of two applications using technologies still in development/standardization that have not yet been widely adopted, and never been ported to a mobile application;
- It is a project that involves close collaboration with Industry, with two active departments (health-care and M2M departments), which shows *PT Inovação*'s interest in this project;
- Given the Innovative nature of the project there are a few risks:

  The project has a high dependency on the platform developed by the Industry partner, and is subject to its delays;

  The ETSI M2M Standard is fairly recent and since it has not yet been widely implemented. This has caused some interpretation issues about the document, which will have to be overcome together with the Industry partner.

This document's conclusion is that even though it is an innovative project with some risks, there is a solid work-plan with feasible tasks, that can be completed successfully by the end of the Semester.

Appendix A

Smart2M API Details

In this Appendix, the architecture of the Smart2M API [45] is going to be detailed, with focus on the important aspects to this project. First of all, the resources are going to be defined, to ease the understanding of the remaining specifications.

User Represents a system user. The user can be a platform administrator, client user or developer.

Feature Features represent a permission, or a group of permissions. These will restrict the users in the scope of a Commercial Service subscribed by the client. A Commercial Service is a group of Features. It can be subscribed with a Service Level, which is just the subscription of a sub-groups of Features within the Commercial Service. Users associated with a Commercial Service will only have access to that sub-group of features, which can represent a restricted account.

Session The API usage is an authenticated service, and this resource exists to allow the credentials and realm of the user's login to be stored and processed correctly.

Device The Device resource allows access to events, capabilities and acting power over the device. It is also possible to subscribe events related to a devices from this resource.

Capability The Capability resource allows for categorization of devices capacities, on the Smart2M platform. A subscription, to receive notifications, can be done to just a specific sub-group of capabilities of a device, without the need to analyze all the data producing capabilities of a device. It is also possible to subscribe capabilities irregardless of the producing device/system.

Application This resource allows for application management, since their creation to their termination in the platform. Through this resource it is also possible to obtain additional information that can be used, for instance, in contextual subscription of events directed to a specific application.

Token This entity supports the authorization mechanisms based in the entities or resources that do not support username/password authentication.

Figure 5.1 represents the API and Resources tree, together with the available RESTful request for each element. Since REST APIs are based in HTTP, the answer success codes are in table 5.1, while the error codes are in table 5.2.

Code	Meaning	Name
200	Undefined Success	Ok
201	Created	Indicates resource successfully created
202	Accepted	Asynchronous session started
204	Indicates the body is purposefully empty	No content
301	Permanently moved	Indicates a new URI was permanently attributed to the resource asked by the client
303	See other	Sent to return results considered optional
304	Not modified	Sent to preserv bandwidth (with conditional GET)
307	Temporary redirection	Indicates a temporary resource was attributed to the resource required by the client

Table 5.1: Smart2M API Success Codes. From [45].

Code	Meaning	Name
400	Bad Request	Unspecified client error
401	Not authorized	Sent when the client sends invalid credentials or no credentials
402	Forbidden	Sebt to deny access to protected resource
404	Not found	Sent when the client tries to interact with an URI the REST API can't find.
405	Method forbidden	Sent when the client tries to interact with an unsupported HTTP method
406	Not accepted	Sent when the client tries to request data in an unsupported media format
409	Conflict	Indicates that the client tried to violate the resource state
412	Pre-condition failed	Indicates that at least one of the pre-conditions failed
415	Unsupported media type	Sent when the client submits data in an unsupported media format

Table 5.2: Smart2M API Error Codes. From [45].

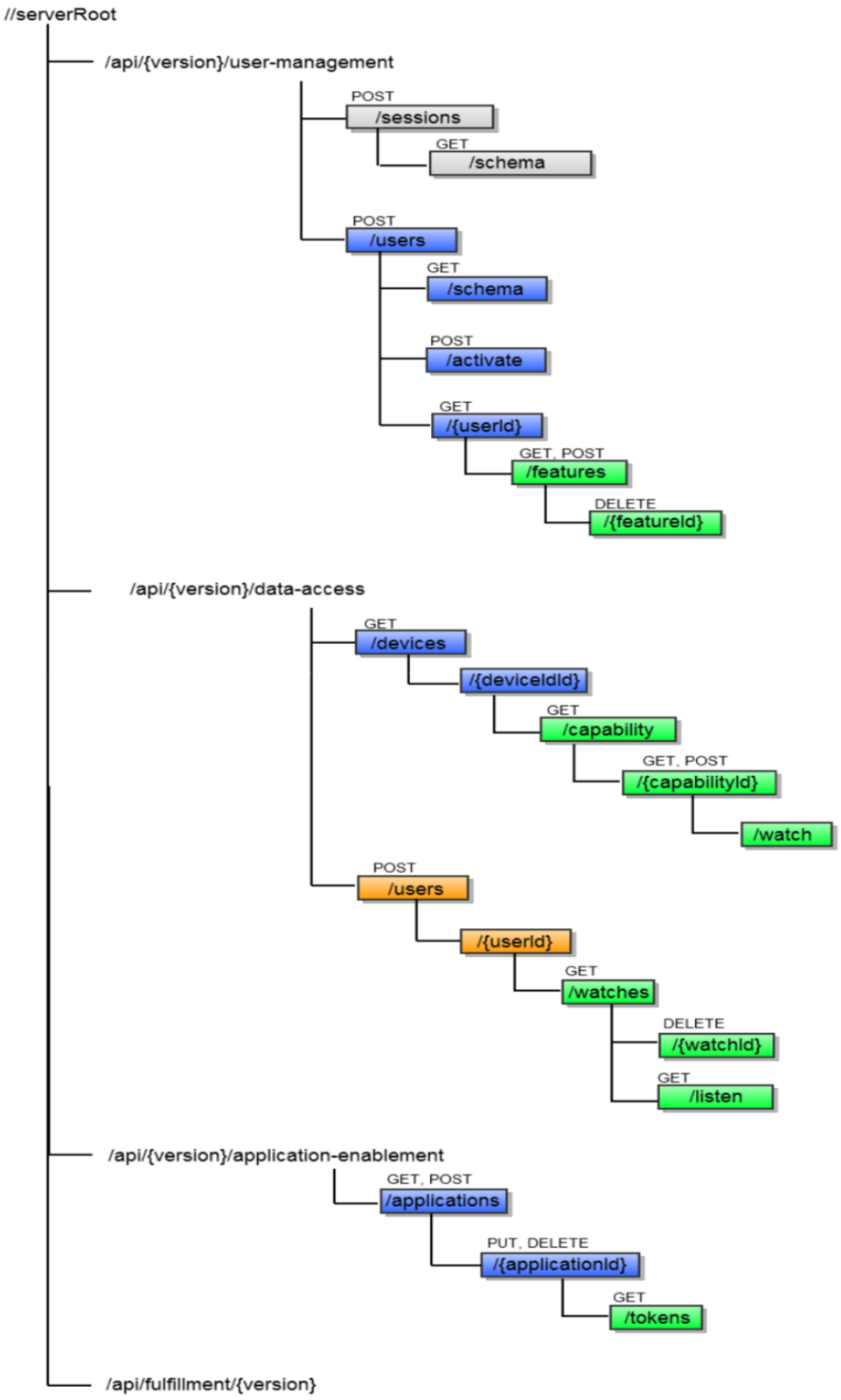


Figure 5.1: Smart2M API and Resources Tree. From [45].

The API is very complete, and this dissertation's use cases don't require implementing the connection to the entire API. The Smart2M API calls required for the project's use cases completion are stated below.

Authentication

The authentication to the Smart2M is done through a the call on Table 5.3.

Resource	URL	Operation	Description
Session	/sessions	@POST	Allows for user identification, creating a session for him

Table 5.3: Authentication: login. From [45].

For the user login to be completed, the POST call must have the following key-value pairs:

```
{
  "password" : "Password",
  "login" : "Login",
  "realm" : "realm"
}
```

Consult Features

To consult the permissions of each user, the call on Table 5.4 must be executed.

Resource	URL	Operation	Description
Feature	/users/{userId}/features	@GET	Returns the list of features (permissions) associated with the userId queried

Table 5.4: Authentication: login. From [45].

To add permissions to a User, fist the call on Table 5.5 must be executed. It yields a JSON answer as:

```
{
  "name" : "featureName",
  "id" : featureId,
  "type" : "FeatureType"
}
```

Then, the call on Table 5.6 allows to add permissions to the User, using the *featureId* of the desired permission.

Resource	URL	Operation	Description
Feature	users"/{userId}/features/ /schema	@GET	Allows to consult a Feature's schema

Table 5.5: Get Feature Schema. From [45].

Resource	URL	Operation	Description
Feature	users/{userId}/features	@POST	Allows the attribution of Features to a User

Table 5.6: Add Features. From [45].

Devices

To list the available devices, the call on Table 5.7 must be executed.

Resource	URL	Operation	Description
Device	/devices/	@GET	Allows for the consultation of the devices to which the user has at least reading permissions

Table 5.7: List Devices. From [45].

To list the device capabilities, the call on Table 5.8 has to be executed.

Resource	URL	Operation	Description
Device	/devices/{deviceId}/ /capabilities	@GET	Allows for the consultation of the capabilities (type of events) that the device supports and/or offers

Table 5.8: Get Device Capabilities. From [45].

Communication synchronously with the devices, to list the events of a capability, Table 5.9's call has to be executed.

Resource	URL	Operation	Description
Capability	/devices/{deviceId}/ /capabilities?from= =timestamp&to= =timestamp& &timeUnit=day& &eventType=2	@GET	Allows the consultation of events the device supports

Table 5.9: List Event. From [45].

To send commands to the device, the Table 5.10's call has to be executed. This is one of the most important functions for this dissertation's project, because it will allow the proof-of-concept application to control the M2M Gateway and its sensors.

Resource	URL	Operation	Description
Capability	devices/{deviceId}/ /capabilities/ /{capabilityId}	@POST	Allows the dispatch of values (commands) for a device's capability

Table 5.10: Send Commands to Device. From [45].

To communicate asynchronously with the devices, Table 5.11's call has to be executed.

Resource	URL	Operation	Description
-	/devices/{deviceId}/ /capabilities	@POST	Creates a new subscription for that capability's events on the Smart2M platform

Table 5.11: Subscribe Event. From [45].

To check the active subscriptions for each user, the call on Table 5.12 has to be executed.

Resource	URL	Operation	Description
Watch	/users/{userId}/watches	@GET	Lists active subscriptions

Table 5.12: List Subscriptions. From [45].

To remove event subscriptions, the call on Table 5.13 has to be executed.

Resource	URL	Operation	Description
-	/users/{userId}/watches/ /{watchId}	@DELETE	Removes the subscription

Table 5.13: List Subscriptions. From [45].

To receive event notifications, the call on Table 5.14 has to be executed. If there are no subscriptions, there return code 204 will be received, signaling the absence of watches "No-body".

Resource	URL	Operation	Description
Event	/users/{userId}/watches/ /listen	@GET	This channel does not guarantee the delivery of events that happened before the connection

Table 5.14: List Subscriptions. From [45].

Appendix B

Continua Health Alliance sensor procurement

The Continua Health Alliance sensor procurement focused on yielding a sphygmomanometer and a scale to be integrated with the mobile M2M Gateway. However, as explained in section 3.3.1, the protocol is proprietary, and as such this procurement had no effect in the final sensor decision. The results below were obtained after a selection from the Continua List, that had around 60 sensors.

Certification Date	Manufacturer	Name	Price	Open API for Android
2010-01-08	OMRON	OMRON Bluetooth® Home Blood Pressure Monitor	Unavailable	Unavailable
2012-07-18	A&D Medical	A&D Medical UA-767PBT-C Blood Pressure Monitor	$158.90 + $16.95 (cuff)	Yes - with signed NDA
2010-04-10	Freescale	Freescale i.MX Linux AHD Reference Platform running LNI HealthLink/OXP-Libraries and Continua certified as a Manager supporting the Blood Pressure Cuf	Unavailable	Unavailable
2011-11-14	TailDoc	FORA D40 Blood Glucose and Blood Pressure Monitor with USB	$270.33	Unavailable

Table 5.15: Sphygmomanometer Procurement

Certification Date	Manufacturer	Name	Price	Open API for Android
2013-02-07	Robert Bosch Healthcare Inc.	WS3000 BT	Unavailable	Unavailable
2010-01-06	OMRON	Body Composition Monitor BF-206BT	Unavailable	Unavailable
2009-01-06	A&D Medical	A&D Medical UC-321PBT-C Weight Scale	$225.61	Yes - with signed NDA
2011-04-13	Stollmann GmbH	BlueMod + P25/G2/IEEE/415	Unavailable	Unavailable

Table 5.16: Scale Procurement

Appendix C

Message Diagrams

As stated in section 3.3, the Demonstration Application features the ability to communicate with the M2M Gateway either through the Smart2M API or the local API present in the Gateway. This appendix shows the message diagrams created, to better visualize the real needs of the M2M Gateway and the Demo Application so that the scenario can be fulfilled.

First off, in figure 5.2, the login is represented. This is the only message that can only be exchanged with the Smart2M API, since this communication yields the API token saved by the application, which is necessary to communicate with the Gateway's API. Then, in figures 5.3 and 5.4, the message exchanges between the Application and the Gateway, through the Smart2M API, are shown. Then, in figures 5.5 and 5.6, the same exchanges are detailed, but through the local Gateway API.

Finally, in figures 5.7 and 5.8, the messages occurring inside the gateway are shown. These are triggered by all requests represented in the previous figures.

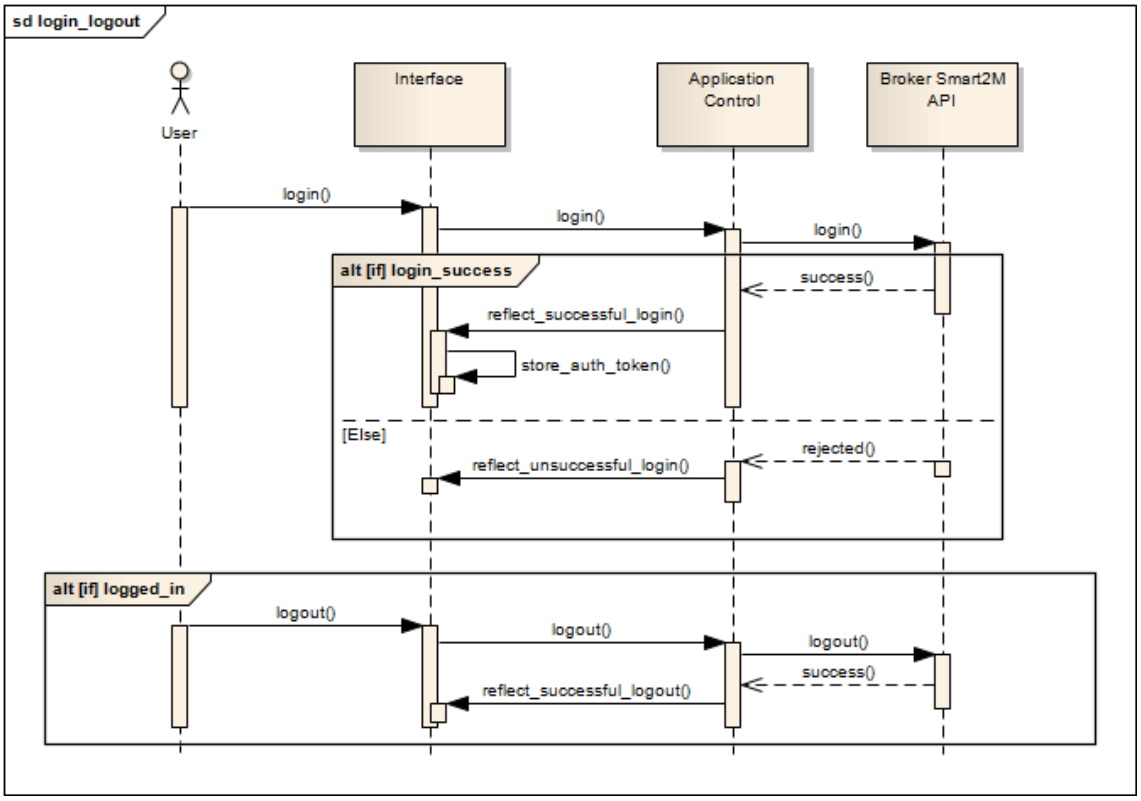


Figure 5.2: Login/Logout Sequence Diagram

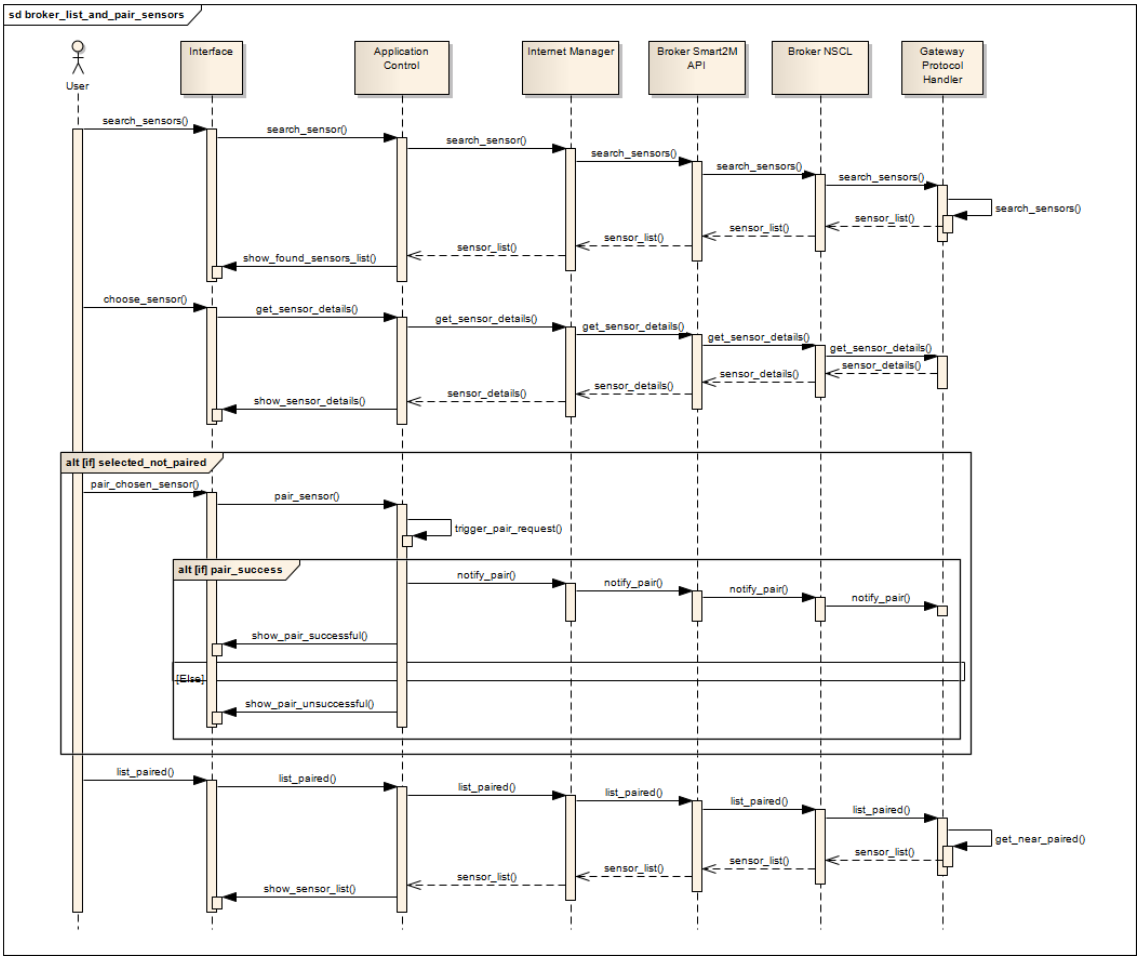


Figure 5.3: Sequence diagram for listing and pairing sensors, through the Smart2M API

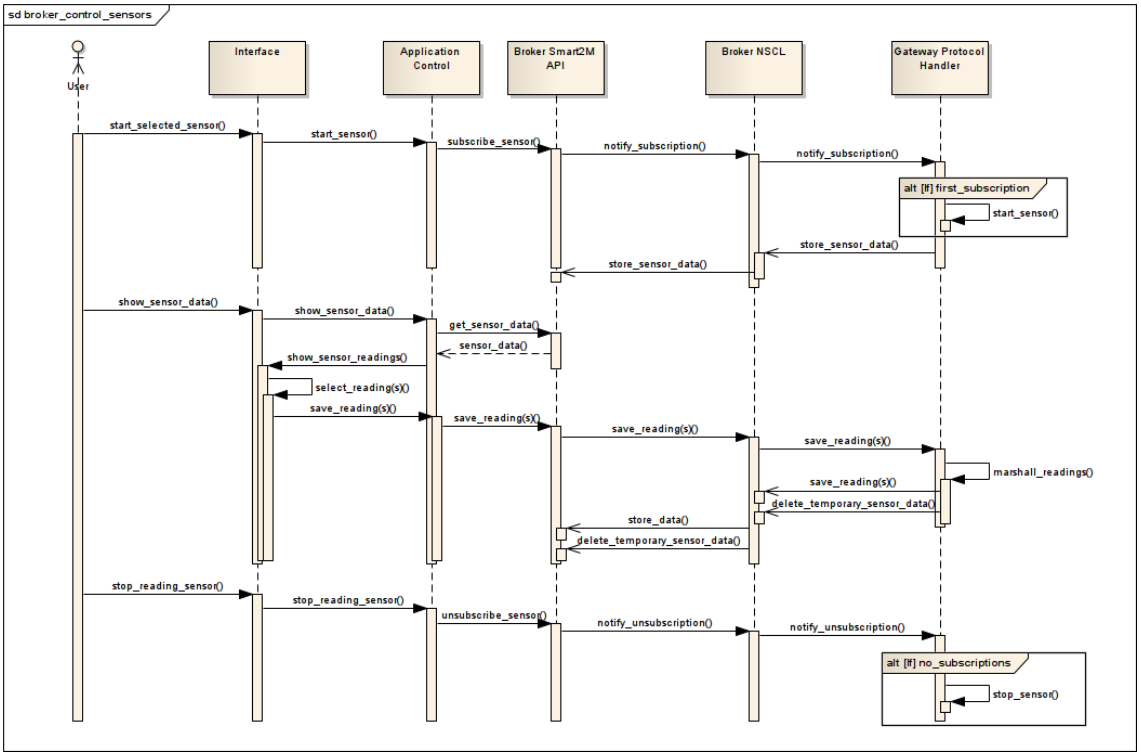


Figure 5.4: Sequence diagram for controlling sensors, through the Smart2M API

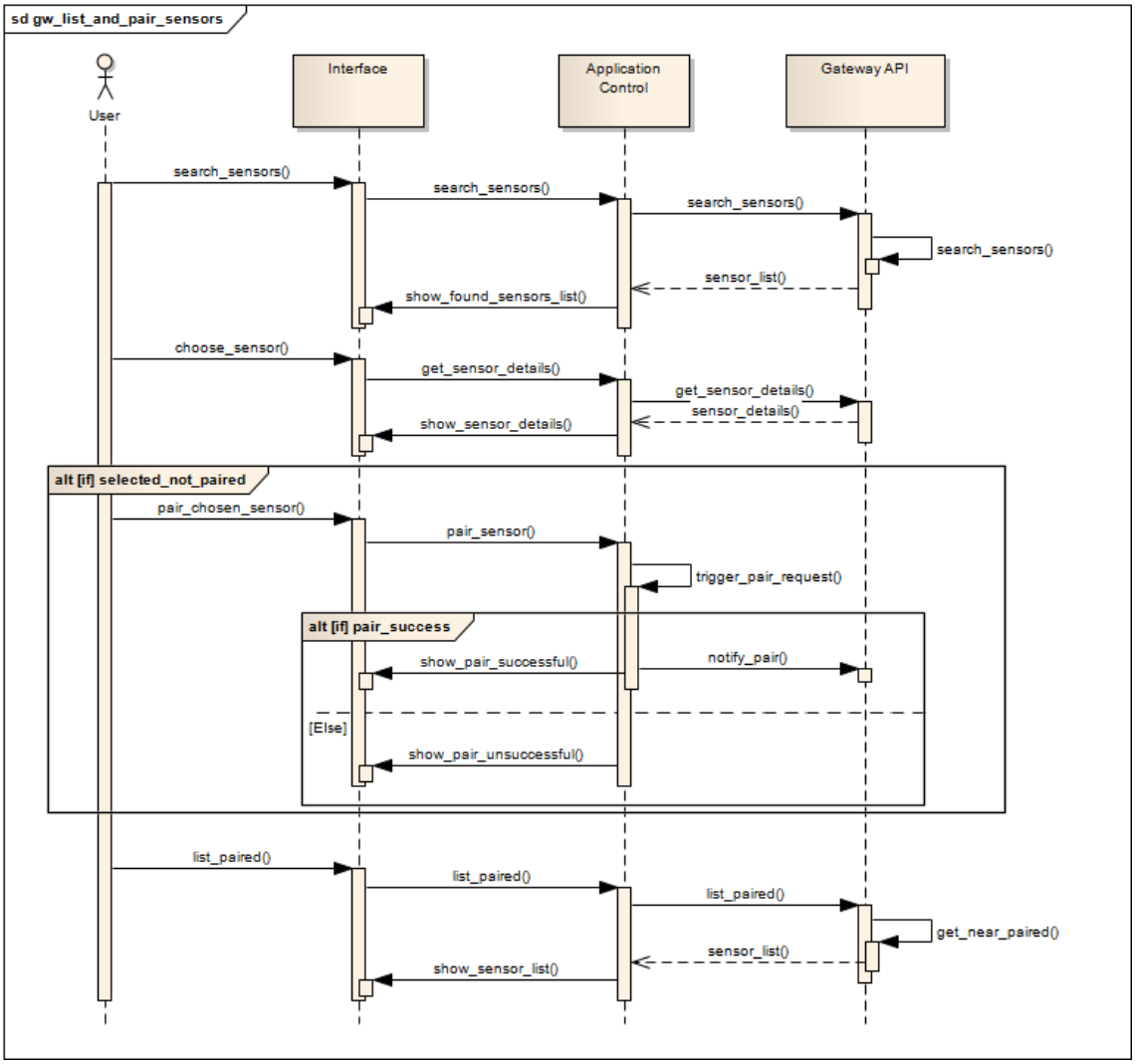


Figure 5.5: Sequence diagram for listing and pairing sensors, through the Gateway API

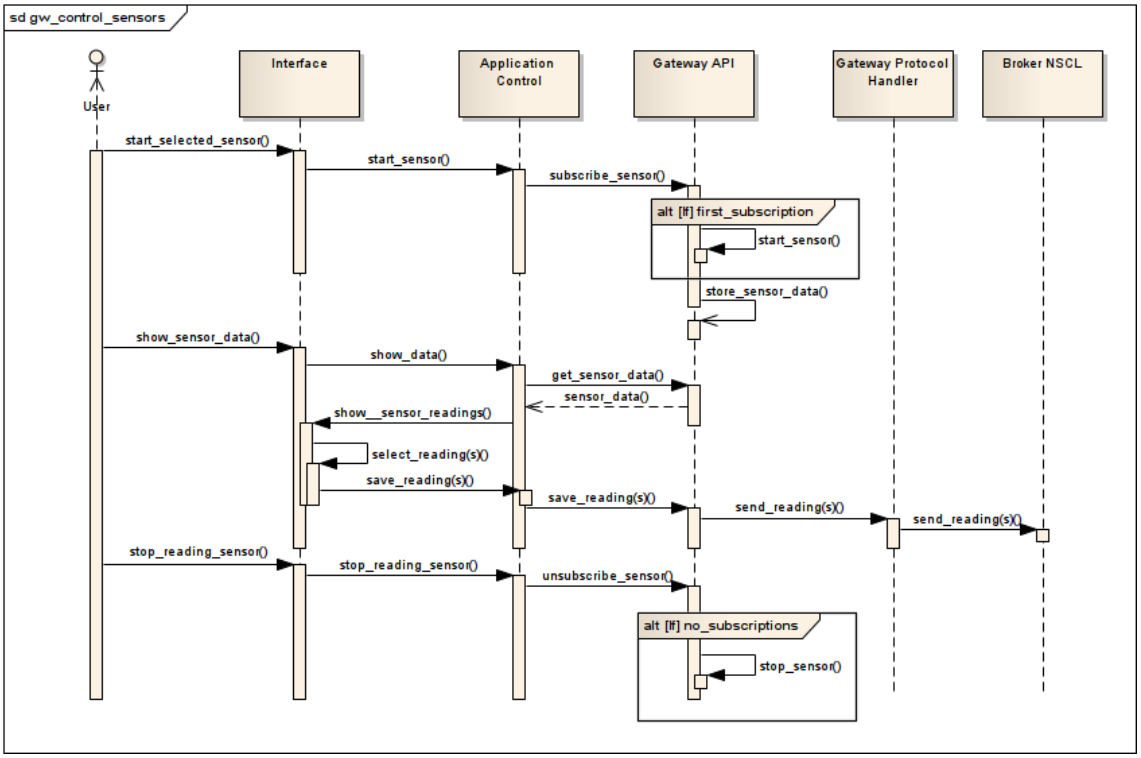


Figure 5.6: Sequence diagram for controlling sensors, through the Gateway API

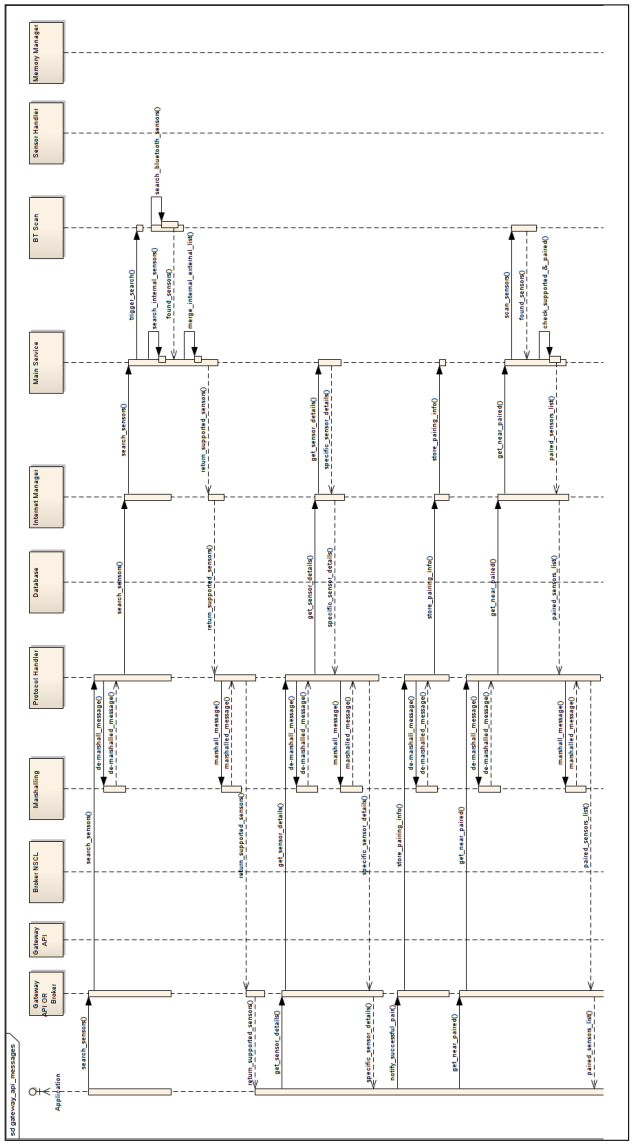


Figure 5.7: Sequence diagrams inside the M2M Gateway (Part 1)

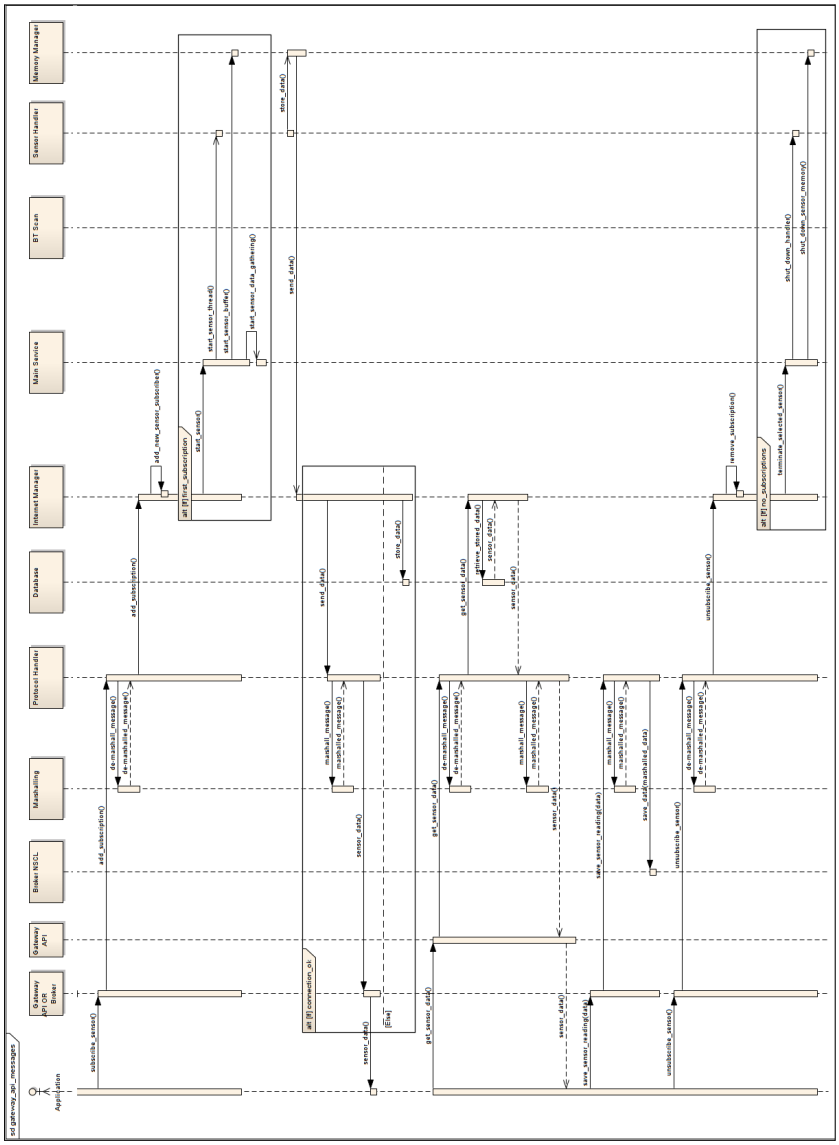


Figure 5.8: Sequence diagrams inside the M2M Gateway (Part 2)

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] S Vinoski. Advanced Message Queuing Protocol. *Internet Computing, IEEE*, pages 87–89, 2006.

[6] GE Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[7] *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.

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

[9] 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.

[10] 3GPP. Service requirements for Machine-Type Communications (MTC); stage 1, release 12. Technical Report TS 22.368 V12.3.0, 3GPP, 2013.

[11] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[12] S Bandyopadhyay and A Bhattacharyya. Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. 2013 International Conference on Computing, Networking and Communications (ICNC 2013), pages 334–340, Tata Consultancy Services, Innovation Lab., Kolkata, India BT - 2013 International Conference on Computing, Networking and Communications (ICNC 2013), 28-31 Jan. 2013, 2013. IEEE.

[13] 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.

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

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

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

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

[18] 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.

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

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

[21] M. Murata E. Whitehead, UC Irvine. XML Media Types. RFC 2376, Fuji Xerox Info. Systems, July 1998.

[22] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, JSON.org, July 2006.

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

[24] 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.

[25] 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.

[26] Robert Battle and Edward Benson. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):61–69, February 2008.

[27] N. Shadbolt, T. Berners-Lee, and W. Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

[28] 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.

[29] 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.

[30] J Postel and ISI. User Datagram Protocol. RFC 768, August 1980.

[31] Suresh Krishnan and Sheila Frankel. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, February 2011.

[32] Eric Rescorla and Nagendra Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, January 2012.

[33] Jon Postel. Transmission Control Protocol. RFC 793, September 1981.

[34] 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.

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

[36] Ed. A. Melnikov, Isode Limited, Ed. K. Zeilenga, and OpenLDAP Foundation. Simple Authentication and Security Layer (SASL). RFC 4422, June 2006.

[37] 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, June 2006.

[38] 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.

[39] 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.

[40] ITU. Information technology - Open Systems Interconnection (OSI). ITU Recommendation X.200.

[41] 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.

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

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

[44] V. Ryan, S. Seligman, R. Lee, and Inc. Sun Microsystems. The Base16, Base32, and Base64 Data Encodings. RFC 4648, Network Working Group, October 2006.

[45] PT Inovação. *Smart2M Application Programming Interfaces, v0.3*, 2013.