Technische Universität Berlin

Institut für Telekommunikationssysteme
Fachgebiet Architektur der Vermittlungsknoten

Fakultät IV
Franklinstrasse 28-29
10587 Berlin
http://www.av.tu-berlin.de

Master Thesis

Design and Implementation of a Device Management Solution for Standard Compliant M2M Platforms

Ranjan Shrestha

Matriculation Number: 0359791
19.1.2015

Supervised by
Prof. Dr. Thomas Magedanz

Assistant Supervisor
Dipl.-Ing. Andreea Ancuta Corici (Fraunhofer FOKUS, Berlin)

FOKUS Institute
Kaiserin-Augusta-Allee 31
10589 Berlin

This dissertation originated in cooperation with the Fraunhofer Institute for Open Communication Systems (FOKUS).

First of all I would like to thank Prof. Dr. Thomas Magedanz, Chair AV, TU Berlin for giving me the opportunity to carry out state of the art research in this field.

A very special thanks to my assistant supervisor Dipl.-Ing. Andreea Ancuta Corici and whole NGNI/OpenMTC team for their excellent supervision and guidance. I am grateful to Ronald Steinke for helping me to translate Abstract page to German.

Furthermore, I would like to thank my parents Associate Prof. Raghubar Shrestha and Anjani Shrestha, my sister Dr. Sanjibani Shrestha and whole Shrestha family for their continuous encouragement and love. Last but not the least, I thank fellow students at EIT ICT Labs Master School for their kind support.

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, 19.1.2015

. . . .

(Signature [Ranjan Shrestha])

Abstract

The Internet of Things(IoT) is a growing and most interesting category of technology of this present and future times. It is a major element of the Future Internet domain and defined as a global network infrastructure containing physical and virtual "things" having proper identities, attributes and able to communicate and exchange information using intelligent interfaces. The IoT composed of physical objects, sensor/controllers/actuators and internet infrastructure that has potential to change the aspect of the economy, society and environment. The scale of the IoT is quite large and predicted to reach 50 billion devices connected to the internet by 2020. The key factors responsible for this rise are due miniaturization of electronic devices, affordability of electronic components due mass production and low cost and de-wireization due efficient wireless technologies[Fel14]. The IoT and M2M system deployment can be seen in industrial management, smart city/home, security and safety, e-health and tele-medicine etc. For the management of wide range of embedded and connected smart devices for M2M communication, a well designed standard is required. Based on ETSI M2M and oneM2M standards, OpenMTC is being developed by NGNI group at Fraunhofer FOKUS, an important middleware for M2M communication.

While facilitating large number of connected devices, it is necessary to monitor, control and exchange information between them. Whether it is related to store device information or controlling devices remotely, Device Management plays an important role. Hence, this thesis focuses on designing an efficient Device Management(DM) service enabler. The DM Server-Client model is based on OMA LightWeight M2M(LWM2M) standard. It features a defined architectural design, a simple and scalable object data model, UDP based Constrained Application Protocol(CoAP) messaging. This well designed standard is particularly interested in managing large and growing category of connected devices called *Constrained Devices.* They are characterized by simple structure, low capacity, low power, low range communication capability. The CoAP is a web transfer protocol, specifically intended for M2M applications and to be used with constrained devices and constrained networks. The CoAP is compared against HTTP by having less header overhead. Hence, LightWeight M2M standard provides a robust solution for Device Management. There are eight standard Management Objects(MO) proposed by OMA LightWeight M2M that characterize the device. Fraunhofer FOKUS is interested to propose a MO *TransportMgmtPolicy* and oneM2M standard also proposes a MO *DeviceCapability* that are implemented in this thesis. An efficient resource model based on LWM2M standard is implemented that manages the MO. A number of proposed DM operations can be performed on the MO and their resources by exchanging CoAP messages.

Zusammenfassung

Das Internet der Dinge (IoT) ist eine wachsende und eine der interessantesten Kategorien der Technik in gegenwärtigen und künftigen Zeiten. Es ist ein wichtiges Element der Future Internet-Domäne und ist definiert als globale Netzwerkinfrastruktur, die physische und virtuelle "Dinge" beinhalten Diese "Dinge" haben geeignete Identitäten und Eigenschaften und sind in der Lage die Kommunikation und den Informationsaustausch mit intelligenten Schnittstellen durchzuführen. Das Internet der Dinge, das aus physischen Objekten, Sensoren/Controllern/Aktoren und der Internet-Infrastruktur besteht, hat das Potenzial den Aspekt der Wirtschaft, Gesellschaft und Umwelt zu verändern. Das Ausmaß des IoT ist recht groß und voraussichtlich werden 50 Milliarden Geräte mit dem Internet 2020 verbunden sein. Ausschlaggebend für diesen Anstieg verantwortlich sind die Erschwinglichkeit von elektronischen Komponenten durch Massenproduktion und die Miniaturisierung von elektronischen Bauelementen sowie die niedrigen Kosten und die Reduzierung drahtgebundener Verbindungen durch effiziente drahtlose Technologien [Fel14]. Das Internet der Dinge und die M2M-Systembereitstellung finden sich im Industriemanagement, in Smart City / Smart Home, in der Sicherheit, in E-Health, in der Telemedizin usw. Für die Verwaltung vieler integrierter und verbundener intelligenter Geräte für M2M-Kommunikation ist ein gut gestalteter Standard erforderlich. Basierend auf den Standards ETSI M2M und oneM2M wurde OpenMTC von der NGNI-Gruppe am Fraunhofer FOKUS entwickelt, einer wichtigen Middleware für die M2M-Kommunikation.

Bei der Bereitstellung einer Vielzahl von angeschlossenen Geräten ist es notwendig, zu überwachen, zu kontrollieren und Informationen zwischen ihnen auszutauschen. Device Management spielt eine wichtige Rolle beim Speichern von Geräteinformationen oder der Steuerung von Geräten aus der Ferne. Daher konzentriert sich die vorliegende Arbeit auf die Gestaltung eines effizienten Device-Management (DM) Service-Enablers. Das DM Server-Client-Modell basiert auf dem OMA Lightweight M2M (LWM2M) Standard. Es verfügt über eine definierte architektonische Gestaltung, ein einfaches und skalierbares Objektdatenmodell und eine UDP Constrained Application Protocol (CoAP)-basierte Nachrichtenvermittlung. Der gut gestaltete Standard behandelt besonders die Verwaltung großer und wachsender Gruppen von angeschlossenen Geräten, die *Constrained Devices* (begrenzte Geräte) genannt werden. Sie zeichnen sich durch eine einfache Struktur, geringe Kapazität, geringem Stromverbrauch und einer geringen Bereich der Kommunikationsfähigkeiten aus. CoAP ist ein Web-Übertragungsprotokoll, das speziell für M2M-Anwendungen bestimmt ist, und bei begrenzten Geräten und Netzwerken verwendet wird. CoAP hat im Gegensatz zu http einen geringeren Header-Overhead. Daher bietet der LightWeight M2M Standard eine robuste Lösung für die Geräteverwaltung. Es gibt acht Standard-Management-Objects (MO), die von OMA Lightweight M2M vorgeschlagen sind und das Gerät kennzeichnen. Fraunhofer FOKUS ist daran interessiert, ein MO *TransportMgmtPolicy* vorzuschlagen und auch der oneM2M-Standard schlägt ein MO *DeviceCapability* vor. Beide sollen in dieser Arbeit implementiert werden werden. Ein effizientes Ressourcenmodell auf Basis des LWM2M-Standards wird implementiert, das die MOs verwaltet. Eine Reihe von vorgeschlagenen DM-Operationen können auf der MO und ihrer Ressourcen durch den Austausch von CoAP-Nachrichten durchgeführt werden.

Contents

List of Figures

List of Tables

1 Introduction

This chapter describes about the role of Internet of Things(IoT) and Machine to Machine(M2M) communication from technical and business prospects and the need of having a robust Device Management enabler. As quoted in [Fel14] , Jim Chase of Texas Instruments offers the definition of IoT:[Cha13] *"The IoT creates an intelligent, invisible network fabric that can be sensed, controlled and programmed. IoT enabled products employ embedded technology that allow them to communicate, directly or indirectly, with each other or the Internet"*. It is estimated that there will be around 25 billion and 50 billion connected devices to the internet by 2015 and 2020[1] respectively which is shown in the figure 1.1 in the form of timeline.

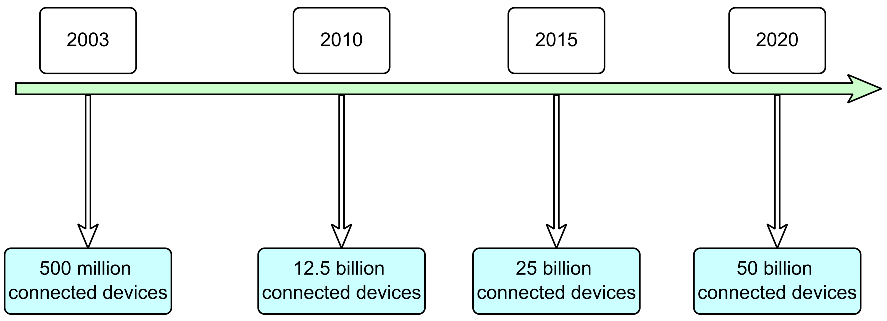

2003
2010
2015
2020
500 million connected devices
12.5 billion connected devices
25 billion connected devices
50 billion connected devices

Figure 1.1: Timeline View of Expected IoT Explosion

According to Kellmereit and Obodovski, three major factors are responsible for the rise of IoT[KO] as quoted in [Fel14]. The advanced fabrication process leads to *Miniaturization* of the electronic devices. The devices become smaller in size, more powerful, energy efficient. The cost of electronics decrease due to mass production and easy process leads to *Affordability.* The wireless communication is gaining popularity and easy to connect feature leads to *De-wireization*[KO]. So, what does IoT consist of? Regarding the composition of IoT, McEwen and Cassimally propose a simple equation[MC] quoted in [Fel14]: *Physical Object + Controllers, Sensors and Actuators + Internet = Internet of Things.* From the equation, it is seen that the spectrum of IoT is quite large. It has a huge impact in future technology and businesses. The McKinsey Global Institute predicts IoT would contribute to global economy from \$2.7 trillion to \$6.2 trillion annually by 2025[Ins13]. The Machine-to-Machine(M2M) is the ability of devices to communicate with each other and share raw or processed data and the services. The

[1] http://share.cisco.com/internet-of-things.html

IoT can be visualized as a superset and considered as a global IP based network that enables M2M communication and its services and further incorporating the concept of Machine to Human(M2H) and Human to Human(H2H) interactions as a whole. More and more intelligent sensor devices having IP connectivity(preferably IPv6) and other devices connected via gateways to reach the internet exchanging huge amount of information. The need of managing these huge number of connected devices provisioned the requirement of a robust Device Management(DM) platform. There exists DM platforms for the remote management of devices from Open Mobile Alliance(OMA)[2] that are already operating such as TR-069, OMA-DM(v1.3, v2.0) and the most recent one OMA Lightweight M2M(v1.0) which is particularly of interest in this thesis.

1.1 Motivation

The need of devices automation give birth to the concept of IoT. The devices are intelligent sensors that sense and produces analog/digital data that can be processed to give some relevant information with meaning. It started with machine to machine interacting and exchanging information. Now, in IoT, in a broader spectrum, it involves any imaginable objects interacting with themselves, possessing IP connectivity to access the mesh networks(e.g. internet) and involves human interactions. With more devices connected to the networks and internet everyday, it is desirable to have a proper bandwidth managed transport mechanism for information transfer and organized, light device management enabler to control all these devices. The million of devices that makeup the M2M system which are part of IoT need to be configured, updated with latest firmware, monitored, queried for data, recovered from error conditions, repairing connectivity issues and all these happen remotely. Hence, an efficient solution for management of devices and applications is required which contains various elements as shown in the figure 1.2.

The possibility of remote monitoring, controlling, machine automation leads to rapid development in the field of IoT and M2M communication. The concept of IoT can be exploited and has a huge research and business prospects in the application fields of Smart City, E-Health, Smart Home, Efficient Energy, Safe Automobiles, Security/Surveillance and etc. The wearable technologies embedded in the smartphone, smart-watch helps to monitor the health of the people. The need of automobiles communicate with each other and the satellites using GPS for navigation and controlling and possibly avoiding road accidents. The need of having energy efficient systems and avoiding possible leakage of energy. Every citizen deserves all kind of security. Upon threat such as earthquake, natural calamities and other accidents, auto-activation of devices is necessary to contact emergency services. Such possibilities can be envisioned in IoT domain.

Related to my Thesis topic on Device Management, there already exists device man-

[2]A standardizing body that develops open standards for mobile phone industry. www.openmobilealliance.org

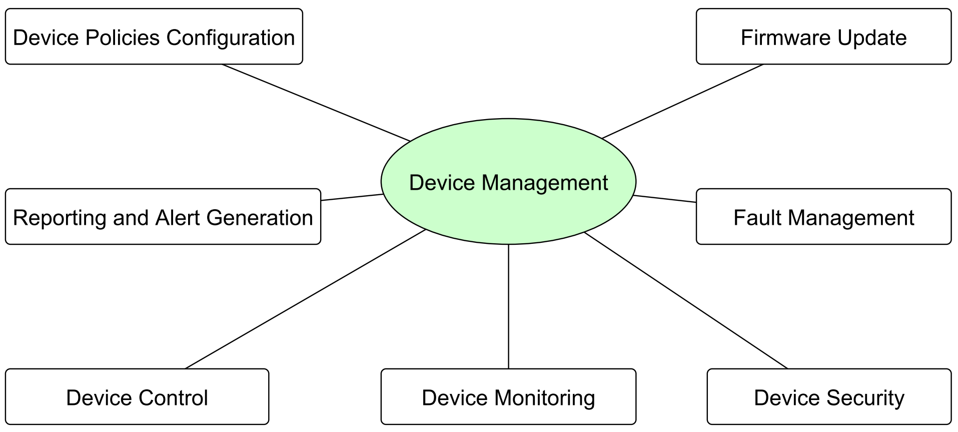


Figure 1.2: Motivational Factors for an Efficient Device Management Solution

agement enabler standards in operation such as OMA DM, TR-069. But, these industry standards for the remote management of devices such as routers and smartphones may not be interesting and efficient for the management of large and growing category of connected devices, often referred to as constrained devices which have limited network bandwidth, computing power and memory, limited battery lifetime and etc. Hence, to address the low cost remote management and service enablement mechanism for constrained devices, a need of simple, lightweight, robust, object oriented complying with RESTful architecture, and designed with state of the art architecture known as OMA Lightweight M2M (LWM2M) is proposed in 2013. This device management enabler would be able to deal with the requirements of constrained devices. This M2M management and service enablement platform has some significant characteristics that makes it special such as extensible and simple object oriented data model, designed for constrained devices, supports efficient data transfer standard called CoAP and RESTful architecture. The CoAP against HTTP is based on UDP which is lightweight and mainly focuses on constrained devices having low bandwidth.

1.2 Objective

The number of connected devices from different manufacturers are increasing rapidly. A better selection of connectivity based on location, time, requirements, a light device management enabler, proper use of intelligent sensors and lots of application fields would help to build an efficient IoT platform. The figure 1.3 shows the objective of having an efficient IoT ecosystem with various involving modules. For the proper organization, management and control of these devices, efficient device management enabler can be the best solution. The traditional device management solutions like OMA DM, TR-069 are most likely to be used for remote router configuration and for mobile devices. Some are used for cellular devices and mostly they are proprietary. But the new issues are

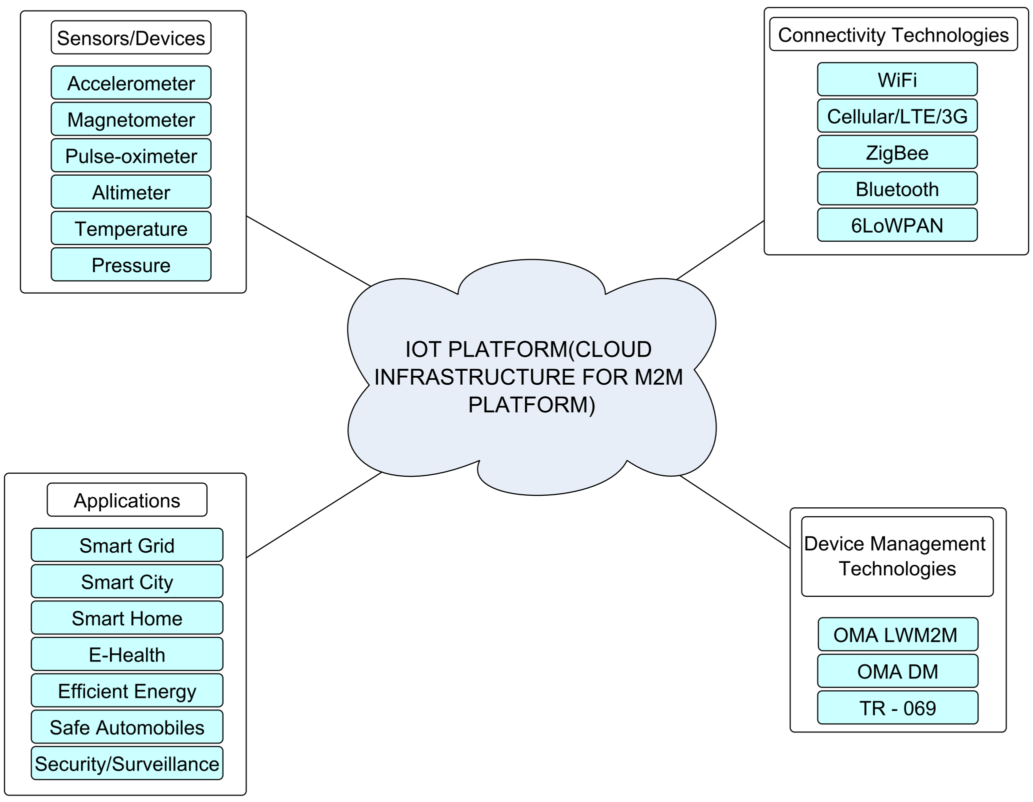


Figure 1.3: Applications of IoT and its Ecosystem

handling the constrained devices. Now-a-days, sensor devices are gaining popularity which are considered as constrained devices because of their behaviour. These are low powered devices, limited battery lifetime, generates less data and hence low bandwidth. Hence, an efficient, lightweight, state of the art device management architecture is required that can address the issues of constrained devices. The solution could be the use of OMA Lightweight M2M. Because of the properties it holds, it can be easily integrated and behaves well in constrained environment. Hence, this thesis describes about the implementation of OMA LWM2M for device management for managing devices. After the implementation of OMA LWM2M, the aim is to find the performance of this technology in constrained environment, to find the robustness, efficiency under different environment conditions.

1.3 Scope

The figure 1.4 shows the bigger picture of the integration of Device Management platform with OpenMTC. The M2M Server and M2M Gateway are the prime components of OpenMTC implementation. The LWM2M DM Server is a plugin to the M2M Server

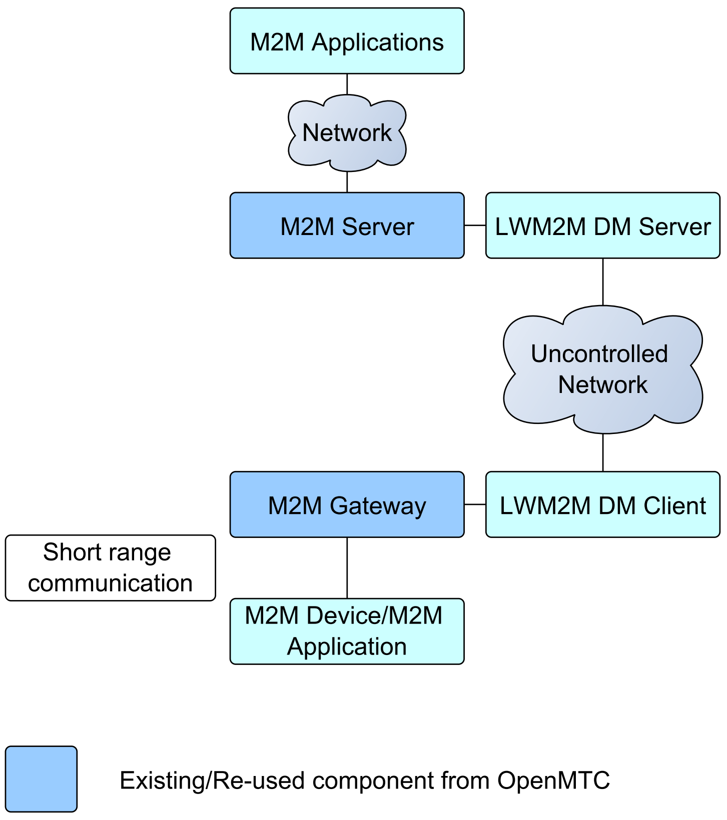


Figure 1.4: High Level Diagram of OMA LWM2M Device Management Integration with OpenMTC

and is responsible for managing management objects of the LWM2M DM Clients that are associated with the M2M Devices. These management objects are responsible for characterizing, monitoring and controlling the M2M devices. The LWM2M DM Client is associated with the M2M Device and responsible for managing it. The M2M Devices interact with the M2M Gateway to access the global network. Usually the interaction is done using short range communication technology. The M2M Network Applications receive processed data coming from M2M Devices/Sensors in form of graph, table etc for visualization and can send commands to control them.

The implementation of LWM2M Device Management is a major spotlight of this thesis. The LWM2M consists of DM Server and DM Client based on its specification. The implementation involves a number of clients connected to the DM server. The DM Client represents the M2M Device and to manage that device, it exposes a number of management objects approved by OMA, few proposed by third parties SDOs and Fraunhofer FOKUS is also willing to propose one management object. An efficient resource model is used to store the resource information of those management objects in both client and server side. The resource model is composed of endpoints, objects, object instances, resources and resource instances. Each resource is uniquely identified by a Resource ID. The DM Server and DM Client exchange requests based on four interfaces such as Bootstrap, Client registration, Device management & Service enablement and Information reporting. All the requests use CoAP over UDP which is a scalable connectionless oriented protocol supported with reliability. The figure 1.5 shows the LWM2M DM Server and LWM2M DM Client interactions.

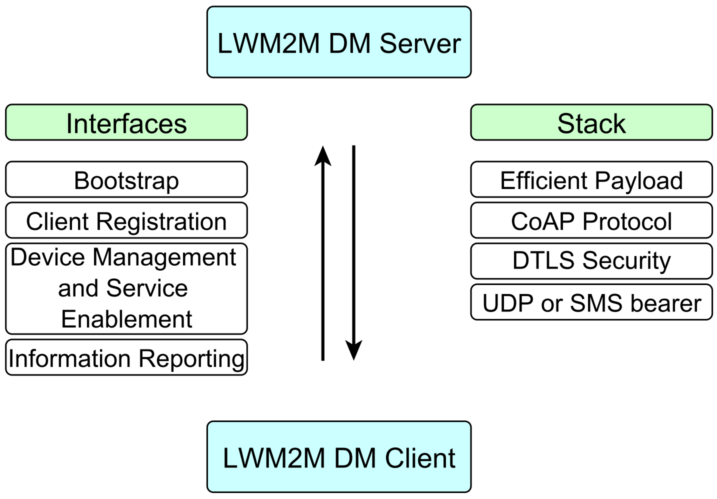


Figure 1.5: Interactions in LWM2M Enabler

To integrate the LWM2M Device Management with OpenMTC , there is a need of adapter that translates the CoAP messages to ETSI M2M compliant messages on Gateways(on DM Client side) and M2M Server (on DM Server side). So, two adapters are also implemented. The figure 1.6 shows the requirement of the adapters for translating messages.

There are number of operations that can happen between a DM Server and a DM Client. First of all, the DM Client registers to the DM Server with all of its registration parameters. The DM Client can send periodic registration updates to the DM Server to inform that the DM client is alive. On request from the Network Application, the DM Server can send requests such as creating objects/resources in the client, writing new values to the resources and their attributes, subscribing to the resource. When there is an update on the DM Client, the DM Server is informed with a notification system about the update of that resource if the resource is subscribed.

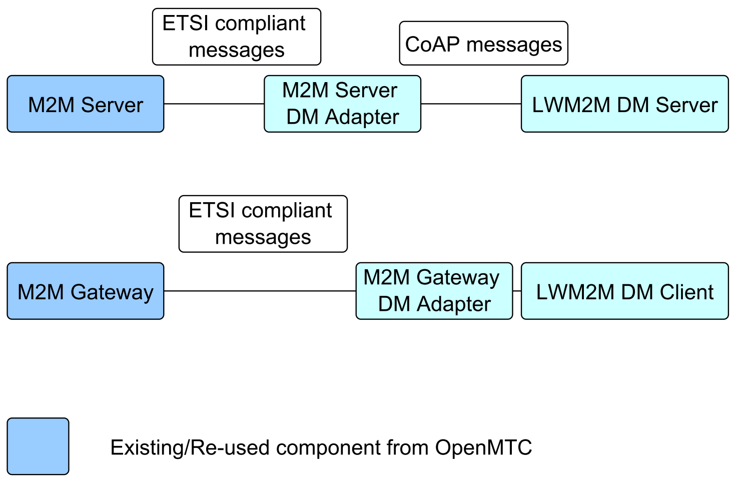


Figure 1.6: Requirement of Adapters for Message Translations

1.4 Outline

This section gives a brief introduction to the main chapters of the work.

This thesis is separated into 7 chapters.

Chapter 2 is related to the 'State of the Art'. The relevant topics related to standards and recent technologies that are closely associated to this thesis are discussed here. An elaborate discussion on LWM2M Device Management, CoAP would be the interesting topics.

Chapter 3 starts with some use cases in smart metering, e-health and telemedicine, connected consumer and the need of Device Management. It is followed by the technical requirements of the Device Management component.

Chapter 4 is related to the proposed design concept of LWM2M Device Management and its integration with OpenMTC middleware platform. The involved components of LWM2M Device Management are minutely discussed.

Chapter 5 describes the implementation of the proposed design concept of LWM2M Device Management. This chapter explains the implementation of involved components.

Chapter 6 is related to evaluation and performance measurements. The evaluation setup is discussed and graph plots are used to measure various elements for performance of the implemented components. The scalability of the system is discussed with the support of graph plots.

Chapter 7 is related to summary of the thesis work. This chapter summarizes the overall work and the role of Device Management. It is followed by some discussion on the targeted group and some future work.

2 State of the Art

The goal of this chapter is to introduce the technologies and standards that are directly or indirectly related to designing and implementing Device Management in M2M platform and Open Machine Type Communication(OpenMTC)[Fra14]. These standards are base for the development of these platforms.

2.1 M2M Standardization Bodies

The standards helps to maintain inter-operability and quality. They help hardwares and softwares communicate and produce an expected result. When many parties are involved for building of a platform, technical specifications and standards are the basic necessities. These standards are developed by standardization bodies. These bodies are described below which are related to this Thesis.

2.1.1 ETSI

The European Telecommunication Standards Institute(ETSI)[1] is an independent, non-profit standardization organization. It's main responsibilities include standardizing tools for Information and Communication Technologies. Founded on 1988 and officially recognized by European Commission, it has over 750 members in 63 countries including hardware manufacturers, network operators, administrators, service providers and research bodies[ETSb].

The ETSI M2M provides a framework for developing services independently of the underlying network. Its architectural design provides a generic set of capabilities for M2M services. It facilitates deployment of vertical domains like e-health, smart home and etc. ETSI M2M adopts RESTful architecture style and standardizes the resource structures residing in the M2M Service Capability Layer(SCL). The M2M application or M2M SCLs exchange information by means of these resources over defined reference points[ETSb][ETSa].

ETSI TS 102 690: This specification[ETS10b] describes the high level architecture as shown in the figure 2.1. Broadly, it consists of a Device and Gateway Domain and a Network Domain.

[1]http://www.etsi.org/

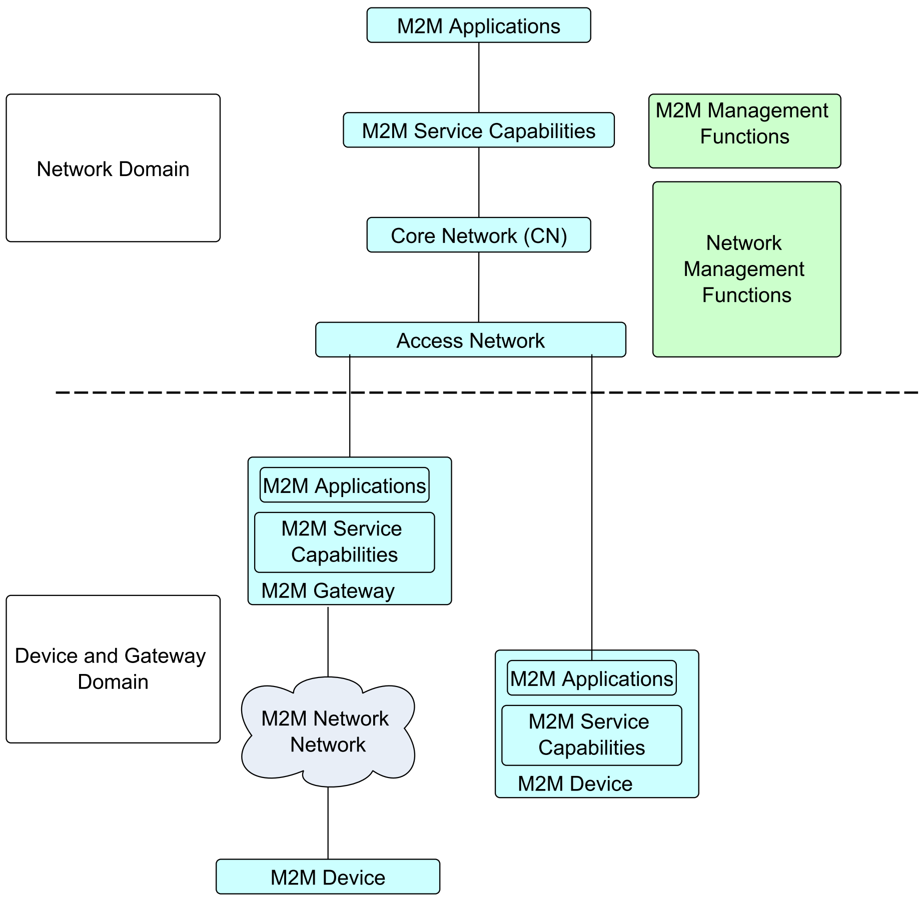


Figure 2.1: ETSI M2M Functional Architecture

The Device and Gateway Domain consists of the following elements[ETS10b]:

M2M Device: A device that runs M2M application using the M2M service capabilities. It can connect the Network Domain via Access Network or via M2M Gateway. In most cases, M2M device can't connect directly, hence, M2M Gateway acts as a proxy to provide services that are hidden from Network Domain. The M2M Devices can be serviced via multiple M2M Gateways.

M2M Area Network: It provides connectivity between M2M Devices and M2M Gateways in a short distance. The examples include Personal Area Network(PAN) technologies such as IEEE[2] 802.15.1, Zigbee[3], Bluetooth[4] and etc.

[2]Institute of Electrical and Electronics Engineers: https://www.ieee.org/
[3]www.zigbee.org
[4]http://electronics.howstuffworks.com/bluetooth.htm

The Network Domain is composed of the following elements[ETS10b]

Access Network: It is an entry point in the Network Domain for M2M devices to access to the Core Network. It includes W-LAN, WiMAX[5], xDSL and etc.

Core Network: It provides service and network control functions.

M2M Service Capabilities: A set of M2M functional interfaces are exposed that are shared by different applications. It simplifies and helps to optimize application development by concealing network complexity.

M2M applications: The applications that run the service functions exposed by M2M service capabilities.

Network Management Functions: It consists of functions required to manage the Access and Core Networks.

M2M Management Functions: It consists of the functions required to manage M2M Service Capabilities in the Network Domain.

ETSI M2M MgmtObjs for Device Management

ETSI M2M offers hierarchical way of storing the information. All the elements are considered as resources which are uniquely addressable entities in the RESTful architecture. It is addressed using URI and can be manipulated using Create Read Update Delete(CRUD) methods. The sub-resource is referenced in the parent resource to access it. The lifetime of the sub-resource is associated and dependent on the lifetime of the parent resource. All the attached devices are represented by the *attachedDevices* resource. It is used to collect management information of all M2M D' devices that are attached to a M2M Gateway. Here, D' represents the attached device. Under each *attachedDevice*, there is *mgmtObjs* resource. The *mgmtObjs* resource contain *mgmtObj* resources. The *mgmtObj* resource holds the management data representing an entity that can be M2M device/gateway. Multiple *mgmtObj* resources can be created that characterize M2M device/gateway and for different management purposes. The *mgmtObjs* model is supported by well defined attributes and parameters as shown in the figure 2.2. The *mgmtCmd* resource is used to model non-RESTful management commands[ETS10c].

The *mgmtObjs* can be created at multiple places of <sclBase>tree[ETS10d].

- <sclBase>/scls/<scl>/mgmtObjs is used to manage service capabilities and for other management functions of single <scl>resource.

[5]http://www.wimaxforum.org

- <sclBase>/scls/<scl>/attachedDevices/<attachedDevice>/mgmtObjs is used for management of each D' device attached to M2M Gateway represented by <scl>resource.

- <sclBase>/scls/<scl>/attachedDevices/mgmtObjs is used to manage M2M Area Network Management of each D' device attached to M2M Gateway.

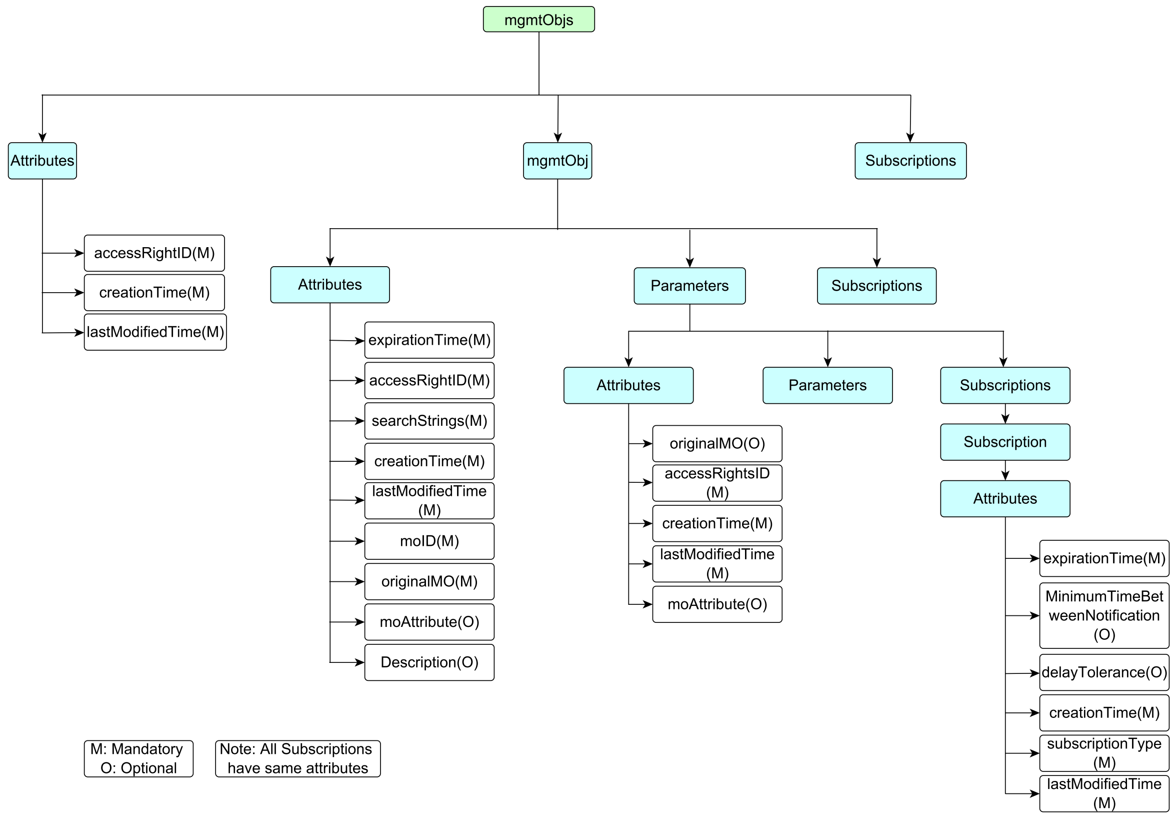


Figure 2.2: ETSI M2M Management Object Model

2.1.2 oneM2M

oneM2M[6] was launched in 2012 by seven Standards Developing Organization(SDO)'s Association of Radio Industries and Businesses(ARIB), Telecommunication Technology Committe(TTC), Alliance for Telecommunications Industry Solutions(ATIS), Telecommunication Industry Association(TIA), China Communications Standards Association(CCSA), ETSI, Telecommunications Technology Association (TTA). Later OMA, Home Gateway Initiatives and Continua Health Alliance joined as partners. By 2014, the number of partners and members reached to 208[one][one14a].

[6]www.onem2m.org

The oneM2M functional architecture[one14c] as shown in the figure 2.3 is split into Field Domain and Infrastructure Domain. It comprises of the following functions.

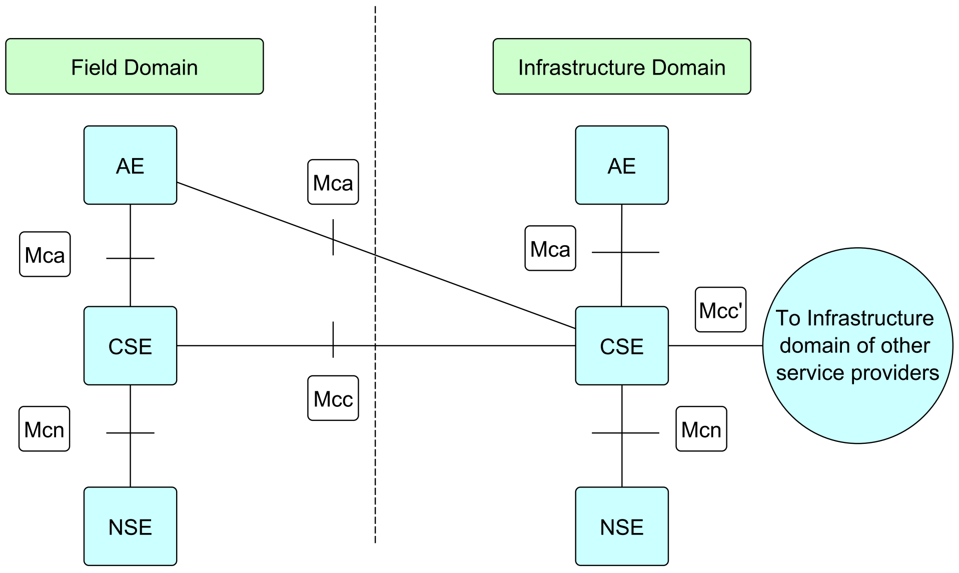


Figure 2.3: oneM2M Functional Architecture

Application Entity(AE): It represents an instance of application logic for end to end M2M solutions. Each AE is equipped with a unique AE-ID. Some examples of AE could be remote blood sugar monitoring application, power metering application.

Common Services Entity(CSE): It represents an instance of set of common functions of M2M environment. Such service functions are exposed to other entities through reference points Mca and Mcc. The Mcn reference point is primarily used to access underlying network service entities. Each CSE is equipped with a unique CSE-ID. Some examples could be Device Management, Data Management, M2M Subscription Management.

Network Services Entity(NSE): This entity provides services from underlying network to CSEs. The examples could be device management, location services, device triggering and etc.

The reference points acts as bridges between different entities in the system. It represents the communication flow between the entities.

Mca Reference Point: It represents the communication flow between AE and CSE cross the Mca reference point. It enables AE to use the services supported by CSE and for CSE to interact with AE.

Mcc Reference Point: It represents the communication flow between the CSEs cross the Mcc reference point. It enables one CSE to use the service of another CSE.

Mcn Reference Point: It represents the communication flow between CSE and NSE cross the Mcn reference point. It enables a CSE to use the services exposed by NSE.

Mcc' Reference Point: It represents the communication flow between two CSEs in infrastructure nodes that are based on oneM2M standard and are placed in different M2M SP domains across the Mcc' reference point. It enables a CSE of an infrastructure node residing in Infrastructure Domain of M2M service provider to communicate with a CSE of another infrastructure node residing in Infrastructure Domain of another M2M service provider to use the supported services.

2.1.2.1 oneM2M support for Device Management

The Device Management(DMG)[one14b] supports management of device capabilities on gateways and M2M devices, also on the devices that reside within M2M Area Network as shown in the figure 2.4.

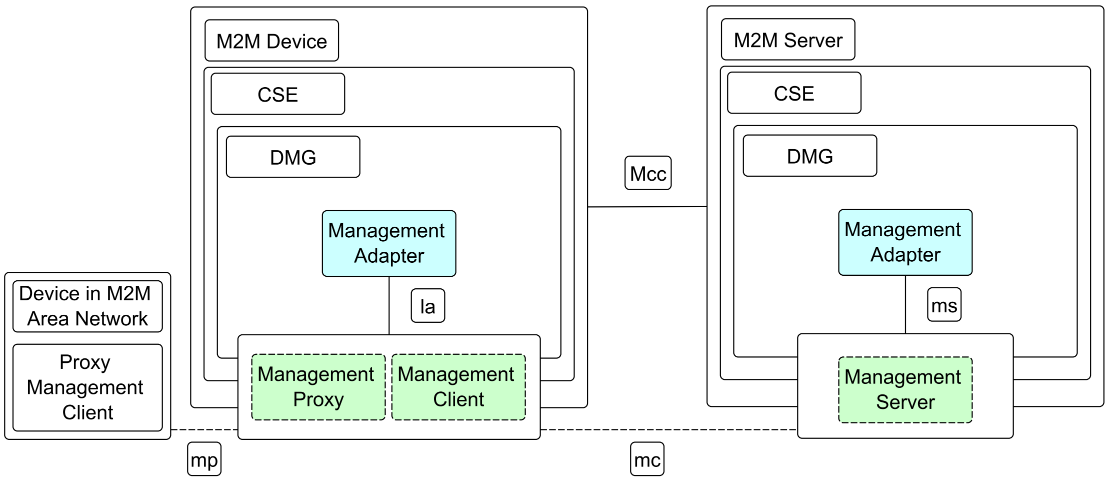


Figure 2.4: OneM2M Device Management Architecture

The DMG can utilize the existing device management technologies like OMA-LWM2M, OMA-DM, in addition to organization of Management Resources across Mcn reference point. A functional component known as Management Adapter is used for both message translations and adaptations. The Management Adapter in DMG of M2M Server (M2M server - DM- MA) performs the adaptation between DMG and Management

Servers, while the Management Adapter in DMG of M2M Device (M2M device - DMG-MA) performs translation and adaptation between DMG and Management Client. The Management Adapter uses the *la* interface to communicate with Management Client for the discovery of external management objects supported by Management Client. The Management Adapter maps between external management objects to the resources. The DMG in the M2M Device can create those resources in the M2M Server and that can be used for managing devices. The *mc* interface bridges Management Server and Management Client and is subject to device management technology being used. The Management Server and Management Client can be implemented as an entity external to the node or they can be implemented as entity embedded within the node. The M2M Device has device management proxy functionality that can interact with Proxy Management Client using *mp* interface[one14b].

2.1.2.2 Management of Device Resources

oneM2M DMG implements the management of devices using well defined resources. There are operations for the creation, update, deletion of the resources. The *mgmtObj* resource contain management data characterizing the M2M device. The oneM2M DMG can further be mapped to external device management technologies like OMA-DM, OMA-LWM2M. Each instance of *mgmtObj* has the following structure as shown in the figure 2.5. The *mgmtObj* has attributes and sub-resources. Also shown in the figure 2.5 is the *mgmtCmd* resource hierarchy. It represents management procedures to model the commands and to execute on the specified resources. It is also comprised of attributes and sub-resources. The number on each attribute/sub-resource indicates the instances that can be created[one14b].

2.1.3 Comparison between ETSI M2M and oneM2M

oneM2M is a global standardization body for machine to machine communication. There are defined names for M2M server as IN, gateways are MN and devices are either ADN or ASN in oneM2M. The interfaces are also named different. The ETSI *mId* would be *Mcc* in oneM2M. Similarly, ETSI *dIa* and *mIa* interfaces are equivalent to *Mca* interface. The *Mcn* interfacing CSE and NSE is new.

oneM2M is an expanded version of ETSI covering other standardizing bodies too in order to bring uniqueness and inter-operability. oneM2M has elaborated the *mgmtObj* resource structure over ETSI with few more attributes like *objectID*, *ResourceID*, *mgmtLink*.

2.2 Device Management Protocol Standards

With the introduction of Internet of Things(IoT), huge number of devices are interconnected to the internet via different network channels. The concept of machine to machine communication is becoming interesting and essential with little or no human intervention. While facilitating the large number of connected devices, it is necessary

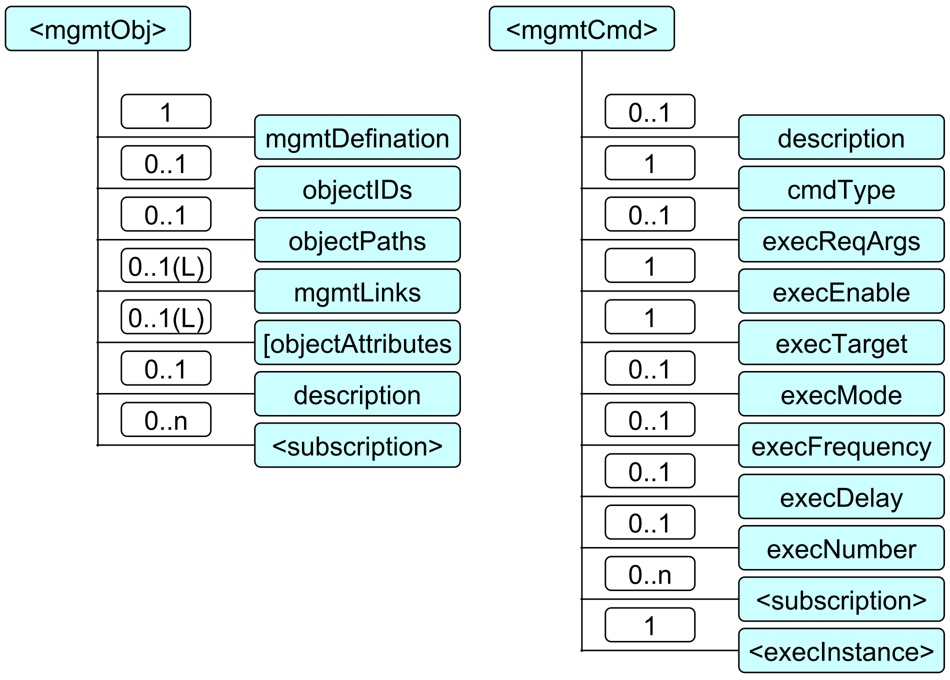


Figure 2.5: Structure of *mgmtObj* and *mgmtCmd* Resources

to keep track, control and exchange information between them. Whether it is related to storing device information or controlling/handling the devices remotely, Device Management plays a major role.

Device Management refers to the management of device configurations and other managed objects of devices. It includes setting initial configuration of the devices, updates of information in the devices, retrieval of device information, execution of commands on devices and etc.

The Open Mobile Alliance[7] is a pioneer in Device Management. The OMA Device Management Working Group specifies the protocols and mechanisms to achieve the management of mobile and connected devices. The list of active OMA Members[8] in Device Management includes Alcatel-Lucent, China Mobile, Ericsson, Gemalto N.V., Intel, Interop Technologies, LG Electronics Inc., NEC, Oberthur Technologies, Red Bend Software, Samsung, Sony Mobile Communications AB, Telecom Italia S.p.A, Vodafone and Sensinode Ltd.

[7]http://openmobilealliance.org/technology
[8]http://openmobilealliance.org/membership/current-members/

2.2.1 OMA Device Management

OMA Device Management[9] is a protocol specified by OMA DM Working Group and Data Synchronization Working Group. The OMA DM is a successful standard for connected device management, preferred by customers and used in millions of devices in this field. The release version 1.2 addresses the protocol and mechanism that allows third parties to carry out procedures like configuring devices on behalf of the end user. The third parties would indicate operators, service providers etc. With the help of DM, third party can remotely set parameters, do troubleshooting servicing, install and upgrade software. It mainly consists of the three parts in version 1.2 as quoted in version 2.0[OMA13b]:

Protocol and Mechanisms: The protocol used between a management server and a device.
Data Model: The model for hierarchical storage of device information.
Policy: The policy decides if the particular object can be accessed in the device.

The design of the architecture follows OMA architecture principle of Network Technology Independence by separating bearer-neutral requirements from bearer specific bindings. There are three parts of the object schema that separates the more general and more specific parameters i.e. a top level Management Object which is a bearer neutral, a set of bearer specific parameters, sub-trees for exposing vendor specific parameters in version 1.2 as quoted in version 2.0[OMA13b].

In OMA DM version 1.3[OMA12], it reused the same architecture from version 1.2. Additionally, it introduced new notification and transport protocol and a new DM server to DM server interface.

In OMA DM version 2.0[OMA13b], it reused the Management Objects designed for version 1.3. This new version integrated new client-server protocol, simplified transaction model of DM command and introduced web based user interaction using SHOW command and management data delivery using HTTP.

The OMA DM 2.0 protocol runs by launching a DM session. DM sessions are always initiated by DM client, but, it can be triggered by DM server using DM notification to the DM client. The DM server can send DM commands and receives responses from the DM client. The DM client informs the DM server about all the activities taking place in the device via Generic Alerts. The DM server can terminate the DM session by sending END command to the DM client[OMA13b].

The management data delivery is carried out using HTTP. The DM server is in synchro-

[9]http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/dm-v1-3

nization to web pages and enables web based interaction with the user. Using HTTP, there are defined DM commands, DM server can send to DM client for retrieving or sending management data to/from the data repository. It supports the notion of DM packages. The sender should wait for response before sending another DM package as the processing can consume unpredictable amount of time. The OMA DM protocol doesn't specify any timeouts between DM packages[OMA13b].

2.2.1.1 OMA DM Architectural Model

The below figure 2.6 shows the Device Management architectural diagram offered by OMA DM 2.0[OMA13b].

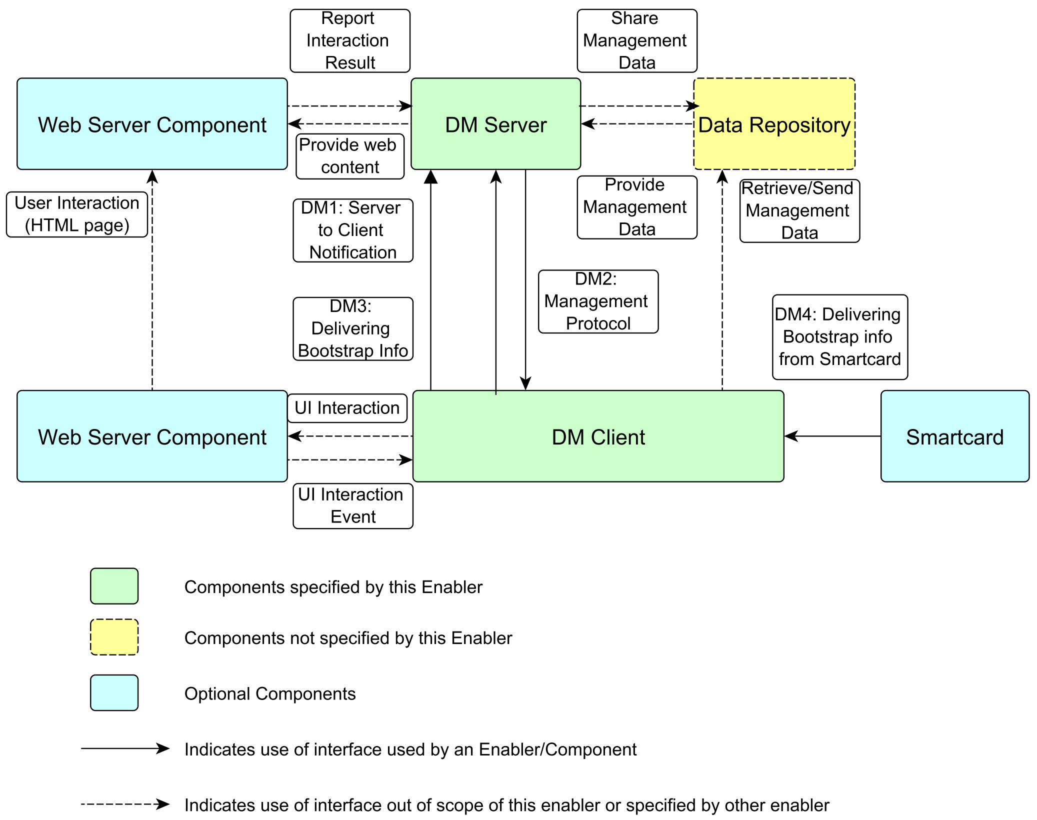

Report Interaction Result
Share Management Data
Web Server Component
DM Server
Data Repository
Provide web content
User Interaction (HTML page)
DM1: Server to Client Notification
Provide Management Data
Retrieve/Send Management Data
DM2: Management Protocol
DM3: Delivering Bootstrap Info
DM4: Delivering Bootstrap info from Smartcard
UI Interaction
Web Server Component
DM Client
Smartcard
UI Interaction Event
Components specified by this Enabler
Components not specified by this Enabler
Optional Components
Indicates use of interface used by an Enabler/Component
Indicates use of interface out of scope of this enabler or specified by other enabler

Figure 2.6: OMA DM 2.0 Device Management Architecture

Functional components and Interfaces

DM Client: It is a component that reflects the device and helps communicate with the other components in the architecture.

DM Server: It manages all the resources of the devices and sends commands to manip-

ulate the devices.
Web Server Component: It is responsible to deliver web contents for UI interaction. It is not specified in the enabler.
Web Browser Component: It is responsible for providing the UI interaction functionality to the DM client. It is not specified in the enabler.
Data Repository: It is a logical server for management of data storage and retrieval using HTTP or other transport methods.
DM-1 Server to Client Notification Interface: It is used by server to send device management notification to the clients to initiate a device management session.
DM-2 Device Management Protocol: It is used by server to send device management commands to clients and in response, clients return status and alert information to the servers. It may use HTTP or HTTPS for communication.
DM-3 Retrieving Bootstrap Information: It provides interface to retrieve bootstrap information.
DM-4 Delivering BootStrap Information from Smartcard: The data stored in the smartcard maybe needed for DM Client for bootstrapping. This is one way interface.

2.2.1.2 Hyper Text Transport Protocol

The Hyper Text Transfer Protocol[10] is a standardized application level protocol running over reliable TCP connections. The principle is based on request-response in client-server computing model. The coordination between Internet Engineering Task Force[11](IETF) and World Wide Web Consortium[12](W3C) resulting in series of publication of RFCs, most notably RFC 2616 that defines commonly used HTTP/1.1[HTT].

In client-server model, the client requests a webpage by providing the resource address in form of Universal Resource Locator(URL) and the server responses with the requested page if the resource path is correct.

HTTP defines methods to perform actions on the interested resources. The HTTP/1.0 specification defined GET, POST and HEAD methods and HTTP/1.1 specification added five new methods OPTIONS, PUT, DELETE, TRACE and CONNECT[HTT].

The status codes are an important part of HTTP standard. It describes how the requests are handled by the server. HTTP status codes are primarily divided into five groups to explain request-response in client-server model as Informational 1XX, Successful 2XX, Redirection 3XX, Client Error 4XX and Server Error 5XX.

Persistent Connection: In HTTP/0.9 and 1.0, the active connection is closed after a single request-response. So, each time a request is made, a new TCP connection has to

[10]http://www.w3.org/Protocols/rfc2616/rfc2616.html
[11]https://www.ietf.org/
[12]www.w3.org

be established. This results in increase of latency of the overall action and degrades the performance. In HTTP/1.1, a keep-alive mechanism keeps the connection existing and can be re-utilized for more than one request-response. Such persistent connection reduce the latency as there is no need of a new TCP three way handshake connection after a first request. HTTP/1.1 is more bandwidth optimized over HTTP/1.0. In HTTP/1.1 introduced chunked transfer encoding allowing content to be streamed rather buffered. HTTP pipelining allows simultaneous requests to be sent and reduces lag time as they don't have to wait for the responses. Another improvement is byte serving where server transmits a portion of the resource requested by the client[HTT].

An example to HTTP request-response:

Request:
GET /pub/WWW/TheProject.html HTTP/1.1
HOST: www.w3.org

Response:
HTTP/1.1 301 Moved Permanently
Date: Thu, 23 August 2014 10:44:32 GMT
Server: Apache/2
Location: http://www.w3.org/TheProject.html
Cache-Control: max-age=21600
Expires: Thu, 23 August 2014 18:24:33 GMT
Content Length: 241
Content-Type: text/html; charset=iso-8859-1

```
<! DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently </title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved. <p>
<href="http://www.w3.org/TheProject.html">here</a></p>
</body></html>
```

2.2.2 OMA LightWeight M2M Device Management

OMA LightWeight M2M version 1.0 emerged as a new technical standard for remote management of constrained devices and provides service enablement and application management for them. It is a communication protocol understood by LWM2M clients available in M2M devices and the central LWM2M server. The architecture and design principle of LWM2M protocol is to create a mechanism that can handle constrained devices having low network bandwidth, works well in unreliable and lossy networks, low power, low cost sensor devices with fewer aspects that can be remotely managed[OMA13a].

The RESTful architectural style is used to make the protocol easily understandable. The items to be managed on a remote device are considered as resources. Each resource is equipped with an address and identified using the URL. The similar resources are grouped together and managed by objects. Hence, LWM2M standard defines an organized, efficient, scalable and simple object model. The LWM2M client-server interaction is done using the underlying data transfer protocol over UDP or SMS known as CoAP. The CoAP is standardized by IETF Constrained RESTful Environment(CoRE)[IET12] working group as a variation to HTTP. The main goal of CoRE Working Group is to keep small message overhead, support multicast, data fragmentation and a simple protocol for M2M communication. Hence, the CoAP is optimized for communication in constrained environment. The LWM2M specification comes with a set of predefined objects and resources that can be easily handled during client-server interaction. The set of objects is extensible as organizations can define and propose the need of new objects that could be important for a robust device management platform[OMA13a].

The first release of OMA LWM2M standard specifies an initial set of objects for device management[OMA13a].

LWM2M Security: It is for handling security aspects between servers and client on the devices.
LWM2M Server: It is for defining data model and functions related to management servers.
Access Control: It is for defining access rights for each data object on the client.
Device: It is for detailing resources on M2M device related to device specific information.
Firmware: It is for detailing resources on M2M device for firmware upgrades.
Location: It is for grouping the resources that provide geographic and positional information of the M2M device.
Connectivity Monitoring: It is for grouping the resources that monitors the connectivity status of the network connections.
Connectivity Statistics: It is for grouping the resources that holds the statistics of the existing network connection.

2.2.2.1 Basic LWM2M Architecture

According to the LWM2M architecture, there are mainly four interfaces as shown the figure 2.7: Device Registration and Discovery, Bootstrap, Device Management and Service Enablement and Information Reporting. The logical operation on these interfaces can be classified as uplink and downlink operations[OMA13a]. The device registration, discovery of all the available devices and notifications upon information changes of subscribed devices are of particular interest as shown in Table 2.1

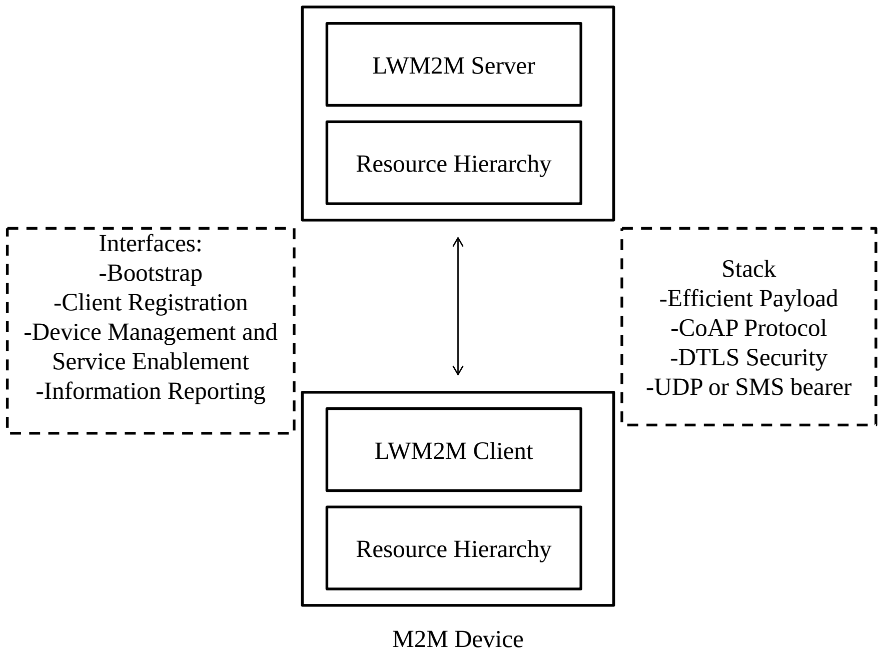


Figure 2.7: Basic LWM2M DM Architecture

Interface	Direction	Logical Operation
Device Discovery and Registration	Uplink	Register, Update and De-register
Device Discovery and Registration	Downlink	Discovery
Device Management and Service Enablement	Downlink	Read, Create, Delete, Write, Execute
Information Reporting	Downlink	Observe, Cancel Observation
Information Reporting	Uplink	Notify

Table 2.1: LWM2M DM logical operations

2.2.2.2 LWM2M DM Resource Model

The figure shows the resource model of the LWM2M Enabler that consists of Objects, Object Instances, Resources and Resource Instances. There are eight predefined objects. Some of the objects can have multiple instances. Each of the object instance has fixed set of well defined resources that describe the client/device and its behavior. Some resources may have resource instances too. Hence, a well defined resource tree can be visualized with the help of these elements. Any of the resources in the resource tree can be easily accessed by using hierarchical path structure like object_id/object_instance_id/resource_id/resource_instance_id[OMA13a].

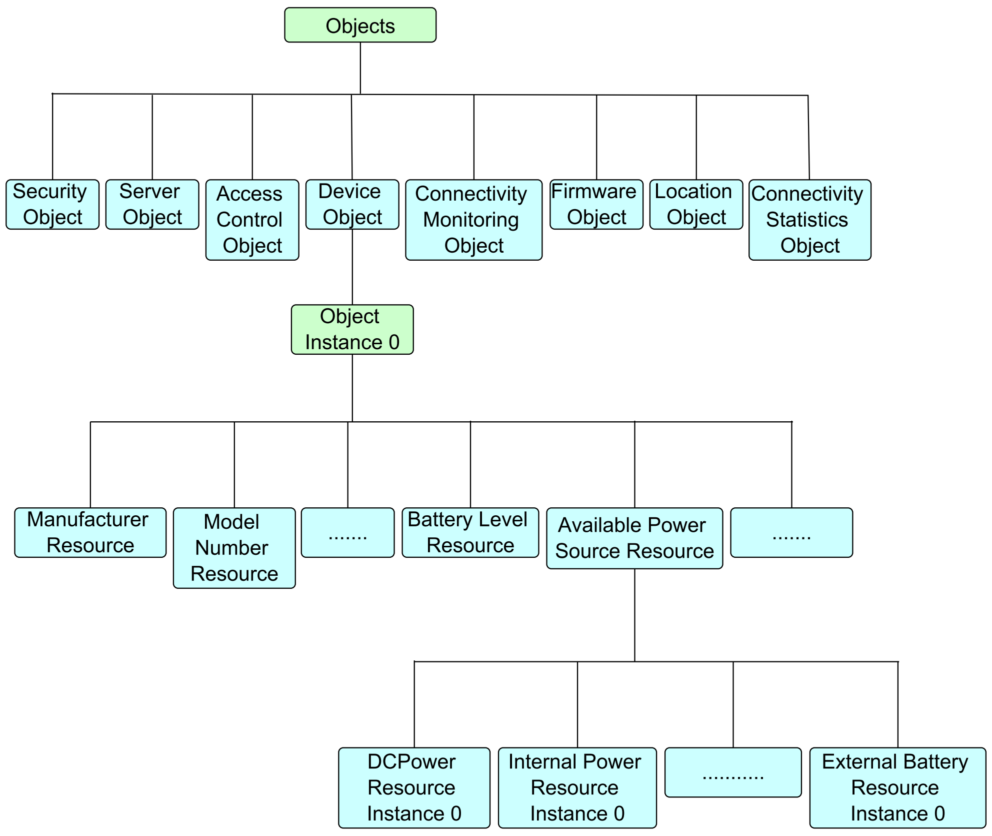


Figure 2.8: LWM2M Device Management Resource Model

2.2.2.3 Proposed New Management Objects

Apart from the management objects proposed in LWM2M Device Management specification, oneM2M in "Management Enablement(OMA)"[one14d] proposed a new management object called *DeviceCapability* dedicated to manage capabilities of devices like USB devices, webcams, actuators and etc. It includes resources describing the *Property* name, *Attached* status of the devices, *Enabled* status, *DenyUserEn* indicating if the user is denied the device capability enablement, *NotifyUser* to indicate if the user should be notified upon the Enabled status change and *opEnable/opDisable* to enable or disable the device capability.

Similarly, Fraunhofer FOKUS also has a new management object called as *TransportManagementPolicy* object. It stores the policy information of the network applications. When the sensor measurement data has to be sent to a desired application, the policy stored in this object is used to retarget to the correct application. It has resources like *IP*, *Port* for retargeting to the desired application and *appidPolicyMapping*, *InitProtocol*, *FinalProtocol* to decide on the policy.

2.2.2.4 Constrained Application Protocol

The Constrained Application Protocol(CoAP) running over UDP is a request-response protocol for machine to machine applications. It is designed to be used with constrained nodes and constrained networks. The constrained nodes refer to low power, low capacity, low computing capability devices. The constrained networks refer to IPv6 over low power wireless personal area network having higher packet error rates and low throughput. It supports the specialized requirements like multicast, very low overhead and simplicity for constrained environments[ZSB14].

The features of CoAP are:

- Fulfills the requirements of constrained behavior of M2M applications
- UDP bindings
- Asynchronous message exchanges
- Low header overhead and parsing complexity
- Security bindings with DTLS

2.2.2.5 Messaging Model in CoAP

The CoAP uses a fixed four bytes header which can be extended to binary options and payload. The message format is applicable to both requests and responses. Each message contains a message ID to avoid duplicates during re-transmissions. The size of message ID is fixed to 16 bits and is randomly generated for each request. Each request-response pair is assigned a unique token ID independently from underlying messages[ZSB14].

The CoAP has optional fields such as URI path, media type, payload and etc that can be included in the same message. To a response of a request, same message ID is used. Here, the response refers to an ACK or a message reply piggybacked with the ACK. CoAP exchange messages asynchronously over a datagram oriented transport such as UDP[ZSB14].

The reliability of messaging is defined by four types of messages. Confirmable(CON), Non-Confirmable(NON), Acknowledgment(ACK), Reset(RST). The confirmable message expects an acknowledgement. This is to confirm that the recipient correctly received the message. A reset message is expected when the server can't process the request. The acknowledgement can be sent separate or piggyback on a response message. The non-confirmable message doesn't expect an acknowledgement. Those messages which are repeatedly sent to the application, such as sensor readings, don't need acknowledgments[ZSB14].

2.2.2.6 Request and Response Model in CoAP

The client requests an action using predefined method code on a resource identified by a URI path on the server. The server replies with a response using a predefined response code. The client requests contain GET, PUT, POST, DELETE methods to manipulate the resources. The response depends on whether the request is confirmable or non-confirmable. The confirmable message expects to receive a response message piggybacked in the acknowledgement. If the message can't be processed, a reset message is sent in a response. If the server isn't able to respond to the confirmable request, it should send an empty acknowledgement message. When the server has the response ready, it sends another confirmable message which is called separate message[ZSB14].

The example showing CoAP confirmable request-response:
Request:
CON [0xbc90]
GET /temperature
(Token 0x71)

Response:
ACK [0xbc90]
2.05 Content
(Token 0x71)
"25.9 C"

2.2.2.7 Observation and Notification in CoAP

The client which desires information regarding a particular resource sends an observe value set in the request to the server. The server responses with an acknowledgement with the observe option value set to confirm that this particular resource is under observation. The server retains a list of observed resources. Hence, whenever there is an update in the resource content, the client is notified of its changes. The notification could be periodic or random and infrequent[Har14]. The observation to a particular resource is set by assigning observe option to zero (observe=0). When it is set, it modifies the GET method so it doesn't only retrieve a representation of the current state of the resource identified by the requested Universal Resource Identifier(URI), but also add the client to the list of observers of the requested resource. The server responses with an acknowledgement containing the present status of the resource. The response contains the same token ID and media type as that of the request is maintained. Also, the observe option is assigned a value which is the time stamp value from the point of starting of observation. Upon changes in the observed requests, notifications are generated as content messages having same token ID and observe option value equivalent to the timestamp value. Notifications have a 2.05 response code (for content) in most cases[Har14]. A simple observation-notification system in CoAP is shown in the figure 2.9.

When there is no need to observe the resource anymore, a request can be sent with observe option value equal to 1(observe=1). This ensures the client is no longer inter-

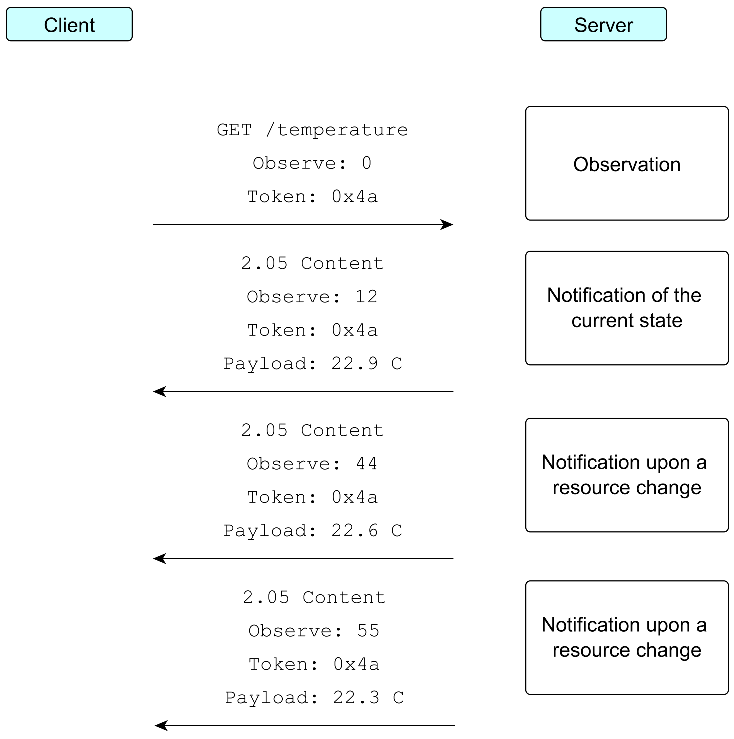


Figure 2.9: Observing Resource as Request and Notifications as Response

ested in that resource updates, hence, the server removes the resource from its observed list[Har14].

2.2.2.8 Comparison between HTTP and CoAP

The CoAP is introduced as a simpler alternative to HTTP for connecting constrained devices. The advantage of using CoAP are it has simpler hardware requirements reducing the cost and power consumption, UDP based transport and communication overhead of the protocol is small and compact. The support of the asynchronous information push when there is a change in the resource, thus, allowing objects to be asleep and hence reducing the power consumption. The CoAP specifies a minimal subset of REST requests, supports resource caching and built-in discovery, provides reliability with re-transmission mechanism. On the other hand, HTTP is usually based on TCP, has a larger header size for reliable transmission and can't deal with the constrained devices. Hence, with the appealing features that supports the constrained devices, CoAP is best suited for M2M communication[TLS].

2.3 Related Work

After the release of OMA LWM2M 1.0 standard in 2013, there have been several attempts to design Device Management based on it. The implementation[13] of this standard in C is done by Intel Corporation. The *lwm2mclient* features four LWM2M objects: Device, Server, FirmwareUpdate and Test. The Erbium's CoAP engine[14] is modified and utilized in this implementation. The lwm2mclient opens UDP port 5683 and registers to the LWM2M Server at 127.0.0.1:5684 which features a basic command line interface. Leshan[15] is a Java implementation of OMA LWM2M by Julien Vermillard. The Eclipse Leshan is based on Eclipse IoT Californium project for CoAP and Datagram Transport Layer Security(DTLS) implementation. It provides a Device Management Server library, Client library, a web user interface displaying a list of connected clients and their resources, and a bootstrapping server[16]. Wakaama[17] is an implementation of OMA LightWeight M2M protocol. It is written in C and contains files to be built with an application. There are two modes: LWM2M_CLIENT_MODE enables LWM2M Client interfaces and LWM2M_SERVER_MODE enables LWM2M Server Interfaces. It provides APIs for server application to send commands to registered LWM2M Clients. On the other hand, on client applications side, Wakaama checks for the syntax and access rights of received commands[18]. After the compilation of the server, it listens on UDP port at 5683. The client defines four LWM2M objects such as Server, Device, FirmwareUpdate and Test (with some definition). The client opens udp port 56830 and tries to register to the server at 127.0.0.1:5683[19].

[13]https://github.com/01org/liblwm2m
[14]http://people.inf.ethz.ch/mkovatsc/erbium.php
[15]https://github.com/jvermillard/leshan
[16]https://projects.eclipse.org/proposals/leshan
[17]https://github.com/eclipse/wakaama
[18]www.eclipse.org/proposals/technology.liblwm2m
[19]https://github.com/eclipse/wakaama

3 Requirements

This section discusses about the requirements for building an effective M2M system and Device Management platform for handling intelligent devices. The requirements can be seen from different use cases in different scenario.

3.1 Use Cases

The Use Cases illustrate practical applicability of M2M system and Device Management in real life scenario. There are different fields where Device Management plays an important role.

1. Smart Metering: Monitor Power Quality Data
2. E-Healthcare and Tele-medicine
3. Connected Consumers

3.1.1 Smart Metering: Monitor Power Quality Data

Smart Metering is an intelligent way of improving the end user energy consumption. It consists of the utility meters(electricity, gas, water, heat meters) that makes customers aware of the energy consumption and helps to make smarter decisions on their usage, provides the distributors with efficient tools for monitoring and managing the energy distribution networks. Among the many use cases of Smart Metering, *Monitor Power Quality Data*[ETS10a] is chosen. It deals with the monitoring of energy from point of production to point of consumption including various factors such as weather, quality of wiring, device malfunction, failing or energy loss in power delivery, operational issues which might affect the quality of supply. It deals with the Smart Metering Information System to provide information on power quality data.

The Distribution Network Operator request for information regarding power quality to the Smart Metering Information System. After all the related information are retrieved along with time-stamps, it is sent to the Distribution Network Operator. Then, the customers receive the requested power quality information. This is the basic flow of information. If the Smart Metering Information System can't process the request, error details are sent back. It is also possible that Smart Metering Information System fails to retrieve data due to hardware failure which is then reported to Distribution Network Operator. It becomes easy to address the issue[ETS10a].

There are few potential requirements for an efficient monitoring system. A secure and accurate time synchronization M2M system should be present for accurate results. A periodic reporting time can be assigned by M2M application for some specific M2M device(s). The M2M system should be able to retrieve scheduled and on demand information for the customers. Authentication and authorization should be maintained throughout to avoid leakage and tampering of information[ETS10a].

3.1.2 E-Healthcare and Tele-medicine

One of the major application areas for M2M systems can be found in E-healthcare sector. It includes the remote monitoring of the patient health and fitness information through use of various appropriate sensor devices, triggering of alarms when critical conditions are detected, remote controlling of certain medical devices for treatment. The person being monitored usually wears one or more sensor devices that record health and fitness indicators such as blood pressure, body temperature, heart rate, weight and etc. These sensor devices can collectively send information to a device/gateway(e.g. a cellphone acting as a gateway device) through short range communication. These information are then sent to a backend entity(M2M server, device management server) by the gateway device via WiFi or LTE/3G networks that is supposed to store and possibly react according to the sensor information. Sometimes the sensors themselves have connectivity capabilities to create a WAN link with the backend server without the need of a gateway device[ETS13a].

The use case for E-Healthcare can be Remote Patient Monitoring(RPM)[ETS13a]. This use case focuses on monitoring the patient's health condition based on the information collected from the sensor devices attached to him. The whole M2M system involves bi-directional transfer of messages, network availability during critical situations, information and device security and etc. The figure 3.1 shows the high level architecture for RPM concept using eHealth services.

The monitoring sensor devices attached to the patient are first registered to the central back-end server(e.g. M2M server and device management server). The registration includes the potential to maintain the information describing the remote monitoring of devices, the patient being monitored and M2M applications which constantly keep track of the data. These sensor devices collect various information of interest from the patient. The patient measurements could be done in hospital or at home, work or while travelling. These information maybe sent to the gateway device(e.g. cellphone) that has WAN connectivity via LTE, WiFi or RFID tag networks . Since these sensor devices are low powered, less complex, so, they likely to be accompanied by a gateway for communication. The device management server can instruct the devices how frequently and through which route these devices should send data. The applications under clinical supervision monitor the information received from the devices attached to patient's body. Based on the measurements, an alert or emergency can be declared upon the radical

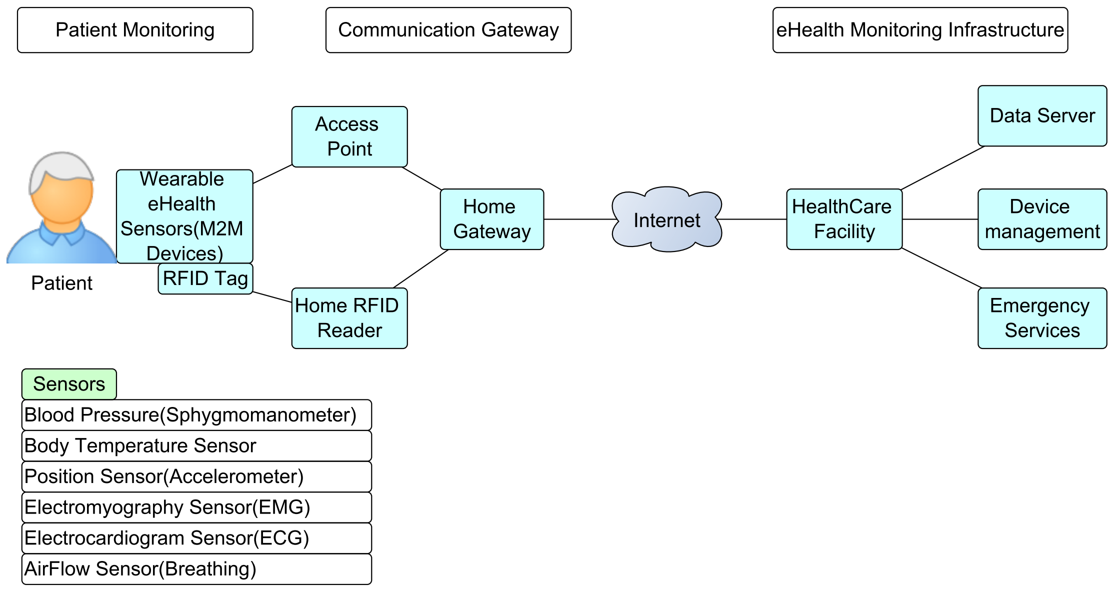

Patient Monitoring
Communication Gateway
eHealth Monitoring Infrastructure
Access Point
Data Server
Wearable eHealth Sensors(M2M Devices)
Home Gateway
Internet
HealthCare Facility
Device management
Patient
RFID Tag
Home RFID Reader
Emergency Services
Sensors
Blood Pressure(Sphygmomanometer)
Body Temperature Sensor
Position Sensor(Accelerometer)
Electromyography Sensor(EMG)
Electrocardiogram Sensor(ECG)
AirFlow Sensor(Breathing)

Figure 3.1: High Level Diagram for Remote Patient Monitoring using eHealth Services

change or high fluctuations in the monitored values or if it falls out of the predetermined safe levels. The device management server can issue command to initiate to find current GPS location of the patient(based on attached device information), calling emergency services or remote controlling of the devices. The device management enabler along with connectivity services should be able to route the critical information via a secure and reliable channel for timely delivery[ETS13a].

3.1.3 Connected Consumer

The *Connected Consumer*[ETS13b] refers to the humans who are equipped with communication tools that allow monitoring and remote control of the devices via M2M services and applications. The use case related to this scenario could be the *Remote control of Home Appliances*. The home appliances with communication capabilities which act as M2M devices and can do short range communication can be interfaced into M2M network. They are generally connected to M2M gateway to access the WAN. The application acting as a remote service such as a website, remote device equipped with connectivity features, that receives and sends information and commands to control home appliances via device management server. The home appliances can be registered to the backend server with all the appropriate details. With the regular data available from the sensor devices in the home, the devices can be remotely monitored and controlled such as turn on/off the heating system, downloading movies from the remote media server, downloading recipes to prepare special dish for dinner, activating alarm system and etc. The figure 3.2 shows the high level architecture for the remote control of home appliances.

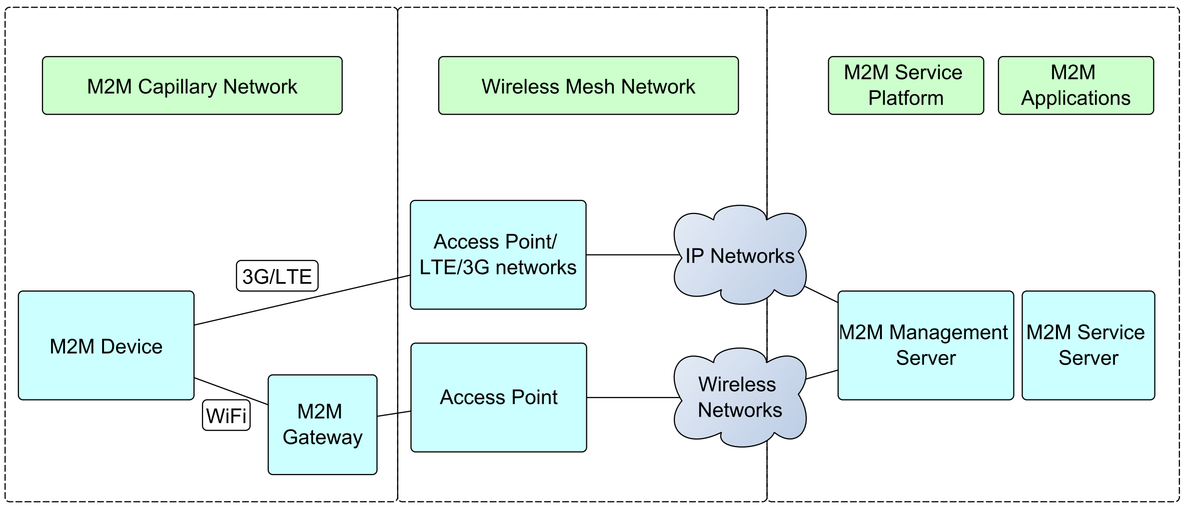


Figure 3.2: High Level Diagram for Remote Control of Home Appliances

3.2 Technical Requirements

The following subsections describe the technical requirements of M2M system and Device Management.

1. **Scalability**
The number of M2M devices are increasing very rapidly because of their usability in every sector. Hence, the M2M system should be able to cope with the ever increasing number of M2M devices that want to connect to the system. The device management server-client model based on LightWeight M2M architecture could be the best effort to address these all low powered, less capacity devices called constrained devices. The UDP based CoAP reduces the payload size making easy and quick transport of messages. The handling of all the resources of thousands of devices seem easy with the hierarchical resource tree model of LWM2M Device Management Server. Hence, the overall system should be able to scale according to the need.

2. **Interoperability**
The components of the M2M system are based on ETSI M2M, Device Management implementation is based on Lightweight M2M, the M2M devices maybe from different vendors, the communication between the components maybe based on

different protocols/standards. But, when they are combined together, they should be able to communicate with each other. Hence, all the components of the whole system should be strictly standards based, so that, they can be made interoperable.

3. **Link Quality**
For the transport of messages in the M2M system, the link quality should be less lossy. The LWM2M device management enabler also has confirmable and non-confirmable message types to indicate if the message delivery is acknowledged or not. Beside these, it stores a list of IP addresses in connectivity management object to route messages using a less congested path. In emergency scenario like fire hazard, accidents, disease outbreak, natural calamities, the messages should be delivered to the system quickly and reliably so that the M2M system can react to it quickly and remotely control the devices or report to the concerned authority.

4. **Information Privacy**
The sensor devices generate huge amount of data; that maybe from patient's body like heart rate, blood pressure, sugar level etc, from different parts of the vehicle like current location, engine performance, fuel consumption rate etc, from household appliances like status of the appliances-period of operation, energy consumption and etc. These information are critical and M2M system should ensure that these information should be recorded securely. So, there are implementations of access rights and local policies for authorization and authentication of users to control access to critical components and protects information from illegal use.

5. **Compatibility**
The M2M system should be flexible enough to work with older components which is backward compatibility and should be able to address possible changes in the future which is known as forward compatibility. As the components are completely based on standards and defined protocols, the compatibility issue should be minimum.

6. **Easy Configuration**
The M2M devices should be easily or in most cases, auto configured and get registered to M2M server and LWM2M device management server. The LWM2M device management server should send regular firmware updates to keep the devices upto date.

7. **Tolerance Level**
The M2M system should be aware of the possible faults that can occur in the system such as loss of connection, congestion in the network. Besides these, some messages are critical and should be reported to the system so that it can act quickly.

8. **Availability of Resources**
The M2M system resources should be made available to critical messages. When critical messages are generated that demand quick action like during earthquake,

cyclones, accidents, the M2M system should give priority to it and process accordingly. The device management enabler should issue commands to activate/deactivate devices and inform the concerned authority.

9. **Energy Efficiency**
   The LWM2M protocol which is based on CoAP aim to reduce the size of payloads which contributes to enhance the spectrum efficiency. As the transmission time is reduced due to payload size reduction, the energy and time efficiency are enhanced.

10. **Low Cost Solutions**
    The low complexity of the solutions contributes to reduce the cost of devices.

4 Design and Specification

This chapter introduces the detailed architectural design concept of OMA LightWeight M2M Device Management, operation types, components description and integration with OpenMTC[Fra14] which is based on ETSI M2M and oneM2M standards. There are various operations that can occur between the LWM2M DM Server and LWM2M DM Client. For the components to understand and exchange ETSI compliant messages and DM messages, there are the need of adapters that translate between them. The DM GUI is also an important part that gives the visual representation of devices and their characteristics involved. The figure 4.1 gives an overall picture of the whole concept.

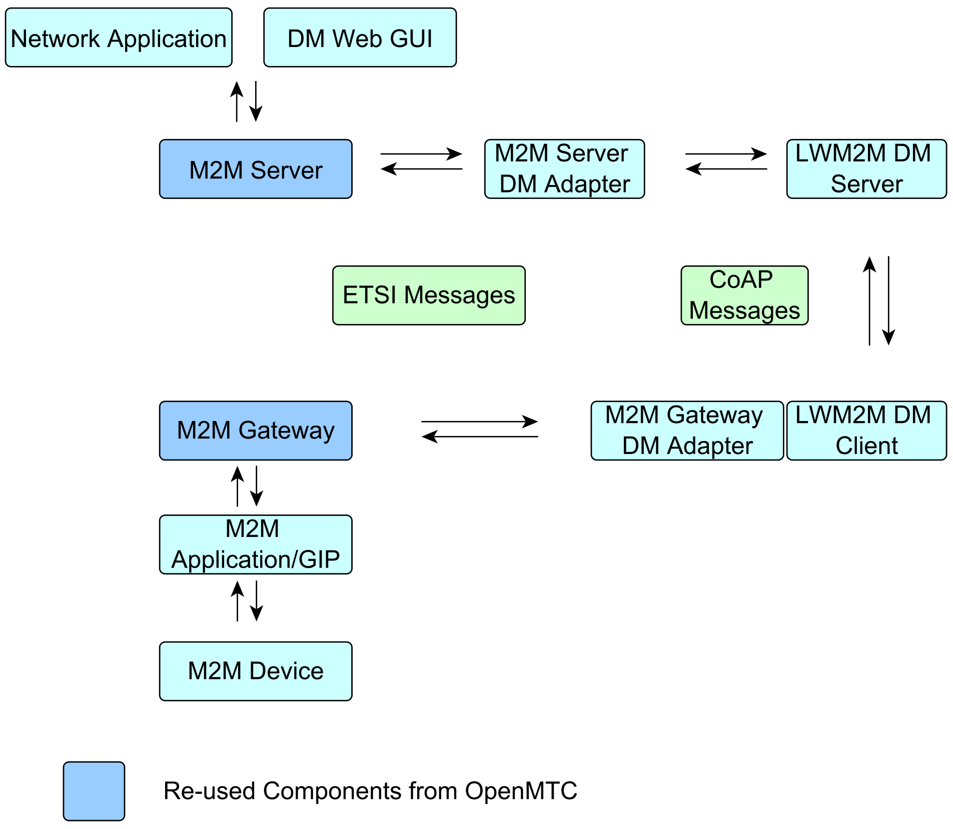


Figure 4.1: Device Management Concept Design

4.1 OMA LWM2M Device Management

The OMA LWM2M Device Management consists of a DM Server and DM Clients. There are four interfaces through which the DM Server and DM Clients exchange CoAP messages to carry out various possible operations as specified in the OMA LWM2M v1.0 specification. These logical operations in the interfaces can be uplink (direction: client to server) and downlink(direction: server to client) which is already presented in Table 2.1. The LWM2M specification[OMA13a] identifies several objects and resources as main components of its resource model. Each object and its resources are identified by unique ID and several characteristics.

The four interfaces proposed in the LWM2M specification are

1. Device Discovery and Registration
2. Bootstrap
3. Device Management and Service Enablement
4. Information Reporting

4.1.1 Device Discovery and Registration Interface

This interface is used by a LWM2M DM Client to register its registration parameters, maintain registration status and de-register with a LWM2M DM Server.

1. **Client Registration**
   While registering, the client sends its endpoint name which is unique in the server, lifetime period that it expects to be alive and list of objects that it supports during registration. The list of objects and object instances paths are described using CoRE Link Format[IET12] and payload format used is application/link-format which are included in the payload of the request and rest of the registration parameters are in the query of the request. Also, along with these registration information, DM Server keeps track of the IP addresses of the clients. The payload supporting Device, Firmware, Location objects and their object instances would simply be </3/0>, </5/0>, </6/0>. The registration is soft state which is indicated by registration lifetime, and once the lifetime period is exceeded, the client registration is removed from the DM server. Hence, the client has to re-register again to the server[OMA13a]. The activity diagram illustrates the process of client registration in figure 4.2.

2. **Registration Update**
   The client can send its registration parameters like lifetime, binding mode and others with the updated values. This also ensures that the client is active and interested to interact and exchange messages with the server[OMA13a].

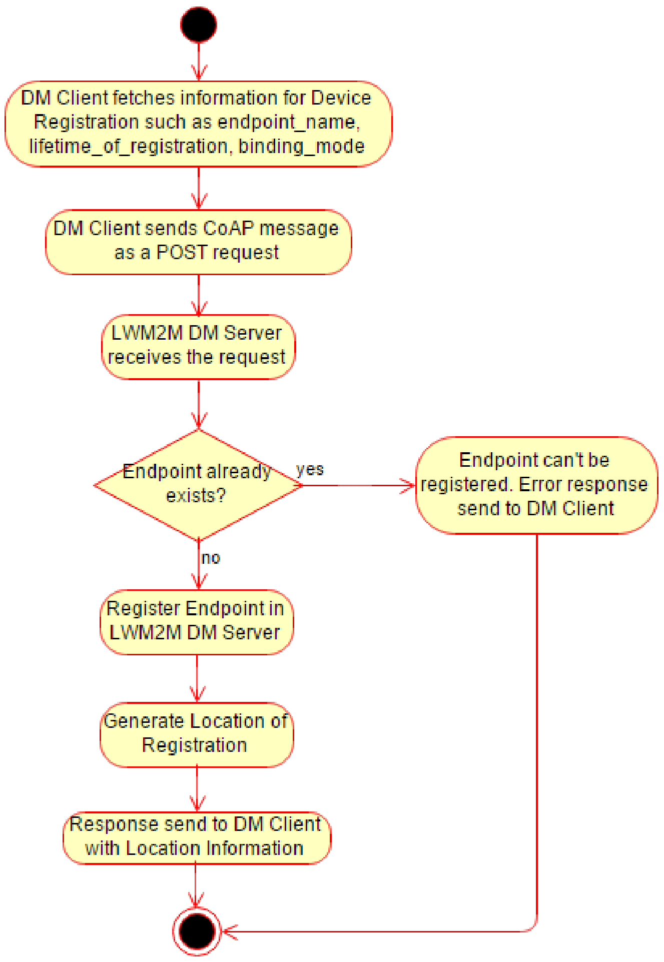

DM Client fetches information for Device Registration such as endpoint_name, lifetime_of_registration, binding_mode
DM Client sends CoAP message as a POST request
LWM2M DM Server receives the request
Endpoint already exists?
yes
Endpoint can't be registered. Error response send to DM Client
no
Register Endpoint in LWM2M DM Server
Generate Location of Registration
Response send to DM Client with Location Information

Figure 4.2: Activity Diagram depicting Device Registration

3. **De-register**
   If the client is no more interested or not needed to be associated with the server, it can do de-register by sending the request to the server to remove its registration information from its database[OMA13a].

4.1.2 Bootstrap Interface

This interface is used to provision essential information of the LWM2M client to make the LWM2M client be able to register to a LWM2M DM server[OMA13a].

4.1.3 Device Management and Service Enablement Interface

This interface is used by LWM2M Server to access resources available from LWM2M Client. There are number of proposed operations such as Create, Read, Write, Delete, Execute, Discover that a LWM2M Server can use to get service related to resources from LWM2M Client.

1. **Create**
   This operation is used by the DM Server to create the object instances and resources in the DM Client. The information is sent in the payload of the request. It has following parameters as shown in Table 4.1[OMA13a].

Parameter	Required	Default Value	Notes
Object ID	Yes	-	Indicates Object
Object Instance ID	No	-	Indicates Object Instance. If this isn't specified, client decides ID of Object Instance
New Value	Yes	-	New value is included in the payload to create object instance

Table 4.1: Operation Create

2. **Delete**
   This operation is used by LWM2M Server to delete an object instance in the LWM2M Client. The object instance that is deleted using this operation must have been created using the Registration operation. It has following parameters as shown in the Table 4.2[OMA13a].

Parameter	Required	Default Value	Notes
Object ID	Yes	-	Indicates Object
Object Instance ID	Yes	-	Indicates Object Instance to delete

Table 4.2: Operation Delete

3. **Discover**
   This operation is used to discover the resources implemented in the object. It is also used to discover the attributes of the individual resource, resources of an object instance or all object instances of an object. The response to this request includes list of resources as described in CoRE Link Format (RFC 6690) along with their attributes.It has following parameters as shown in the Table 4.3[OMA13a].

4. **Read**
   This operation is used to access the value of a resource, an array of resources, an object instances. It has following parameters as shown in the Table 4.4[OMA13a].

Parameter	Required	Default Value	Notes
Object ID	Yes	-	Indicates the Object.
Object Instance ID	No	-	Indicates Object Instance.
Resource ID	No	-	Indicates the Resource.

Table 4.3: Operation Discover

Parameter	Required	Default Value	Notes
Object ID	Yes	-	Indicates the Object
Object Instance ID	Yes	-	Indicates Object Instance. If resource ID isn't specified, whole object is returned.
Resource ID	No	-	Indicates the resource to read

Table 4.4: Operation Read

5. **Write**
This operation is used to change the value of a resource, an array of resource instances or multiple resources from the object instances at once. It has following parameters as shown in the Table 4.5[OMA13a].

Parameter	Required	Default Value	Notes
Object ID	Yes	-	Indicates the Object
Object Instance ID	Yes	-	Indicates Object Instance. If resource ID isn't specified, the included payload is an object instance containing resource values to be written.
Resource ID	No	-	Indicates the resource to be written. The value of the resource is contained in the payload.
New Value	Yes	-	The new value in the payload to update the resources.

Table 4.5: Operation Write

6. **Execute**
This operation is used to carry out some action and performed on individual resource. It contains following parameters as shown in the Table 4.6[OMA13a].

7. **Write Attributes**
This operation is used to update the attributes of the object instances and resource instances. There are six attributes defined such as Maximum period, Minimum period, Greater than, Less than, Step and Cancel. It has following parameters as

Parameter	Required	Default Value	Notes
Object ID	Yes	-	Indicates the Object
Object Instance ID	Yes	-	Indicates Object Instance.
Resource ID	Yes	-	Indicates the resource to execute.

Table 4.6: Operation Execute

shown in the Table 4.7[OMA13a].

Parameter	Required	Default Value	Notes
Object ID	Yes	-	Indicates the Object
Object Instance ID	Yes	-	Indicates Object Instance.
Resource ID	Yes	-	Indicates the resource to execute.

Table 4.7: Operation Write Attributes

4.1.4 Information Reporting Interface

This interface is used by a LWM2M Server to observe any changes in a resource present in a LWM2M Client and receiving notifications when new values are available. The LWM2M Server sends an observation request to the LWM2M Client for observing an object instance or resource instance. This operation may contain a number of optional attribute settings that defines how the notifications should reach the LWM2M Server. An observation ends when cancel observation operation is performed on that object instance or resource instance in the LWM2M Client[OMA13a].

1. **Observation**
   The LWM2M server initiates an observation by sending a request to monitor a specific resource, resources within an object instance or all the object instances of an object with LWM2M client. The observation request is dependent on few attributes that are illustrated in Table 4.7 and can be modified using the Write Attribute operation as discussed above. It has following parameters as shown in the Table 4.8[OMA13a][Har14].

2. **Notify**
   This operation is executed from the LWM2M Client to the LWM2M Server if there is a change or an update in the observed object instance or resource. The notification message should include the updated values in the object instance or resource along with other information like same token id as that of the request,

Parameter	Required	Default Value	Notes
Object ID	Yes	-	Indicates the Object
Object Instance ID	No	-	Indicates Object Instance to observe. If not specified, all object instance of the object are observed and resource IDs must not be specified.
Resource ID	No	-	Indicates the resource to observe. If not specified, whole object instance is observed.

Table 4.8: Operation Observation

observation time and etc. While sending the notification, it should meet all the criteria specified in the attributes of object instance or resource as shown in Table 4.7. It has following parameters as shown in the Table 4.9[OMA13a][Har14].

Parameter	Required	Default Value	Notes
Updated Value	Yes	-	The new value is about object instance or resource.

Table 4.9: Operation Notify

When an observation is sent on an object or a resource, a Token ID is maintained besides a Message ID in the request message. When notifying back upon the observed resource being modified, same Token ID is used. Also, observe time, which is the time taken to send a notification after the observation started, is also sent along with the response. The figure 4.3 and figure 4.4 show the activity diagrams for General and Specific Observation and Notification[OMA13a][Har14].

3. **Cancel Observation**
   If the object instance or resource observation is no more required, it can be cancelled by sending a Cancel Observation request from the LWM2M DM Server to the LWM2M DM Client. This operation doesn't contain any parameters at the LWM2M layer. The LWM2M Server can do this operation in two ways[OMA13a][Har14].

   - By sending "Cancel Observation" operation
     When the LWM2M Server is no more interested being notified about the object instance or resource updates, the LWM2M Server can send a Cancel Observation request to end the observation.
   - By sending "Write Attributes" with cancel parameter
     If the LWM2M Server executes the "Write Attributes" operation with cancel parameter to a certain URI in the LWM2M Client, the LWM2M Client must cancel observation for that specified URI.

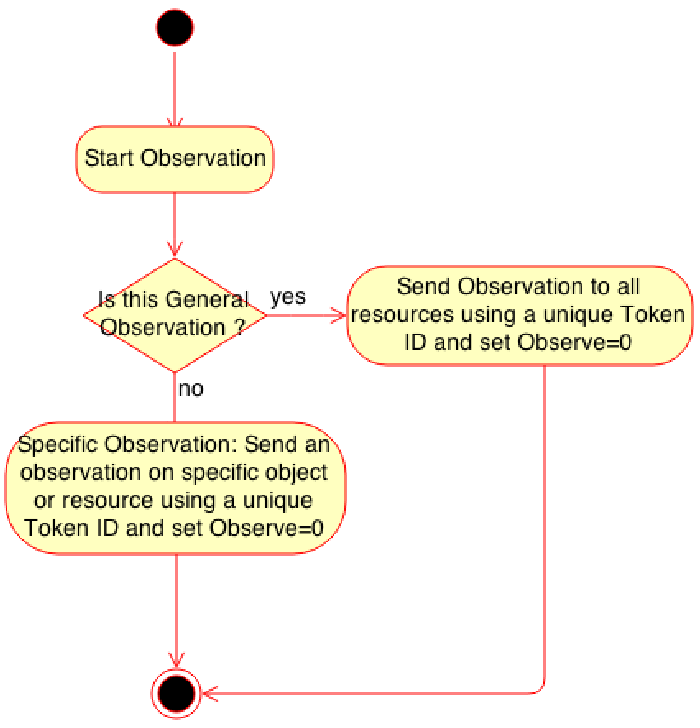

Start Observation
Is this General Observation ?
yes
Send Observation to all resources using a unique Token ID and set Observe=0
no
Specific Observation: Send an observation on specific object or resource using a unique Token ID and set Observe=0

Figure 4.3: Activity Diagram depicting Resource Observation

4.2 OMA LWM2M Device Management Resource Model

The resource model plays an important role in the organization of data stored in the memory or the database. It is directly related with the scalability, accessibility in short time, readability, organization and management of data. According to the specification, resource model is a hierarchical system consisting of Objects, Object Instances, Resources, Resource Instances <object_id/object_instance_id/resource_id/resource_instance_id>. The concept is to use the python dictionary to store the data and class objects in JSON format in each hierarchical level to reduce the access time. The time complexity is O(1) in almost all operations related to dictionary except the iteration which is O(N)[1]. The figure 5.2 shows the design and implementation of DM resource model.

4.3 OpenMTC Middleware Platform

OpenMTC[Fra14] is a standard compliant implementation of ETSI M2M and further extended to meet the need of oneM2M standard. It is a middlware platform enabling generic M2M communication and acts as a facilitator between different service platforms.

[1]https://wiki.python.org/moin/TimeComplexity

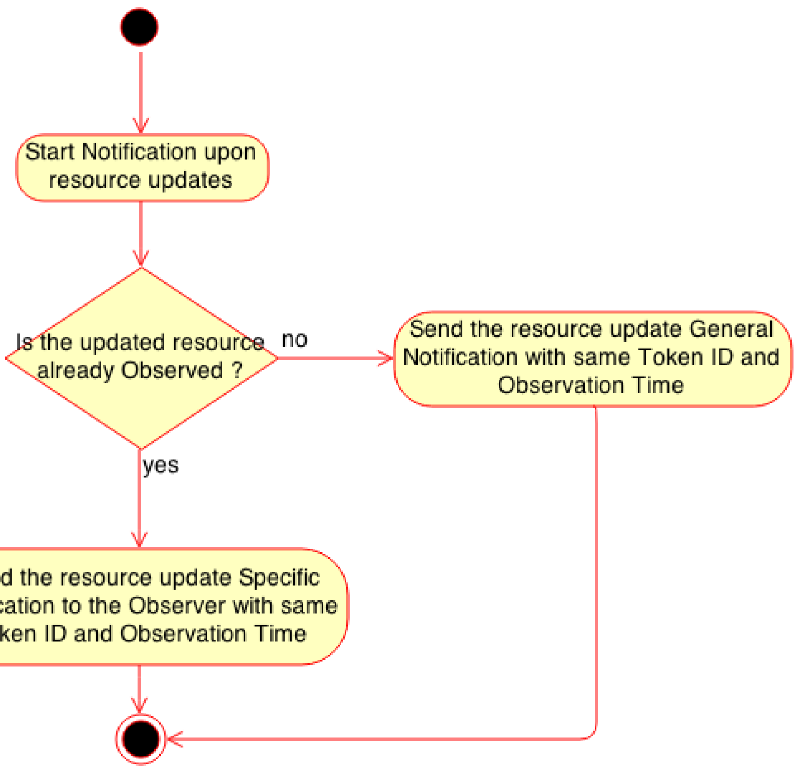

Start Notification upon resource updates
Is the updated resource already Observed ?
no
Send the resource update General Notification with same Token ID and Observation Time
yes
Send the resource update Specific Notification to the Observer with same Token ID and Observation Time

Figure 4.4: Activity Diagram depicting Resource Update Notification

The core network allows flawless communication management of different terminals, sensors, actuators and etc. It is designed to act as a horizontal convergence layer for machine type communication that supports multiple vertical application domains[Fra14]. It extends its support in M2M market segments such as smart city building(TRESCIMO[2] project), e-Health(FI-STAR[3] project), automotive, surveillance and monitoring and etc.

OpenMTC consists of service capability layers such as Gateway Service Capability Layer(GSCL) and Network Service Capability Layer(NSCL). Additionally, it also supports Device Service Capability Layer(DSCL) for devices that don't need gateway for communication with the core networks and applications.

OpenMTC supports Management Objects(*mgmtObjs*) as part of its resource tree as defined in the standards. These resources are part of Network Reachability, Addressing and

[2]www.trescimo.eu
[3]www.fi-star.eu

Repository(NRAR) capability and also deals with the Device Management[Fra14][ETS10d]. The *mgmtObjs* can be located under *scl* paths such as

- <sclBase>/scls/<scl>/mgmtObjs is used to manage service capabilities and for other management functions of single <scl>resource.
- <sclBase>/scls/<scl>/attachedDevices/<attachedDevice>/mgmtObjs is used for management of each D' device attached to M2M Gateway represented by <scl>.
- <sclBase>/scls/<scl>/attachedDevices/mgmtObjs is used for M2M Area Network Management of each D' device attached to M2M Gateway.

M2M Server

It refers to M2M service capabilities in the network domain and is employed as a back-end component in the cloud. It provides a managed resource tree for the front-end components such as GSCLs and DSCLs which are registered. The M2M devices which needs public connectivity access can be registered to a Gateway which is further registered to M2M Server in the backend under *attachedDevices* resource. Similarly, the devices with connectivity access can directly register as *scl* resource in the M2M Server[ETS10d].

In connection to the Device Management, M2M Server allows *mgmtObjs* and its ETSI defined resources and attributes/parameters to be placed in various locations as mentioned in section 4.3 and in the figure 2.2. Further, the resources that are compliant to the LWM2M objects are placed under respective *mgmtObjs* as flexible attributes.

4.4 Adapters

The main idea to have Adapters in the design and concept is to translate the messages that are compliant to ETSI M2M (understood by OpenMTC) and LWM2M Device Management (understood by LWM2M DM Server and Client). There are two adapters:

1. **M2M Gateway DM Adapter** : between M2M Gateway and LWM2M DM Client.
2. **M2M Server DM Adapter** : between M2M Server and LWM2M DM Server.

The LWM2M DM Server and Client understand CoAP aligned messages and objects which use custom built APIs to hide the underlying mechanisms. The type of the messages sent are POST, PUT, GET and DELETE. The OpenMTC platform uses RequestIndication objects that use APIs such as *CreateRequestIndication*, *UpdateRequestIndication*, *DeleteRequestIndication* to create, update and delete the ETSI resources same as REST interfaces. So, there is the need of above listed adapters that translate information and create messages and objects that handle the resources respectively to avoid the compatibility issues.

The adapters use internal events to subscribe to the ETSI defined resources such as *scl*, *attachedDevice*, *mgmtObj*, *mgmtCmd* and etc. When a new resource is created or updated in the M2M Server/M2M Gateway resource tree, the adapter gets notified about the changes. Then, the adapter translates this message compliant to LWM2M standard in order to push the changes in the LWM2M Device Management Server and Client. On the other hand, these adapters are also subscribed LWM2M Device Management Server and Client via General and Specific subscriptions to get notified about the changes in the Device Management. Similarly, the adapters push the changes by using RequestIndication API calls and RequestIndication objects.

4.5 Device Management Web GUI

The web GUI gives a visual depiction of devices and gateways registered in the Device Management Server with all the important details using an easy interface. It is a good way of knowing about the status of devices and gateways connected to the system. Furthermore, the interfaces in the GUI that helps to remotely control the devices by sending the commands using buttons would be interesting, e.g. turning on/off the remote devices(sensors/actuators) by pressing buttons on the GUI.

4.6 Cloud Infrastructure

The concept for including cloud infrastructure is to check the viability of running large number of components in virtual environment. With the components deployed in the cloud, lot of advantages can be observed such as reduction in operational cost, sharing of processing units, storage and network infrastructures. The OpenSDNCore[4] which is the implementation for Cloud based services based on ETSI Network Functions Virtualisation(NFV)[5] standards, and is used for demonstrating the scalability of Device Management concept in FOKUS FUSECO Forum 2014. It receives the topology request describing a set of service/network functions and the Device Management Server and Device Management Clients running emulated devices are deployed in virtual machines using OpenStack[6]. The orchestrator does the management and provide an uninterrupted communication between them. Each virtual machine can be designated with different flavors which contain certain number of processing cores, memory size as per the availability of resources. A number of network applications are also deployed as per need in the cloud. The whole concept is more detailed in section 5.3.6 and Chapter 6 and a high level design showing policy handling for transporting M2M traffic is shown in the figure 4.5.

[4]www.opensdncore.org
[5]http://www.etsi.org/technologies-clusters/technologies/nfv
[6]www.openstack.org

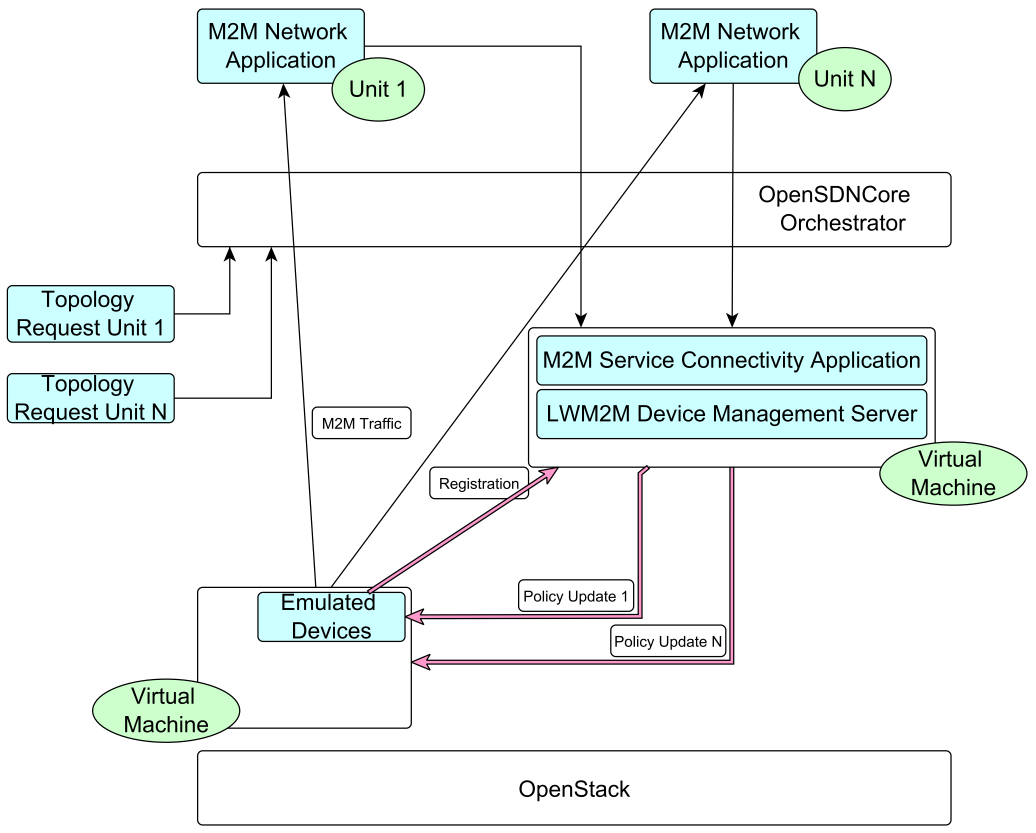


Figure 4.5: High Level Concept Design of Transport Policy Handling for M2M Traffic

5 Implementation

This chapter provides the insights to the implementation of Device Management enabler. It starts with the description of the needed environment, project structure, implementation of the Device Management components based on the concept discussed in Chapter 4.

5.1 Environment

For the implemention, the following environment is used.

5.1.1 Ubuntu 12.04 LTS

The operating system Ubuntu[1] 12.04 LTS which is a widely used GNU Linux distribution is chosen and the workstation with hardware configuration of Quad-core processor and 4 GB of DDR3 RAM are used. This version of Ubuntu is quite stable and has a good support for programming languages. Easy access to its own repositories, third party repositories, availability and support for large number packages are the features for choosing this distribution. Also, JDK/JRE 1.7 packages are used for ubuntu environment.

5.1.2 Python 2.7

Python[2] 2.7 is a widely used high level, general purpose, object oriented programming language. The flexibility of its syntax allows programmers to express the concept in fewer lines of codes than would be possible in other programming languages such as C++ or Java. Python 2.7 comes as pre-installed package in Ubuntu 12.04 LTS.

5.1.3 Komodo 8 IDE

Komodo[3] 8 Integrated Development Environment(IDE) is a text editor for many popular programming languages. This editor is used to write Python 2.7 based library functions and plugins related to Device Management. It supports user customization through plugins and macros. The ability to find out syntax errors related to Python 2.7 and easy interface makes it a better choice among developers.

[1]www.ubuntu.com
[2]www.python.org
[3]www.komodoide.com

5.1.4 Gevent 1.0

Gevent[4] 1.0 is an event based library providing asynchronous I/O API and can scale its number of execution units known as greenlets, according to processing load. It features more consistent API, simpler implementation and better performance.

5.2 Project Structure

The Device Management modules are the part of OpenMTC and run as plugins. The root folder of OpenMTC is *openmtc-python*. All the library modules, plugin modules and configuration files related to Device Management are located under this folder. The figure 5.1 shows the folders hierarchy.

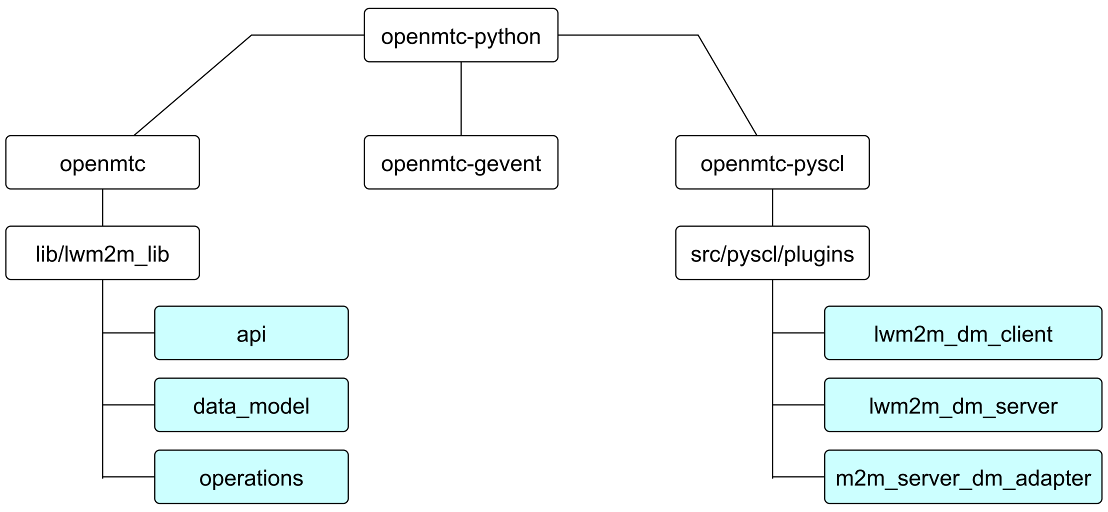


Figure 5.1: Project Structure Implementation Hierarchy

1. **Library Modules**
   All the library modules related to Device Management can be found under *openmtc-python/openmtc/lib/lwm2m_lib*. The *lwm2m_lib* folder is further sub-divided into three folders:
   - **api**: This folder contains the list of functions related to LWM2M Device Management operations. These functions are used by LWM2M Server and Client for sending POST, PUT, GET, DELETE requests based on LWM2M operations.
   - **data_model**: This folder contains files related to resource model for LWM2M Device Management.

[4]www.gevent.org

- **operations**: This folder contains files related to various LWM2M operations.

2. **Plugins**
All the plugins related to Device Management can be found under *openmtc-python/openmtc-pyscl/src/pyscl/plugins*.
    - **lwm2m_dm_client**: This folder contains files related to LWM2M Device Management Client and M2M Gateway Adapter.
    - **lwm2m_dm_server**: This folder contains files related to LWM2M Device Management Server.
    - **m2m_server_adapter**: This folder contains files related to M2M Server Adapter that resides between M2M Server and LWM2M Device Management Server.

3. **Configuration Files**
The configuration files contains information about the plugins and the initial parameters related to plugins can be modified using these files. These files are JSON formatted. There are atleast two configuration files to launch a M2M Server and a M2M Gateway. These basic configuration files are located at *openmtc-python/openmtc-gevent*. Other configuration files used for the purpose of evaluation will be stated in the respective section.
    - **config-nscl.json**: This configuration file is used to launch the M2M Server. ./run_nscl -f path_to_server_configuration_file
    - **config-gscl.json**: This configuration file is used to launch the M2M Gateway. ./run_gscl -f path_to_gateway_configuration_file

Listing 5.1: Portion of Configuration file

```
{
        "name":"lwm2m_dm_server",
        "package":"pyscl.plugins.lwm2m_dm_server",
        "disabled":false,
        "config":{
                "lwm2m_dm_server_ip":"127.0.0.1",
                "lwm2m_dm_server_port":5684,
                "client_ip":"127.0.0.1",
                "client_port":38000
        }
}
```

5.3 Important Implementation Aspects

This section describes the important building blocks for the implementation of LWM2M Device Management. All the components need a listener and client for handling CoAP requests. So, during the implementation phase, they are assigned the address localhost:port where port is defined in the configuration files. Based on the concept design on figure 4.1, below are the components and operations descriptions.

5.3.1 LWM2M Device Management Server

The LWM2M DM Server is used for handling requests, sending requests and managing the management objects. The Gevent version of Datagram server is created to listen for the CoAP requests at standard port 5684. The server handles CoAP confirmable(POST, PUT, GET, DELETE) and non-confirmable requests. An acknowledgement is sent upon the request is handled. The table shows the type of requests and the associated LWM2M operations that are handled by the server. A CoAP client is used to send the requests to

Request Type	LWM2M Operations
POST	Registration, Create, Execution, Notification
PUT	Registration Update, Write, Write Attributes
GET	Discovery, Read, Observation(observe=0), Cancel Observation(observe=1)
DELETE	Delete

Table 5.1: Types of Requests and Associated LWM2M Operations

other listeners. The client port starts from 38000. Every request is sent with a unique port(i.e. port number is incremented by 1). This is not a mandatory requirement. But, when the listener is flooded with the requests using same port number, before any of them is acknowledged, there exists some socket issues to handle the acknowledgments. This is the limitation of CoAP implementation.

The LWM2M DM Server maintains a Resource Tree for hierarchical storage of objects and resources. It uses python dictionary data structure to store them. The information retrieval in dictionaries with key:value pair is quite fast with time complexity of O(1) in most dictionary operations.

5.3.2 LWM2M Device Management Client

The LWM2M DM Client represents a M2M Device. It is used to manage the device. A listener is implemented using Gevent Datagram server to listen the CoAP requests coming from the LWM2M DM Server. It also handles the LWM2M operations specific to the associated device as discussed in the Table 5.1. A CoAP Client is also used to send the requests to the LWM2M DM Server. Each client is assigned a unique port for the listener and CoAP Client. The client also stores the objects and resources information relevant to the associated device.

5.3.3 Adapters

The Adapters are implemented between M2M Gateway and LWM2M DM Client called as M2M Gateway Adapter, also between M2M Server and LWM2M DM Server called as M2M Server DM Adapter. The adapters address the need of translating the CoAP and ETSI compliant messages for compatibility.

In order to receive the CoAP messages, a listener is created using Gevent Datagram Server listening at port 5911 for M2M Server DM Adapter. The requests are handled as discussed in above sections. A CoAP client is also needed to send the request to LWM2M DM Client and Server by the respective adapters. The client port for M2M Server DM Adapter starts from 35000 and is unique for each request sent.

The adapters are subscribed to various Events such as *scl*(created, updated), *attached-Devices*(created, updated), *mgmtObjs*(created, updated), *mgmtCmd*(created, updated) and etc. For each event, a handler is created. Upon the trigger due to any of the events, the handler function is called. These handlers manipulate the RequestIndications and extract information. On the other hand, there are functions to send the RequestIndications such as *CreateRequestIndication*, *UpdateRequestIndication* to create the resources in the M2M Server and M2M Gateway.

Example of *CreateRequestIndication*

```
mgmt_object = mgmtObj(id="DeviceCapability_0", moID="urn:oma:lwm2m:ext:4200", Property="Pulse", Enabled=True)
request = CreateRequestIndication(
path="/m2m/scls/gscl/attachedDevices/PulseOximeter/mgmtObjs", resource = mgmt_object)
response = self.api.handle_request_indication(request)
```

Request

```
curl -X GET localhost:14000/m2m/scls/gscl/attachedDevices/PulseOximeter/mgmtObjs/
DeviceCapability_0
```

Response

Listing 5.2: Curl Response

```
{
        "mgmtObj":{
                "parametersCollection" : {
                        "namedReference":[]
                },
                "Property":"Pulse",
                "Enabled":"True"
                "creationTime":"2014-12-07T11:55:23.662642+00:00"
                "moID":"urn:oma:lwm2m:ext:4200"
```

```
                    "searchStrings":{
                            "searchString":[]
                    }
                    "expirationTime":"2014-12-07T12:55:23.662642+00:00"
                    "subscriptionReference":"/m2m/scls/gscl/attachedDevices/
                     PulseOximeter/mgmtObjs/DeviceCapability_0/subscriptions"
          }
}
```

5.3.4 Organization of Management Objects in Resource Model

The LWM2M DM standard specifies objects and resources hierarchy. So, in the implementation, all the objects and resources are python class objects with a set of attributes associated with each of them. The data structure *Dictionary* is used to store these objects. It is used for the quick retrieval of objects as the time complexity is constant, i.e. O(1). Hence, the resource tree contains information about the endpoints, associated objects & object instances, and resources and resource instances. The implemented objects and their resource details are illustrated in Annex. The figure 5.2 shows the implemented resource model of OMA LWM2M Device Management. The *endpoint* dictionary maintains a list of registered endpoints to the LWM2M DM Server in a form of key:value pair. It is further extended to store the *Location* of the registered endpoints so that the *Location* can be used for object update/delete later. The *object_dict* stores a list of registered objects to each of the endpoint in the form of key:value pair. It also stores the object instance which contains various properties related to resources, object attributes(pmax, pmin) and object observation instance. The *resource_id_dict* stores list of python class objects that contain properties for LWM2M resources(name, id, type and few other descriptors), resource attributes(pmax, pmin and etc) and resource observation instance. The object names and resource names are extended by "_X" where X represents the instance id if they support multiple instances which are discussed in LWM2M specification. For example, Device_0, Location_0 indicate the first instance of that management objects. Hence, this resource model addresses the LWM2M objects and resources by providing a robust and scalable system.

5.3.5 LWM2M Device Management Operations

The LWM2M DM operations are maintained in separate python files as library components. The LWM2M DM Server and LWM2M DM Client use these library functions to do LWM2M operations. The request and response are sent using CoAP Client objects from CoAP Client library. Some of the important DM operations are described below.

5.3.5.1 Client Registration

The LWM2M Client registers to the LWM2M DM Server with an unique endpoint name, registration parameters. The unique endpoint name and registration parameters can be

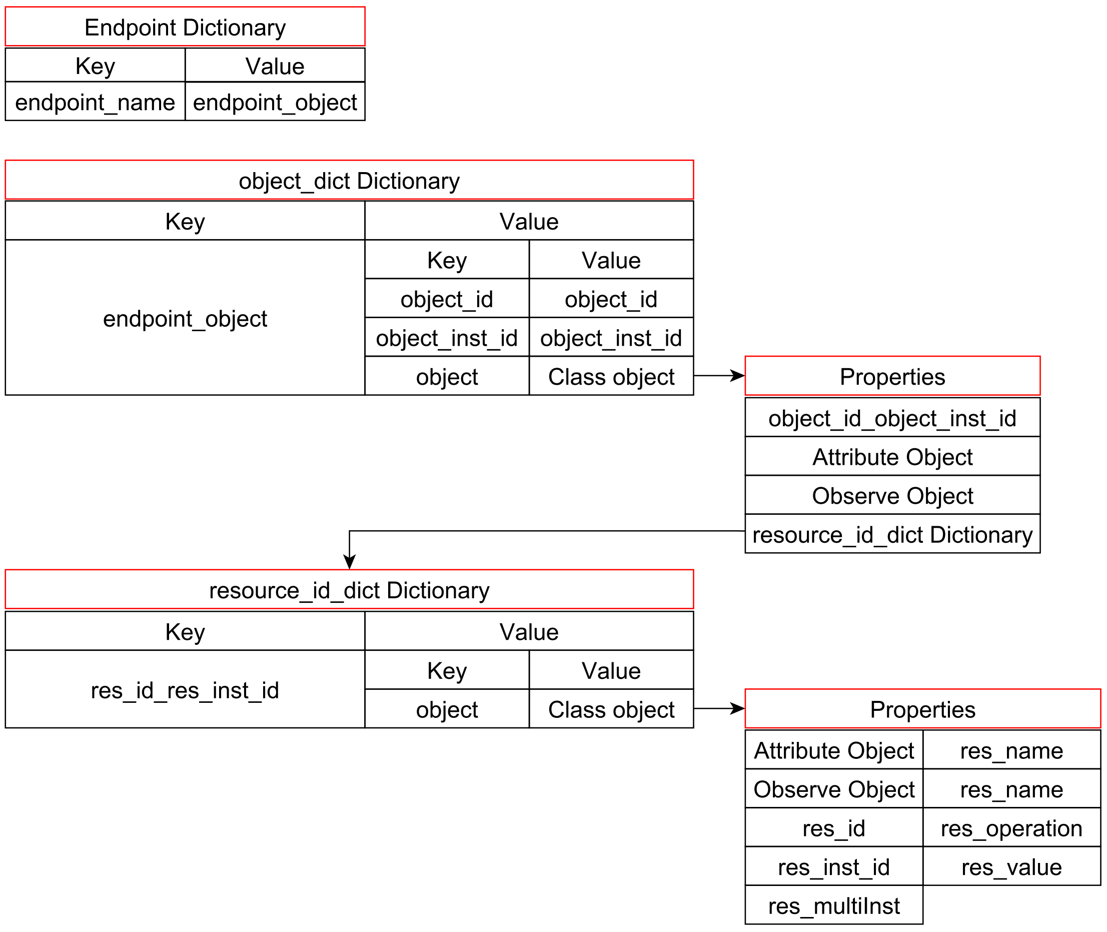


Figure 5.2: Resource Model in Device Management

used from the configuration file. A *ClientEndpoint* is implemented as a Python class. Each object of this class associates a number of object attributes such as the endpoint name, its registered location, sender and listener IPs and ports. A Python *Dictionary* data-structure is used to associate each endpoint with its endpoint object. The LWM2M Client uses a CoAP Client and library function to send the registration request to the LWM2M DM Server. The request is of type POST. On successful registration, the LWM2M DM Server responses with a unique 15 digit location address as the request is a Confirmable(CON) message. This unique location address is further used to send the updates on registration parameters or delete the Client if not required anymore.

Client Registration
```
client = CoapClient("coap://" + server_ip_port, client_port=client_port)
response = client.post(path+query, payload, timeout=10000, content_type=content_type)
return response
```

5.3.5.2 Resource Discovery

It is used to discover the registered clients along with associated resources and their attributes. Two types of discoveries are proposed in this implementation. If the discovery is done in an entry point which is a string "/.well-known/core", the LWM2M DM Server returns all the endpoint links. Attributes information of a registered object/resource can be accessed by requesting a discovery on a particular resource path.

Resource Discovery
```
query="?method=discover"
client = CoapClient("coap://" + server_ip_port, client_port=client_port)
response = client.get(path+query, data=payload, timeout=10000, content_type=content_type)
return response
```

5.3.5.3 Resource Observation and Notification

When an object or resource is being observed, a Token ID is assigned to that observation. The request is of type GET and it contains an optional parameter observe which is set to 0 to indicate the request is observation. A CoAP Client is used to send the request. When there is an update in that observed resource, the same Token ID is used for the notification. Also, the observation time is returned in the response. It is the time elapsed since the observation request is processed. This ensures the observation-notification pair. In this implementation, two types of observations are considered. General Observation and Specific Observation. A General Observation is subjected to all the resources change and uses a single Token ID for notification. A Specific Observation is specific to a particular object or resource and a different Token ID is used for each of them.
To cancel the observation, the operation Cancel Observation can be used. A CoAP client is used to send GET request on that observed resource with observe set to 1. This cancels the observation and no further updates on that resource are notified.

Resource Observation
```
client = CoapClient("coap://" + server_ip_port, client_port=client_port)
response = client.get(path, data=payload, timeout=10000, observe=observe, content_type=content_type)
return response
```

5.3.6 Use Case Additions

For realizing and evaluating the scalability of the emulated devices, some more components are implemented. The figure 5.3 shows the design and implementation of the components[5] for scalability test that use cloud infrastructure.

[5]These components are used for 5th FOKUS FUSECO Forum 2014, Berlin

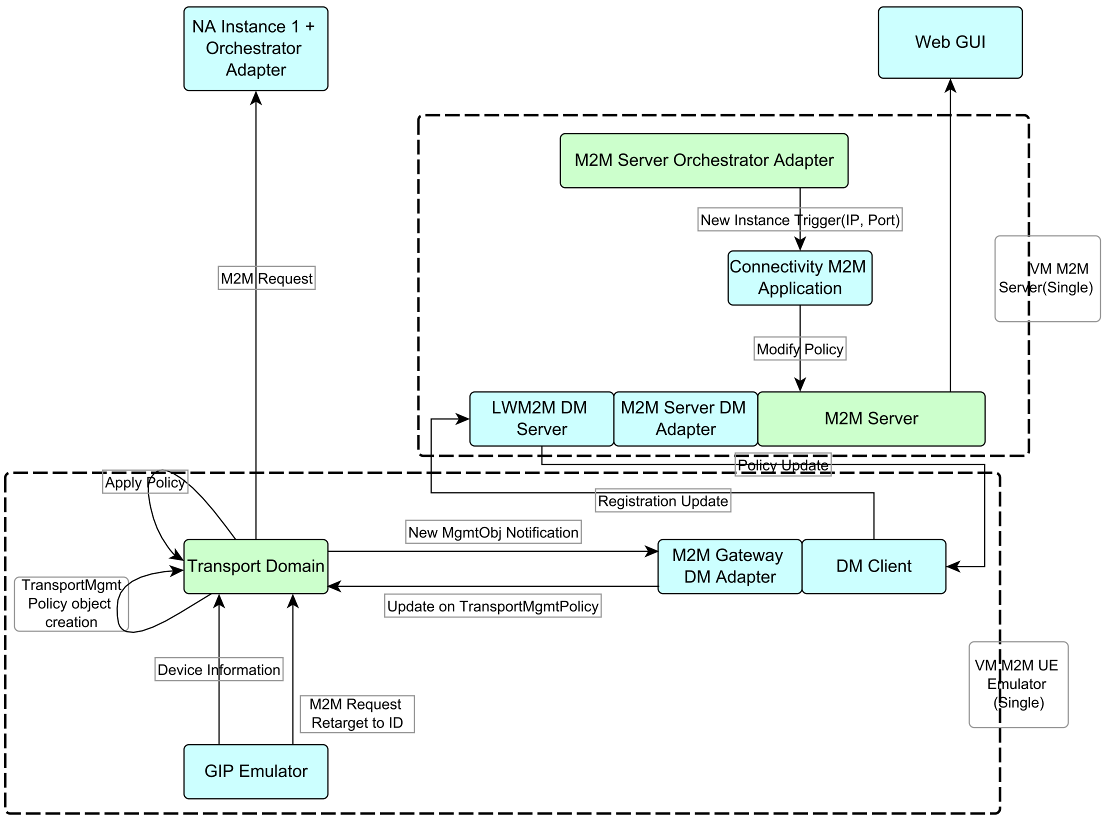

NA Instance 1 + Orchestrator Adapter
Web GUI
M2M Server Orchestrator Adapter
New Instance Trigger(IP, Port)
Connectivity M2M Application
M2M Request
VM M2M Server(Single)
Modify Policy
LWM2M DM Server
M2M Server DM Adapter
M2M Server
Policy Update
Apply Policy
Registration Update
New MgmtObj Notification
Transport Domain
M2M Gateway DM Adapter
DM Client
TransportMgmt Policy object creation
Update on TransportMgmtPolicy
Device Information
VM M2M UE Emulator (Single)
M2M Request Retarget to ID
GIP Emulator

Figure 5.3: Components for Scalability Test in Device Management

5.3.6.1 Applications

Two applications are created: back-end connectivity application and back-end application. The back-end application is created when it is registered to the M2M Server. This application forwards the transport policy information such as IP, Port, Transport protocol type via orchestretor of the OpenSDNcore to the back-end connectivity application registered at M2M Server. The back-end connectivity application stores these information as the resource attributes of the TransportMgmtPolicy *mgmtObj* object. This action triggers the M2M Server DM Adapter to handle this *mgmtObj* updates. The detailed evaluation setup analysis is done in Chapter 6.

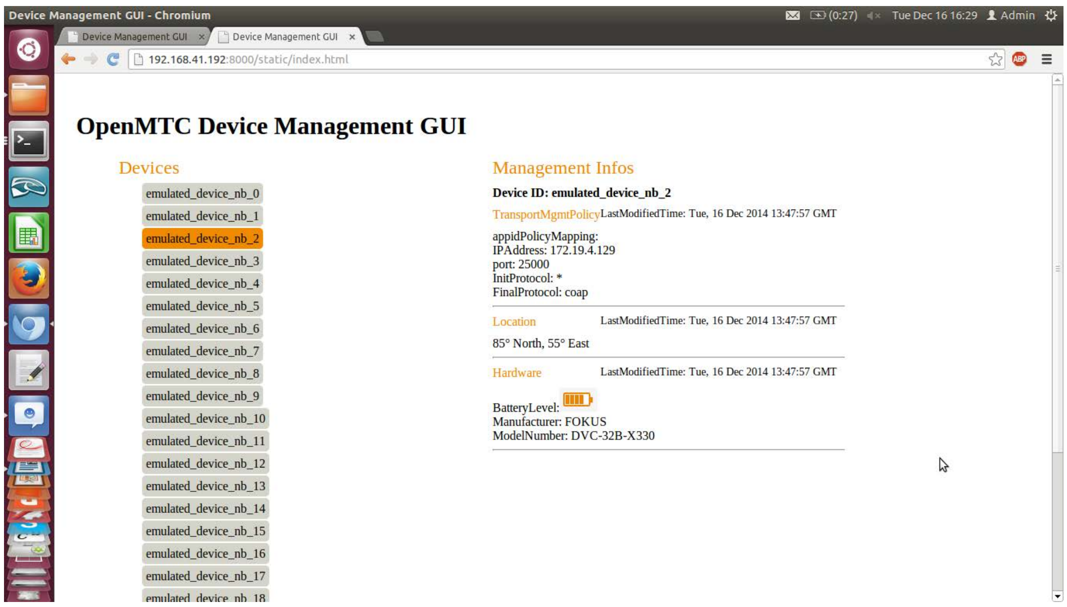
5.3.6.2 Emulated Devices

A 1000 emulated devices are registered in the front-end as DSCLs. Along with the basic registration, three management objects are also created such as TransportMgmt-Policy(proposed by FOKUS), and other two registered at OMA LWM2M: Device and Location. When they are created, M2M Gateway DM Adapter listening on the events of *scl* and *mgmtObj* create/update gets triggered and handled accordingly to create

LWM2M DM Client and getting them registered at LWM2M DM Server.

5.3.6.3 Device Management Web GUI

An interface to visualize the registered emulated devices and their *mgmtObj* attributes in the web-browser is developed using JavaScript and JQuery. The web GUI uses Open-MTC Socket IO plugin to establish connection with the M2M Server to extract information from it. The GUI component is subscribed to the M2M Server for changes in the *mgmtObj*. Whenever there is an update on the *mgmtObj* resources, the component gets triggered and then updates the GUI with new values. In this implementation, GUI is only used to monitor the *mgmtObj* changes. But, it can be further extended to send the commands to carry out some actions in the devices. The figure 5.4 shows the Device Management web GUI in a browser. The figure 5.5 shows 1000 emulated devices represented by square boxes and IDs. They change color whenever the emulated devices resource updates are triggered.


Device Management GUI - Chromium
(0:27)
Tue Dec 16 16:29
Admin
Device Management GUI
Device Management GUI
192.168.41.192:8000/static/index.html

OpenMTC Device Management GUI

Devices

emulated_device_nb_0
emulated_device_nb_1
emulated_device_nb_2
emulated_device_nb_3
emulated_device_nb_4
emulated_device_nb_5
emulated_device_nb_6
emulated_device_nb_7
emulated_device_nb_8
emulated_device_nb_9
emulated_device_nb_10
emulated_device_nb_11
emulated_device_nb_12
emulated_device_nb_13
emulated_device_nb_14
emulated_device_nb_15
emulated_device_nb_16
emulated_device_nb_17
emulated_device_nb_18

Management Infos

Device ID: emulated_device_nb_2

TransportMgmtPolicy LastModifiedTime: Tue, 16 Dec 2014 13:47:57 GMT

appidPolicyMapping:
IPAddress: 172.19.4.129
port: 25000
InitProtocol: *
FinalProtocol: coap

Location LastModifiedTime: Tue, 16 Dec 2014 13:47:57 GMT

85° North, 55° East

Hardware LastModifiedTime: Tue, 16 Dec 2014 13:47:57 GMT

BatteryLevel:
Manufacturer: FOKUS
ModelNumber: DVC-32B-X330

Figure 5.4: Device Management Web GUI

5.4 Documentation

The important function documentation is added in the python files. Further documentation is also added in internal OpenMTC wiki page.

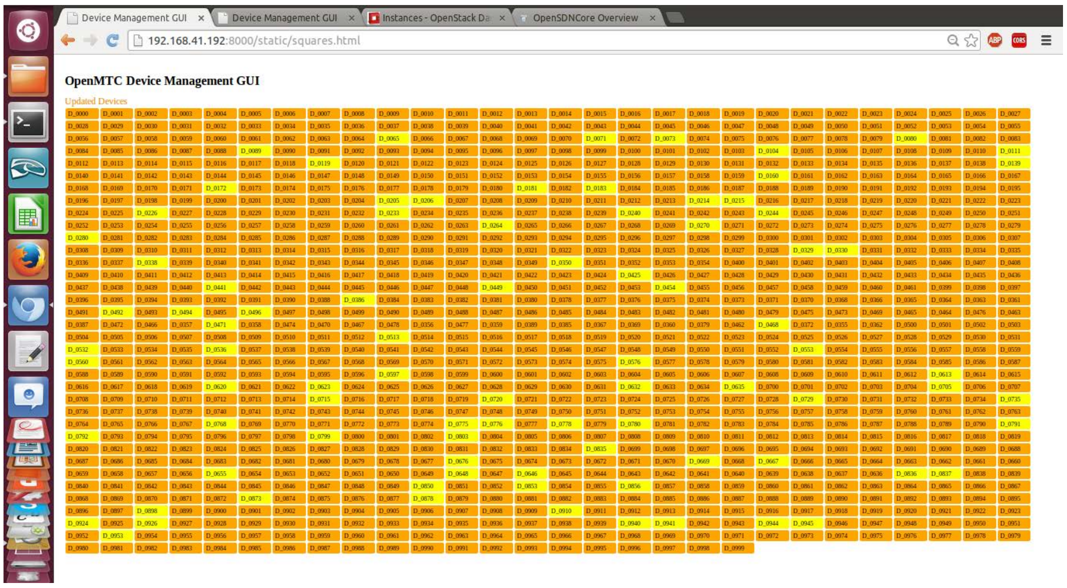


Figure 5.5: GUI showing Emulated Devices Resource Updates

6 Evaluation

In this chapter, the implementation of LWM2M Device Management is evaluated using a setup which is discussed in the section 6.1. It is followed by some data analysis and performance measurements under different environments.

6.1 Test Environment

A demo setup related to Device Management at FOKUS FUSECO[1] Forum 2014 is used to carry out tests for the evaluation and performance measurements. The figure 6.1 illustrates the demo setup.

The virtual machines in the OpenStack are used to run the M2M Server and Emulated devices in different datacenters managed by OpenSDNCore/Orchestrator. Unlike local machine that uses localhost, these virtual machines use 172.19.0.0/16 for communication.

A 1000 emulated devices are registered along with three management objects TransportMgmtPolicy, Device and Location in a short span of time. This triggers the internal events in the M2M Gateway DM Adapter. Then, 1000 LWM2M DM Clients are created which are registered to the LWM2M DM Server. The M2M Server DM Adapter translate these messages in CoAP to ETSI compliant messages to be registered in M2M Server. The web GUI displays the information regarding all the emulated devices in web browser when it is notified of the updates on *mgmtObjs*.

When there is a Network Application(NA) topology request, the Orchestrator in the OpenSDNCore deploys it and when the state becomes *Ready*, the transport policy information is forwarded to the Connectivity M2M application. The container TransportManagement of Connectivity M2M application contains all the information about transport policy which is further saved in the TransportMgmtPolicy object under *mgmtObjs* resource. This update triggers a handler in the M2M Server DM Adapter which translate the information and sends a CoAP message with Write Operation to the LWM2M DM Server. The LWM2M DM Server updates its resource tree and forwards the Write operation request to the LWM2M DM Client. The Client also updates its resource tree. The transport policy is further sent to Transport Domain via the M2M Gateway DM Adapter. Now, the Transport Domain has the information about which NA to send the data from the device. When the device generates the data, the Transport Domain retargets it to the correct NA. Furthermore, all these information updates on the management

[1]http://www.fokus.fraunhofer.de/fb1668ca4a126cd2/5th-fokus-fuseco-forum

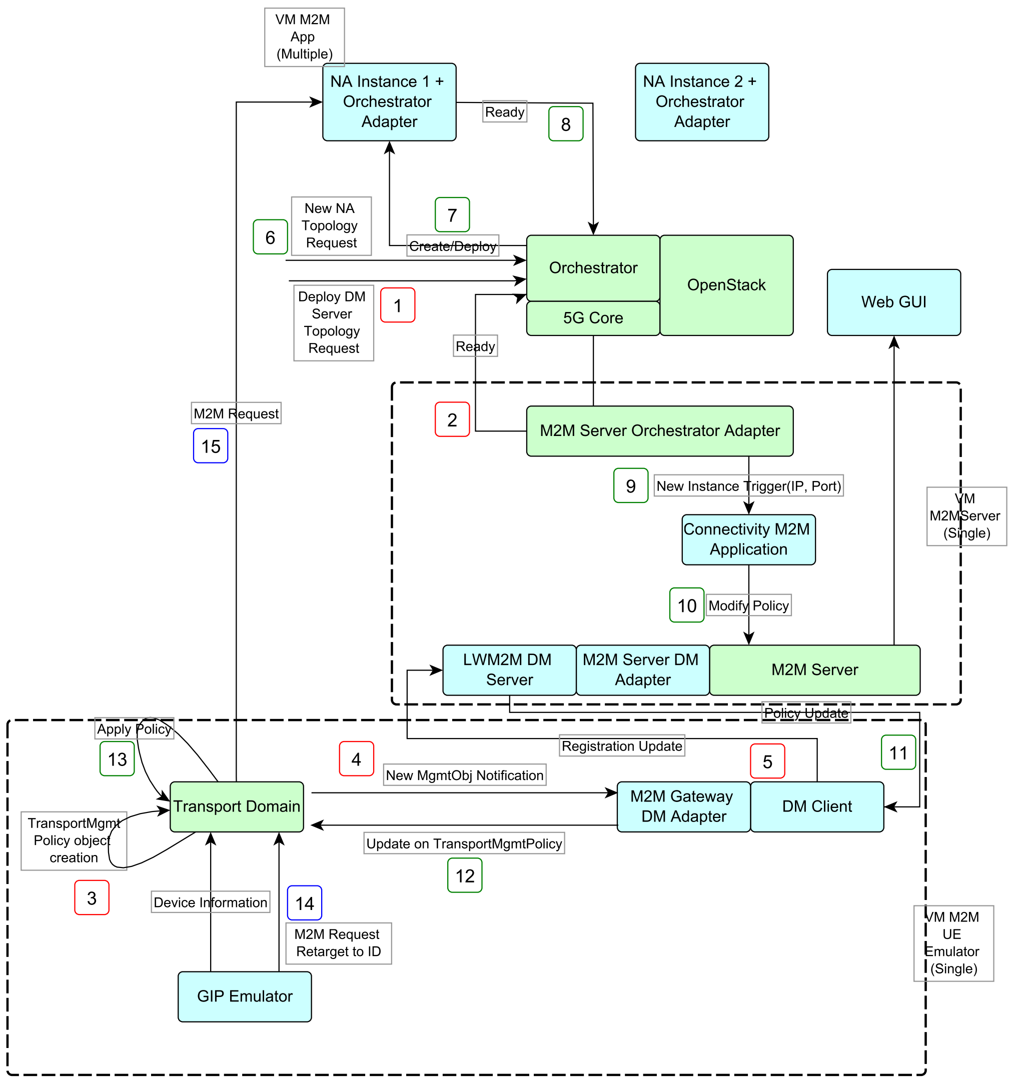


Figure 6.1: Overall Setup for Emulation of Devices in Device Management for Scalability Test

objects can be visualized in the web GUI attached to the M2M Server.

6.2 Scalability

The setup in the figure 6.1 is to test the scalability of LWM2M Device Management to handle the devices. A 1000 emulated devices are generated that depict the real devices with instantiation of three management objects that characterize these emulated devices. Initially there were issues with the sockets being re-used. It was resolved using semaphore to allow every CoAP request to have different ports. So, every time a port is assigned from a pool of ports to each module. This really enhanced in scaling the number of emulated devices that can be used. As the 1000 emulated devices start their registration almost all together, there is a huge traffic on the LWM2M DM Server to handle the simultaneous requests coming from the Clients. Hence, few of the requests aren't served. The CoAP library has a feature to re-transmit the requests. This is explained in the section 6.3. Also, instead of sending small payload sized request every time, bundling of the messages can be done but not to exceed the size of payload too high in every request. This reduces the port usage and the number of requests sent to the LWM2M DM Server. Hence, the use of different ports for requests, re-transmission feature, randomized back-off timer, bundling of messages to make appropriate sized payload are some of the possible ways that we encounter which helps to scale the number of emulated devices.

6.3 Congestion Control

The congestion can be observed at the server side when there are many requests arriving at the same time. This causes the request queue size to be full and additional incoming requests get lost. Hence, these requests are never acknowledged. In the client side, a time-out is maintained in the CoAP library after each request is sent, after which if not acknowledged results in the re-transmission of the request. The re-transmission of the request is done after a certain time interval and it is randomized using a small mathematical formula. Every time there is a re-transmission, the waiting time is increased for that particular request. This randomization in time cause the re-transmission of the requests to be spread over time, making them possible to reach the server at different times. This reduces the load in the server at any instant of time. This increases the efficiency of the server and less re-transmission of the requests is achieved. This also helps to reduce the utilization of the network bandwidth.

6.4 Performance Measurements

The performance evaluation and measurement helps to know the quality of the system. It can be done in various ways by plotting the data and showing them in graphs, in tabular form, data comparisons and etc. The Wireshark[2], Tshark[3] and TCPDump[4] are used to collect the packets and process during the analysis.

There are two scenarios considered to evaluate the performance. The same experiment is carried out in the local machine and between the virtual machines in the OpenStack under different conditions such as with uniform back-off timer and random back-off timer. This helps to understand the behavior of the system in different environments. All the experiments are repeated three times and their average are taken into account. A total

[2]www.wireshark.org
[3]https://www.wireshark.org/docs/man-pages/tshark.html
[4]www.tcpdump.org

of $60(20 \times 3)$ wireshark traces for each case resulting to $240(60 \times 4)$ wireshark traces for four different cases are processed to generate the results. To calculate the total round trip time of requests, a bash script is used.

6.4.1 Total Round Trip Time of Requests

The total round trip time of a request is the aggregation of RTT for the basic registration and the registration of the three management objects as mentioned before. The six plots are created with 300, 600, 1000 emulated devices, three with uniform+random back-off timer in local machine, three with uniform+random backoff timer in virtual machines.

Comparing the plots in the figures 6.2, 6.3 and 6.4, 6.5 and 6.6, 6.7, we observe that as the number of emulated devices increase, the total RTT also increase from around 0.5 seconds to 20 seconds. This is because of the more re-transmissions of the requests as the server becomes overloaded at one point and can't acknowledge back before the time-out period. It is also interesting to see that total average RTT is less in local machine compared to virtual machine. In general, in the local machine, the requests reach the server quite fast, get processed and are acknowledged back before the time-out period preventing most re-transmissions.

Each plot consists of two curves based on random back-off timer and uniform back-off timer. The random back-off timer helps to randomly distribute the sending of the requests over time to lower the load in the server. The uniform back-off timer also distributes but the block of requests over time which might affect the server performance. It can't be determined exactly at which point the re-transmissions occur heavily but from the figures 6.2, 6.3 and 6.4, 6.5 and 6.6, 6.7, it can be observed that at certain intervals, there are peak points indicating loss of the requests(server overloads) resulting into more re-transmissions and ultimately higher average RTT. It can be seen that using the random back-off timer improves the performance by random distribution of requests and hence, lowering the total average RTT in almost all cases.

6.4.2 Average Re-transmissions per Request

When the acknowledgement isn't sent back as a response within a specified time-out period defined in CoAP library, the request has to be re-transmitted. It could be because of several reasons like the request is lost on its way or never reached the server, limitation in the server's request queue size. In this experiment, the latter reason is quite dominating. The maximum number of re-transmissions per request is set to 15. It means each request can be re-transmitted maximum 15 times upon failure. The experiments are carried out in Local Machine and in Virtual Machines deployed in OpenStack. Two scenarios are considered: Uniform back-off timer and Random back-off timer to study the behavior of re-transmissions. Each experiment is performed three times to have a better average data readings. The total time taken for each experiment isn't

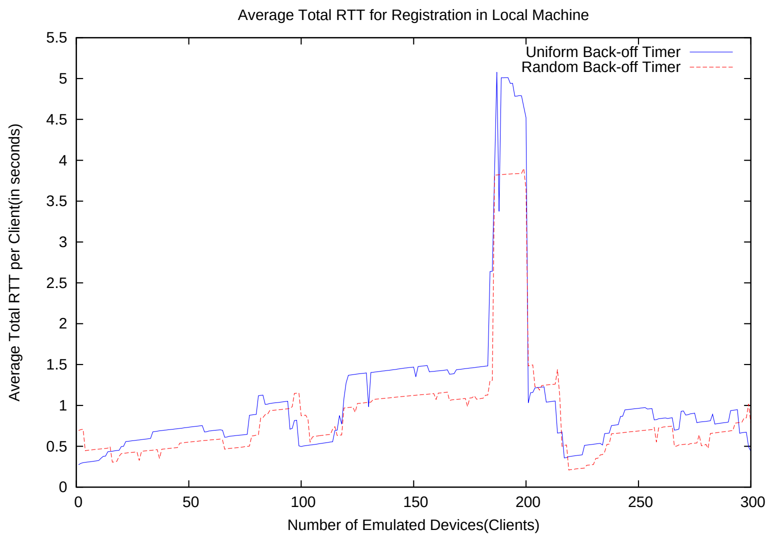


Figure 6.2: Total RTT of Client Registration for 300 Emulated Devices in Local Machine

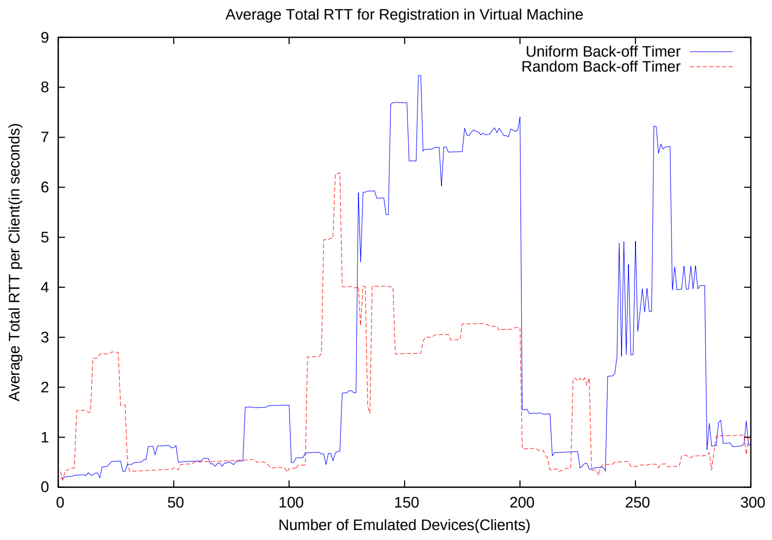


Figure 6.3: Total RTT of Client Registration for 300 Emulated Devices in Virtual Machine

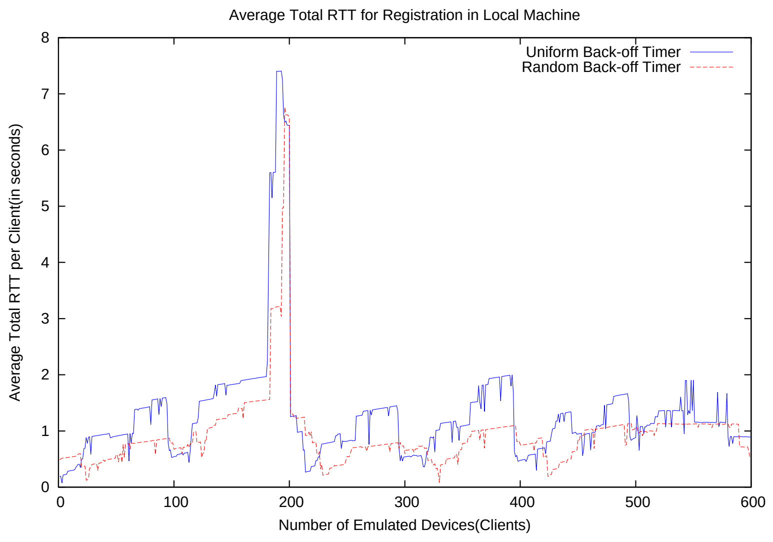


Figure 6.4: Total RTT of Client Registration for 600 Emulated Devices in Local Machine

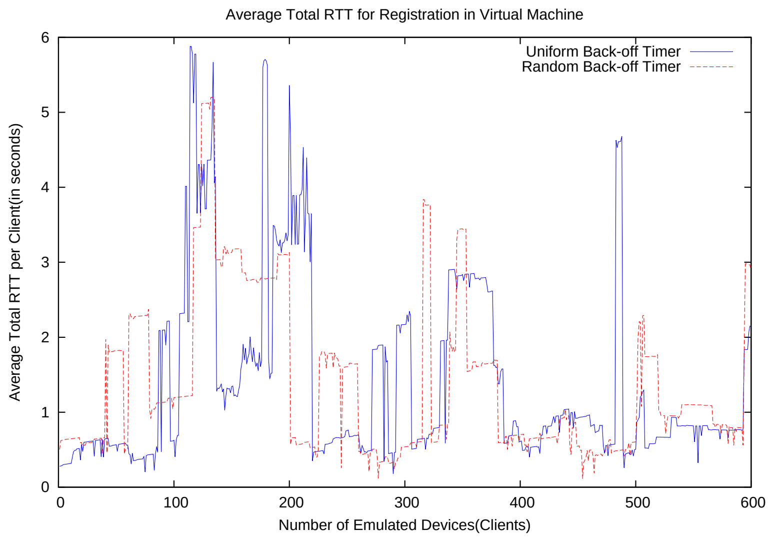


Figure 6.5: Total RTT of Client Registration for 600 Emulated Devices in Virtual Machine

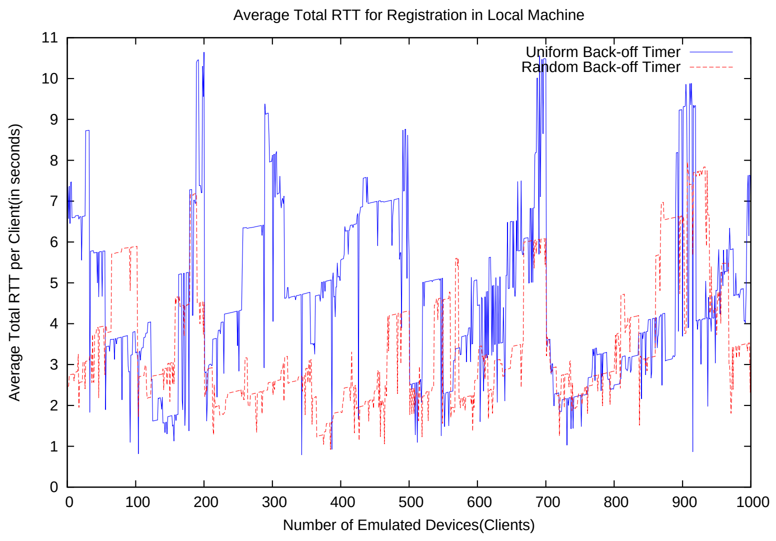


Figure 6.6: Total RTT of Client Registration for 1000 Emulated Devices in Local Machine

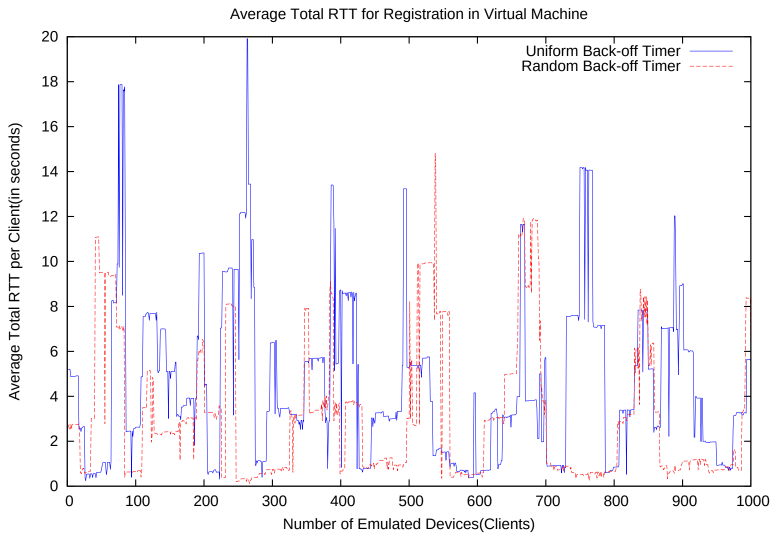


Figure 6.7: Total RTT of Client Registration for 1000 Emulated Devices in Virtual Machine

taken into consideration for this case. Also, the launch time of the DSCLs that emulate 500 devices each also varies for each experiment which might slightly affect the outcomes.

The plot in the figure 6.8 compares the average re-transmission per request experimented in local machine. It can be seen that using random back-off timer, the average re-transmission per request is less. In uniform back-off timer, a block of requests are re-transmitted at the same time causing the server to handle all of them at almost the same time. The random back-off timer spreads the arrival of the requests to the server over time, which reduces the load in the server. Hence, less re-transmissions per request can be achieved. The higher spike at around 900 emulated devices can be explained by the fact that at that time, more re-transmissions occurred which is totally a random case.

The plot in the figure 6.9 compares the average re-transmission per request experimented in virtual machine. Here, also, the experiment with the random back-off timer proven to produce better results against the uniform back-off timer. At around 200-250 emulated devices, the re-transmissions per request is observed to be high as can be estimated that it reached its bottleneck. The peaks in the curve represent more re-transmissions of requests at those points that occur at certain intervals as the server gets overloaded. Later at higher number of emulated devices, the re-transmitted requests are spread over time reducing the average re-transmission per request to a lower value.

The plot in the figure 6.10 and 6.11 compare the average re-transmission per request in local and virtual machines experimented using uniform back-off timer and random back-off timer respectively. In both cases, the average re-transmissions per request in local machine seem less which indicates to a better performance. A series of peaks can be observed in figures 6.10 and 6.11 at intervals which indicate more re-transmissions and server overload. In local machine, the response is sent quite quickly and in most cases, the acknowledgments reaches the clients before the time-out period. Thus, there are less re-transmissions lowering the average re-transmission per request.

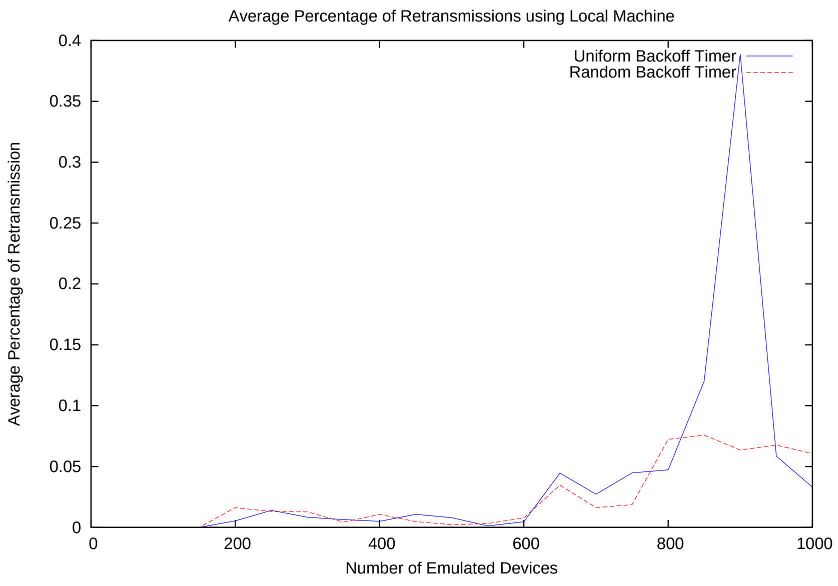


Figure 6.8: Average Retransmissions per Request in Local Machine

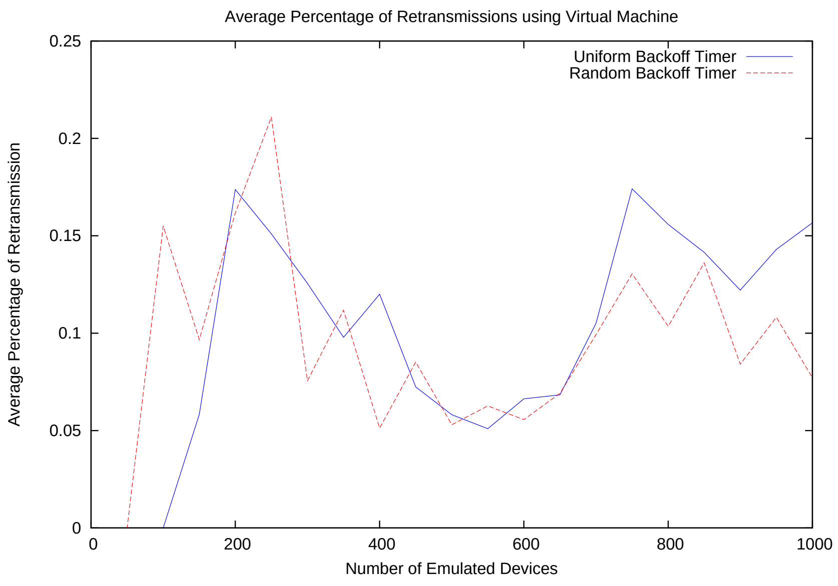


Figure 6.9: Average Retransmissions per Request in Virtual Machine

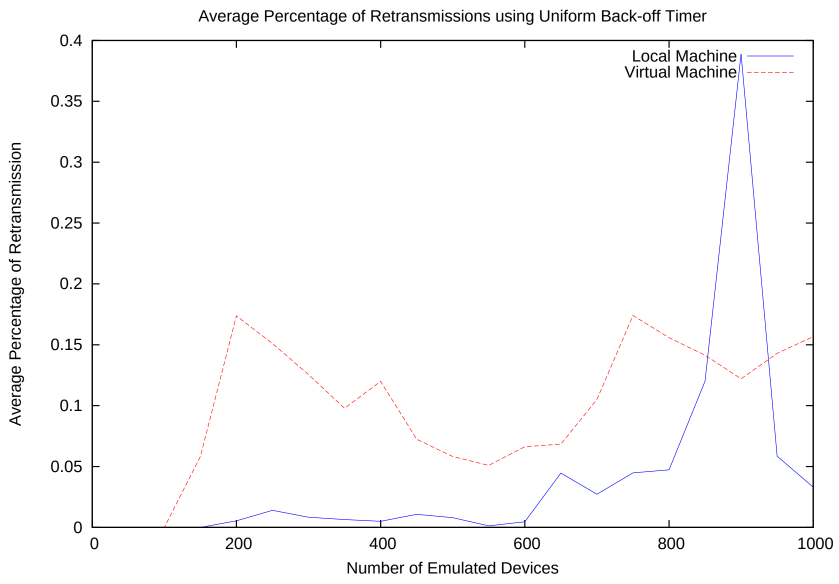


Figure 6.10: Average Retransmissions per Request with Uniform Back-off Timer

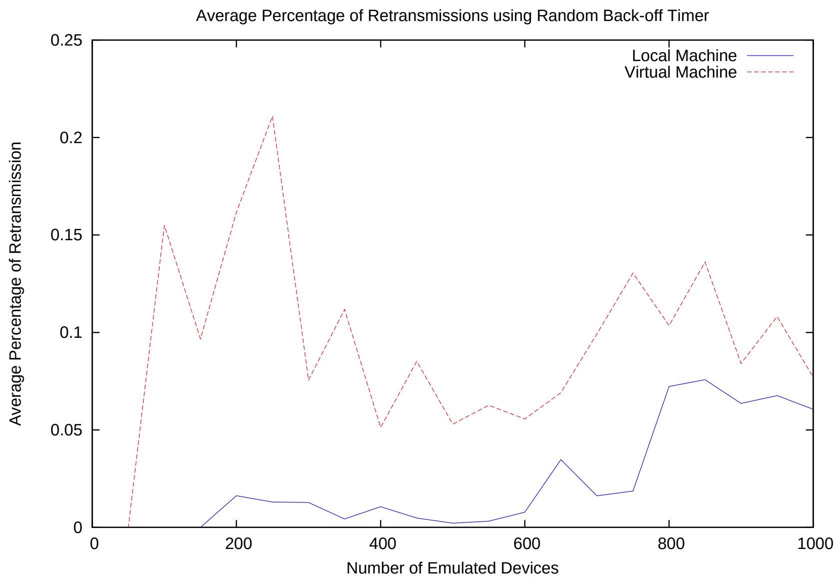


Figure 6.11: Average Retransmissions per Request with Random Back-off Timer

7 Conclusion

This chapter concludes the overall thesis work. It summarizes the concept of machine to machine communication in the field of Internet of Things, detailed implementation and importance of device management platform for controlling the sensor devices, other logical implementation that makes device management more scalable and robust. The summary is followed by other sections that discuss about the implementation fields of the components in real project, problems encountered and limitations, and future work.

7.1 Summary

The Internet of Things is a major breakthrough in the field of technology. The machine to machine communication and machine automation are the promising concepts for the current and future generation. The scope of the IoT is so wide that there is huge potential in every field like Smart Home/City, Surveillance and Security, E-Health and Tele-medicine, Automobiles and Traffic Control and etc. From the business prospective, there is a huge revenue potential. Many research institutions like Fraunhofer FOKUS, T-Labs etc and companies like Cisco, Deutsche Telekom etc are working on ways to make IoT a major success.

The machine to machine communication involves automation of devices. These sensor devices collect information from the environment and send it to the central system through some proper channel like gateways where it is processed and disseminated to the applications that are subscribed to them. These sensor devices are generally characterized by low-powered, low capacity, low communication range of few meters. Hence, in order to reach to the global internet, they should be accompanied by gateways. The gateways manage the attached devices and the part of the front-end of the system. The gateways are connected to the back-end of the system through the internet.

The OpenMTC middleware is a platform for generic machine to machine communication and is based on ETSI M2M and oneM2M standards. It acts as a bridge between different service platforms and the core network to enable seamless communication management of sensors and actuators. It is designed as a horizontal convergence layer for machine type communication and supports vertical application domains. The typical M2M market vertical segments can be deployed as a part of common platform. It consists of service capability layers such as gateway service capability layer and network service capability layer[Fra14]. The components are implemented as the plugins which can be managed using configuration files. One of the components is related to device management which is the topic of this thesis.

As the domain of the IoT is quite huge, the number of sensor devices connecting to the internet is also increasing rapidly. This raises the issue of handling these devices in a more scientific and systematic way. Hence, the need of Device Management enabler is a must for managing and controlling these devices. There are different flavors of device management enablers like OMA-DM, TR-069 that are likely to manage and configure remote devices with high capacity and uses HTTP and TCP for transport. These technologies are still in practice but to support the ever increasing number of devices which are also called constrained devices because of its characteristics being low powered, low capacity, less communication range, there is a need of a different standard. This standard is OMA LightWeight M2M. The OMA LWM2M is characterized by an efficient and scalable object model, supports constrained devices, REST architecture, CoAP using UDP, designated resource descriptions of the objects and etc. Based on this OMA LWM2M standard, the Device Management enabler is written in Python 2.7 and implemented as plugin to the OpenMTC platform.

The OMA LWM2M Device Management consists of Device Management Server and Clients. The DM Clients are connected to the LWM2M DM Server. They exchange CoAP messages to share the management objects information or to send the control information. Each device is represented by a DM Client and characterized by several pre-defined management objects. Both of them maintain an efficient objects hierarchy to store and retrieve the management object information. The Python dictionary data-structure is used for it and its time complexity for data retrieval being O(1) for most operations is an advantage. Both the DM Server and DM Clients maintain listener and client ports to listen and send the requests. There are various DM operations proposed in the standard. The operations such as Read, Write, Create, Delete, Execute, Write Attributes, Discovery, Observation & Notification are implemented based on the standard. In order to make this module inter-operable with OpenMTC, two adapters in the M2M Gateway and M2M Server are implemented. The main task of these adapters is to translate the CoAP messages and ETSI/oneM2M complaint M2M messages.

Adding more to the proposed and implemented Device Management, based on the design in the figure 6.1, the evaluation and performance based on total round trip time, number of retransmissions and scalability of emulated devices are carried out. These factors check the efficiency of the LWM2M Device Management implementation. The experiments are done in the Local Machine and in the Virtual Machines running in OpenStack. The random back-off timer and uniform back-off timer are also considered while re-transmitting the lost or not acknowledged requests. The results are then compared. The random back-off timer improves the system by spreading the requests being sent over time. This lowers the number of requests arriving in the DM Server and reduces the loss of the requests.

Concluding briefly chapter-wise, Chapter 1 introduces Internet of Things, importance of machine communication and automation and their scope in present and future scenario. Chapter 2 introduces the State of the art concepts on various dominating technologies in IoT and governing standards/specifications relating to Device Management. Chapter 3 talks about the technical requirements of the proposed system by illustrating some Use Cases. Chapter 4 discusses about the concept design of the Device Management enabler in details and other involved components. Chapter 5 explains the way LWM2M Device Management and various other components are implemented and merged together for inter-operability. Chapter 6 refers to the evaluation and performance tests of LWM2M Device Management based on various factors like scalability, efficiency, request re-transmissions, total request time etc.

7.2 Dissemination

The implementation of LWM2M Device Management is well integrated into OpenMTC platform. The OpenMTC middleware is used as a generic machine to machine communication platform in European Union(EU) project like Future Internet - Social Technological Alignment in Health Care(FI-STAR) and EU-South Africa collaboration project Testbed for Reliable Smart City Machine-to-Machine Communication(TRESCIMO). The e-Health and tele-medicine is the major concern in FI-STAR project and Device Management plays important role in remote management of devices. The TRESCIMO project is more concerned about making the city smart in the domain of energy consumption, environment, transport, health, education. This project is critical with the fact that urbanization is increasing rapidly and proper management of all the resources is mandatory. This can be envisioned with smart city concept.

Based on the FOKUS FUSECO 2014 demo on Device and Connectivity Management, we are writing a paper for ICO ICT[1] with the title: *Device Management based Software Defined Solution for Provisioning Reliable M2M Infrastructures*[2].

7.3 Problems Encountered

Few problems were encountered during the implementation and evaluation phases. When each GSCL was associated with an emulated device/client, the memory consumption was high as 32 MB. This directly impacted the scalability factor as number of emulated devices that can be experimented at a time dropped to 200-300. This problem was overcome by handling number of emulated devices by each DSCL. This reduced the physical limitation of memory in OpenStack. Hence, thousands of emulated devices could be experimented afterwards. Another problem was related to loss of requests due to the limited queue size in DM Server. The uniform re-transmission of the request

[1] www.icoict.org

[2] Authors: A. Corici, R. Shrestha, G. Carella, A. Elmangosh, R. Steinke, T. Magedanz, Fraunhofer FOKUS.

wasn't of much help because the re-transmitted requests would arrive the DM Server at the same time. Hence, randomization in time was introduced while re-transmitting the requests. This reduced the load in the DM Server and possible requests loss by some proportion. Another problem and limitation was the number of listening sockets that could be opened at a time per a python process. This limitation could be associated with Ubuntu 12.04 itself.

7.4 Outlook

The security of information is a major concern and that is also stated in the LWM2M Device Management standard. The Datagram Transport Layer Security (DTLS) protocol provides communication privacy over UDP. This has to be implemented in connection with the LWM2M Device Management enabler. Another important feature that can be added is the Store and Forward (SAF) for the requests. The Device Management enabler should be able to coordinate with the Connectivity Management module to have a knowledge of existing connection path to reach the destination. If the proper channel or connection is not available, those requests should be buffered. The module should be aware of availability of possible connection and once it is available, the requests should be delivered. This is the concept of connectivity awareness and Store and Forward(SAF). These concepts can further be extended to the TRESCIMO/FI-STAR projects based on their requirements. Regarding the inter-operability test, the LWM2M Client is tested with Leshan Server(described in section 2.3) for the client registration and looking forward to other operational tests.

List of Acronyms

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
CRUD	Create Read Update Delete
DM	Device Management
DSL	Digital Subscriber Line
FOKUS	Fraunhofer-Institut fur Offene Kommunikationssysteme
GUI	Graphical User Interface
GPS	Global Positioning System
GSCL	Gateway Service Capability Layer
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
JDK	Java Developer Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LTE	Long Term Evolution
LWM2M	LightWeight Machine to Machine
M2M	Machine to Machine
MO	Management Object
NGN	Next Generation Network
NSCL	Network Service Capability Layer
OMA	Open Mobile Alliance
OpenEPC	Open Evolved Packet Core
OpenMTC	Open Machine Type Communication
OpenSDNCore	Open Software Defined Network Core
PAN	Personal Area Network
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request For Comments
SDK	Software Developer Kit
SMS	Short Message Service
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier

W3C	World Wide Web Consortium
WiMAX	Worldwide Interoperability for Microwave Access

Bibliography

[Cha13] CHASE, JIM: *The Evolution of the Internet of Things, p1*, 2013.

[ETSa] ETSI: *Machine to Machine Communication.* `http://www.etsi.org/technologies-clusters/technologies/m2m`.

[ETSb] ETSI: *Wikipedia reference on ETSI.* `http://en.wikipedia.org/wiki/ETSI`.

[ETS10a] ETSI: *ETSI TR 102 691 v1.1.1: Machine to Machine Communication(M2M); Smart Metering Use Cases*, 2010. `http://www.etsi.org/deliver/etsi_tr/102600_102699/102691/01.01.01_60/tr_102691v010101p.pdf`.

[ETS10b] ETSI: *ETSI TS 102 690 v1.1.1, Machine to Machine Communication: Functional Architecture*, 2011/2010. Page 17, Technical Specification.

[ETS10c] ETSI: *ETSI TS 102 690 v1.1.1, Machine to Machine Communication: Functional Architecture*, 2011/2010. Page 103, Technical Specification.

[ETS10d] ETSI: *ETSI TS 102 690 v1.1.1, Machine to Machine Communication: Functional Architecture*, 2011/2010. Technical Specification.

[ETS13a] ETSI: *ETSI TR 102 732 v1.1.1: Machine to Machine Communication(M2M); Use Cases of M2M applications for eHealth*, 2013. `http://www.etsi.org/deliver/etsi_tr/102700_102799/102732/01.01.01_60/tr_102732v010101p.pdf`.

[ETS13b] ETSI: *ETSI TR 102 857 v1.1.1: Machine to Machine Communication(M2M); Use Cases of M2M applications for Connected Consumer*, 2013. `http://www.etsi.org/deliver/etsi_tr/102800_102899/102857/01.01.01_60/tr_102857v010101p.pdf`.

[Fel14] FELL, MARK: *Roadmap for The Emerging "Internet of Things"*, 2014.

[Fra14] FRAUNHOFER FOKUS: *Open Machine Type Communication*, 2014. `http://www.open-mtc.org`.

[Har14] HARTKE, K.: *Observing Resources in CoAP draft-ietf-core-observe-16*, 2014.

[HTT] HTTP: *Wikipedia Reference on HTTP.* http://en.wikipedia.org/wiki/ Hypertext_Transfer_Protocol.

[IET12] IETF: *CoRE Link Format*, 2012. https://tools.ietf.org/html/rfc6690.

[Ins13] Institute, McKinsey Global: *Disruptive technologies: Advances that will transform life, business, and the global economy, p51*, 2013.

[KO] Kellmereit, Daniel and Daniel Obodovski: *The Silent Intelligence-The Internet of Things, p14*.

[MC] McEwen, Adrian and Hakim Cassimally: *Designing the Internet of Things, p11*.

[OMA12] OMA: *OMA Device Management Protocol, Candidate Version 1.3*, 2012. `http://technical.openmobilealliance.org/Technical/Release_Program/docs/DM/V1_3-20130422-C/OMA-AD-DM-V1_3-20120306-C.pdf`.

[OMA13a] OMA: *Lightweight Machine to Machine Technical Specification, Candidate Version 1.0*, 2013.

[OMA13b] OMA: *OMA Device Management Protocol, Candidate Version 2.0*, 2013. `http://technical.openmobilealliance.org/Technical/Release_Program/docs/DM/V2_0-20131210-C/OMA-TS-DM_Protocol-V2_0-20131210-C.pdf`.

[one] oneM2M: *Introduction to oneM2M*. http://www.onem2m.org/news-events/news/2-leading-ict-standards-development-organizations-launch-onem2m.

[one14a] oneM2M: *oneM2M Members*, 2014. http://www.onem2m.org/membership/current-members.

[one14b] oneM2M: *oneM2M TS 001- Functional Architecture*, 2014. Page 26, Technical Specification.

[one14c] oneM2M: *TS 001- oneM2M Functional Architecture*, 2014. Page 19, Technical Specification.

[one14d] oneM2M: *TS 005- oneM2M Management Enablement*, 2014. Technical Specification.

[TLS] Tapio Leva, Oleksiy Mazhelis and Henna Suomi: *Comparing the cost efficiency of CoAP and HTTP in Web of Things Applications.*

[ZSB14] Z. Shelby, K. Hartke and C. Bormann: *The Constrained Application Protocol*, 2014.

Annex

Appendix A. LWM2M Objects defined by OMA

The LWM2M Objects defined by OMA LWM2M 1.0 specification are as follows:

Object	Object ID
LWM2M Security	0
LWM2M Server	1
Access Control	2
Device	3
Connectivity Monitoring	4
Firmware	5
Location	6
Connectivity Statistics	7

Some of the important LWM2M Objects are detailed below.

A.1 LWM2M Object: Device

This object provides device related information.

Object Definition

Object	Object ID	Instances	Mandatory	Object URN
Device	3	Single	Mandatory	urn:oma:lwm2m:oma:3

Resource Definitions

ID	Name	Operations	Instances	Mandatory	Type	Range	Units	Description
0	Manufacturer	R	Single	Optional	String			Manufacturer Name
1	Model Number	R	Single	Optional	String			A model identifier
2	Serial Number	R	Single	Optional	String			Serial Number

3	Firmware Version	R	Single	Optional	String			Current Firmware Version
4	Reboot	E	Single	Mandatory				Reboot LWM2M Device
5	Factory Reset	E	Single	Optional				Perform factory reset on device
6	Available Power Sources	R	Multipe	Optional	Integer	0-7		0-DC power, 1-Internal Battery, 2-External Battery, 4-Power over Ethernet, 5-USB, 6-AC power, 7-Solar
7	Power Source Voltage	R	Multiple	Optional	Integer		mV	Present voltage of each resource instance
8	Power Source Current	R	Multiple	Optional	Integer		mA	Present current of resource instance
9	Battery Level	R	Single	Optional	Integer	0-100	%	Contains current battery level percentage
10	Memory Free	R	Single	Optional	Integer		KB	Current available storage space
11	Error Code	R	Multiple	Mandatory	Integer			0-No Error, 1-Low battery power, 2-External power supply, 3-GPS module failure, 4-Low received signal, 5-Out of memory, 6-SMS failure, 7-IP connectivity failure, 8-Peripheral malfunction
12	Reset Error Code	E	Single	Optional				Delete all error code resource instances and set error code-0

13	Current Time	RW	Single	Optional	Time			Current UNIX time of LWM2M Client
14	UTC Offset	RW	Single	Optional	String			Indicates UTC offset currently in effect for LWM2M Device.
15	Timezone	RW	Single	Optional	String			Indicates where LWM2M is located in IANA Timezone format.
16	Supported Binding and Modes	R	Single	Mandatory	String			Indicates the supported binding and modes in LWM2M Client.

A.2 LWM2M Object: Location

This object provides device related information.

Object Definition

Object	Object ID	Instances	Mandatory	Object URN
Location	6	Single	Optional	urn:oma:lwm2m:oma:6

Resource Definitions

ID	Name	Operations	Instances	Mandatory	Type	Range	Units	Description
0	Latitude	R	Single	Mandatory	String		Deg	Decimal notation of latitude
1	Longitude	R	Single	Mandatory	String		Deg	Decimal notation of longitude
2	Altitude	R	Single	Optional	String		m	Height from sea level (in meters)
3	Uncertainty	R	Single	Optional	String		m	Accuracy of position (in meters)
4	Velocity	R	Single	Optional	Opaque			Velocity of device as defined in 3GPP 23.032 GAD specs. The device is static if value is not set.

5	Timestamp	R	Single	Mandatory	Time	0-6		Timestamp when location measurement was done

A.3 LWM2M Object: TransportMgmtPolicy
This object provides device related information.
Object Definition

Object	Object ID	Instances	Mandatory	Object URN
DeviceMgmtPolicy	4300	Single	Optional	urn:oma:lwm2m: ext:4300

Resource Definitions

ID	Name	Operations	Instances	Mandatory	Type	Range	Units	Description
0	appidPolicy Mapping	RW	Single	Mandatory	String			ID of the application for mapping
1	IPAddress	RW	Single	Mandatory	String			IP address of the application
2	port	RW	Single	Mandatory	String			Port on which application is listening
3	InitProtocol	RW	Single	Optional	String			Initial Protocol being used
4	FinalProtocol	RW	Single	Mandatory	String			Final Protocol to be used

A.4 LWM2M Object: DeviceCapability
This object provides device related information.
Object Definition

Object	Object ID	Instances	Mandatory	Object URN
DeviceCapability	4200	Single	Optional	urn:oma:lwm2m: ext:4200

Resource Definitions

ID	Name	Operations	Instances	Mandatory	Type	Range	Units	Description
0	Property	R	Single	Mandatory	String			Property Name
1	Group	R	Single	Optional	String			Group Name
2	Description	R	Single	Optional	String			Description of Device Capability
3	Attached	R	Single	Optional	Boolean			Indicates if it is currently attached to the device
4	Enabled	R	Single	Mandatory	Boolean			Indicates if Device Capability is enabled or not
5	OpEnable	E	Single	Mandatory				Command use to transfer Device Capability to transfer from Disabled to Enabled state
6	OpDisable	E	Single	Mandatory				Command use to disable Device Capability to transfer from Enabled to Disabled state
7	DenyUserEn		Single	Optional	Boolean			Specifies if the user is able to enable Device Capability
8	NotifyUser		Single	Optional	Boolean			Specifies whether the user is notified when enable/disable primitive is executed