The Design and Implementation of OMA RESTful Location Services in Wireless Sensor Environments

Md. Asadul Islam

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

April 2012

©Md. Asadul Islam, 2012

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis proposal prepared

By: **Md. Asadul Islam**

Entitled: **The Design and Implementation of OMA RESTful Location Services in Wireless Sensor Environments**

and submitted in partial fulfilment of the requirements for the degree of

Master of Applied Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

____________________________________Dr. Sheldon Williamson, Chair

____________________________________Dr. N. Bouguila, External examiner

____________________________________Dr. A. Agarwal, Internal Examiner

____________________________________Dr. F. Khendek, Supervisor

____________________________________Dr. R. Glitho, Supervisor

Approved by____

William E. Lynch, Chair of Department, Electrical and Computer Engineering

____

Robin Drew, Dean of the Faculty of Engineering and Computer Science

Abstract

The Design and Implementation of OMA RESTful Location Services in Wireless Sensor Environments

Md. Asadul Islam

Open Mobile Alliance (OMA) RESTful location services are standard RESTful Web services for terminal location. They are location technology-independent and enable applications' portability and interoperability. Wireless sensors are electronic devices that can sense context: space, environment and physiology. Location is a key element of space context information. Wireless sensors can sense location with a level of accuracy that most other technologies cannot provide, which has made them the technology of choice for several applications. This thesis is about the design and implementation of OMA RESTful location services in wireless sensor environments for improved accuracy. A novel architecture is proposed. The architectural components and operational procedures are defined and implemented. The proof of concept prototype has been realized, along with the measurements for a preliminary performance evaluation. Several lessons were learned. For instance, it is possible to map location information for each of the OMA services to the sensor-based location information. However, using geographic coordinates (i.e. geographic latitude, longitude and altitude) to describe terminal location does not match with the fine-grained location accuracy provided by WSNs.

Acknowledgements

This thesis is a part of research work done in the Telecommunication Service Engineering (TSE) research lab. This research project would not have been possible without the support of many people. It is my pleasure to thank those people who have helped me and made the completion of this thesis possible. First, I would like to express my heartfelt gratitude to my supervisors, **Dr. Ferhat Khendek** and **Dr. Roch Glitho**, for their inspiration, valuable advice and continuous support during the thesis. Their critical comments, considerate attitude and immense knowledge helped me a lot during my research and thesis-writing. Their patience and willing to help their students, made my research life remarkable. Their constructive ideas played a vital role for the accomplishment of this thesis.

I would like to give special thank **Dr. Fatna Belqasmi** who helped me a lot during my research and gave me clear guidance in every stage of my research and saved me a lot of time. I am also grateful to all of my friends and teammates in TSE lab: **Majid, Razi**, for their constructive ideas, comments and sharing of knowledge and experiences with me.

I am also grateful to my brother **Sayed Hafizur Rahman**, for his encouragement, solid guidelines and constructive ideas in every stage of my thesis. His valuable advice was always a source of my motivation in my research. His generosity to help me in every stage of my life means a lot to me.

I would like to acknowledge the financial support from Concordia University, Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson Canada Inc.

Last, but not least, I would like to thank my wife, **Sayeda Marina Khatun**, for her understanding and love during the past few years. Her support and encouragement was in the end what made this thesis possible. My parents, **Md. Nur Ali Sarder** and **Rabiya Khatun**, and family members receive my deepest gratitude and love for their dedication, many years of support during my undergraduate studies that provided the foundation for this work and the opportunity to be where I am.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ADC	Analog to Digital Converter
A-GPS	Assisted Global Positioning System
API	Application Programming Interface
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GSMA	GSM Association
HAN	Home Area Network
HGD	Home Gateway Device
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IPC	Inter-process Communication
JAXB	Java Architecture for XML Binding
JCP	Java Community Process
JSON	JavaScript Object Notation
KB	Kilobyte
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
OGC	Open Geospatial Consortium
OMA	Open Mobile Alliance
OSGi	Open Services Gateway Initiative
PC	Personal Computer
PDA	Personal Digital Assistant

REQ	Request Message for Single Terminal
REQM	Request Message for Multiple Terminals
RES	Response Message for Both Single and Multiple Terminals
REST	Representational State Transfer
RF	Radio Frequency
RFID	Radio Frequency Identification
ROA	Resource Oriented Architecture
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SWF	Slide Window Filter
TCP	Transmission Control Protocol
TDOA	Time Difference of Arrival
UDP	User Datagram Protocol
UKF	Unscented Kalman Filter
URI	Uniform Resource Identifier
URL	Universal Resource Locator
W3C	World Wide Web Consortium
WADL	Web Application Description Language
Wi-Fi	Wireless Fidelity
WSN	Wireless Sensor Network
WWBAN	Wearable Wireless Body Area Network
WWW	World Wide Web
XHTML	Extensible Hyper Text Markup Language
XML	Extensible Markup Language

Chapter 1 Introduction

1.1 Research Domain

A Wireless Sensor Network (WSN) is formed by a set of small sensors equipped with processors, memory and short range wireless communication capabilities. Sensors are responsible to collect information and aggregate data regarding the physical phenomena under observation. A conventional WSN architecture includes three major components: sensor, sink and gateway. The sensors are responsible for actual sensing, while the sinks collect information from all the sensors and send them to the application through the gateway. The gateway acts as a dual network interface, a link between the WSNs and the outside worlds by performing the protocol translation and necessary mapping. Sink-less WSN architecture is another potential scenario where sensors are responsible for communicating with external network(s) or application(s) directly as done in [1].

A sensor is a source of contextual data such as spatial (e.g. location), environmental (e.g. temperature, humidity and light etc.) and physiological (e.g. heartbeat, body temperature, blood pressure and glucose level etc.). WSNs are deployed in a wide range of application domains including healthcare applications, environmental monitoring, military applications, commercial applications, home automation, structural monitoring, exploration, and agricultural applications. To accurately locate a specific person or a specific object in many of these applications are very noteworthy. In this occasion, the sensors responsible for collecting location data of a specific object or a particular person such as MIT (i.e. Massachusetts Institute of Technology) Cricket sensor [2] are known as location sensor. The information collected by the location sensor is called location information which will be relayed to a fixed and centralized gateway. After that the gateway will transmit the information to the end-user applications residing in infrastructure-based networks through standard or non-standard interfaces. The provided location data can be used in many value-added services.

Academia and industry have a lot of interest in this research by introducing many exciting issues in hardware, networking, and application level. Designing of efficient routing protocols for power consumption optimization of sensor nodes, robustness and security of the network, distributed data processing, and mobility and dynamicity of different WSN entities are currently ongoing research [7].

Location services are defined as the services that could deliver information about the current geographic location (e.g. spatial coordinates or civic location) of a person or a mobile device as well as mobile asset. The term “location services” refers to mobile services that integrate mobile device’s location or position with other information in order to add value to the service as a whole [4]. The location information in that case could be either geographical coordinates (i.e. latitude and longitude) or civic address (i.e. EV.15-163, Concordia University) generated by any given location technology such as WSNs, Cell-Id, Global Positioning System (GPS) etc. Location services are becoming more popular and very useful in our everyday life. Location plays a vital role in many location based applications. A task recognized by location services in healthcare system includes keeping track of patients with critical situation, doctors, nurses, and expensive devices in order to provide better and faster service. Another task acknowledged in large museums and exhibitions take account of, keeping track of visitors, guiding them inside the museums and exhibition halls, and finding lost friends and family members and the nearest washroom based on their current locations. An overview of location based applications and the level of accuracy required for each of the application is well presented in [4]. Providing more accurate and standard location services at the application level is still a challenging issue.

There are a few standard and non-standard location services. Open Mobile Alliance (OMA) defined standard RESTful location services. These are standard RESTful web services for terminal location that follows the Representational State Transfer (REST) architectural style [5]. They are location technology independent and enable applications portability and interoperability. OMA RESTful location services deliver information (i.e. location, distance) about the current geographic location of any mobile terminal (e.g. Cell phones, PDAs, and Laptops etc.) in order to provide added value to a user.

1.2 Problem Statement and Motivations

OMA RESTful location services are standard Web services for terminal location that follows the REST design style. OMA RESTful location services are location technology-independent and enable applications portability and interoperability. A list of technologies that provide location information include GPS and Cell-Id of the currently serving cell in the cellular network. GPS is one of the most well-known technologies [6]. It works very well in outdoor environments but its usage in indoor environments is limited as the signals from GPS satellites are too weak to penetrate most of the buildings. Furthermore, along as the Cell-Id, the GPS does not provide the enough accuracy required by several applications such as people/asset tracking, healthcare environments and pervasive gaming.

Another alternative to get location information is using WSNs. Sensors can sense location with an accuracy (e.g. within 1-3 centimetres) that most technologies cannot provide, making them the technology of choice for applications that require high accuracy. To illustrate the potential benefits of accuracy that can be obtained from the implementation of OMA RESTful location services in wireless sensor environments, let us consider the following situation when someone enters a large building (airport, museum, shopping mall, fair center etc.) that he/she is not familiar with. He/she may get lost. One potential solution is that this person could use his/her cell phone or a PDA equipped with a client application based on OMA RESTful location services to locate herself/ himself. The application can also guide the person step by step to visit the whole building via the user interface (with a floor plan). A user could also use the same application to find his/her friends, relatives and colleagues inside a building, and can even ask to be notified when some of her/his friends/relatives/colleagues are located in the proximity of their current location in a specific area or within a specific distance. Other services can be provided, such as locating meeting rooms, the nearest toilets or auto-teller machines.

Our main research goal is to implement OMA RESTful location services in wireless sensor environments in order to provide more accurate location information. There is no ready-to-use architecture for providing OMA RESTful location services in wireless sensor environments.

The highlight of our research work is an architecture for providing OMA RESTful location services in wireless sensor environments. We design, implement this architecture and perform preliminary performance evaluation.

1.3 Contribution of the Thesis

The main contributions of the thesis are as follows:

- We performed a detailed survey and evaluation of the state-of-the-art on the implementation of location services with existing technologies.
- More important, we developed an architecture for the implementation of OMA RESTful location services in wireless sensor environments. As part of this architecture, we designed and implemented a WSN gateway architecture based on MIT Cricket sensors [2].
- In order to establish a communication with the WSN gateway we derived a set of requirements for designing the interfaces between REST gateway and WSN gateway. We defined an UDP-based interface between REST gateway and WSN gateway that met all of our requirements.
- We implemented the proof-of-concept prototype for the design and implementation of OMA RESTful location services in WSN environments. We implemented all the architectural components of REST gateway, and WSN gateway. All the OMA defined services have also been implemented.
- We evaluated the performance of our system based on four different performance metrics such as end-to-end delay, server capacity, network load, and bandwidth consumptions. We gathered all the performance data using the Apache JMeter [8], an open source testing tools.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows:

Chapter 2 provides the necessary background information on WSN, REST and RESTful web services, and OMA RESTful location services which are required for understanding this research work.

Chapter 3 delivers a detailed review of the state-of-the-art on the implementation of location services with any existing location technology including WSN. We conclude this chapter by describing the requirement/criteria for the design and implementation of OMA RESTful location services in wireless sensor environments.

Chapter 4 elaborates on the detailed design and implementation of OMA RESTful location services in wireless sensor environments. Firstly, we described the design and implementation of the architectural components of a REST gateway. Secondly, we explained the design and implementation of a WNS gateway. We conclude this chapter by discussing the overall implementation of the system.

Chapter 5 is devoted to the prototype implementation as a proof-of-concept for the design and implementation of OMA RESTful location services in wireless sensor environments. The performance evaluation and related measurements are also presented. This chapter is concluded with the discussion of performance metrics and result analysis.

Finally, Chapter 6 concludes this research work with some potential future remarks.

Chapter 2 Background Information

This chapter focuses on necessary background information on WSNs, REST and RESTful web services, and OMA RESTful Location services.

2.1 Wireless Sensor Networks (WSNs)

In this section we first introduce WSNs as well as WSNs protocol stack. Afterwards, WSN architectures, hardware components and software components of a sensor node are discussed. We conclude this section by describing WSN applications and WSN challenges.

2.1.1 Introduction

A sensor is a tiny electronic device that can measure, detect and gather data from physical environment or surroundings and converts it into an electrical signal for further processing and communication to other network entities. The so-called sensor nodes are equipped with on-board multifunctional miniature sensors, processor, and communication components which are low-cost and low-power intelligent devices that are small in size and communicate untethered in short distance. Due to the recent advancements in hardware and wireless network technologies most of the sensor devices are equipped with wireless interface for inter-node communication. The collaboration between sensors makes up a network defined as Sensor Network or Wireless Sensor Networks (WSNs). In short, WSNs is formed by a set of sensors equipped with processors, memory and short range wireless communication capabilities [7]. Modern sensor networks represent significant improvements over traditional sensors, which are deployed in the following two ways [7] [9]:

- Sensors can be positioned far from the actual phenomenon (i.e. something known as sense perception). In this approach, a large sensor, that may use some complex techniques to distinguish the targets from environmental noise, is required.
- Several sensors that perform only sensing can be deployed. In this case, the positions of the sensors and communications topology need to be carefully engineered. They transmit timed series of the sensed phenomenon to the central nodes where the actual computations are performed and data are fused.

The positions of the sensor node do not need to be engineered or pre-determined. We can deploy them randomly in inaccessible topographies or disaster relief operation without any human intervention or configuration. WSN network protocols and algorithms must own self-organizing capabilities. Sensor nodes are cooperative with each other. They can process the raw data instead of sending them to the node responsible for fusion using their on-board processing capabilities in order to carry simple computations and convey only the desired and partially processed data. The aforementioned features of WSN open the door for a wide range of new applications. A variety of potential applications which are categorized into health, military, environmental and commercial applications.

2.1.2 Wireless Sensor Network (WSN) Protocol Stack

Many protocols have been developed for WSNs considering energy as a critical issue. A generic WSN protocol stack consists of a set of communication and management protocols are shown in Figure 2.1. The communication protocol stack is made as usual of physical, data-link, network, transport and application layer. The physical layer is responsible for sensing, actuation and signal processing. The link layer is responsible for channel sharing and timing issues where the network layer is responsible for topology

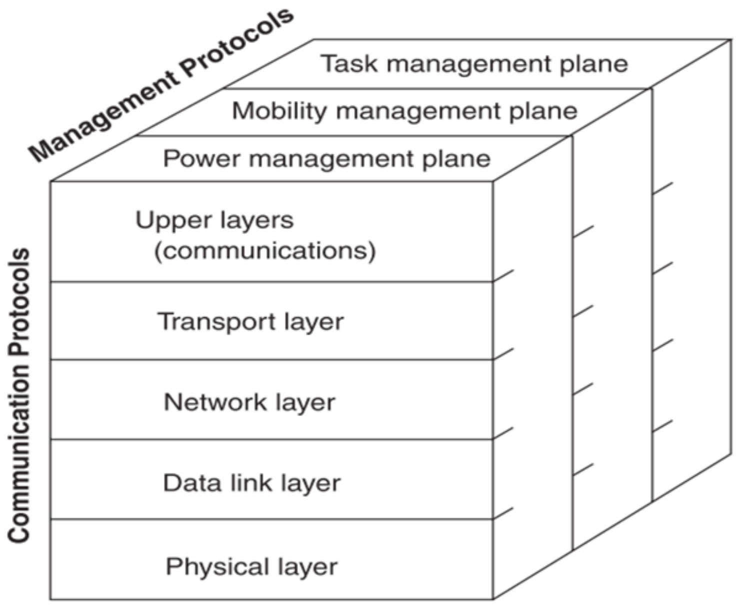


Figure 2.1: A generic WSN protocol stack

management and routing data supplied by transport layer, which is responsible for data dissemination, accumulation, cashing and storage. The application layer is responsible for application processing, data aggregation, and query processing. Depending on the sensing capabilities, different types of application software can be built and used in this layer. The management protocol stack consists of power, mobility and task management plane which operate across all layers. The management protocol stack will assist sensors in coordinating the sensing and lower the overall power consumption. Sensor nodes power is controlled by power management plane. For instance, the sensor may keep its radio turn off after receiving a message from one of its neighbors in order to avoid receiving duplicate message. When the sensor is running low in energy, it will inform low power to its neighbors and not be able to participate in routing messages. The remaining power of the sensors suffering from low energy is reserved for sensing only. The mobility management plane is responsible for maintaining the full operation of sensor mobility. This will aid a sensor node to detect the route by keeping track of its neighbors. The task management plane should be capable of coordinating all nodes toward a common objective in a power-aware manner [7] [10].

2.1.3 Wireless Sensor Network (WSN) Architectures

The design of the WSNs architecture is influenced by many factors such as sensing, fault tolerance, hardware constraints, network topology and power consumption etc. All these factors are well explained in [7]. Figure 2.2 depicts a typical WSN network architecture which consists of three main components: sensors, sinks and gateways. The

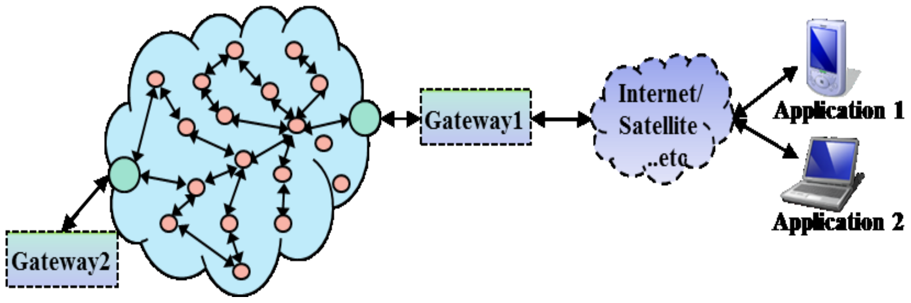


Figure 2.2: A typical wireless sensor network

sensors are responsible for actual sensing, while the sinks collect information from all the sensors and send them to the application through the gateway. The gateway acts as a dual network interface, a link between the WSNs and the outside worlds by performing the protocol translation and necessary mapping. A more detailed description of WSNs architectural entities are given in [1]. Sink-less WSN architecture is another potential scenario where sensors are responsible for communicating with external network(s) or application(s) directly as done in [1].

2.1.4 Hardware Components of a Sensor Node

The significant hardware components of a sensor node are shown in Figure 2.3. The components are: sensing unit, processing unit, communication unit and power unit.

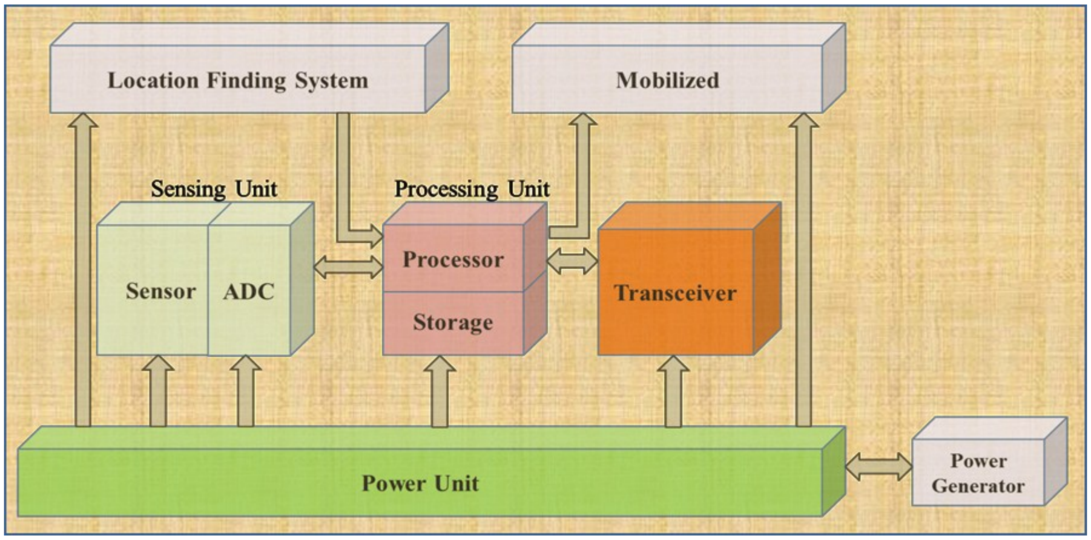


Figure 2.3: Hardware components of a sensor node

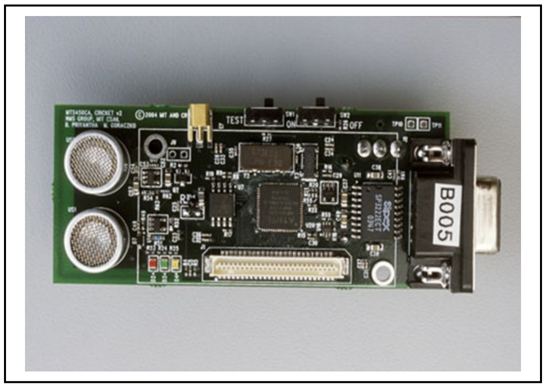
Sensing Unit: The sensing unit comprises of two sub modules: single or multifunctional sensors and a converter called Analog to Digital converter (ADC). Sensors are responsible for sensing physical phenomenon such as light and temperature, and also produce analog signal based on the observed phenomenon. The ADC is responsible for converting sensed analog signals into digital signals. After that the converted digital signals fed into the processing unit for further processing.

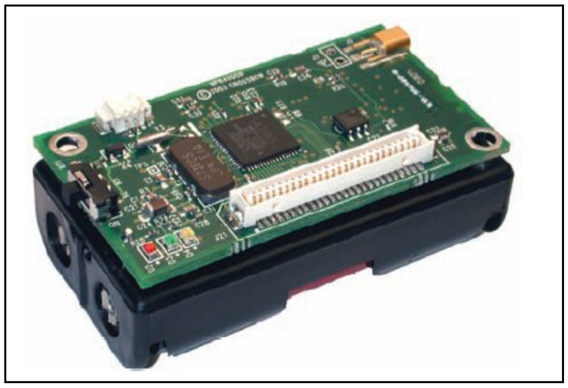
Processing Unit: The processing unit is responsible for processing and storing the sensed data. It is also responsible for local processing and aggregation of sensed data using on-board processing capability.

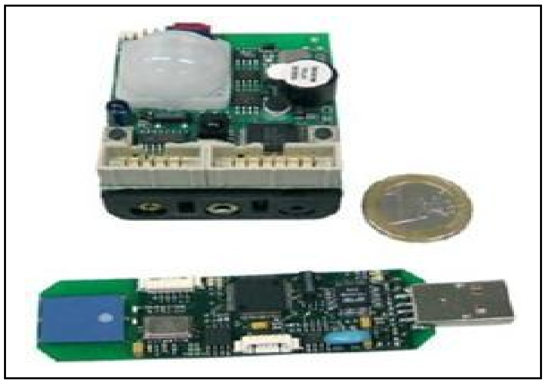
Communication Unit: Communication unit provides functionalities to communicate with their neighboring nodes. It has a full-duplex built-in transceiver that could be used to transmit and receive data simultaneously.

Power Unit: Power unit is responsible to supply power to the sensor node. Batteries are the most common source of power supply. However, there is some energy scavenging techniques that could generate power from ambient sources.

The most commonly used sensor hardware platforms in academic research, commercial applications as well as industry includes MIT Cricket for location detection from MIT [2], motes platform (i.e. MICA, MICA2 and TelosB) from crossbow designed at University of California Berkley [11] and TMote-Sky from moteiv [12]. Scatterweb [13] is another platform from Freie Universität Berlin and BTnodes with Bluetooth wireless interface developed at ETH Zurich [14]. Some common sensor platforms are shown in Figure 2.4.


(a) MIT Cricket Beacon


(b) Crossbow Motes (MICA2)


(c) ScatterWeb

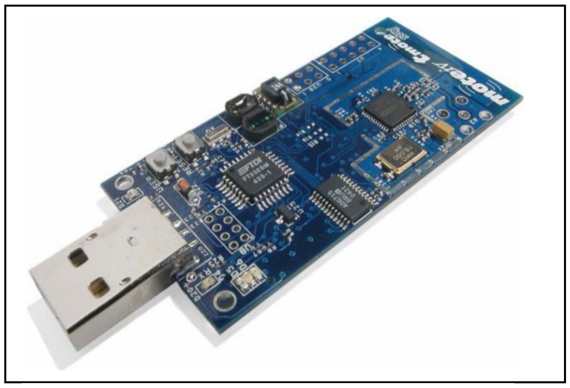

(d) TMote-Sky

Figure 2.4: Various sensors platform currently used in academia and industry

2.1.5 Software Components of a Sensor Node

In addition to hardware components, sensors have also two software components: Operating System and Middleware/Applications. Operating system and middleware are used to provide an environment for WSN application deployment and execution. One of the very commonly used operating systems for WSNs is TinyOS [15]. In order to provide sensor data processing and accessing, TinyOS is ported with applications, such as TinyDB [16] and TinyREST [17]. These applications are programmed in a special language known as nesC [18]. Figure 2.5 shows the software components (subsystem) of a sensor node.

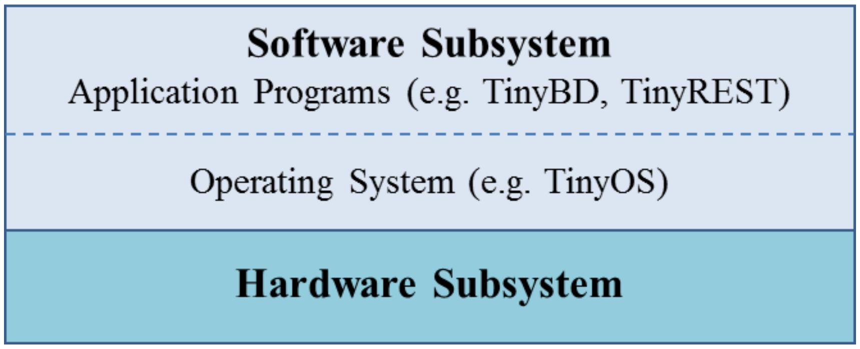

Software Subsystem
Application Programs (e.g. TinyBD, TinyREST)
Operating System (e.g. TinyOS)
Hardware Subsystem

Figure 2.5: Software components of a sensor node

2.1.6 Wireless Sensor Network (WSN) Applications

Due to the technology advancement, researchers have developed numerous different kinds of sensors such as infrared, ultrasound, seismic, thermal, visual and acoustic, radar and combination of infrared and radar. These sensors open the doors for application developers to develop a wide range of WSNs applications. WSNs applications categorized in [7] are:

(i). People/Asset tracking applications

(ii). Healthcare applications

(iii). Environmental applications

(iv). Military applications

(v). other commercial applications

Beside these categories it is also possible to extend the applications as space exploration, chemical processing and disaster relief operations. In the following section we are going to discuss two major application categories of people/asset tracking and health application areas that could be used to motivate our technical problem.

2.1.6.1 People/Asset Tracking Applications

People/asset tracking inside large buildings is one important application area that could benefit from the availability of contextual information (i.e. location information) provided by the wireless sensors. For example, when a person enters into a large building (e.g. airport, museum, shopping mall, fair center etc.), he/she is not familiar with, and may get lost. One potential solution is that the person can use a cell phone or a PDA equipped with a client application based on the OMA RESTful location services to locate her/him. The application can also guide the person step by step to visit the whole building as a guide throughout the user interface (i.e. floor plan). The person can also use the same application to find his/her friends, relatives and children inside the building. Other services like finding meeting rooms, nearest toilets, and auto-teller machines can also be provided. Moreover, the person may ask to be notified when some of her/his friends/relatives/children's is located in the proximity of their current location located in a specific area or a specific distance [19].

2.1.6.2 Healthcare Applications

Another important WSNs applications area is healthcare application. The following example shows a health monitoring application that tracks the current location of older patients with dementia. Typically, the patients suffering from dementia may try to escape from the hospital against orders. In this case a sitter may need to be assigned to monitor continuously which is not cost effective and suitable in future healthcare environment. Instead of assigning a sitter for each patient, we can bind a cell phone with attached sensor into their body which can sense the location information. Application can subscribe (with the subscription information) about the cell phone attached to the patient body, and notify at later time when patients will try to escape or out of bed from hospital. In a hospital or clinic, location sensors can also be attached to track the doctor's current position in order to inform other personnel and ensure more efficient collaboration among them.

Tele-monitoring or telemedicine is another direction in health applications, aims at providing automated, accelerated and more efficient healthcare assistance. This could be done by integration of some lightweight and small sensors installed on the patient bodies. These sensors are responsible for measuring the patient's biomedical and physiological data such as heart rate, blood pressure, body temperature, respiratory rate, mental status and oxygen saturation, temperature and location information. The sensors installed on the patient's body together form a Wearable Wireless Body Area Network (WWBAN) [20]. WWBAN allows doctors, nurses and other medical stuff to monitor and analyze patient's physiological, environmental and location information remotely. The emergency applications, such as triage and treatment of seriously injured people in a very efficient and coordinated manner, are already realized to overcome the critical bottlenecks of emergency activities. These emergency applications are based on WSN technology [21].

Several projects and research prototypes have been developed for healthcare applications using WSNs. Mobi-Health European project [22] provides a continuous monitoring of patients' health status while they are outside of the hospital environment.

CodeBlue [23] provides a wireless infrastructure and architecture which can be deployed in emergency medical care. This project combines low-power wireless biomedical sensors, PDAs and PCs. [24] describes some illustrative applications in the wireless healthcare application domain and also introduce detailed challenges to WSNs with respect to the required level of reliability, privacy and security of medical data.

2.1.7 Wireless Sensor Networks (WSNs) Challenges

Despite the numerous benefits of WSN applications, WSN also poses many challenges for the design of communication protocols, hardware architectures and software modules. The factors that influence the design of the above protocols are mainly focused on the optimization of power consumption, security, fault tolerance, network routing, self-organization, network discovery, robustness, and data processing. Among all of them, mobility of sensors is a growing research topic now-a-days. Due to the mobility of sensor nodes and constraints of WSNs, network topology may vary. In order to cope with this dynamic topology, special mechanisms are needed. To-date, many researchers have been done for designing energy efficient routing protocols [25] [26] [27]. WSN security is another challenging issue as described in [28] [29].

2.2 REST and RESTful web services

In the following section, we will discuss the background information on REST and RESTful Web services.

2.2.1 Introduction

REST was first defined by Roy Fielding in his PhD dissertation [3]. The aim is to design distributed networked applications by taking benefit of the existing technologies and protocols of the World Wide Web (WWW). REST uses the Web's basic technology (e.g. HTML, XML and HTTP) as a platform to build and provision distributed services [30]. It is one of the key players of Web 2.0, a concept associated with the web applications that promotes interactive information sharing and collaboration over the web. REST adopts the client-server architecture of the web and does not restrict client-server communication to a particular protocol but more work has been done on using REST with Hypertext Transfer Protocol (HTTP) since HTTP is the primary transfer protocol of the web [30].

A RESTful web service is designed following REST design principles. RESTful web services can be described using the Web Application Description Language (WADL [80]). A WADL file describes the request that can be addressed to a service, including the service's Uniform Resource Identifier (URI) and the data the service expect and serves [30].

REST models the information to operate on (e.g. a user location) as resources and identifies each resource using a URI. A standard and unified interface is then used to access the defined resources. This interface consist of the HTTP methods GET, POST, PUT and DELETE, which are used to read, create, update and delete a resource, respectively. A more details description on REST and RESTful web services can be found in a recent survey paper [30].

REST is not an architecture, but a set of design criteria. Resource Oriented Architecture (ROA [45]) is a RESTful architecture that provides a set of commonsense rules for designing RESTful web services. The REST architectural style is based on the following principles: resources, addressability, statelessness, connectedness, uniform interface and cache-ability [31] [37] [33]. We will be discussing ROA and properties of REST in following sections.

2.2.2 Resource Oriented Architecture (ROA)

As name implies ROA is all about resources. It is based on the concept of resource. Resource is a distributed component that can be accessible through a standard common interface. The main ROA concepts are: resource, resource name, resource representation, resource link and resource interface.

A resource is anything that can be named and that is important enough to be referenced as a “thing” itself (e.g. a document, a row in a database, a search result, an item etc.). It can be a physical object or an abstract concept. A resource is the first thing that we should consider when designing REST based applications. On the other hand, a resource can be defined by simply an entity, an item or a thing that we want to expose. Each resource should have at least one URI in order to uniquely identify it on the web. Without an URI, it is not a resource and not possible to find it on the web. It is also not possible to access two different resources by using the same URI. However, it is possible to access same resource by using two different URIs.

Each resource has a representation that represents the current state of a resource. It can be represented in any format or any media type. For instance: Extensible Markup Language (XML [70]), JavaScript Object Notation (JSON [71]), Extensible Hyper Text Markup Language (XHTML) or plain text representation. We can specify resource representation format or media type inside the URIs itself (e.g. resource/location.xml) but it is not a good practice to do. Rather, Content-Type header specifies the representation needed to display the entity-body. On the human web, web browser displays it inline or by running an external program. On the programmable web, web service client decide which parser to apply to represent [34]. Resource can be linked to same or other resources via hyperlinks. We can access the resource and manipulating it through a uniform interface. Uniform interface is described in Section 2.2.3.4. A more information about resource can be found on [31] [34].

2.2.3 Properties of REST

The properties of REST are: addressability, statelessness, connectedness, uniform interface, and cache-ability.

2.2.3.1 Addressability

It is the idea that every resource in the system is addressable through URIs. The number of URIs depends on the number of resources. With this addressability one can bookmark a specific page. Furthermore, we can email the URI of a resource to anybody else. A more details description with an example is presented in [31].

2.2.3.2 Statelessness

Statelessness is a property of HTTP requests that are performed in a complete isolation. When the client makes an HTTP request, it should include all necessary information for the server to process that request. The server should never rely on information from previous requests. If the server needs some information from the previous request, the client should send that information again with the new request since the server does not keep any information about the client. This principle makes the system simpler, reliable, scalable and cacheable. In a stateless scenario, each request is separated from other request. This allows the client to make request for a specific resource, any number of times and in any order. For instance, page-2 can be requested before requesting page-1. A client makes a request to server A and it is successful. The client sends another request to the same server but at that time the server goes down or unable to response. In that case, a new server can serve the request since all the information it requires was given to make the request successful at the time of requesting.

2.2.3.3 Connectedness

In RESTful web services, the client can navigate from one state to another by sending links inside the representation. Representations are hypermedia, and documents that contain not only data but also links to other resources. Human web is easy to use as it is well connected but programmable web is not yet easy to use. In RESTful fashion, it is obvious that resources should be connected and linked together. This allows the client to discover the interface by traversing hyperlinks between each of its resource representation.

2.2.3.4 Uniform Interface

Unified interface is used to access the defined resources. This interface consists of several HTTP methods including GET, POST, PUT and DELETE, which are used to read, create, update and delete (i.e. CURD) a resource respectively. Each of the HTTP methods is discussed as follows:

HTTP GET: The GET HTTP method is used to retrieve whatever information is identified by the request-URI [35]. This information is identified by the URI, which is a representation of a resource. GET HTTP method is safe, meaning that it does not change the state of the server; it only retrieves information from the server. HTTP GET method is also considered as idempotent, meaning that it can be applied multiple identical requests having the same effect as a single request. If the request is succeeded and the resulting resource is returned in the message body, the appropriate response code is 200 (i.e. OK). If the resource does not exist, the response code is 404 (i.e. Not Found).

HTTP POST: The POST HTTP method is used to create a new resource with the data enclosed in the request as a new subordinate of the given URI. When a resource is created, it returns a 201 (i.e. Created) response code. It contains an entity which describes the status of the request and refers to the new resource and a location header.

HTTP PUT: The HTTP PUT method is used to store the enclosed entity under the supplied request-URI [35]. If the URI refers to an existing resource, it will update the previously stored data on the server with the enclosed data. If the requested URI does not point to an existing resource and is also capable of being defined as a new resource, the origin server can create the resource with that URI. If a new resource is created, a 201 (i.e. Created) response code returns. If an existing resource is modified, a 200 (i.e. OK) response code should be sent to indicate successful completion of the request. If the resource is not created or modified, an appropriate error response should be given that reflects the nature of the problem.

PUT will create a new request where the client is in charge of creating the new resource URI (i.e. given by the client). In order to modify an existing resource, PUT modifies an existing resource identified by the URI that already exists. On the other hand, POST creates a new resource where the server is in charge of creating the new resource URI. In short, HTTP PUT request identifies the entity enclosed with the request where

POST request identifies the resource that will handle the enclosed entity. Furthermore, PUT HTTP method is idempotent like GET method but the POST HTTP method is not since every HTTP POST request has different result.

HTTP DELETE: The HTTP DELETE method is used to delete the existing created resources identified by the requesting URI. If the action is performed successfully, a successful response status code 200 (i.e. OK) will return. Otherwise, an appropriate error response code will return indicating the nature of the problem. The HTTP DELETE method is also an idempotent like other GET and PUT HTTP methods.

Besides the aforementioned four main HTTP methods, there are two other HTTTP methods; HTTP HEAD and HTTP OPTIONS are also considered as uniform interface. HTTP HEAD method is used to fetch meta-data about a resource whereas HTTP OPTIONS method is used to discover HTTP methods which are allowed for specific resources.

2.2.3.5 Cache-ability

The REST architectural style allows caching the resources whenever possible with an expiration date and time. Cache-ability offers faster response and better loading time by decreasing the load on the server. HTTP has two caching mechanism such as Expiration and Validation. A more details description of REST properties can be found in [31].

2.2.4 Development of a RESTFul web service

Before staring the implementation of RESTful web services, it is important to analyze the procedures to follow. The procedures are as follows [31]:

1. Figure out the data set

2. Split the data set into resources

For each kind of resource:

3. Name the resources with URIs

4. Expose a subset of the uniform interface

5. Design the representation(s) accepted from the client

6. Design the representation(s) served to the client

7. Integrate this resource into existing resources, using hypermedia links and forms

8. Consider the typical course of events: what is supposed to happen?

9. Consider error conditions: what might go wrong?

2.3 OMA RESTful Location Services

In the following sections, we will discuss OMA RESTful location services along with their resources, data structures and sequence diagrams.

2.3.1 Introduction

OMA RESTful location services allow an application to obtain the current location of a given terminal and the distance of a terminal from a given location, as well as the distance between two terminals. These services also allow for an application to manage client subscriptions for periodic location notifications (i.e. to be periodically notified about the current location of a terminal), area/circle notifications (i.e. to be notified when a given terminal enters or leaves the target area), as well as periodic distance notifications (i.e. to be notified when the distance between a monitored and a reference terminal exceeds or goes below a certain threshold) [5].

2.3.2 OMA RESTful Location API Definition

In this section, we present a detail description of OMA RESTful location services. First, we will be describing the resource structure for defining OMA RESTful location services and their operations. Second, we will be presenting a few sequence diagrams to demonstrate the interaction between server and client applications. Finally, we will conclude this section by providing an example of request and response for accessing a simple query resource (e.g. for accessing location information of a single terminal).

2.3.2.1 OMA RESTful Location Resources

OMA RESTful location services define a set of server-side resources plus one client-side resource that is used to send notifications to the client. Figure 2.6 shows the resource structure for server-side resources. The server-side resources are either of the query (i.e. request/response) or subscription type (i.e. a request followed by multiple notifications). Query resources include terminal location (e.g. current location) and terminal distance (e.g. the distance of a terminal from a given location or the distance between two terminals). Terminal location and terminal distance resources are accessed via an HTTP GET request only. The GET request has no message body, whereas the response body carries the requested information.

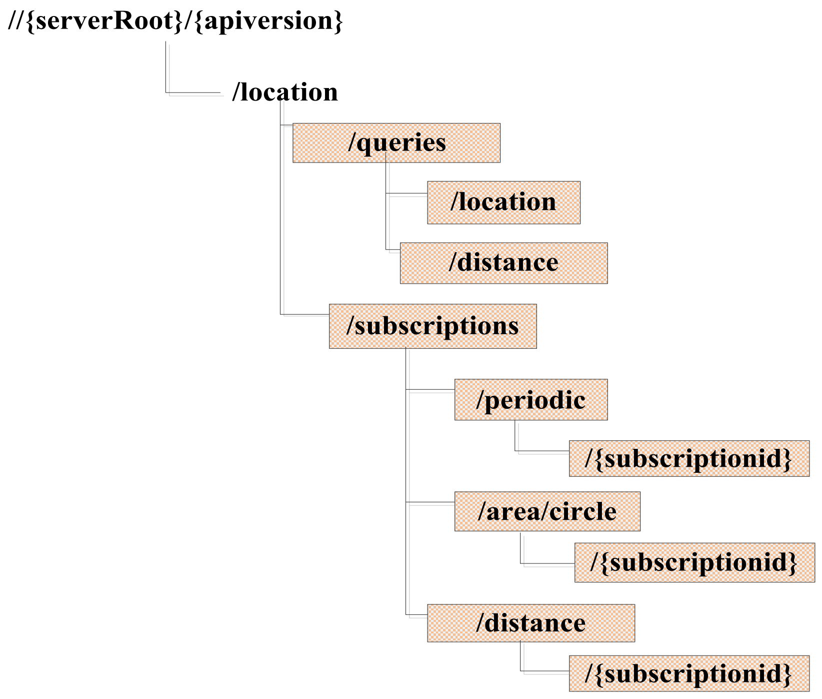


Figure 2.6: OMA RESTful server-side resource structure for terminal location

The subscription resources include: the list of subscriptions for periodic location notifications, the list of subscriptions for area notifications, and the list of subscriptions for distance notifications. The supported HTTP methods are GET and POST. GET returns the list of active subscriptions, whereas POST creates a new subscription under the target resource. The other subscription resources are: an individual subscription for periodic location notifications, an individual subscription for area notifications, and an individual subscription for distance notifications. For these resources, the supported HTTP methods are GET, PUT and DELETE. GET returns the different parameters (e.g. the notifications' frequency) of a specific subscription identified by its subscription Id, PUT modifies the subscription parameter, and DELETE removes the subscription and stops the related notifications.

Resources	**Base URL:** http://{serverRoot}/{apiVersion}/location	HTTP action
Terminal location	/queries/location?address={terminalId} /queries/location?address={terminalId1}&address={terminalId2}	GET: return current location of a terminal or multiple terminals
Terminal distance	/queries/distance?address={terminalId}&latitude={lat}&longitude={lon} /queries/distance?address={terminalId1}&address={terminalId2}	GET: return current distance of a terminal from a given location or distance between two terminals
The list of subscriptions for periodic location notifications	/subscriptions/periodic	GET: return all active subscriptions POST: create a new subscription
Individual subscription for periodic location notifications	/subscriptions/periodic/{subscriptionId}	GET: return specific subscription PUT: update specific subscription DELETE: delete one subscription
The list of subscriptions for area notifications	/subscriptions/area/circle	GET: return all active subscriptions POST: create a new subscription
Individual subscription for area notifications	/subscriptions/area/circle/{subscriptionId}	GET: return specific subscription PUT: update specific subscription DELETE: delete one subscription
The list of subscriptions for distance notifications	/subscriptions/distance	GET: return all active subscriptions POST: create a new subscription
Individual subscriptions for distance notifications	/subscriptions/distance/{subscriptionId}	GET: return specific subscription PUT: update specific subscription DELETE: delete one subscription
Callback resource (client side resource)	URI provided by the client at subscription time	POST: post the notification information to the client application

Table 2.1: OMA RESTful location resource

The client-side resource is known as the callback resource. When a client application requests the creation of a new subscription, it should provide the URI of the callback resource that is to receive the periodic notifications. The only HTTP method supported for this resource is POST. Table 2.1 summarizes the resources, their URIs and the methods they accept. The data structures are used for all these resources are described in [5]. We will only be discussing terminal location resource in details along with an example in Section 2.3.2.3. All other resources are explained in [5] [36].

2.3.2.2 Sequence Diagrams

This section presents two sequence diagrams, one for accessing server-side query resource and another one for accessing server-side subscription resource. In order to access server-side query resource, we present the scenario where the client/application needs the location information of a single or multiple terminals. For server-side subscription resource, we present the scenario in order to control the subscriptions for periodic notifications of a terminal. Figure 2.7 depicts the sequence diagram in order to acquire the location information of a single or multiple terminals. The client/application sends HTTP GET request (step-1) with necessary information in the requested URI as a parameter (e.g. terminal address, or terminal addresses, desired accuracy) and receives the location information as a response (step-2).

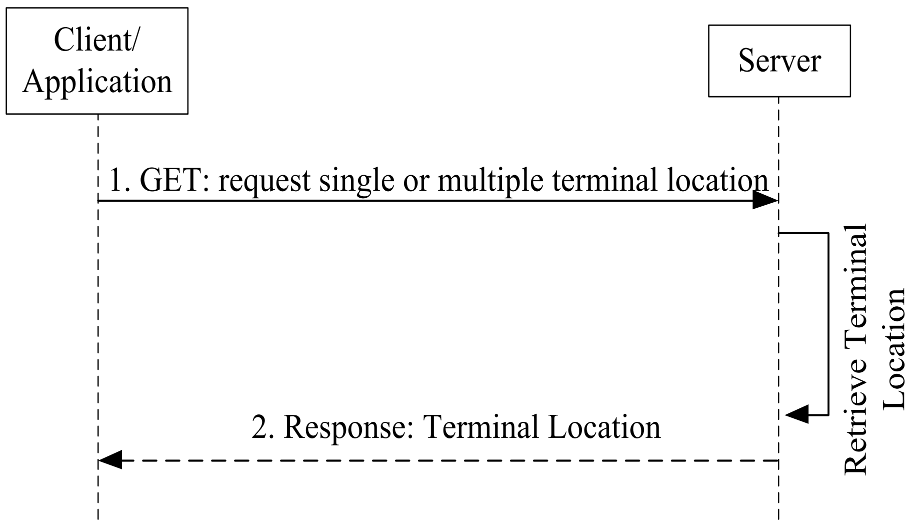

Client/ Application
Server
1. GET: request single or multiple terminal location
Retrieve Terminal Location
2. Response: Terminal Location

Figure 2.7: Getting location information of a single or multiple terminals

Figure 2.8 shows the sequence diagram to control the subscription for periodic notification of a terminal. Client/application creates a new periodic subscription by sending HTTP POST request with the subscription information in the requested body (Step-1) and receives the created resource URI containing the subscription-Id (Step-2).

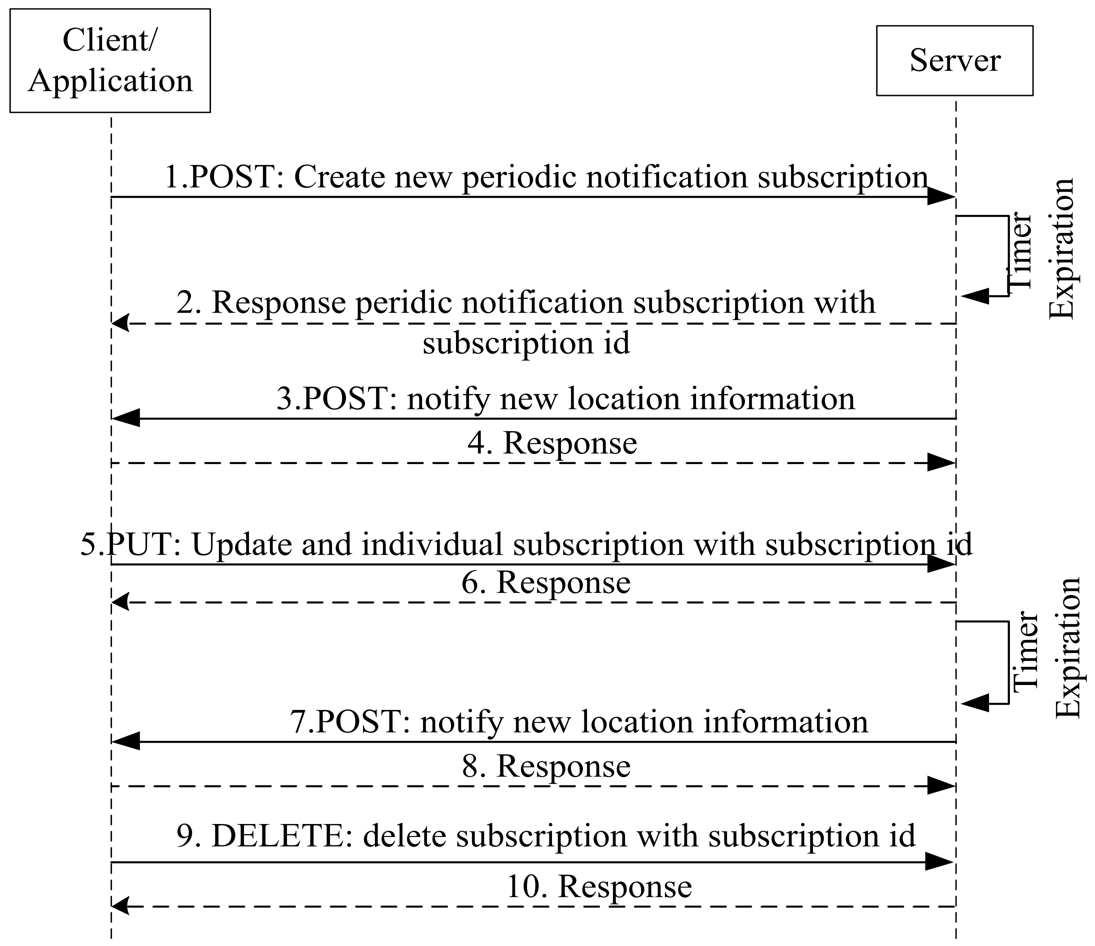

Client/ Application
Server
1.POST: Create new periodic notification subscription
Timer Expiration
2. Response peridic notification subscription with subscription id
3.POST: notify new location information
4. Response
5.PUT: Update and individual subscription with subscription id
6. Response
Timer Expiration
7.POST: notify new location information
8. Response
9. DELETE: delete subscription with subscription id
10. Response

Figure 2.8: Steps to control the periodic subscription notification

When the timer expires, the server sends back the notification to the client by the client-provided callback URI (Step-3 & Step-4). After some times, the client/application modifies the previously created subscription by sending the HTTP PUT request to the resource URI with subscription-Id (Step-5 & Step-6). After the modification, the server sends the notification again until the subscription ended by the client/application (Steps-7 & Steps-8). Finally, client/application deletes the subscription by sending the HTTP DELETE request to the URI with subscription-Id which stop the notification as well (Step-9 & Step-10).

2.3.2.3 Examples of Request/Respone for Terminal Location Resource

This section is discussed with an example of request and response of a terminal location resource in details Terminal location resource is used to return the current location of a single terminal or multiple terminals. The URI is used to access the terminal location resource is *http://{serverRoot}/{apiVersion}/location/queries/location*. The request URI variables and their description are shown in Table 2.2. The request URI variables are also common for all other HTTP requests. Each request associated with a specific response code in the response depending on the action taken by the server. The common HTTP response code is used for OMA RESTful location services shown in Table 2.3.

Name of the Variables	Description
serverRoot	serverRoot defines the servers base url: *hostname+port+basepath.* Example: http://localhost:8080/location
apiVersion	This indicate the API version of the ParlayREST API that the client wants to use (e.g. 1 for version 1.x)

Table 2.2: Request URI variables

Response Code	Description
200	**Success**
201	**Created**
204	**No Content**
304	**ConditionNotMet** (Not Modified): The condition specified in the conditional header(s) was not met for a read operation.
400	**Invalid parameters in the request**
401	**Authentication failure**
403	Application don't have permissions to access resource due to the policy constraints (request rate limit, etc.)
404	**Not Found:** The specified resource does not exist
405	**Method not allowed by the resource**
409	**Conflict**
411	**Length Required:** The Content-Length header was not specified.
412	**Precondition Failed:** The condition specified in the conditional header(s) was not met for a write operation.
413	**RequestBodyTooLarge (Request Entity Too Large):** The size of the request body exceeds the maximum size permitted.
416	**InvalidRange (Requested Range Not Satisfiable):** The range specified is invalid for the current size of the resource.
500	**Internal server error**
503	**ServerBusy (Service Unavailable):** The server is currently unable to receive requests. Please retry your request.

Table 2.3: Common HTTP response code used for OMA RESTful location services

An example of a request and response for terminal location resource is shows in Table 2.4. The request, which is a HTTP GET request, is responsible for receiving the current location of a specific terminal (i.e. 5145817818). In the response, the response code is 200 (i.e. OK) which indicates the successful completion of the request. It also specifies the content type, content length and the time inside header and the actual information inside the response body.

Request	GET http://localhost:8080/location/queries/location?address=5145817818 HTTP/1.1 Host: localhost:8080	
Response	HTTP/1.1 200 OK Content-Type: application/xml Content-Length: 1234 Date: Thu, 04 Jun 2009 02:51:59 GMT	**Content -Header**
	<? xml version="1.0" encoding="UTF-8"?> <terminalLocationList> <terminalLocation> **<address>5145817818/address>** <locationRetrievalStatus>Retrieved</locationRetrievalStatu <currentLocation> **<latitude>-80.86302</latitude>** **<longitude>41.277306</longitude>** **<altitude>1001.0</altitude>** <accuracy>100</accuracy> <timestamp>2011-06-03T00:27:23.000Z</timestamp> </currentLocation> </terminalLocation> </terminalLocationList>	**Content-Body**

Table 2.4: Example of request and response to get location information of a single terminal

2.4 Chapter Summary

This chapter focused on a brief description of WSNs along with its architectures, protocol stack, hardware and software components, REST and RESTful web services and

OMA RESTful location services with its resources and their operations. In the following chapter, we will explore the state-of-the-art for the implementation of location services with existing technologies.

Chapter 3 Location-Services: State-Of-The-Art

This chapter reviews and evaluates the existing solutions for location services such as OMA RESTful location services [5], GSMA OneAPI RESTful location services [38], and OGC OpenLS [43] location services using different technologies (i.e. WSNs, GPS, A-GPS, and Mobile Mast). Other non-standard approaches are also discussed.

3.1 Introduction

Schiller and Voisard defined location services as “services that integrate a mobile device’s location or position with other information to provide added value to a user” [4]. Standard location service specifications are provided by OMA [57], GSM Association (GSMA) [58] and Open Geospatial Consortium (OGC) [42]. Beside these standard specifications there are a few other approaches [46] [47] [48]. Implementation of such standard and nonstandard specifications with different technology opens the door for application developers to offer more attractive location based services and speed up their development.

3.2 Categorization of Location Services

We categorized location services into four groups: OMA RESTful location services, GSMA OneAPI RESTful location services, OGC OpenLS location services and a few other proprietary interfaces as shown in Figure 3.1. This categorization is based on the standard and non-standard location interfaces and location APIs. The aforementioned interfaces will not be able to provide location information without using any other locator technologies. Different locator technologies can provide different level of accuracy. We further categorized the related work into sub-categories based on the used implementation technologies: WSN, GPS, Assisted Global Positioning System (A-GPS) and Cell-Id (as shown in Figure 3.1). The proprietary approaches will be discussed in Section 3.2.1. The implementations of standard interfaces (i.e. location services) with different technologies are dealt with in Section 3.2.2.

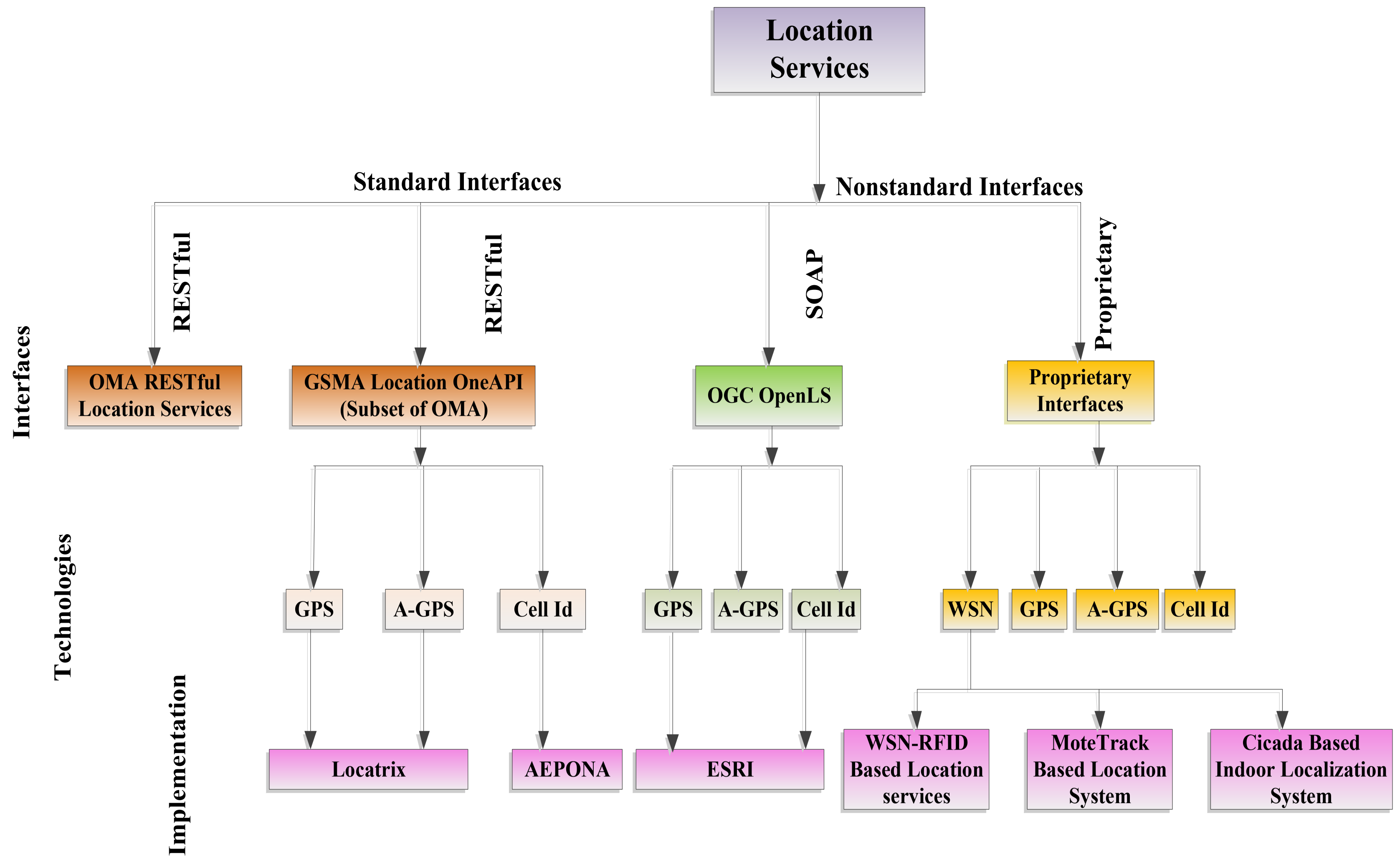


Figure 3.1: Location services: State-of-the-art

3.2.1 The Selected Proprietary Interfaces

The most significant existing proprietary solution includes WSN-RFID based location services [46], Mote-Track based location system [47], and Cicada based indoor localization system [48].

3.2.1.1 WSN-RFID Based Location Services

The work presented in [46] proposed an internal location based system using Radio Frequency Identification (RFID) and Wireless Fidelity (Wi-Fi) technology in order to accurately determine people's location in indoor environments. RFID is used to identify user's location in indoor environment with WSNs. Each RFID reader is attached to a Millennial-Net [49] end point in order to create a wireless network. The user whose location information needs to know is tagged with a RFID tag. When a user passed or crossed the end points, RFID reader will detect the tag and send this data to store into a database through a gateway application. At later time, application uses this data to detect the location of a user and respond accordingly. Figure 3.2 shows the system architecture of WSN-RFID based location services system. Millennial-Net sensor network is a low-powered, self-configuring WSN. Once the nodes of Millennial-Net sensor network are deployed in real environments, it requires minimum maintenance which minimizes the network deployment cost as well.

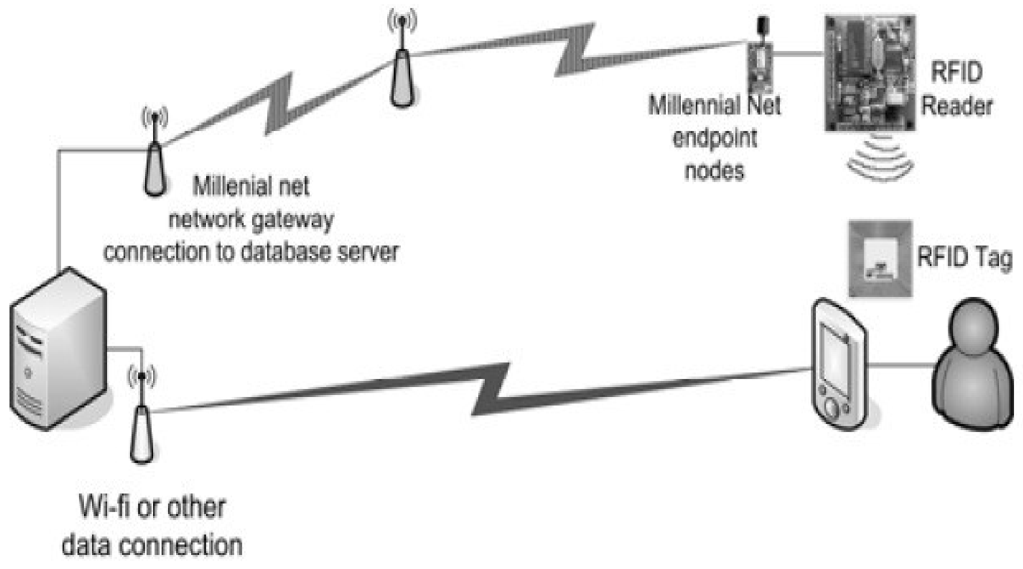


Figure 3.2: Overall system architecture of WSN-RFID based location services

It was shown that, a combination of sensor nodes can achieve the required level of accuracy in order to create usable location based services [46]. However, their work did not mention any standard payload that may be used to exchange information. Furthermore, they also did not provide any Application Programming Interface (API) description for application development. In addition, their work did not support heterogeneous WSNs and reusability of existing WSNs gateways as well as existing technology/standards.

3.2.1.2 Mote-Track Based Location System

An important research work is presented in [47]. This work analyzed the design and implementation of a sensor-network based location system that provides sufficient location accuracy only for home applications. They mainly focused on the location determination system, a location storage system, and a middleware interface. The location determination system adopts and extends Mote-Track [50] WSN. The location storage system is responsible for storing the location information into the location database. The middleware interface is based on Open Services Gateway Initiative (OSGi) [51]. It allows accessing the current and past location information from the system. This work is a part of a Home Area Network (HAN) middleware, a component of the TRLabs "Smart Home Framework" shown in Figure 3.3. The Home Gateway Device (HGD) is the main component in TRLabs Smart Home Framework. HGD is an embedded computational device that controls operations in the HAN and also serves as the home's link to the Internet. It is assumed that all HAN middleware services are running in the HGD which is the source of stored location information. This work also provides a service layer that supports service discovery and fully automated service composition [52]. The sensor platform, location determination and middleware interfaces are discussed as follows:

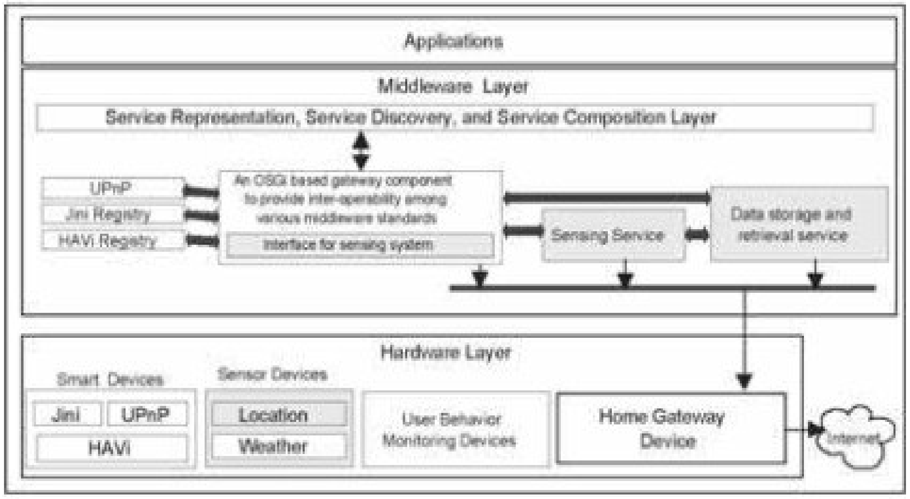

Applications
Middleware Layer
Service Representation, Service Discovery, and Service Composition Layer
UPnP
Jini Registry
HAVi Registry
An OSGi based gateway component to provide inter-operability among various middleware standards
Interface for sensing system
Sensing Service
Data storage and retrieval service
Hardware Layer
Smart Devices
Jini
UPnP
HAVi
Sensor Devices
Location
Weather
User Behavior Monitoring Devices
Home Gateway Device
Internet

Figure 3.3: TRLabs Smart Home Framework

Sensor platform: Mica2 and Mica2Dot sensors [53] are used as the hardware platform for the location determination system. Mica2 sensors run the TinyOS [54] operating system and nesC [55] was used for programming. Mica2 and Mica2Dot are also used to form the sensor network in order to gather information from the beacon nodes. Mica2 sensor nodes act as the beacon nodes (B). Beacon nodes are deployed at fixed locations in home environments that transmit signals using multiple frequencies and power levels to improve accuracy [56]. The beacon nodes are also deployed in an equally spaced grid topology within the home. The nodes are placed in such a way that the entire home is covered by their collective radio range. Figure 3.4 depicts the sensor network deployment. On the other hand, Mica2Dot sensor nodes act as either mobile node (M) or base station node (BS). Mobile nodes are carried by the home residents and received messages from the fixed beacon nodes. The nodes calculate their positions using the information received from the fixed beacon nodes. Mica2Dot sensor nodes also perform as a base station node connected to the HGD. The base station node is responsible for collecting and storing data received from the mobile node. Moreover, software is developed to capture the sensor data received by the BS through serial port. The software component calculates and stores the location reference point signature during the setup phase. It also displays the location data on a graphical map of the home.

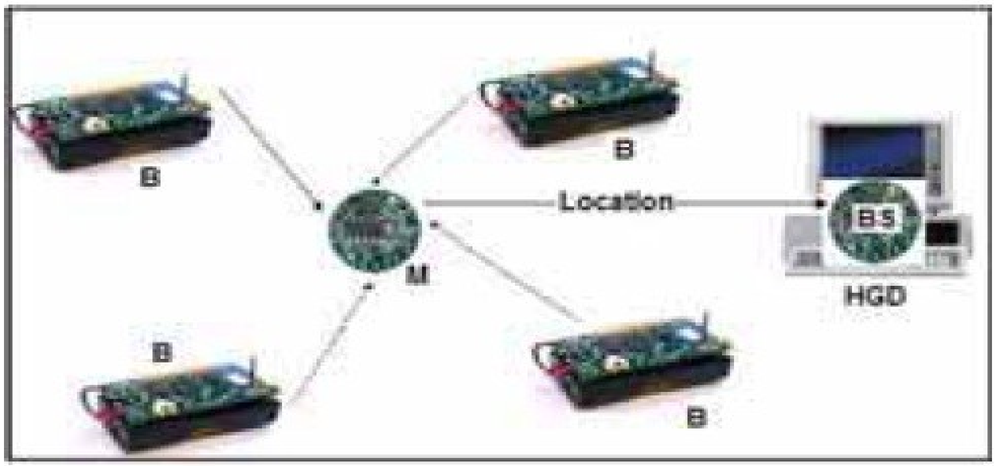


Figure 3.4: Mote-Track sensor network deployment

Location determination: Beacon nodes broadcast their signals periodically at initial data collection phase. A beacon signal contains the beacon node id, signal sequence number, and the power level and frequency channel. The mobile node receives this signal and collects the information in it. It also measures the received signal strength. After that, mobile node sends all the received information to the base station. Then the information is written into a record for the appropriate reference point in the reference signature database. During normal operations, beacon nodes broadcast signals in the same way as the data collection phase. When a mobile node receives signals it creates a signal signature from the received signals. After that, signal is matched against entries in the reference database and mobile node. It also calculates its position using the matched entries and forwards this location data to the HGD for storage purpose [47].

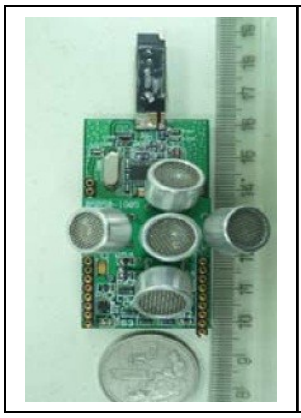
Middleware service components: An OSGi interoperability component providing access to the location determination and storage services was developed. The location sensing bundle delivers services including the determination of current location of a person based on his/her Id. It also retrieves all past locations of a person [47].

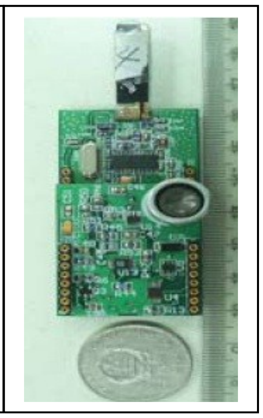
However, they did not specify the APIs for application development. Moreover, their work also did not support heterogeneous WSNs and reusability of existing WSNs gateways as well as existing technology/standards.

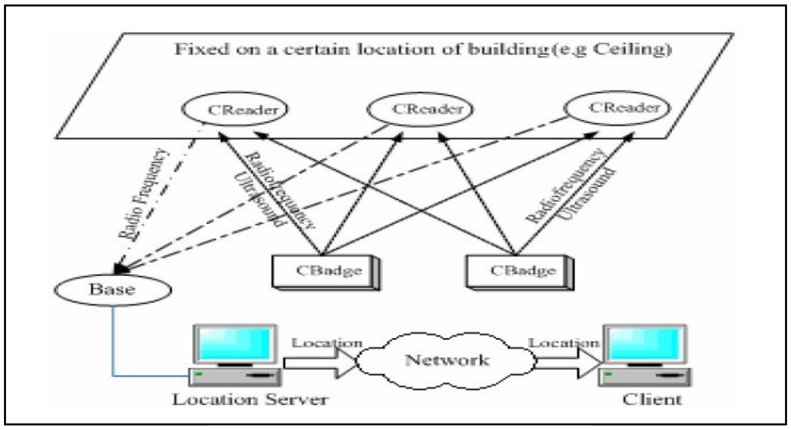
3.2.1.3 Cicada Based Indoor Localization System

The research work presented in [48] describes the design and implementation of the Cicada based indoor localization system. Their work mainly focuses on sensor specific information. For instance, in order to measure the distance, the system uses Time Difference of Arrival (TDOA) between Radio Frequency (RF) and ultrasonic. To calculate the rough distance correction, it uses SWF (i.e. slide window filter) and least square fitting. In order to estimate the coordinates UKF (i.e. unscented Kalman filter) is adopted by the system. Finally, the system sends the estimated location coordinate to the network in order to provide location services.

Cicada wireless infrastructure is shown in Figure. 3.5. The system is consisted of two embedded hardware devices: CBadge and CReader. CBadge unit is used to send the RF and ultrasonic signals periodically. On the other hand, CReader unit is used to receive the signals sending by CBadge. CReader unit acts as a base station used to collect RF signals transmitted by the CReaders. Finally, the base station sends the collected information to the location server in a fixed data format. The location server is responsible for storing the obtained position coordinate and providing location information towards the applications. However, their research work did not define the interface between the application and the location server. Besides, they also did not


(a) CBadge


(b) CReader



(c) Overall architecture

Figure 3.5: Cicada sensor nodes and overall architecture

provide any API descriptions for application development. Furthermore, their work did not support heterogeneous WSNs and reusability of existing WSNs gateways as well as existing technology/standards.

The aforementioned three proprietary solutions are location technology dependent. Their solutions mainly focused on a specific sensor and on the proprietary interfaces. Furthermore, they did not provide interoperability and application portability. Due to these limitations of proprietary solutions, the aforesaid approaches are not taken into account for providing location services. Now in the following section we will analyze the standard approaches.

3.2.2 Standard Location Services

OGC OpenLS location services, GSMA OneAPI RESTful location services and OMA RESTful location services are the standard location services. We will contrast REST with SOAP based web services in the following section.

3.2.2.1 REST versus SOAP

We hereafter contrast REST with Simple Object Access Protocol (SOAP) based solutions according to three criteria to demonstrate the superiority of REST based solutions.

Simplicity: RESTFul web services are very lightweight and simple compared to SOAP based web services. REST leverages existing well-known W3C/IETF standards (i.e. HTTP, XML, URI, and MIME) and necessary infrastructure has already become pervasive [37]. On the other hand, SOAP based web services are deemed to be complex, especially because of the use of the complex SOAP messages and Remote Procedure Call (RPC). In SOAP based web services it is required to support SOAP and RPC on both client and server applications. However, SOAP based web services are suitable for enterprise application integration scenarios whereas RESTFul web services are suitable for ad-hoc integration scenarios [37].

Flexibility of Data Representation: SOAP based web services for resource constrained devices impose XML for data representation. However, RESTFul web services provide a greater flexibility to use multiple resource representation formats. This enables a variety of possibilities such as XML, JSON and plain text.

Easy to use and develop the Application: SOAP based web services require some specific toolkits for developing both client and server applications. SOAP based services require greater implementation effort and understanding of these toolkits for both client and server side, whereas REST based web services require only greater implementation effort on the server side. Therefore, RESTFul web services are very easy to use and develop client application as it does not require any prior knowledge about the toolkits. We can develop client application by simply writing the HTML code in Notepad.

Table 3.1 summarizes the comparison of REST versus SOAP. From Table 3.1, it is easily visible that REST is the most suitable choice between the two approaches. For instance, a standard location service such as OpenLS [43], proposed by Open Geospatial

Criteria	**REST**	**SOAP**
1. Simplicity	Simple	Complex
2. Flexibility in terms of data representation	Yes	No
3. Easy to use and develop the application	Yes	No

Table 3.1: Comparison of REST and SOAP based web services

Consortium Inc. [42], is a SOAP based web services. Hence REST is the best approach rather than SOAP, so OpenLS location services was not taken under consideration for the implementation of location services. In the following sections, existing standard RESTful location services and its evaluation will be discussed.

3.2.2.2 GSMA OneAPI RESTful Location Services

GSMA OneAPI RESTful location services [38] are standard RESTful web services for terminal location. They allow a web application to query the location of one or more mobile device that is connected to a mobile operator network. GSMA RESTful location service is a subset of OMA RESTful location services [39]. Only the service "query the location of one or more terminals" is supported by the GSMA RESTful location services. The other services provide by the OMA RESTful location services are not offered by the GSMA RESTful location services. In order to retrieve the location information (i.e. latitude/longitude) of one or more mobile terminals HTTP GET method is used. POST, PUT and DELTE methods are not used in GSMA RESTful location services. Moreover, to the best of our knowledge the implementation of GSMA RESTful location services with WSN has not been done yet. However, GSMA RESTful location services, a subset of OMA RESTful location services, implemented using technologies other than WSNs.

Reference implementation of GSMA location APIs (i.e. GSMA RESTful location services) are available online for testing against the operator networks in Canada, Europe and Asia pacific region. In Canada and Europe, AEPONA [40] implemented the GSMA RESTful location services using the mobile mast technology. The reference implementation can support both application/XML and/or application/JSON as the resource representation format (i.e. response content type). However, the current reference implementation can only support application/JSON as a representation format. In addition to, Locatrix [41] also implemented the GSMA RESTful location services using the GPS and A-GPS technology in Asia pacific region. However, this implementation can only accept application/XML as a resource representation format.

Moreover, none of the above implementations can offer greater accuracy and also cannot support requesting for multiple terminals. GSMA RESTful location services was not considered for the implementation of location services, as it is a subset of OMA RESTful location services and also its existing implementation cannot provide higher accuracy.

3.2.2.3 OMA RESTful Location Services

OMA RESTful location services [5], discussed in Chapter 2, are standard RESTful web services for terminal location. Since, GSMA RESTful location service is a subset of OMA RESTful location services; we have only one standard solution (i.e. OMA RESTful location services) to be considered in order to implement standard location services. However, OMA RESTful location services could not provide any location information without using any other locator technology. A list of locator technologies that can provide location information includes WSN, GPS, A-GPS and Cell-Id of the currently serving cell in the cellular network. As we mentioned in the introduction chapter, our target is to provide greater accuracy of location information. Accuracy refers to how precise location information a given locator technology may yield [32]. A key question for developing location services is: How much accuracy of location information these locator technologies can provide? We will hereafter compare these different technologies with respect to the accuracy they can provide.

GPS is one of the most well-known locator technologies [6]. It works very well in outdoor environments but its usage in indoor environments is limited as the signals from GPS satellites are too weak to penetrate most of the buildings. Furthermore, along as the Cell-Id and A-GPS, the GPS does not provide the enough accuracy required by several applications. Another alternative to get location information is using WSNs. Wireless sensors can sense location with an accuracy (e.g. within 1-3 centimeters) that most technologies cannot provide, making them the technology of choice for applications that require high accuracy.

Locator Technology / **Criteria**	**GPS**	**A-GPS**	**Cell-Id**	**WSN**
Accuracy	Low	Low	Low	**Higher**

Table 3.2: Comparison of existing locator technologies with their accuracy

Table 3.2 provides a comparison of existing locator technologies and their accuracy. We can see from Table 3.2, WSN provides higher accuracy than any other locator technologies. Therefore, we chose to implement OMA RESTful location services using WSN in order to provide a standard location service.

In the next section we derive the requirements for the implementation of OMA RESTful location services in wireless sensor environments.

3.2.3 Specific Requirements for OMA RESTful Location Services in Wireless Sensor Environments

The specific requirements for the design and implementation of OMA RESTful location services in wireless sensor environments are as follows [19] [59]:

Higher level of abstraction: The end-user should have access to the location information provided by the WSNs, without having to know the internal structure of the WSN networks. The user should not be aware of: how location information can be obtained, what is the network structure of the WNSs is, and how WSNs collaborate to access the information from each other.

Heterogeneity of WSNs: It should be possible to support heterogeneous WSN networks and easily introduce new WSNs.

Reusability of Existing WSNs gateway: It should be possible to reuse existing WSN gateways, if any, to access the information provided by the WSNs. This is important because a plethora of WSNs exist. It is easy to connect new WSNs to the system by reusing existing gateways. This reusability speeds up the development of the system.

Reusability of Existing Technologies/Standards: Existing technologies/standards (e.g. for interconnecting the entity that implements the OMA RESTful location services and the WSNs) should be reused wherever possible.

3.3 Chapter Summary

In this chapter, we discussed the state-of-the-art of standard and non-standard location services and their existing implementation. We also discussed the different locator technologies and derived the specific requirements in order to implement OMA RESTful location services in wireless sensor environments. In the following chapter our novel architecture for the design and implementation of OMA RESTful location services will be discussed.

Chapter 4 An Architecture for OMA RESTful Location Services in WSN Environments

4.1 Introduction

The evaluation of architectures for location services with different technologies were discussed in previous chapter. It has been shown that there is no existing architecture for location service implementation with WSNs. Therefore, we develop a novel architecture for the design and implementation of OMA RESTful location services in wireless sensor environments.

In the next section, we introduce the overall architecture. The detailed design and implementation of our proposed architecture is discussed in the subsequent sections. The architectural design and implementation is divided into two phases; the first phase is involved with the design and implementation of a REST gateway and the second phase is involved with the design and implementation of a WSN gateway.

4.2 Overall Architecture

Figure 4.1 depicts our overall architecture. A REST gateway is used to provide OMA RESTful location services to end-users. To get location information, the gateway is connected to a number of WSNs. Each WSN is divided into a set of spaces/areas, identified by unique identifiers (e.g. Space-a, Space-b and so on). Each space contains a set of sensors and a WSN gateway, which is used to store the location information inside

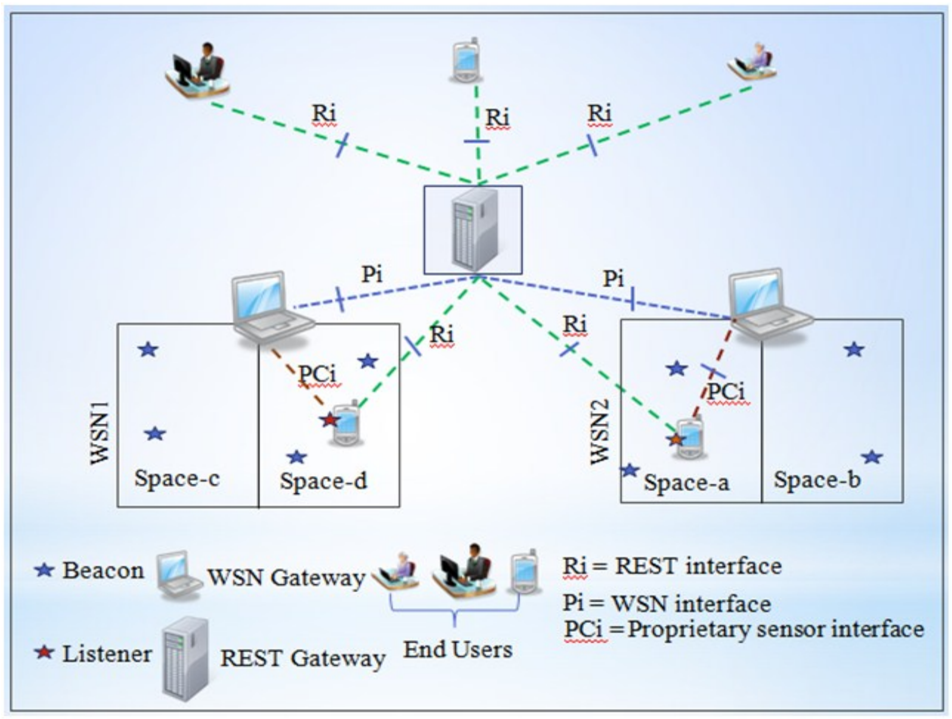


Figure 4.1: Overall architecture

the WSNs and to make this information available to the REST gateway. A sensor is attached to each of the tracked end-user terminals (e.g. cell phone and PDA). When an end-user enters a given space, the location of the sensor attached to his/her terminal is detected and stored in the local WSN gateway (e.g. terminal-1 is currently in Space-b). At later time a user can get the current location of a specific user by sending the request to the REST gateway. The REST gateway receives the request from the end-user and will communicate to the local WSN gateway to get the location information.

The REST gateway communicates with end-users through a REST interface (i.e. Ri), and communicates with the WSN gateways through a UDP-based interface that we define (i.e. Pi). The end-users also can communicate to the REST gateway through the REST interface (i.e. Ri). The WSN gateway communicates with sensor nodes, which is dedicated for collecting location information through proprietary interfaces (i.e. PCi). These interfaces will be discussed in Section 4.7.

4.3 REST Gateway Architecture

The REST gateway is the key component of the system responsible for protocol translation and necessary mappings between REST and WSN. It provides a standard REST interface (i.e. Ri) towards the applications for information exchange. REST gateway composed of a request handler, a subscription repository, a parser/formatter module, a REST-WSN address mapping module, a processing module, and an HTTP client module. Figure 4.2 shows the architectural components of REST gateway architecture.

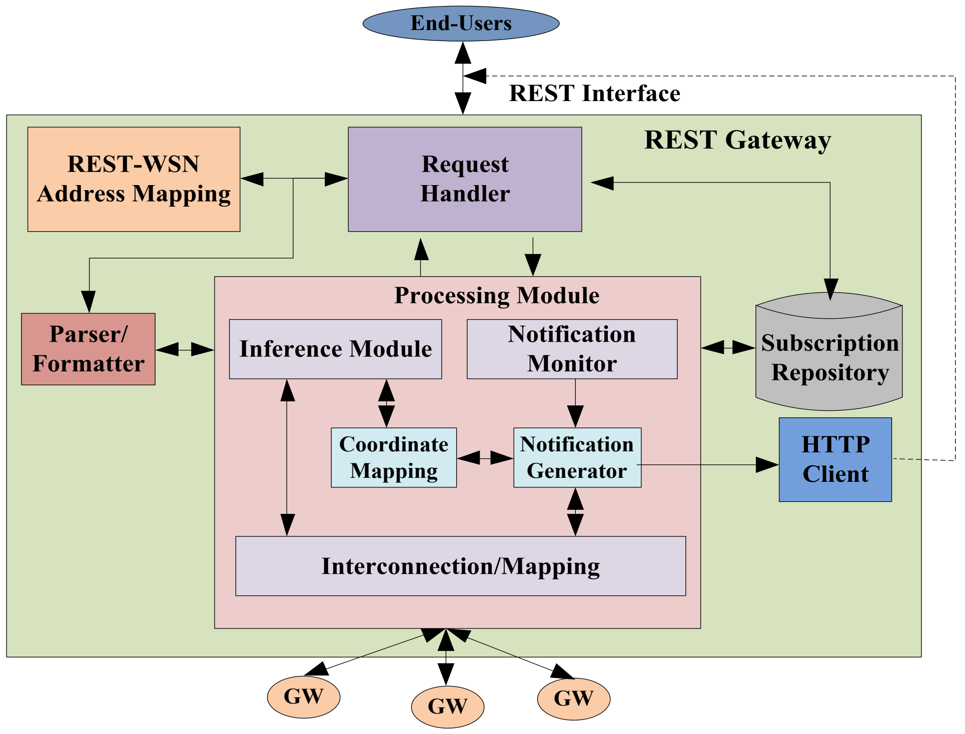


Figure 4.2: REST gateway architecture

4.3.1 Request Handler

The request handler is the entry point of the gateway. It receives and handles the REST requests from end-users. GET requests are forwarded to the processing module, which gets the requested information and sends it to the end-user via the request handler. When a POST request is received, the request handler creates a new subscription, which it then stores in the subscription repository for further processing by the processing module. The PUT/DELETE requests result in the updating/deleting of the appropriate subscriptions.

4.3.2 Subscription Repository

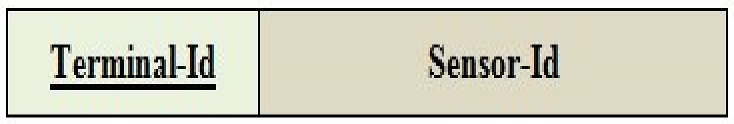
Subscription repository is the storage location responsible for storing all information related to the subscriptions. Application can create a subscription towards the REST gateway by storing the information into the subscription repository. Once the subscription information stored in the repository, application can modify or delete this information at any time by sending request to the REST gateway.

4.3.3 Parser/Formatter Module

The parser/formatter module is used to parse the body of incoming REST requests and to format the responses' contents. This module is also used to transform WSN data into standard REST format.

4.3.4 REST-WSN Address Mapping Module

The REST-WSN address mapping module maps the addresses of the OMA location service elements into sensor specific addresses. A Terminal-Id (i.e. a phone number) is associated to the Id of the sensor that is attached to it. A REST-WSN address mapping table is used to map Terminal-Id to Sensor-Id, which is preconfigured and static. The REST-WSN mapping table format is shown in Figure 4.3.



Terminal-Id	Sensor-Id

Figure 4.3: REST-WSN mapping table format

Terminal-Id is the unique Id of a terminal (i.e. phone number) and Sensor-Id is the Id of the sensor attached with the associated terminal. A sample REST-WSN mapping table is shown in Table 4.1.

Terminal-Id	Sensor-Id
+1 514 581 7818	Sensor01
+1 514 581 7819	Sensor02
+1 514 581 7820	Sensor03
+1 514 581 7821	Sensor04
+1 514 581 7822	Sensor05
+1 514 581 7850	Sensor06
+1 514 581 7851	Sensor07
+1 514 581 7852	Sensor08
+1 514 581 7853	Sensor09

Table 4.1: Sample REST-WSN address mapping table

4.3.5 Processing Module

The processing module is the core module of the REST gateway. It is composed of an inference module, an interconnection/mapping module, a coordinate mapping module, a notification monitor and a notification generator.

Inference Module: The inference module receives GET requests from the request handler. It uses the interconnection/mapping module to get the requested location or distance information from the relevant WSN(s). After that it puts that information into the appropriate format, and passes to the request handler.

Interconnection/Mapping Module: The interconnection/mapping module is responsible for the communication with the WSN gateways. It performs request-mapping between the REST gateway and the WSN gateways.

Coordinate Mapping Module: The location information, which is received from the WSNs (e.g. terminal-1 is in Space-b), is translated into OMA location-information model (e.g. the latitude, longitude and altitude of the current location of terminal-1) by the coordinate mapping module. The coordinate mapping technique and the WSN raw data mapping with OMA location-information model are conversed in Section 4.5.

Notification Monitor: The notification monitor controls the subscriptions (i.e. stored in the subscription repository) and triggers notifications when appropriate. The triggers are fired towards the notification generator.

Notification Generator: Notification generator generates and sends the notifications to the HTTP client module in order to notify it. As with the inference module, the notification generator collects location information from the WSNs through the interconnection/mapping module, translates the WSN specific location information to the OMA location format using the coordinate mapping module, and generates the notification content using the parser/formatter module.

4.3.6 HTTP Client

HTTP client module is a special module required to implement OMA RESTful location services. Notification generator triggers the HTTP client module with the appropriate notifications. HTTP client module forwards them to the concerned end-users through a POST request sent to the appropriate callback URI.

4.4 WSN Gateway Architecture

We have also designed a proprietary WSN gateway based on the MIT Cricket sensors [2]. Since, there is no existing gateway for such type of sensors. The WSN gateway is responsible for collecting location information from the cricket sensors and providing this information towards the REST gateway. It also provides a UDP-based interface (i.e. Pi) on the way to the REST gateway for WSN data exchange. The WSN gateway architecture is shown in Figure 4.4. This architecture is structured into two layers: data provider and connectivity layer.

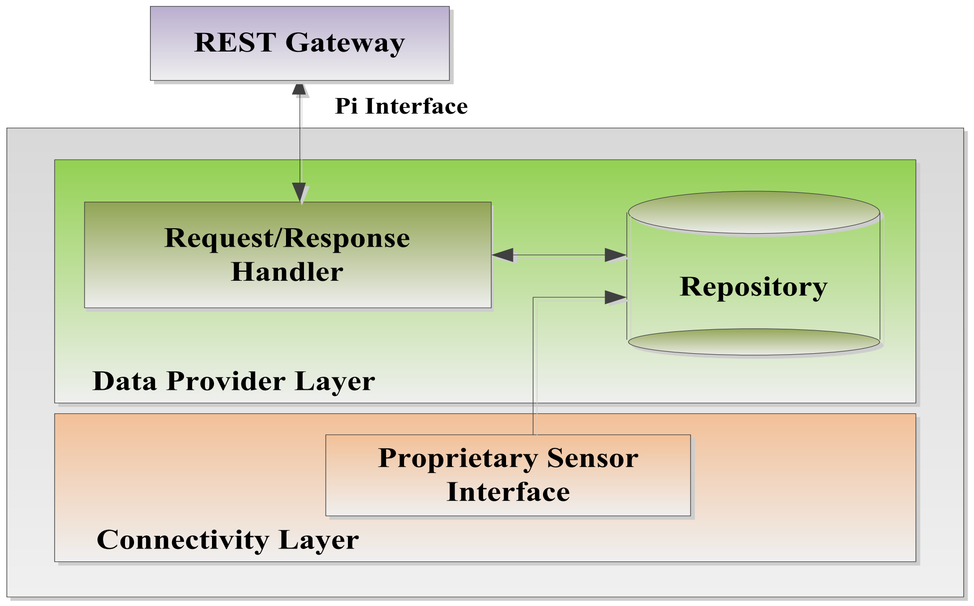


Figure 4.4: WSN gateway architecture

4.4.1 Data Provider Layer

Data provider layer provides WSN data towards the REST gateway. It is responsible for handling request from REST gateway. It consists of a request/response handler and a repository.

Request/Response Handler: Request/response handler is the entry point of the WSN gateway. It receives and handles request from REST gateway. When REST gateway sends the request to the WSN gateway, it also specifies the Sensor-Id in the request messages. The Sensor-Id indicates the specific sensor information required for the REST gateway. When request/response handler receives a request, it will communicate with the repository to obtain the appropriate sensor information (i.e. Space-Id) based on the specified Sensor-Id. If the information found in the repository then request/response handler send back this (i.e. Space-Id) information to the REST gateway.

Repository: The repository is the storage location of WSN related data (i.e. Space-Id). It is also responsible for storing the sensed WSN information for the uses of near future. A sample table of recorded WSN data is shown in Table 4.2. Where, Sensor-Id indicates a unique identifier which is used to identify a sensor uniquely. On the other hand, Space-Id specifies a space where a specific sensor is currently located. If the sensor changes its position from one space to another, it will either update the existing information or create a new record in the table.

Sensor-Id	**Space-Id**
Sensor01	Space-a
Sensor02	Space-b
Sensor03	Space-c
Sensor04	Space-d
Sensor05	Space-e
Sensor06	Space-f

Table 4.2: Sample table of recorded WSN data

4.4.2 Connectivity Layer

In the connectivity layer, the proprietary cricket WSN interface enables communication between WSN gateway and individual cricket sensor node. The proprietary sensor interface is the key module of the connectivity layer.

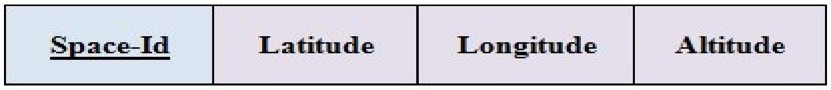
Proprietary Sensor Interface (PCi)*:* The proprietary sensor interface implements a communication stack of a sensor in order to establish a communication between a WSN gateway and individual sensor. The communication interfaces between sensor nodes are also proprietary.

4.5 Mappings

In this section we will be discussing the coordinate mapping technique and the WSN raw data mapping with OMA location-information model.

4.5.1 Coordinate Mapping Technique

OMA location information is based on latitude, longitude and altitude. However, WSN is sensed location information as a proprietary format which is meaningless for OMA services. Therefore, a mechanism is required to transform WSN raw data into OMA location information. Several transformation techniques can be used. Among all of the techniques, we picked up the coordinate mapping technique. Since it does not require any pre-configuration of sensor node deployment and any processing for coordinate transformation from one coordinate system to another. In mapping technique, a preconfigured and static mapping table is used to map the WSN Space-Id with the corresponding latitude longitude and altitude. The coordinate mapping table format is shown in Figure 4.5.



Space-Id	Latitude	Longitude	Altitude

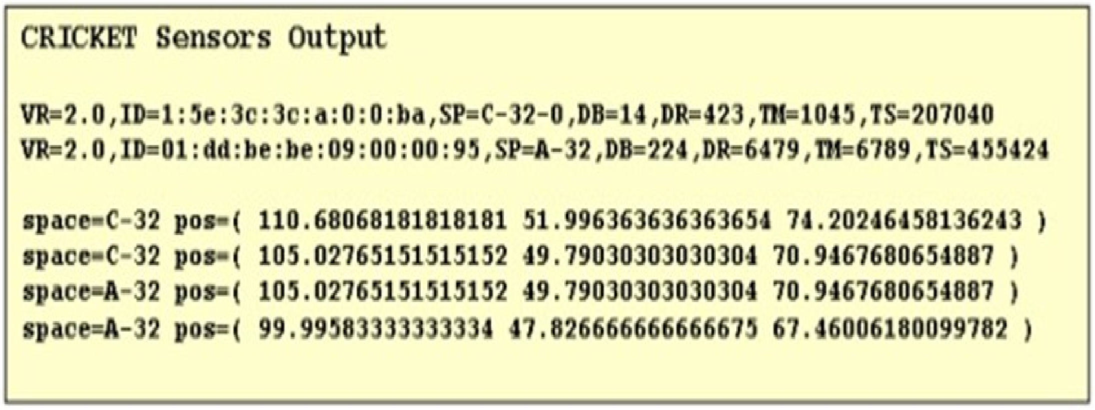
Figure 4.5: REST-WSN mapping table format

Space-Id is the unique space identifier of a space. Latitude, longitude, and altitude are the corresponding values of that associated space. Table 4.3 shows a sample coordinate mapping table.

<u>Space-Id</u>	Latitude	Longitude	Altitude
Space-a	23.00	-712.214	743.121
Space-b	63.00	-912.224	143.121
Space-c	123.00	-12.234	343.121
Space-d	-233.00	-312.234	243.121
Space-e	33.00	-12.234	-243.121
Space-f	522.65	-12.214	-943.121

Table 4.3: Sample coordinate mapping table

Figure 4.6 shows the Space-Id and the corresponding raw data stream captured from the MIT cricket sensor. This sensor raw data should be mapped with OMA location-information model.



```
CRICKET Sensors Output

VR=2.0,ID=1:5e:3c:3c:a:0:0:ba,SP=C-32-0,DB=14,DR=423,TM=1045,TS=207040
VR=2.0,ID=01:dd:be:be:09:00:00:95,SP=A-32,DB=224,DR=6479,TM=6789,TS=455424

space=C-32 pos=( 110.68068181818181 51.996363636363654 74.20246458136243 )
space=C-32 pos=( 105.02765151515152 49.79030303030304 70.9467680654887 )
space=A-32 pos=( 105.02765151515152 49.79030303030304 70.9467680654887 )
space=A-32 pos=( 99.99583333333334 47.82666666666675 67.46006180099782 )
```

Figure 4.6: Cricket sensor output

4.5.2 WSN Raw Data Mapping with OMA Location-Information Model

The complete scenario of mapping between WSN data and OMA location-information model can be realized at different levels of data abstraction shown in Figure 4.7, Figure 4.8, and Figure 4.9. At the lowest level of data abstraction, the WSN raw data is sensed from the individual sensor and processed and transformed into OMA location-information data object. The OMA location-information data object is then formatted into XML or JSON document at the highest level and can be served as OMA location-information. OMA location-information consists of three different data objects include: terminal-location, terminal-distance and subscription-notification.

Terminal-location data mapping: Figure 4.7 shows WSN data mapping with the terminal-location object of OMA location-information model. The terminal-location object is classified into address, location-retrieval-status and current-location element. The address and location-retrieval-status are the elements of OMA location-information model. The address represents the address of a terminal whose location information is requested. Location-retrieval-status represents the success or failure of retrieving the location information for a particular terminal. Current-location element represents latitude, longitude, altitude, timestamp and accuracy. The latitude, longitude and altitude of current-location element are mapped with the converted latitude, longitude, and altitude of corresponding Space-Id of actual WSN data. These converted latitude, longitude and altitude can be found from coordinate mapping table. This latitude, longitude and altitude of the corresponding Space-Id are acquired from the coordinate mapping table.

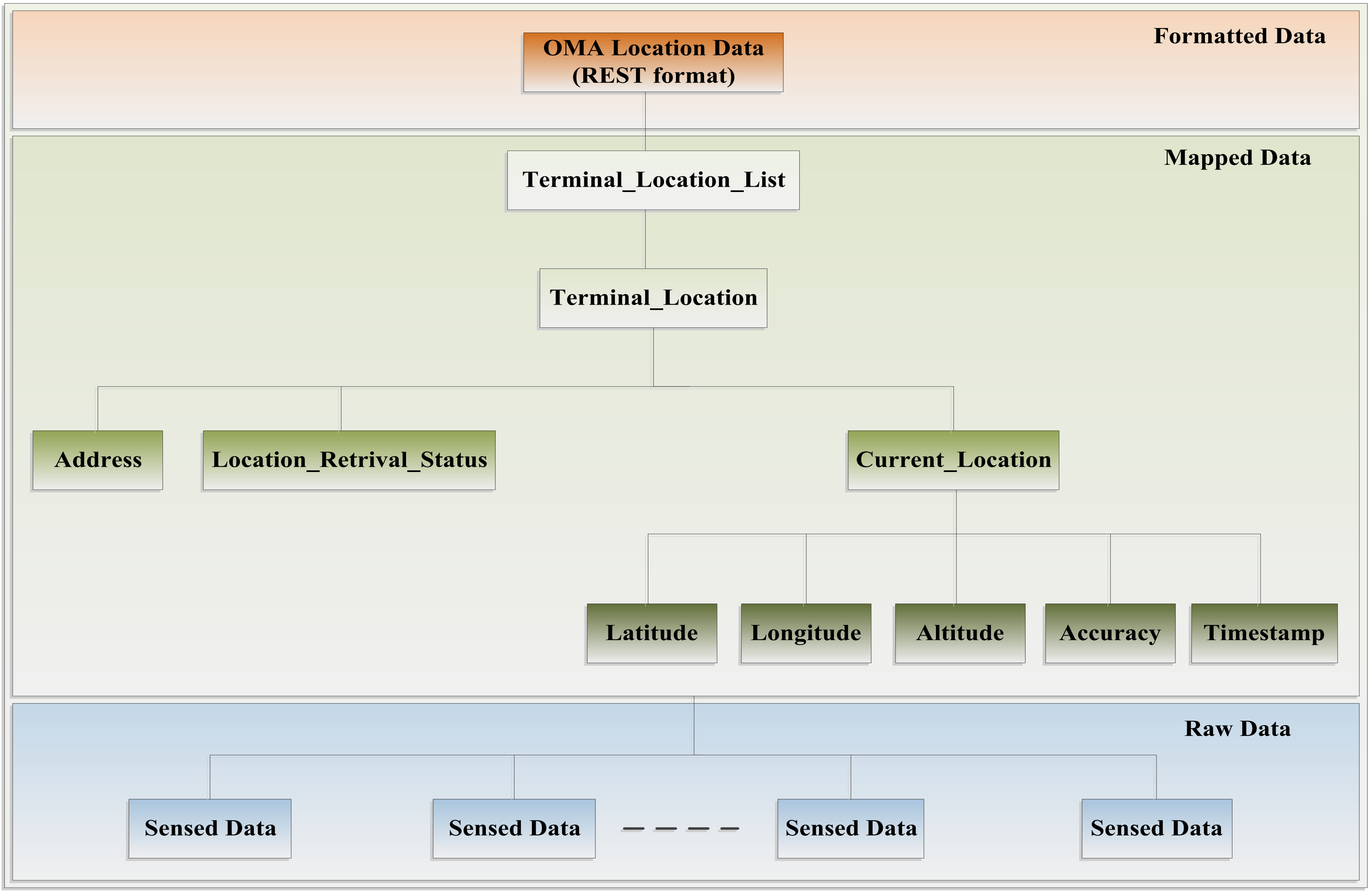


Figure 4.7: Mapping of WSN data with OMA terminal-location object

Terminal-distance mapping: Figure 4.8 shows the mapping of WSN data and terminal-distance object of OMA location-information model. The terminal-distance object is classified into distance, accuracy and timestamp element. The distance element of terminal-distance object is mapped with the calculated distance between two terminals or from a given location. REST gateway performs some additional operations in order to calculate the distance. The additional operations of REST gateway are performed as follows: it receives WSN raw data from WSN gateway and then converted into OMA location format (i.e. latitude, longitude and altitude) using coordinate mapping table. After that it calculates the distance and finally maps with distance element of terminal-distance object of OMA location-information model.

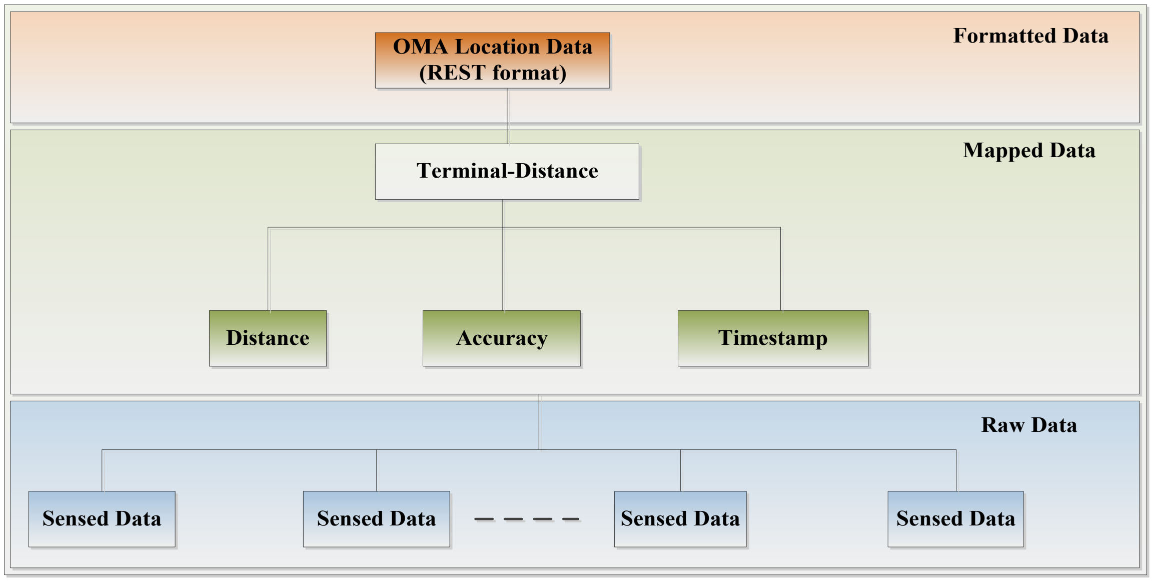


Figure 4.8: Mapping of WSN data with OMA terminal-distance object

Subscription-notification mapping: The WSN data mapping with subscription-notification object of OMA location-information model is shown in Figure 4.9.

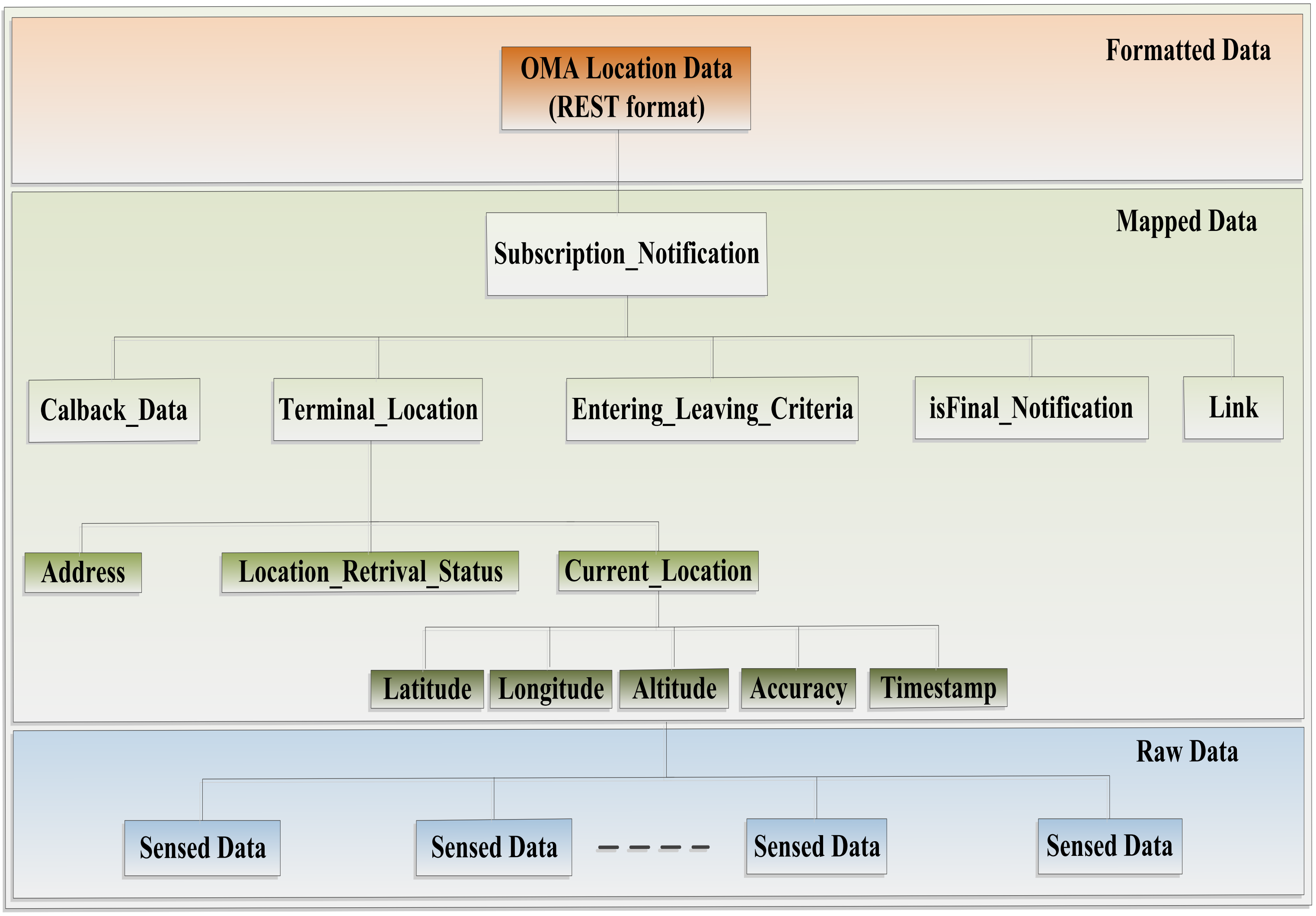


Figure 4.9: Mapping of WSN data with OMA subscription-notification

Subscription-notification object is classified into callback-data, terminal-location, entering-leaving-criteria, isFinal-notification and link elements. The terminal-location element mapping with WSN raw data is already discussed in earlier.

4.6 Operational Procedures

This section describes the procedures used to answer the various REST requests, with a focus on location information mapping between the OMA services and the WSN environment. The operational procedures include the query and subscription procedures are discussed in Section 4.6.1 and 4.6.2 respectively.

4.6.1 Query Procedures

This section discusses the query procedures to answer the REST request. The query procedures are classified into get terminal location, get distance between two terminals and get distance from a given location.

4.6.1.1 Get Terminal Location

Figure 4.10 depicts the procedures for getting terminal location information. After the request handler gets the request (with the Terminal-Id) from the end-user, it uses the REST-WSN address mapping module to get the Id of the sensor attached to the terminal. It then forwards the request with the sensor Id to the inference module, which transmits the request to all of the attached WSN gateways through the interconnection/mapping module. The WSN gateway that senses the current location of the tracked sensor replies with the location information (i.e. the Space-Id). The inference module then uses the coordinate mapping module to get the OMA-specific location information (i.e. latitude, longitude and altitude) associated with the center of the returned space. This module then

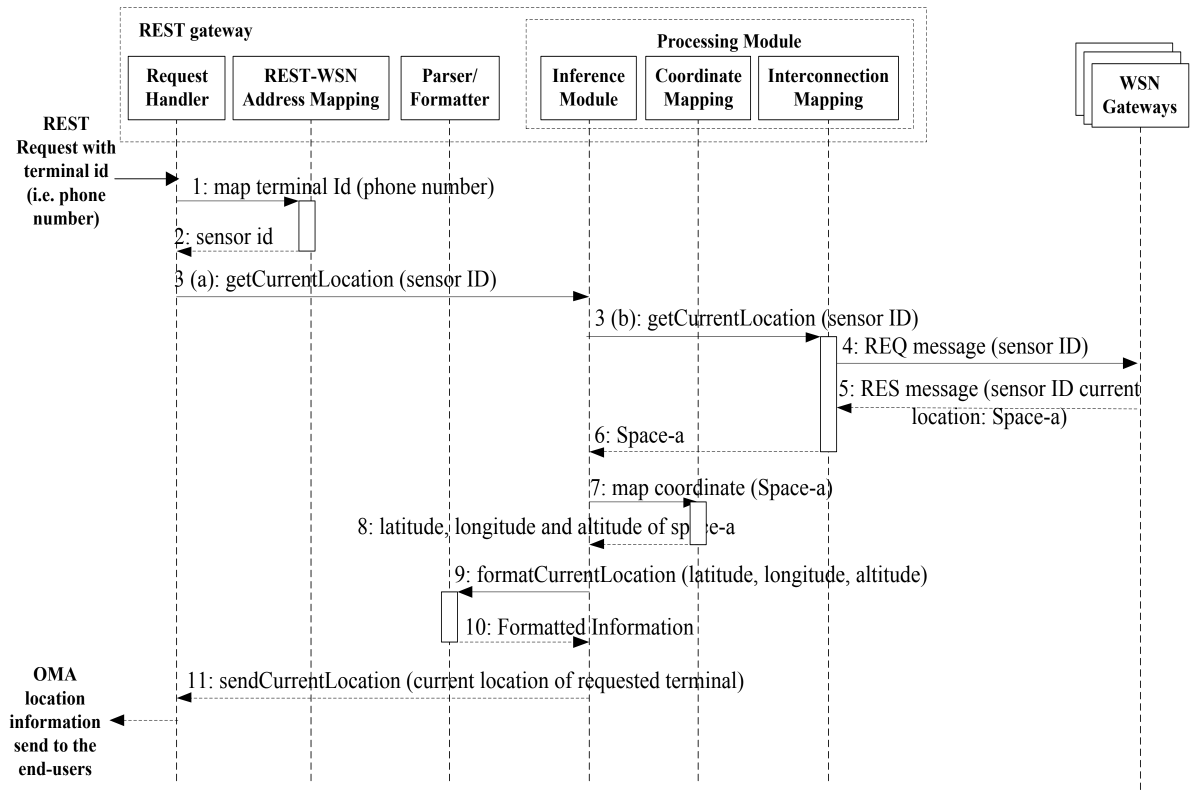

REST gateway
Processing Module
Request Handler
REST-WSN Address Mapping
Parser/ Formatter
Inference Module
Coordinate Mapping
Interconnection Mapping
WSN Gateways
REST Request with terminal id (i.e. phone number)
1: map terminal Id (phone number)
2: sensor id
3 (a): getCurrentLocation (sensor ID)
3 (b): getCurrentLocation (sensor ID)
4: REQ message (sensor ID)
5: RES message (sensor ID current location: Space-a)
6: Space-a
7: map coordinate (Space-a)
8: latitude, longitude and altitude of space-a
9: formatCurrentLocation (latitude, longitude, altitude)
10: Formatted Information
11: sendCurrentLocation (current location of requested terminal)
OMA location information send to the end-users

Figure 4.10: Procedures for getting terminal location

formats this information with the help of parser/formatter module and sent this formatted information to the end-user in the response message. We should mention that the coordinate mapping module maintains, for each space, the mapping between the Space-Id and the OMA coordinates (i.e. latitude, longitude and altitude) of the space center.

4.6.1.2 Get Distance Between Two Terminals

The inference module gets the location of the two terminals (e.g. Space-a and Space-b) from the WSN(s), and transfers them to the coordinate mapping module. The latter gets the OMA coordinates associated with the center of both spaces, and then calculates the distance between them. This can be done mathematically [60]. The procedures for getting distance between two terminals are shown in Figure 4.11.

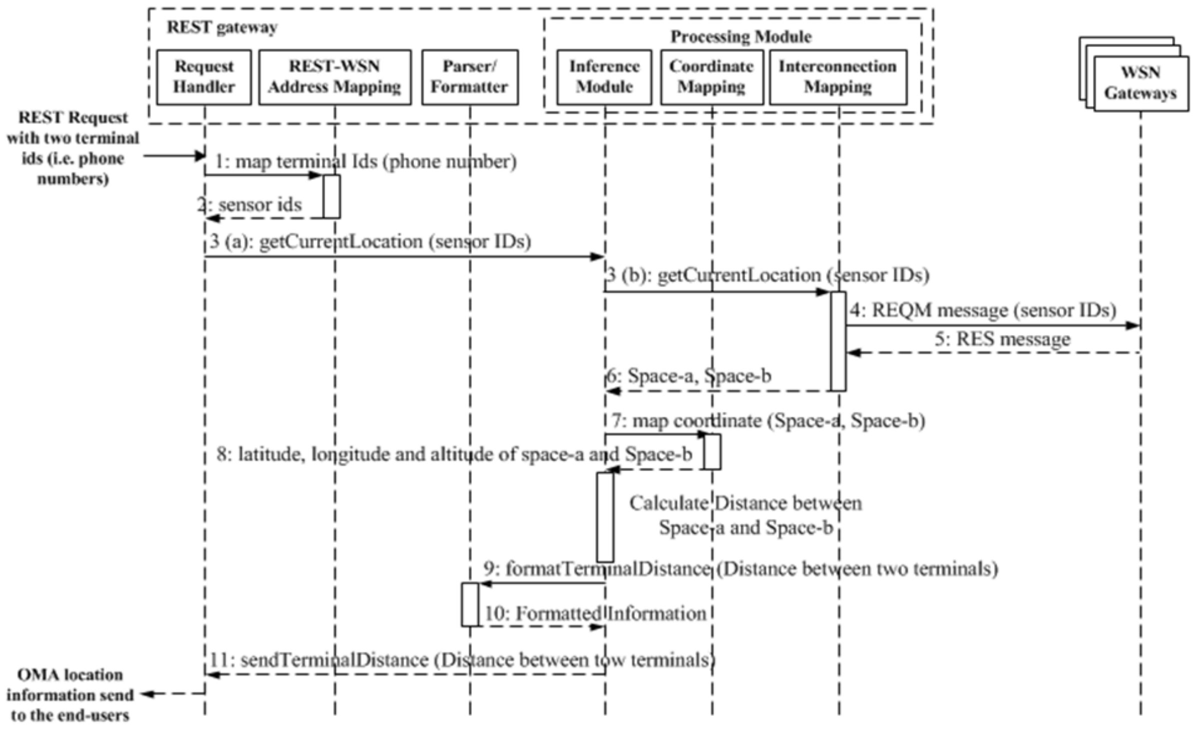

REST gateway
Processing Module
Request Handler
REST-WSN Address Mapping
Parser/ Formatter
Inference Module
Coordinate Mapping
Interconnection Mapping
WSN Gateways
REST Request with two terminal ids (i.e. phone numbers)
1: map terminal Ids (phone number)
2: sensor ids
3 (a): getCurrentLocation (sensor IDs)
3 (b): getCurrentLocation (sensor IDs)
4: REQM message (sensor IDs)
5: RES message
6: Space-a, Space-b
7: map coordinate (Space-a, Space-b)
8: latitude, longitude and altitude of space-a and Space-b
Calculate Distance between Space-a and Space-b
9: formatTerminalDistance (Distance between two terminals)
10: Formatted Information
11: sendTerminalDistance (Distance between tow terminals)
OMA location information send to the end-users

Figure 4.11: Procedures for getting terminal distance between two terminals

4.6.1.3 Get Distance from a Given Location

To get the distance between a terminal and a given location, the REST request from the end-user includes the Terminal-Id and the latitude, longitude and altitude of the location. The processing module gets the latitude, longitude and altitude of the terminal from the relevant WSN and then calculates the distance in the same way as in the distance between two terminals, described above. The total procedures for getting distance from a given location are shown in Figure 4.12.

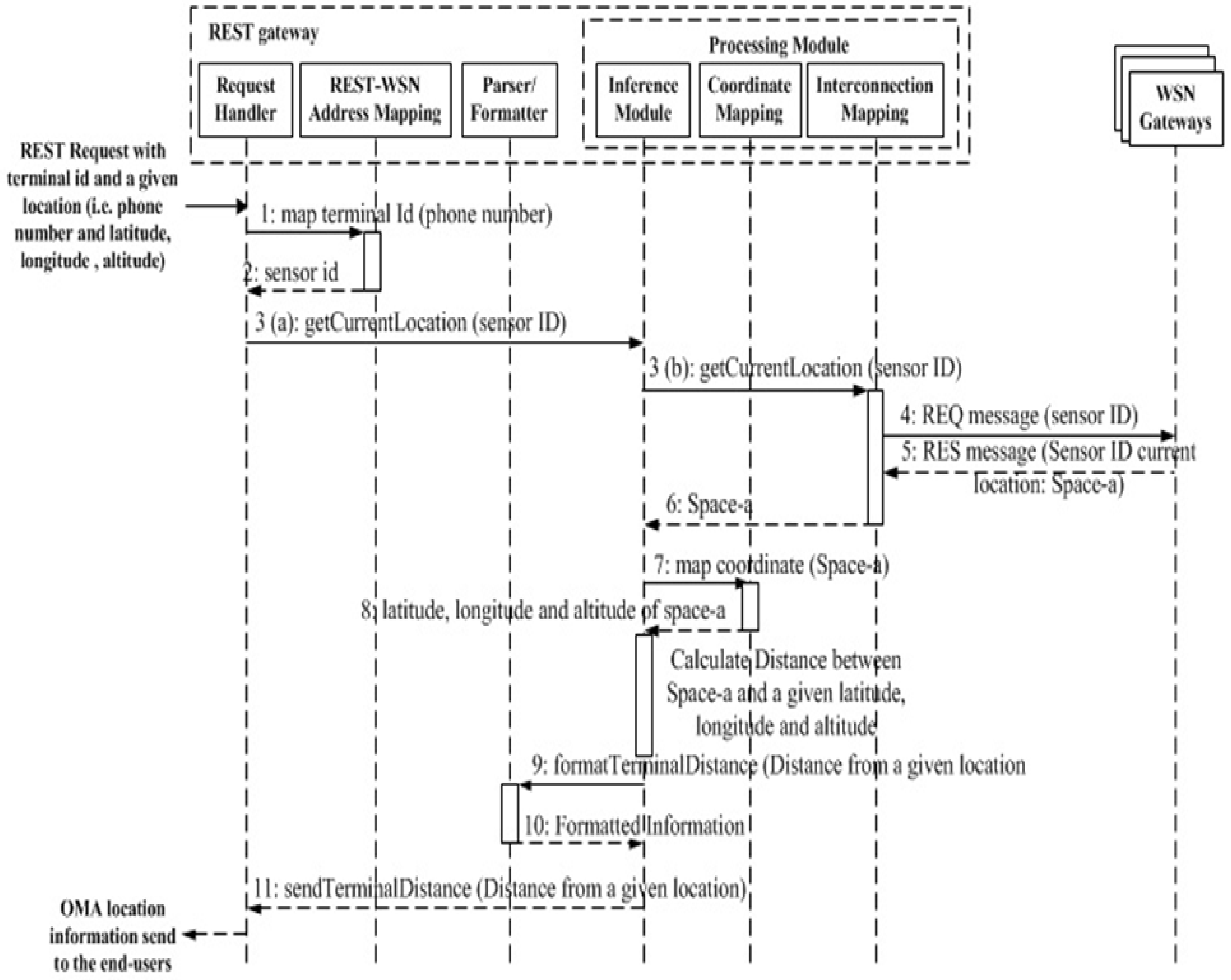

REST gateway
Processing Module
Request Handler
REST-WSN Address Mapping
Parser/ Formatter
Inference Module
Coordinate Mapping
Interconnection Mapping
WSN Gateways
REST Request with terminal id and a given location (i.e. phone number and latitude, longitude , altitude)
1: map terminal Id (phone number)
2: sensor id
3 (a): getCurrentLocation (sensor ID)
3 (b): getCurrentLocation (sensor ID)
4: REQ message (sensor ID)
5: RES message (Sensor ID current location: Space-a)
6: Space-a
7: map coordinate (Space-a)
8: latitude, longitude and altitude of space-a
Calculate Distance between Space-a and a given latitude, longitude and altitude
9: formatTerminalDistance (Distance from a given location
10: FormattedInformation
11: sendTerminalDistance (Distance from a given location)
OMA location information send to the end-users

Figure 4.12: Procedures for getting terminal distance from a given location

4.6.2 Subscription Procedures

In this section the subscription procedures are discussed to answer the REST requests. The subscription procedures are classified into area/circle notification and distance notification. These procedures are shown in Figure 4.13 and discussed in the following sections.

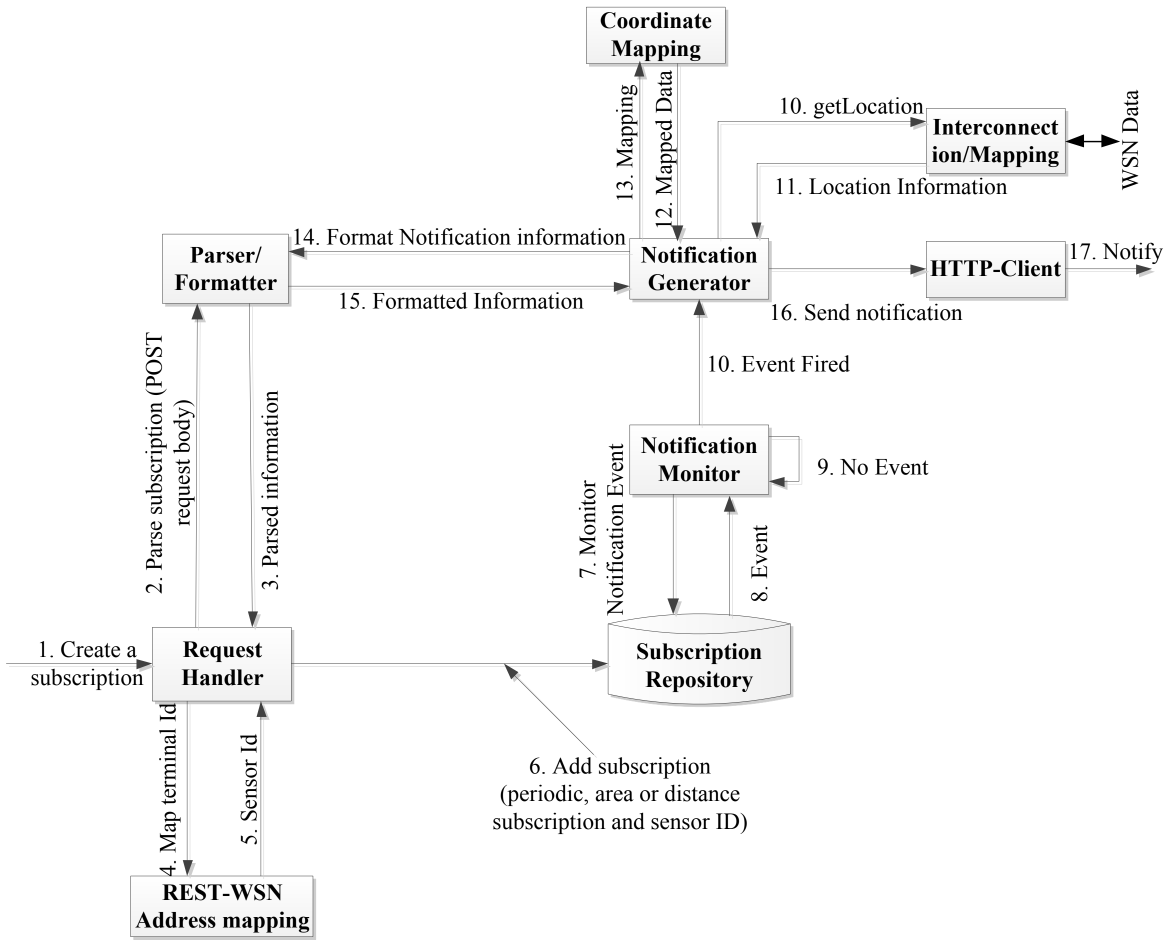


Figure 4.13: Operational procedures for subscription resources

4.6.2.1 Area/Circle Notification

When an end-user creates an area/circle subscription, he/she has to provide the Id of the terminal to be monitored; the OMA coordinates of the center point of the area, as well as the area radius (e.g. 50 cm). The end-user should also specify if he/she wishes to be notified when this terminal enters or leaves the area.

The notification monitor (periodically) triggers the notification generator, which then gets the current location of the terminal (in terms of OMA coordinates) and compares it to that of the area’s center. A notification is sent to the end-user if the difference between the two coordinates is less than the specified radius and the subscription is for ‘entering the area’, or if the difference is greater than the radius and the subscription is for ‘leaving the area’.

4.6.2.2 Distance Notification

To subscribe the distance notification, an end-user should specify the terminal to be monitored, the reference terminal or location (i.e. the OMA coordinates of the reference location), the distance threshold, and the comparison criteria (e.g. the distance exceeds/goes below the threshold). When triggered, the notification generator gets the distance between the monitored terminal and the reference terminal/location, compares the distance to the specified threshold, and sends a notification when the comparison criterion is satisfied.

4.7 Interfaces

This section discusses the different interfaces shown in Figure 4.1. Three interfaces are defined as: REST, WSN, and proprietary sensor interfaces.

4.7.1 REST Interface (Ri)

This is the REST interface provided towards the end-users. It enables to access the OMA RESTful location services discussed in the background section of Chapter 2.

4.7.2 WSN Interface (Pi)

In this section we discuss the interfaces provided by the WSN gateway towards the REST gateway. This interface enables to access WSN data. WSN gateway provides this interface on the way to the REST gateway. We have derived a set of requirements in order to design this interface. The derived requirements are as follows:

Unified Interface: To enable the easy support of heterogeneous WSNs and easy interconnection of new WSNs, the WSN gateways should provide a unified interface.

Simplicity: The second requirement is simplicity. The interface should be simple to use and to implement, and be easy to plug into any WSN gateway. This will further simplify new WSN interconnection.

Efficiency: The third requirement is efficiency in terms of network and time overhead.

Web services, socket and HTTP are existing protocols. This interface can use any one of them. Our goal is to transfer raw WSN information towards the REST gateway in a faster way. To achieve this goal we have chosen socket as a communication protocol to implement this interface. Since, it is very easy to implement rather than other existing protocols. However, socket communication is based on Transmission Control Protocol (TCP) and UDP. We evaluate the TCP and UDP based socket communication according to our requirements. The following sections represent the evaluation of the TCP and UDP based socket communication according to our requirements.

TCP: TCP is a connection oriented protocol. In TCP protocol a connection must be established before the transmission initiates. After the connection is established, it begins to transfer the file until the connection goes down. If the file is lost in between the transmission, the receiver will again request for the lost part of the file. This ensures the reliability but takes some time to establish the connection. However, it introduces much more network overhead due to the reliability and error checking.

UDP: UDP is a connectionless protocol. UDP is much faster than TCP as it does not require establishing a connection. Moreover, it does not introduce much network overhead (e.g. there is no extra overhead for reliability or error checking). In addition, it is easily supported by any WSN gateway. The evaluation summary of TCP and UDP protocol for WSN interface is shown in Table 4.4.

Protocol / **Requirements**	**TCP Socket**	**UDP Socket**
1. Unified Interface	Yes	**Yes**
2. Simplicity	No	**Yes**
3. Efficiency	No	**Yes**

Table 4.4: Evaluation of TCP and UPD protocol for WSN interface

The summary presented in Table 4.4 is shown that UDP is more suitable as it satisfies all of our requirements. So, UDP-based protocol is chosen in order to implement the WSN interface.

The messages, which are exchanged through this interface, are composed of a common header that indicates the message type, followed by a message body that carries the actual raw information (e.g. Sensor-Id or Space-Id). Figure 4.14 shows the message format and the required messages to exchange through this interface. Three types of messages are defined as request message for single terminal (REQ), request message for multiple terminals (REQM), and response message for both single and multiple terminals (RES). REQ is used to request the current location of a single terminal. On the other hand

REQM is used to request the location of more than one terminal. RES is responsible for sending back the current location of the requested terminals.

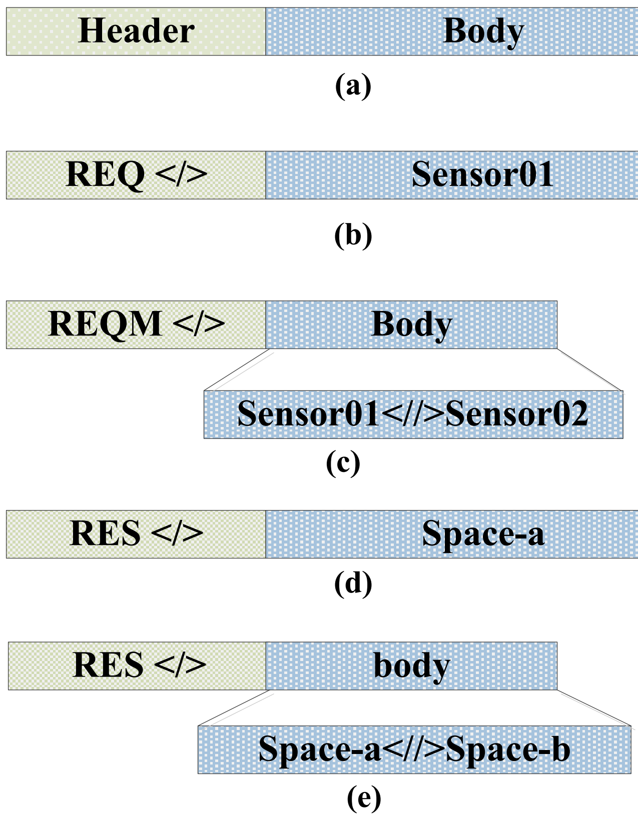


Figure 4.14: (a) Message format, (b) Request message, (c) Multiple request message, (d) Response message and (e) Multiple response messages

The message is formed by a header and a body. Where header contains the type of the message (e.g. request or response) and the body carries the actual raw information (e.g. Sensor-Id or Space-Id). In order to encode and decode the contents of the body we use <//>. After that we use "</>" to encode and decode the message. Figure 4.14 (b) represents the request type message for single terminal where header defines request type message and body contains information of a Sensor-Id (e.g. Sensor01). Figure 4.14 (c) indicates the request type message for multiple terminals where header defines multiple request type message and body contains information of multiple sensor ids (e.g. Sensor01 and Sensor02). Figure 4.14 (d) shows the response type message where header defines response type message and message body carries the actual information of the specified senor-Id, which is a Space-Id (e.g. Space-a). Figure 4.14 (e) depicts the response messages corresponding to Figure 4.14 (c).

4.7.3 Proprietary Sensor Interface (PCi)

Proprietary sensor interfaces between WSN gateway and the sensor nodes dedicated to collect location information. It is dependent on the sensors which are composing the WSN. It may also differ from one WSN to another. A more details information can be found on this proprietary sensor interfaces in [61].

4.8 Implementation

This section considers for the implementation of REST gateway and WSN gateway. It also discusses the implementation environment.

4.8.1 Implementation of REST Gateway

In order to implement REST gateway we have chosen Jersey [62] toolkit to realize the function blocks (i.e. components) of the REST gateway. Jersey is an open source JAX-RS (JSR-311 [69]) reference implementation for building RESTful web services. We have implemented all the components of REST gateway. Initially the implementation of the components of REST gateway architecture is structured into Java packages. Afterwards, Java packages are structured into Java classes. Figure 4.15 (generated by Altova UModel software [63]) shows Java packages and their dependencies.

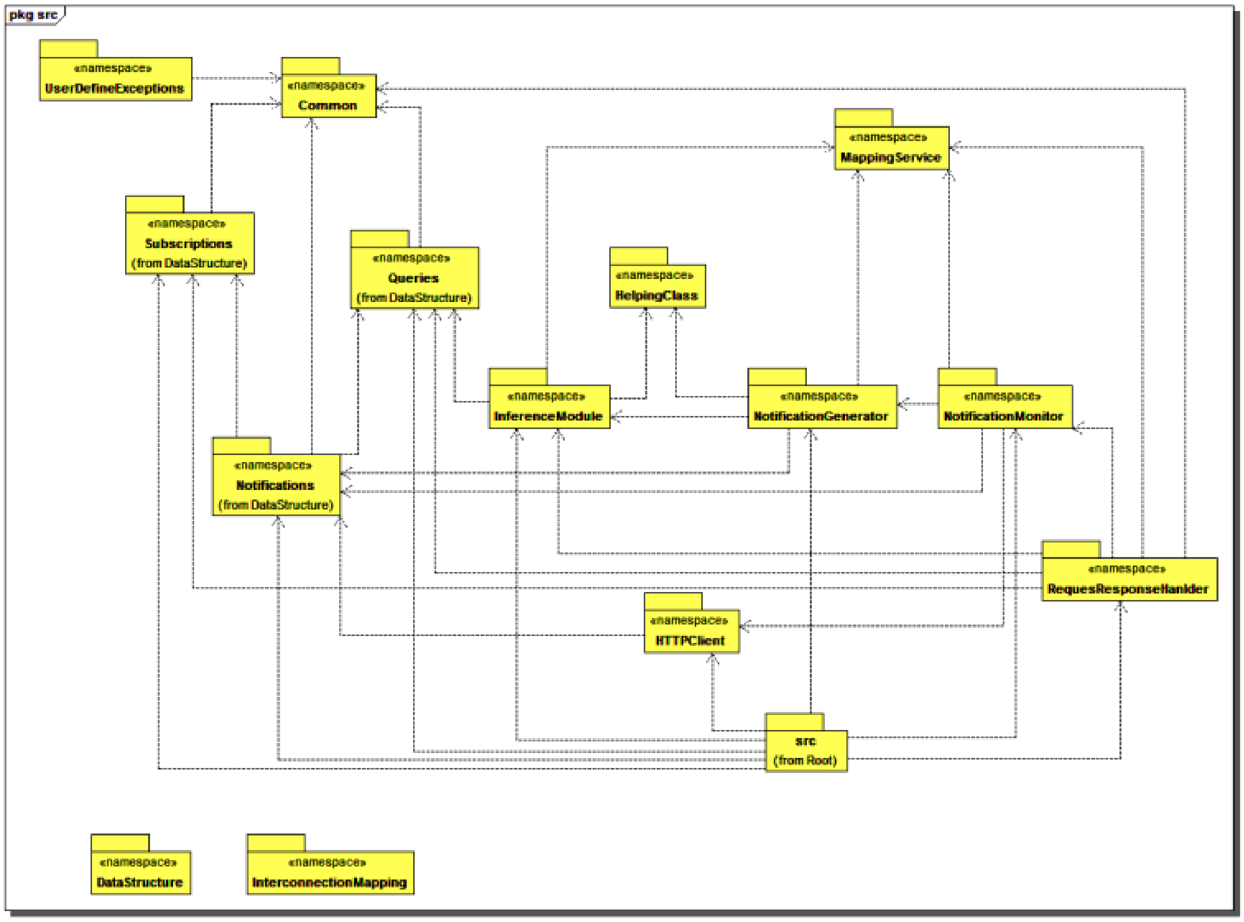
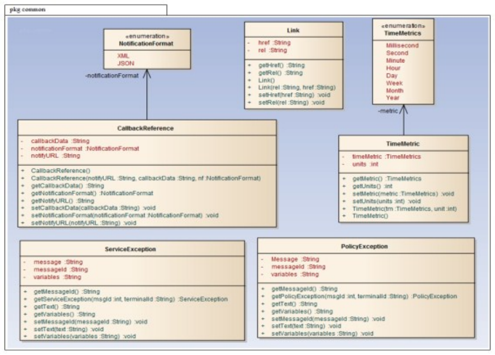

pkg src
«namespace» UserDefineExceptions
«namespace» Common
«namespace» MappingService
«namespace» Subscriptions (from DataStructure)
«namespace» Queries (from DataStructure)
«namespace» HelpingClass
«namespace» InferenceModule
«namespace» NotificationGenerator
«namespace» NotificationMonitor
«namespace» Notifications (from DataStructure)
«namespace» RequesResponseHanlder
«namespace» HTTPClient
src (from Root)
«namespace» DataStructure
«namespace» InterconnectionMapping

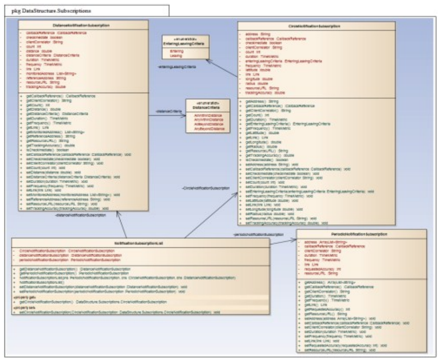
Figure 4.15: Package structure and its dependencies for the implementation of REST gateway

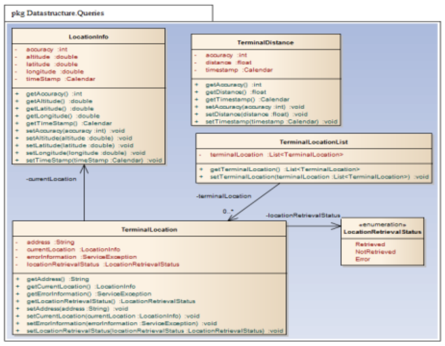
A list of Java packages recognized by REST gateway includes RequestResponseHandler, MappingService, InferenceModule, InterconnectionMapping, NotificationGenerator, NotificatioMonitor, HTTPClient, Datastructure, Common, UserDefinedExceptions, and HelpingClass. In order to implement REST gateway the realization of classes, which is generated by Enterprise Architecture software [64], inside each package are shown in Figure 4.16. '**DataStructure**' package is divided into three sub-packages such as 'Datastructure.Queries', 'Datastructure.Subscription', and 'Datastructure.Notifications'. The classes of data 'DataStructure' and '**Common**' package are responsible for the implementation of all data structures of OMA RESTful location services. ‘**HelpingClass**’ package contains ‘DistanceCalculation’ and String2StringArr class. Where ‘DistanceClaculation’ class is used to calculate the distance between two geographical locations and ‘String2StringArr’ class is used to convert from string to string array.

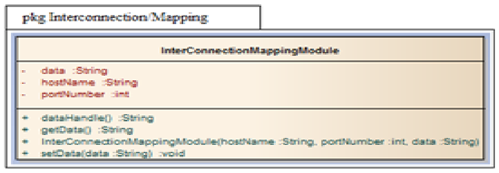
‘**RequestResponseHandler**’ package comprises ‘RequestResponseHandler’ and ‘SubscriptionHandler’ class shown in Figure 4.16 (l). The ‘RequestResponseHandler’ class is responsible for handling the entire query request and the ‘SubscriptionHandler’ class is responsible for handling the entire subscription request. ‘**MappingService**’ package contains all the classes responsible for different mappings shown in Figure 4.16 (i). The main classes of this package are ‘TerminalId2SensorIdmap’, ‘SensorId2LatLonMap’. Where ‘TerminalId2SensorId’ class is responsible for the REST and WSN address mapping and ‘SensorId2LatLonMap’ is responsible for mapping between OMA coordinate and WSN Space-Id.

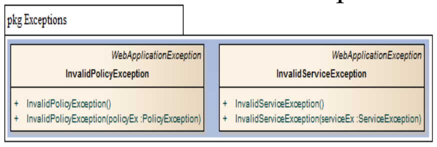
‘**InferenceModule**’ package contains ‘TerminalLocationInfo’ and ‘TerminalDistanceInfo’ classes shown in Figure 4.16 (g). Where ‘TerminalLocationInfo’ is responsible for providing location information and ‘TerminalDistanceInfo’ is responsible for providing terminal distance information triggered by the ‘RequestResponseHandler’ class. ‘NotificationMonitor’ package is composed of ‘AreaNotification’, ‘PeriodicNotification’ and ‘DistanceNotification’ classe. These classes are responsible for detecting the notification event from the subscription repository shown in Figure 4.16 (k).

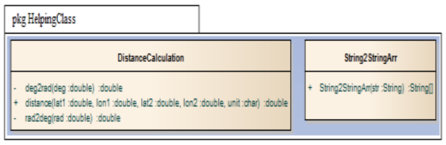

(a) pkg Common

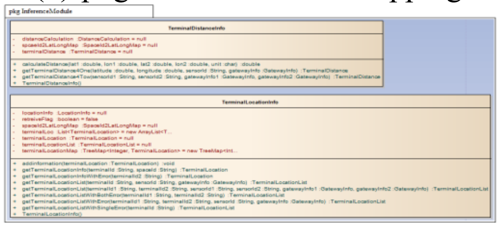

(b) pkg Datastructure.Subscriptions

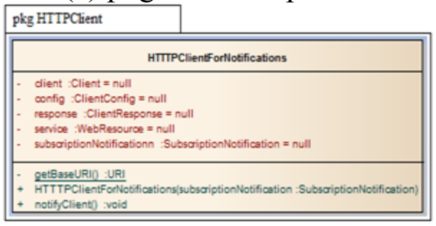

(c) pkg Datastructure.queries

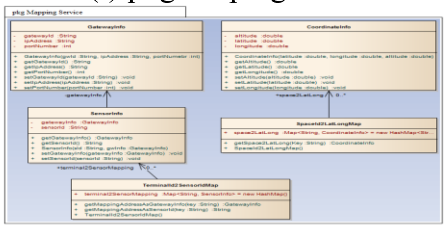

pkg Interconnection/Mapping
InterConnectionMappingModule
- data :String
- hostName :String
- portNumber :int
+ dataHandle() :String
+ getData() :String
+ InterConnectionMappingModule(hostName :String, portNumber :int, data :String)
+ setData(data :String) :void
(d) pkg Interconnection/Mapping

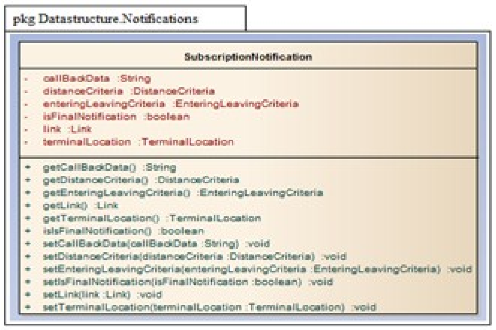

pkg Exceptions
WebApplicationException
InvalidPolicyException
+ InvalidPolicyException()
+ InvalidPolicyException(policyEx :PolicyException)
WebApplicationException
InvalidServiceException
+ InvalidServiceException()
+ InvalidServiceException(serviceEx :ServiceException)
(e) pkg UserExceptions

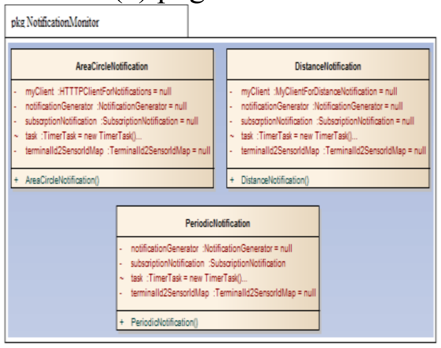

pkg HelpingClass
DistanceCalculation
- deg2rad(deg :double) :double
+ distance(lat1 :double, lon1 :double, lat2 :double, lon2 :double, unit :char) :double
- rad2deg(rad :double) :double
String2StringArr
+ String2StringArr(str :String) :String[]
(f) pkg HelpingClass


(g) pkg InferenceModule

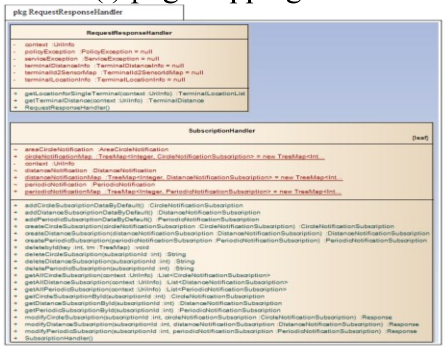

pkg HTTPClient
HTTPClientForNotifications
- client :Client = null
- config :ClientConfig = null
- response :ClientResponse = null
- service :WebResource = null
- subscriptionNotificationn :SubscriptionNotification = null
- getBaseURI() :URI
+ HTTPClientForNotifications(subscriptionNotification :SubscriptionNotification)
+ notifyClient() :void
(h) pkg HTTPClient


(i) pkg MappingService


pkg Datastructure.Notifications
SubscriptionNotification
- callBackData :String
- distanceCriteria :DistanceCriteria
- enteringLeavingCriteria :EnteringLeavingCriteria
- isFinalNotification :boolean
- link :Link
- terminalLocation :TerminalLocation
+ getCallBackData() :String
+ getDistanceCriteria() :DistanceCriteria
+ getEnteringLeavingCriteria() :EnteringLeavingCriteria
+ getLink() :Link
+ getTerminalLocation() :TerminalLocation
+ isIsFinalNotification() :boolean
+ setCallBackData(callBackData :String) :void
+ setDistanceCriteria(distanceCriteria :DistanceCriteria) :void
+ setEnteringLeavingCriteria(enteringLeavingCriteria :EnteringLeavingCriteria) :void
+ setIsFinalNotification(isFinalNotification :boolean) :void
+ setLink(link :Link) :void
+ setTerminalLocation(terminalLocation :TerminalLocation) :void
(j) pkg Datastructure.Notifications



(k) pkg NotificationMonitor


(l) pkg RequestResponseHandler

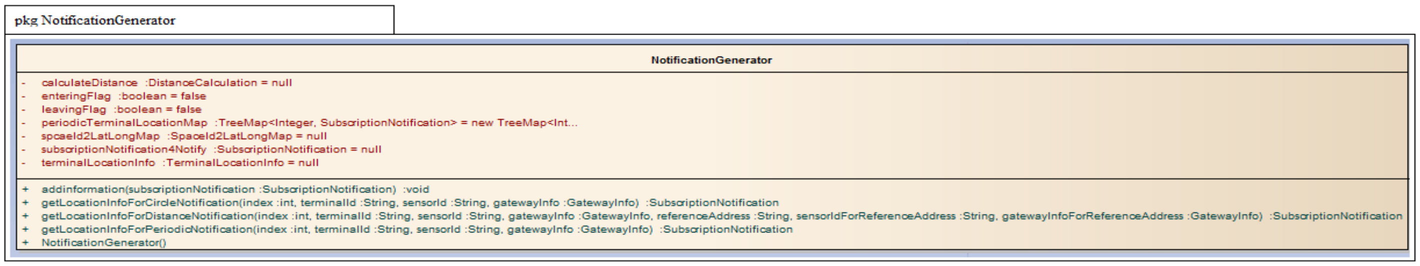

pkg NotificationGenerator
NotificationGenerator
- calculateDistance :DistanceCalculation = null
- enteringFlag :boolean = false
- leavingFlag :boolean = false
- periodicTerminalLocationMap :TreeMap<Integer, SubscriptionNotification> = new TreeMap<Int...
- spcaeId2LatLongMap :SpaceId2LatLongMap = null
- subscriptionNotification4Notify :SubscriptionNotification = null
- terminalLocationInfo :TerminalLocationInfo = null
+ addInformation(subscriptionNotification :SubscriptionNotification) :void
+ getLocationInfoForCircleNotification(index :int, terminalId :String, sensorId :String, gatewayInfo :GatewayInfo) :SubscriptionNotification
+ getLocationInfoForDistanceNotification(index :int, terminalId :String, sensorId :String, gatewayInfo :GatewayInfo, referenceAddress :String, sensorIdForReferenceAddress :String, gatewayInfoForReferenceAddress :GatewayInfo) :SubscriptionNotification
+ getLocationInfoForPeriodicNotification(index :int, terminalId :String, sensorId :String, gatewayInfo :GatewayInfo) :SubscriptionNotification
+ NotificationGenerator()
(m) pkg NotificationGenerator

Figure 4.16: Realization of classes in the packages for REST gateway implementation

‘**NotificationGenerator**’ package, shown in Figure 4.16(m), has only one class called ‘NotificationGenerator’. This class will be triggered by one or more notification monitor class. It also generates the notification information. ‘**HTTPClient**’ package shown in Figure 4.16 (h) contains the classes that are responsible for sending notification towards the application.

4.8.2 Implementation of WSN Gateway

We have chosen Java socket programming APIs [44], a de facto standard for Inter-process Communication (IPC), for the implementation of WSN gateway as the communication between REST and WSN gateway is based on UDP. We have implemented all the components of WSN gateway. The components of WSN gateway are structured into Java classes, which are generated by Enterprise architecture software [64], shows in Figure 4.17. ‘WSNRequestResponseHandler’ class is the core class of WSN gateway responsible for handling request and response messages generated by REST gateway. The raw data are encapsulated in ‘SensorData’ class and then the encapsulated sensor data are stored into the repository. ‘ProprietaryCricketSensorInterface’ class implements WSN communication stack in order to communicate with the individual cricket sensor node. ‘SensorNode’ class is an object representation of a sensor node containing sensed data and the corresponding Sensor-Id.

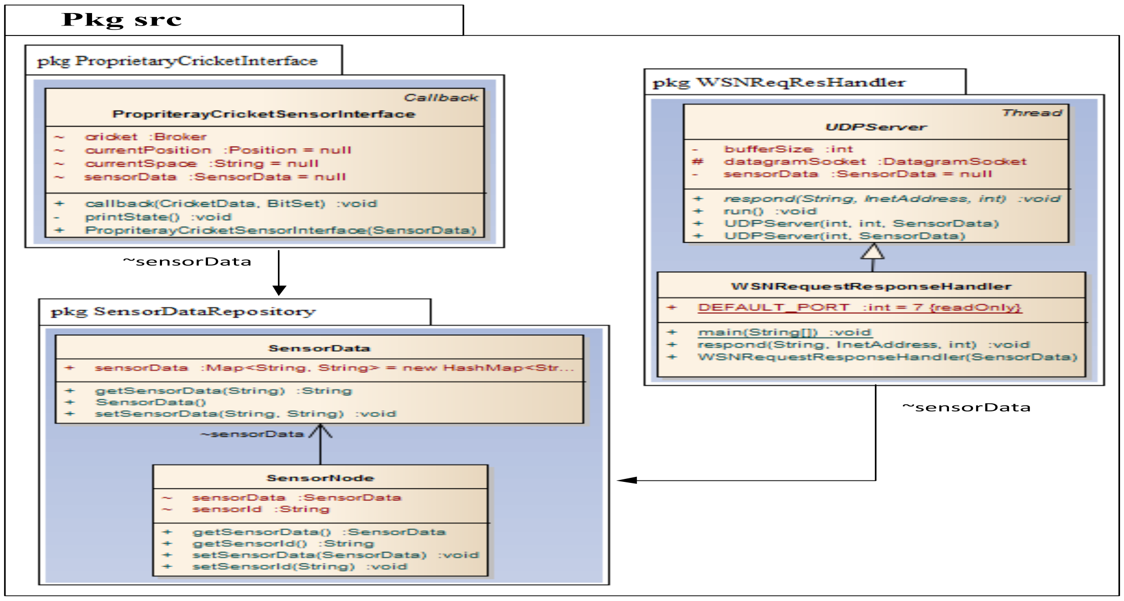

Pkg src
pkg ProprietaryCricketInterface
Callback
ProprieterayCricketSensorInterface
~ cricket :Broker
~ currentPosition :Position = null
~ currentSpace :String = null
~ sensorData :SensorData = null
+ callback(CricketData, BitSet) :void
- printState() :void
+ ProprieterayCricketSensorInterface(SensorData)
~sensorData
pkg SensorDataRepository
SensorData
+ sensorData :Map<String, String> = new HashMap<Str...
+ getSensorData(String) :String
+ SensorData()
+ setSensorData(String, String) :void
~sensorData
SensorNode
~ sensorData :SensorData
~ sensorId :String
+ getSensorData() :SensorData
+ getSensorId() :String
+ setSensorData(SensorData) :void
+ setSensorId(String) :void
pkg WSNReqResHandler
Thread
UDPServer
- bufferSize :int
# datagramSocket :DatagramSocket
- sensorData :SensorData = null
+ respond(String, InetAddress, int) :void
+ run() :void
+ UDPServer(int, int, SensorData)
+ UDPServer(int, SensorData)
WSNRequestResponseHandler
+ DEFAULT_PORT :int = 7 {readOnly}
+ main(String[]) :void
+ respond(String, InetAddress, int) :void
+ WSNRequestResponseHandler(SensorData)
~sensorData

Figure 4.17: Realization of packages and classes for WSN gateway implementation

4.8.3 Implementation of Notification Application

We have developed an application program for the realization of client notifications by the REST gateway. Where, REST gateway sends the notifications (if any). The REST gateway posts the notification information to the application running on the same machine or different machine using client provided callback URI. The application is then responsible for sending notification towards the end-users. We have used Jersey APIs [62] to implement the notification application as well. Figure 4.18 shows the main class for notification application of OMA RESTful location services. All other classes regarding the implementation of OMA RESTful location data structure are already discussed in Section 4.8.1.

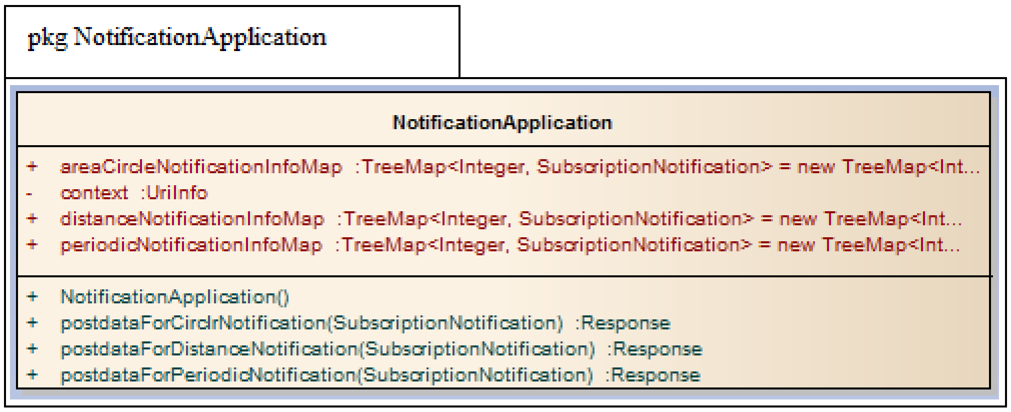

pkg NotificationApplication
NotificationApplication
+ areaCircleNotificationInfoMap :TreeMap<Integer, SubscriptionNotification> = new TreeMap<Int...
- context :UriInfo
+ distanceNotificationInfoMap :TreeMap<Integer, SubscriptionNotification> = new TreeMap<Int...
+ periodicNotificationInfoMap :TreeMap<Integer, SubscriptionNotification> = new TreeMap<Int...
+ NotificationApplication()
+ postdataForCirclrNotification(SubscriptionNotification) :Response
+ postdataForDistanceNotification(SubscriptionNotification) :Response
+ postdataForPeriodicNotification(SubscriptionNotification) :Response

Figure 4.18: Realization of package and class for Notification Application implementation

4.8.4 Implementation Environment

Our implementation environment is involved with sensor hardware and different software tools.

4.8.4.1 Hardware Environment

We used MIT cricket sensor [2] in order to sense location data. Cricket sensor can report location either as a space identifier (Space-Id) or as an assigned coordinate. It can also detect location information either as a Space-Id (i.e. room-1) or as Cartesian coordinate (i.e. 230, 23 and 129) of a space. The Id of a space is the space identifier which is a user defined string such as room-1 or EV.15.163. The coordinates are in the form of (x, y, z) in Cartesian coordinate system. The cricket sensor operates either in beacon or listener mode. Beacons are pre-configured with space identifiers and coordinates. Beacons are actively transmitting data to the listeners and the listeners are passively listening from different beacons at the same time. Listeners are usually attached with the host PC through RS-232 serial interface [65]. A MIT cricket sensor is shown in Figure 4.19.

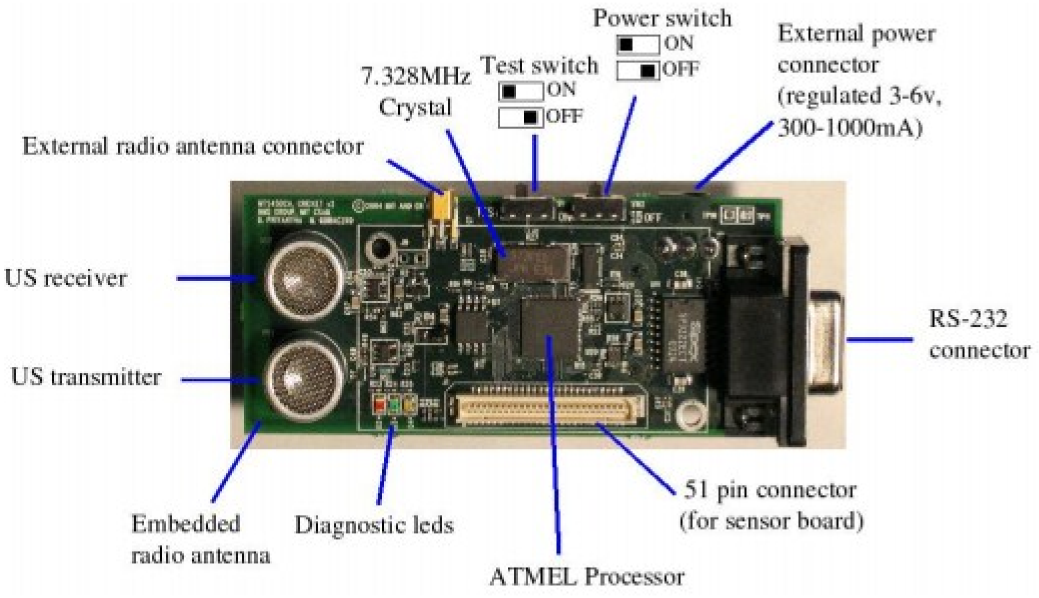


Figure 4.19: MIT cricket sensor

4.8.4.2 Software Environment

We used different types of software tools in order to implement our system, such as Jersey, JAXB and Cricket APIs.

Jersey API: Jersey API is a java based API provides support to create web services according to the REST architectural style defined in JSR-311 [69]. Jersey is an open source, production quality, JAX-RS (JSR-311) reference implementation for building RESTful web services [62]. It also provides an API that enables developers to extend Iersey to satisfy their needs. The REST gateway was implemented using Jersey APIs.

JAXB API: Java architecture for XML binding (JAXB [66]) is a Java based API defined in JSR-31 [81]. The JAXB specification is developed through the Java Community

Process (JCP). The processes of JCP are described in [67]. JAXB API provides a fast and easy way to marshal and un-marshal both XML and JSON payloads. It easily maps XML and JSON documents into Java objects without any requirement for extensive knowledge of XML and JSON programming and vice-versa. This simplifies and speeds up the development of RESTful web services. Figure 4.20 shows the JAXB architecture and JAXB binding process [68].

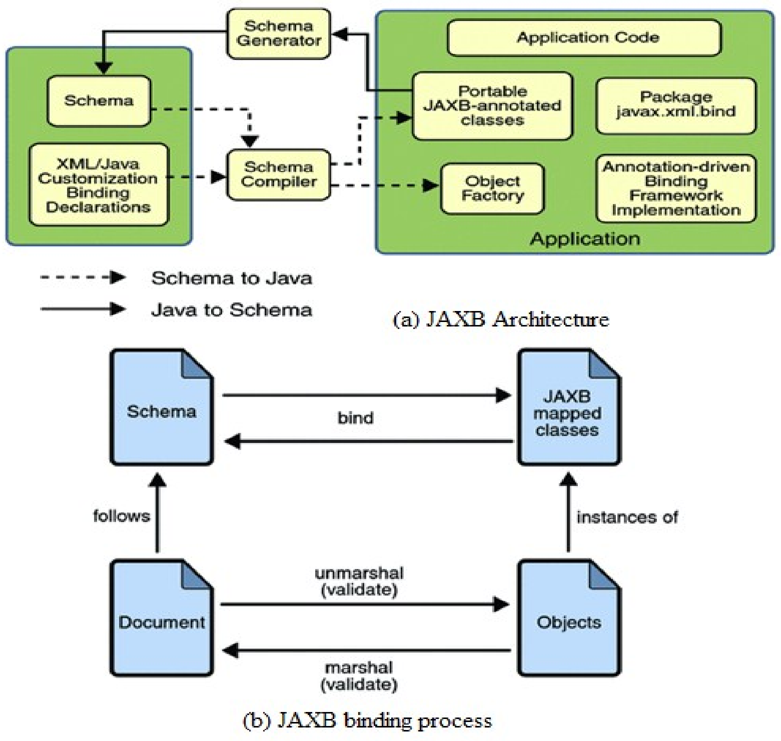


Figure 4.20: JAXB architecture and binding process

Cricket API: MIT Cricket sensor provides Cricket java client API called ClientLib [61] to develop location-based applications. It also delivers software called cricketd and cricketdaemon. These software run on host device with attached listener to retrieve sensor data. MIT Cricket software architecture and ClientLib architecture are shown in Figure 4.21(a) and 4.21(b) respectively.

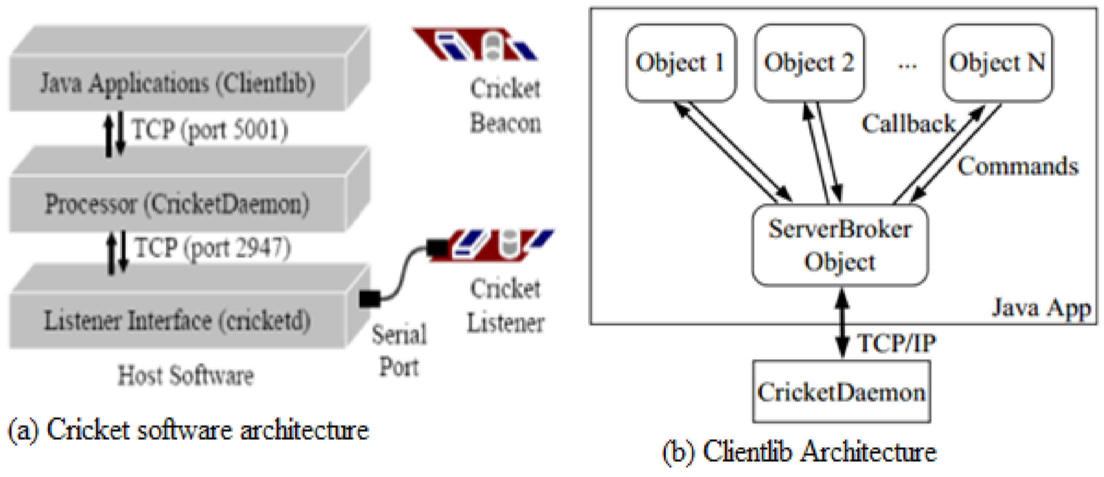


Figure 4.21: Cricket software and ClientLib architecture

4.9 Chapter Summary

This chapter explored the design and implementation of OMA RESTful location services in wireless sensor environments. Several architectural components of proposed REST and WSN gateway were also discussed. Our proposed architecture satisfied all the specific requirements we derived for OMA RESTful location services in wireless sensor environments discussed in Chapter 3. Our implemented architecture provides an abstraction for end-users and application developers to access the location information through the REST gateway. REST gateway hides all the lower layer details from the end-users and application developers, which satisfied our first requirement of higher level of abstraction. In order to support the second and third requirements of heterogeneity of WSNs and reusability of existing WSN gateways we separated WSN gateway component from REST gateway. This flexibility provided us to easily introduce new WSN and existing WSN gateways via the WSN interface (Pi). WSN interface was designed in a unified and simpler way so that it can easily introduce new WSN and plug into the existing WSN gateways easily as well. We reused existing standard APIs such as JAXB, JAX-RS and existing technology such as MIT cricket sensors which satisfied our last specific requirement. The next chapter will focus on the evaluation of our prototype application.

Chapter 5 Prototype Application and Performance Evaluation

5.1 Introduction

The architecture for OMA RESTful location service(s) using WSN provides a framework for the development of location based services and applications in a faster way. The framework provides an abstraction to developers of location based service and application in order to rapid creation of new attractive services through the realization of REST gateway. The REST gateway hides all the physical complexities and lower layer details from the developers. With the help of different operations such as query and subscription procedures one can access WSN information (i.e. location) through the REST gateway. In the following sections, we will explain the proof-of-concept prototype with a prototype application and the performance evaluation of our system.

5.2 Prototype Application

A prototype application of tracking a person/people can be categorized into indoor and outdoor tracking scenario. In outdoor scenario, tracking services work appropriately with GPS since it does not require accurate location information. However, indoor tracking scenario requires more accurate location information. So, our targeted prototype application is tracking a person/people inside a building. This scenario depends on user's context such as location data that can be provided through sensors. Sensors can be deployed inside the building and act as a source of location data. Person/people could use his/her cell phone or a PDA equipped with a client application based on OMA RESTful location services to locate herself/himself. The application can also guide the person step by step to visit the whole building via the user interface with a floor plan.

In the following sections we introduce our prototype setup and a prototyping scenario.

5.3 Prototype Setup

Our prototype setup is shown in the Figure 5.1. It includes a REST gateway, a WSN gateway, a set of MIT cricket sensors [2] acting as a sensor network, a set of end-user terminals, and a REST client that accesses the location services offered by the REST gateway. We implemented all of the location services provided by the OMA.

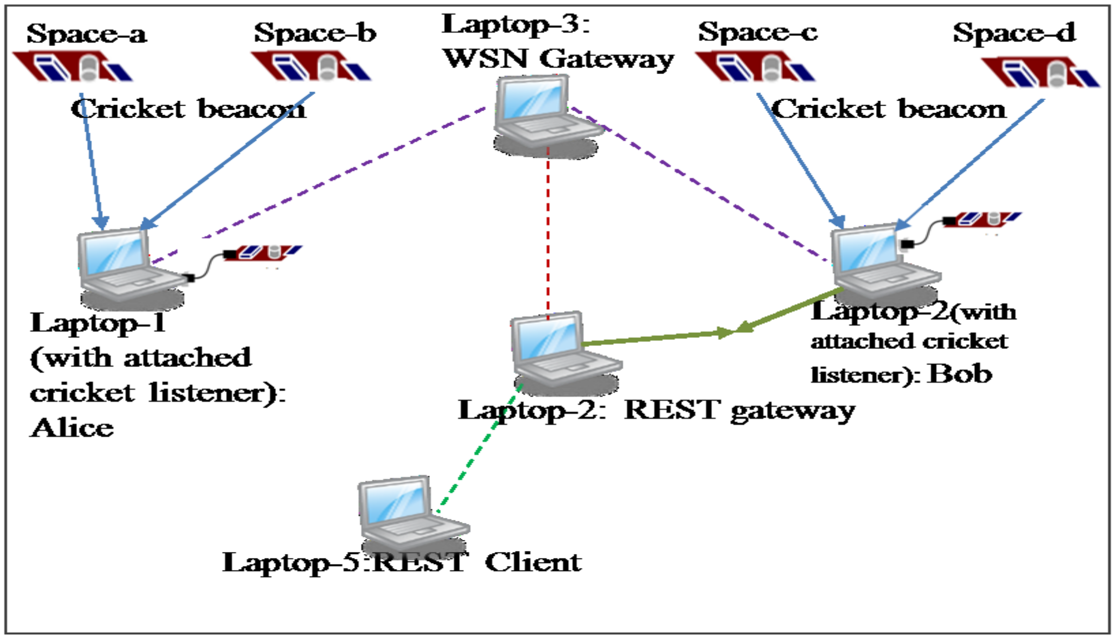


Figure 5.1: Prototype setup

For the WSN environment we divided the working area (i.e. a room) into four spaces (Space-a, Space-b, Space-c, and Space-d). We placed four sensors (beacons) on the ceiling of the room to represent the four spaces (i.e. each beacon is pre-configured with a given Space-Id). We attached a sensor (listener) to each end-user terminal. When moving, an application running on a terminal can automatically detect which space it is on (i.e. the Space-Id of the closest beacon) and send it to the WSN gateway. We assume that the terminals have already discovered the gateway when they enter into the WSN environment.

5.4 Procedures of Periodic Subscription

Figure 5.2 shows the procedures of periodic subscription of our prototyping application. The procedures are as follows:

Step-1: We considered two users Alice and Bob in our scenario. Bob is in Space-a in the beginning. Alice subscribes to the current location of Bob by sending a POST request to the REST gateway.

Step-2: REST gateway creates a new subscription for Alice about the periodic notification of current location of Bob. After that, REST gateway response back to the Alice which indicates a successful creation of periodic subscription.

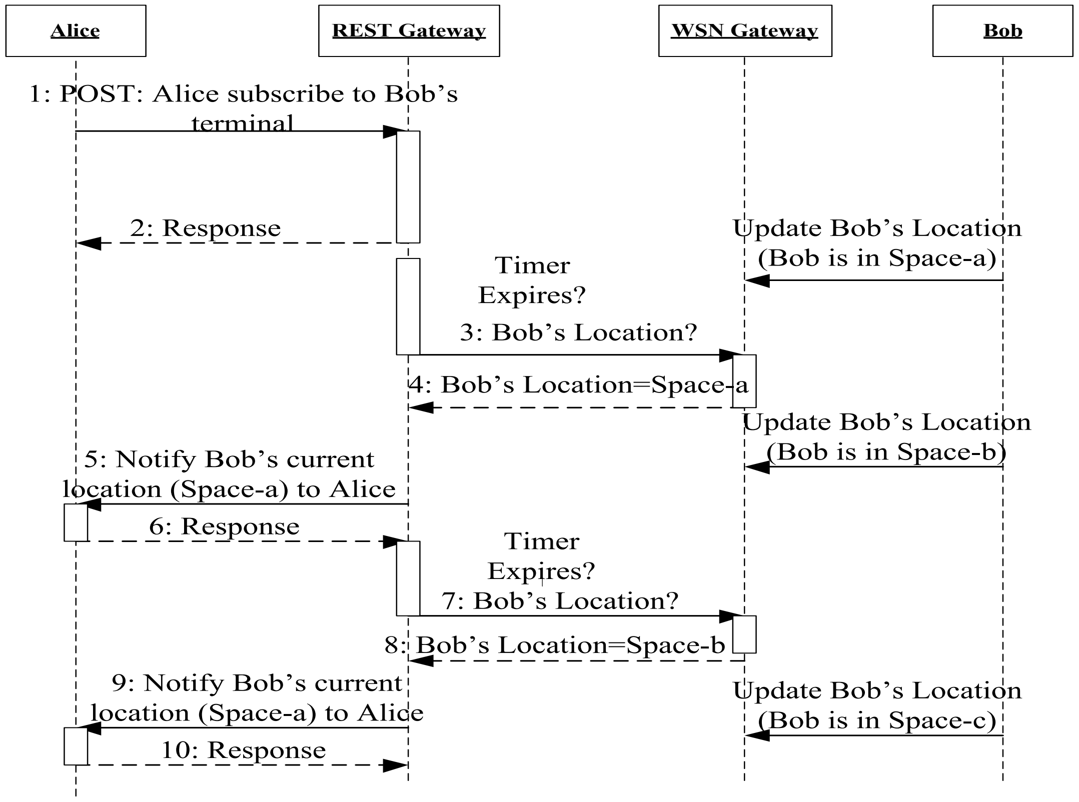
Step-3: When the interval of periodic notification exceeds, REST gateway will communicate with WSN gateway to get the current location of Bob.

Step-4: WSN gateway sends the appropriate information to the REST gateway.

Step-5: When REST gateway receives the appropriate information of Bob current location from WSN gateway; it will send a notification with this information to Alice using POST method.

Step-6: Alice replies back to the REST gateway. In the meantime, Bob already change his position from 'Space-a' to 'Space-b'.

Step-7: Repeat step-3 to step-6 until ending the subscription.



Alice
REST Gateway
WSN Gateway
Bob
1: POST: Alice subscribe to Bob's terminal
2: Response
Update Bob's Location (Bob is in Space-a)
Timer Expires?
3: Bob's Location?
4: Bob's Location=Space-a
Update Bob's Location (Bob is in Space-b)
5: Notify Bob's current location (Space-a) to Alice
6: Response
Timer Expires?
7: Bob's Location?
8: Bob's Location=Space-b
9: Notify Bob's current location (Space-a) to Alice
Update Bob's Location (Bob is in Space-c)
10: Response

Figure 5.2: Procedure of periodic subscription of our prototype application

5.5 Test Environment

Our test an environment was involved with five laptops and a list of software include cricketd [61], cricketdeamon [61], REST gateway, WSN gateway and REST client to run the prototype. Laptop-1 and laptop-2 were used as end-user terminals with attached listener. Cricketd [61] and cricketdeamon [61] software were running on laptop-1 and laptop-2. On the other hand, WSN gateway was running on laptop-3, REST gateway was running on Laptop-4 and REST client was running on laptop-5. The operating system Windows XP was installed on laptop-1 and laptop-2 and Windows 7 was installed on laptop-3 and laptop-5. We used jersey API, an open source reference implementation of JSR-311 [69], to implement the REST interface of the REST gateway. JAXB API [66] was used to implement REST requests and responses' marshaling and un-marshaling. Table 5.1 describes the specification of our test environment.

Laptops	**Software Module**	**Hardware Configuration**		
		CUP Model	**CPU Speed**	**Physical Memory**
Laptop-1	• End user (Alice) • cricketd • cricketdeamon • Windows XP	Genuine Intel CPU T2060	1.60GHz	512MB
Laptop-2	• End user (Bob) • cricketd • cricketdeamon • Windows XP	Genuine Intel CPU T2060	1.60GHz	0.99 GB
Laptop-3	• WSN Gateway • Windows 7	Intel Core i5 2540m	2.60GHz and 2.60GHz	4 GB
Laptop-4	• REST Gateway • Windows XP	Genuine Intel CPU T2060	1.60GHz	0.99 GB
Laptop-5	• REST Client • Windows 7	Intel Pentium Dual CPU T3400	2.16GHz and 2.17 GHz	4 GB

Table 5.1: Specification of our test environment

5.6 Performance Evaluation

In the following section the performance evaluation of our system will be discussed in the context of a set of performance metrics.

5.6.1 Performance Metrics

The intent of performance evaluation was to examine the efficiency and feasibility of location services in wireless sensor environments. The prototype was assessed in terms of time delay, network load, server capacity and bandwidth consumption. From the end-user perspective, a user expects timely response to its request. The delays we measured include query delay, notification delay and subscription delay. Query delay is the time difference between the time an end-user sends a query to the REST gateway and the time he/she receives the response. Subscription delay is the time difference between sending a subscription request to the REST gateway and receiving a 201 Created response. Notification delay is the time difference between the time REST gateways sends a POST request to the end-user with the notification content and the time it receives the response (i.e. 201 Created). Network load indicates the total number of bytes sent and received by end-users for a given request. In order to measure the server performance we need to quantify the number of request a server can handle for a particular time period. Thus, the server capacity indicates how many requests the server can handle to completion with a given amount of time. Bandwidth consumption indicates how much data (average) can be sent over the network in a given amount of time. The delays were measured in milliseconds; the network load was measured in bytes, server capacity was measured in request per second and bandwidth consumption was measured in Kilobyte (KB) per seconds.

5.6.2 Measurements and Analysis

Apache JMeter [8], an open source testing tool from apache and which also acts as a REST client is used to measure the performance data. Each measurement is calculated as an average of hundred experiments. We took the data for 100, 200 and 300 concurrent users (samples) to analyze the performance.

5.6.2.1 Performance Analysis of all OMA Location Services

Table 5.2 shows the average response time for all OMA RESTful location services with respect to the different number of users. The performance result of average response time with respect to Table 5.2 data is shown in Figure 5.3. According to our expectation, the average query delays for accessing query resources (i.e. S1-to-S4) are higher than those of the subscription (i.e. S5-to-S16) and notification (i.e. S17-to-S19) delays. This is because the query requests require a communication with the sensor network to get the information in contrast with the subscription and notification requests. However, the delays should remain acceptable from the end-user point of view. The average delays are 41ms for the query and 17ms for the subscription according to 100 users.

S1 = **get** current location of a terminal

S2 = **get** current location of two terminal

S3 = **get** distance of a terminal from a given location

S4 = **get** distance between two terminals

S5, S6 & S7 = **get** a specific subscription for location, area and distance notification

S8, S9 & S10 = **create** a subscription for location, area, and distance notification, respectively

S11, S12 and S13 = **modify** a specific subscription for location, area, and distance notification

S14, S15 & S16 = **get** all active subscriptions for location, area, and distance notification

S17, S18 & S19 = location, area, and distance **notification callback**

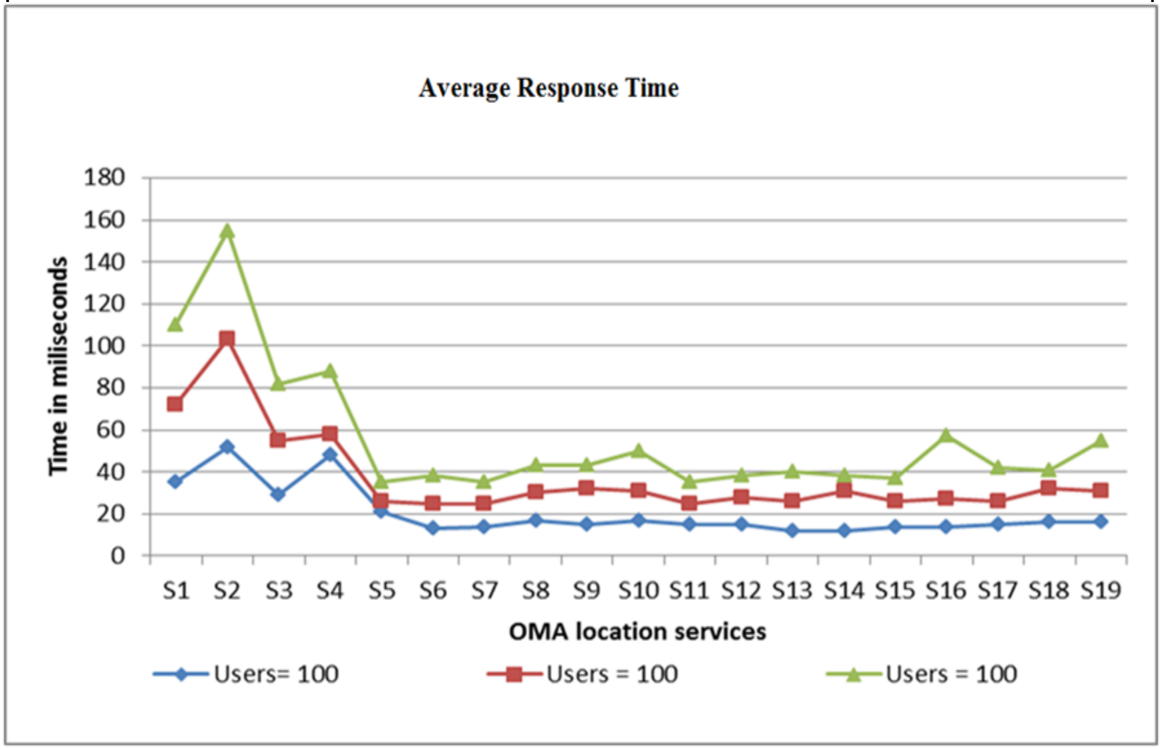


Figure 5.3: Average response time for all OMA RESTful location services

OMA RESTful location services	**Average Response Time (millisecond)**		
	100 Samples	**200 Samples**	**300 Samples**
Get current location of a terminal	35	72	110
Get current location of two terminal	52	103	155
Get distance of a terminal from a given location	29	55	82
Get distance between two terminals	48	58	88
Get a specific subscription for location notification	21	26	35
Get a specific subscription for area notification	13	25	38
Get a specific subscription for distance notification	14	25	35
Create a subscription for location notification	17	30	43
Create a subscription for area notification	15	32	43
Create a subscription for distance notification	17	31	50
Modify a specific subscription for location notification	15	25	35
Modify a specific subscription for area notification	15	28	38
Modify a specific subscription for distance notification	12	26	40
Get all active subscriptions for location	12	31	38
Get all active subscriptions for area notification	14	26	37
Get all active subscriptions for distance notification	14	27	57
Periodic **notification callback**	15	26	42
Area **notification callback**	16	32	41
Distance **notification callback**	16	31	55

Table 5.2: Average response time for OMA RESTful location services

Table 5.3 shows the experimental results of server capacity for all OMA RESTful location services with respect to different number of users. The comparison of server capacity with respect to Table 5.3 data is shown in Figure 5.4. The server capacity for accessing query resources is lower than that of the subscription and notification resources because query requests have higher response time than the subscription and notification requests.

S1 = **get** current location of a terminal S2 = **get** current location of two terminal

S3 = **get** distance of a terminal from a given location

S4 = **get** distance between two terminals

S5, S6 & S7 = **get** a specific subscription for location, area and distance notification

S8, S9 & S10 = **create** a subscription for location, area, and distance notification, respectively

S11, S12 and S13 = **modify** a specific subscription for location, area, and distance notification

S14, S15 & S16 = **get** all active subscriptions for location, area, and distance notification

S17, S18 & S19 = location, area, and distance **notification callback**

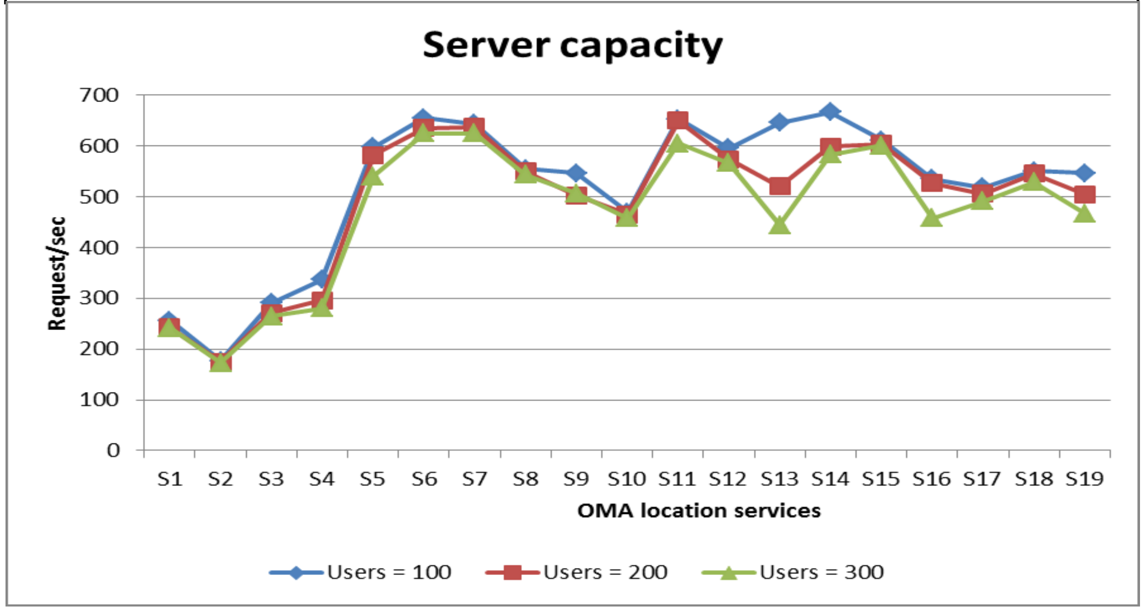


Figure 5.4: Server capacity for all OMA RESTful location services

OMA RESTful location services	**Server Capacity (request/second)**		
	100 Samples	**200 Samples**	**300 Samples**
Get current location of a terminal	255.7545	242.9245	241.3516
Get current location of two terminal	176.6784	174.8252	172.644
Get distance of a terminal from a given location	290.6977	270.8104	264.0583
Get distance between two terminals	336.4066	295.858	280.6361
Get a specific subscription for location notification	597.619	581.3953	539.6588
Get a specific subscription for area notification	655	634.9206	625.1613
Get a specific subscription for distance notification	643.4783	636.9427	625.1613
Create a subscription for location notification	555.2041	549.4505	543.5055
Create a subscription for area notification	546.4481	502.5126	506.7568
Create a subscription for distance notification	469.4836	464.9344	457.707
Modify a specific subscription for location notification	652.4862	649.3506	604.8387
Modify a specific subscription for area notification	595.0495	573.0659	566.0164
Modify a specific subscription for distance notification	645.1613	520.1681	443.787
Get all active subscriptions for location	666.6667	598.4862	582.5243
Get all active subscriptions for area notification	610.8024	604.2296	600.2874
Get all active subscriptions for distance notification	534.7594	526.3158	457.3171
Periodic **notification callback**	518.1347	506.0606	490.9984
Area **notification callback**	550.4836	544.9591	529.1506
Distance **notification callback**	546.4481	505.0505	467.2897

Table 5.3: Server capacity for OMA RESTful location services

Table 5.4 shows the experimental result of average network loads for all OMA RESTful location services with respect to different number of users. The comparison of average network load for different number of users with respect to Table 5.4 data is shown in Figure 5.5. It shows that the average network load for accessing subscription and notification resources is generally higher than the average network load for accessing query resources. This is due to the fact that the responses for the subscription requests and the notification requests have higher payloads.

S1 = **get** current location of a terminal   S2 = **get** current location of two terminal

S3 = **get** distance of a terminal from a given location,

S4 = **get** distance between two terminals

S5, S6 & S7 = **get** a specific subscription for location, area and distance notification

S8, S9 & S10 = **create** a subscription for location, area, and distance notification, respectively

S11, S12 and S13 = **modify** a specific subscription for location, area, and distance notification

S14, S15 & S16 = **get** all active subscriptions for location, area, and distance notification

S17, S18 & S19 = location, area, and distance **notification callback**

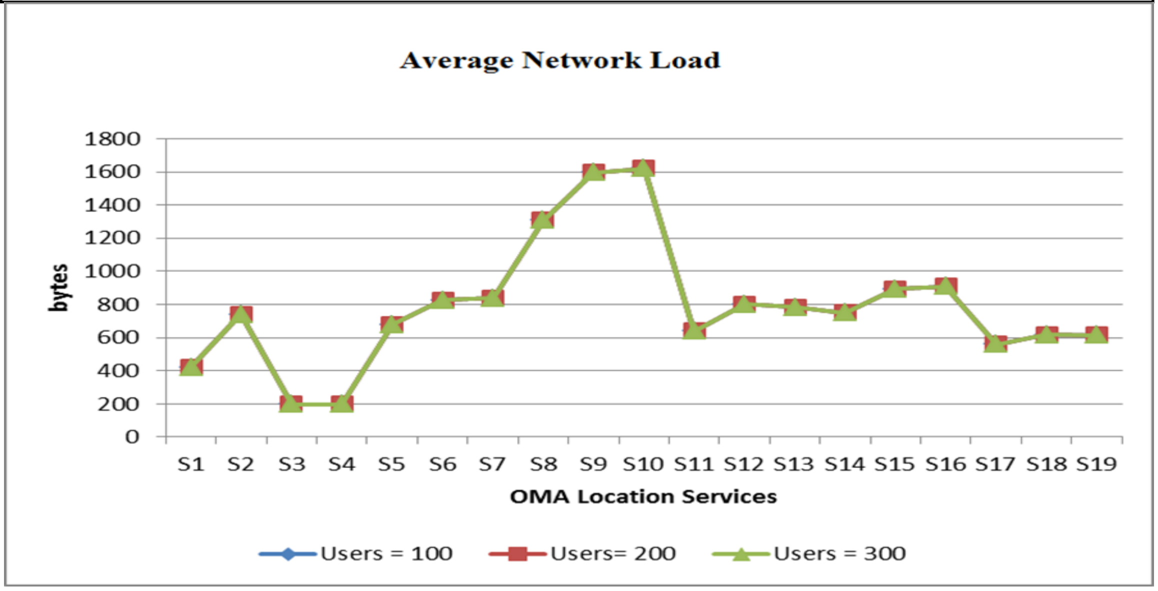


Figure 5.5: Average network load for all OMA RESTful location services

OMA RESTful location services	**Average Network Load (bytes)**		
	100 Samples	**200 Samples**	**300 Samples**
Get current location of a terminal	420	420	420.42
Get current location of two terminal	740	740	739.9733
Get distance of a terminal from a given location	196	196	196.9733
Get distance between two terminals	197	197	197
Get a specific subscription for location notification	679	679	679
Get a specific subscription for area notification	826	826	826
Get a specific subscription for distance notification	838	838	838
Create a subscription for location notification	1307	1307	1307
Create a subscription for area notification	1597	1597	1597
Create a subscription for distance notification	1621	1621	1621
Modify a specific subscription for location notification	639	639	639
Modify a specific subscription for area notification	801	801	801
Modify a specific subscription for distance notification	784	784	784
Get all active subscriptions for location	750	750	750
Get all active subscriptions for area notification	893	893	893
Get all active subscriptions for distance notification	909	909	909
Periodic **notification callback**	559	559	559
Area **notification callback**	616	616	616
Distance **notification callback**	613	613	613

Table 5.4: Average network load for different OMA RESTful location services

Table 5.5 shows the experimental result of bandwidth consumption for all OMA RESTful location services with different number of users. The comparison of bandwidth consumption with respect to Table 5.5 data is shown in Figure 5.6. The bandwidth consumption goes hand in hand with the network load, and it remains same for each service, regardless of the number of users.

S1 = **get** current location of a terminal   S2 = **get** current location of two terminal

S3 = **get** distance of a terminal from a given location,

S4 = **get** distance between two terminals

S5, S6 & S7 = **get** a specific subscription for location, area and distance notification

S8, S9 & S10 = **create** a subscription for location, area, and distance notification, respectively

S11, S12 and S13 = **modify** a specific subscription for location, area, and distance notification

S14, S15 & S16 = **get** all active subscriptions for location, area, and distance notification

S17, S18 & S19 = location, area, and distance **notification callback**

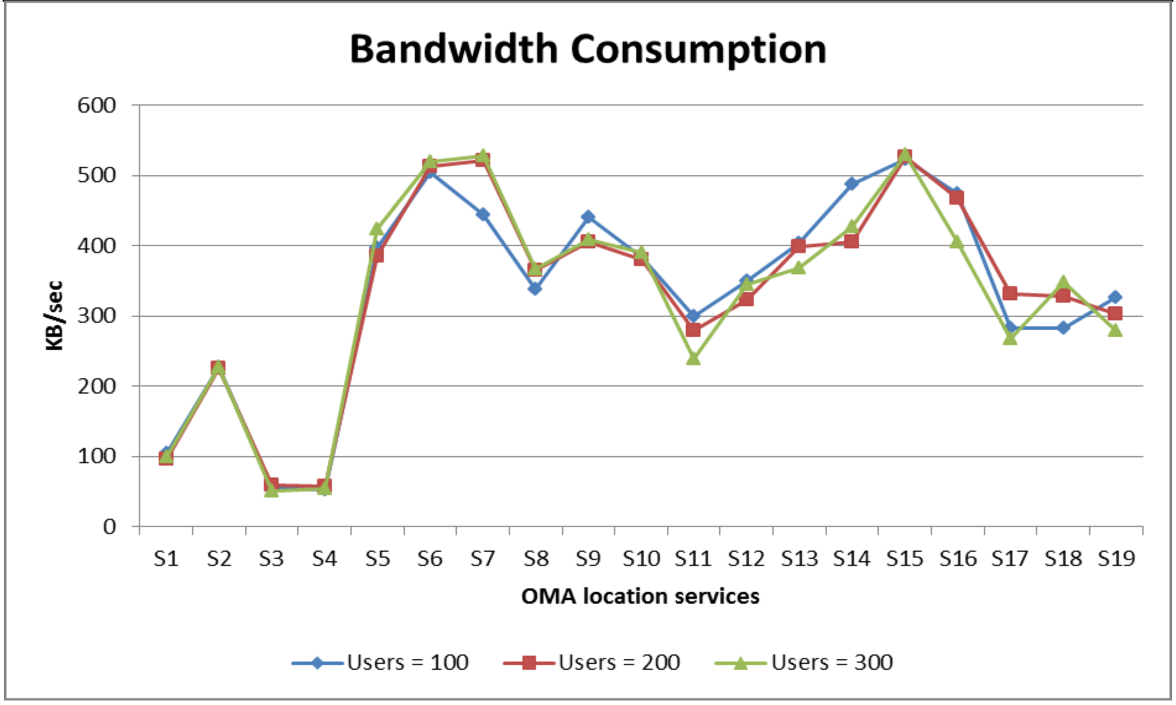


Figure 5.6: Bandwidth consumption for all OMA RESTful location services

OMA RESTful location services	Bandwidth Consumption (KB/second)		
	100 Samples	200 Samples	300 Samples
Get current location of a terminal	104.8993	96.73496	99.09085
Get current location of two terminal	227.6778	226.3385	226.9257
Get distance of a terminal from a given location	55.64135	58.83266	51.75518
Get distance between two terminals	52.48057	56.91799	53.98956
Get a specific subscription for location notification	397.347	385.5151	424.1488
Get a specific subscription for area notification	504.1504	512.1528	520.4133
Get a specific subscription for distance notification	444.7605	521.248	527.9738
Create a subscription for location notification	338.3092	364.3329	367.0217
Create a subscription for area notification	440.7872	405.347	408.7706
Create a subscription for distance notification	384.2063	379.585	390.936
Modify a specific subscription for location notification	298.832	278.832	238.832
Modify a specific subscription for area notification	350.234	323.49	345.32
Modify a specific subscription for distance notification	403.42	398.34	367.98
Get all active subscriptions for location	488.2813	404.653	426.6535
Get all active subscriptions for area notification	522.1978	526.9307	529.5974
Get all active subscriptions for distance notification	474.7034	467.2081	405.9582
Periodic **notification callback**	282.8489	330.8475	268.0352
Area **notification callback**	282.4237	327.827	348.3953
Distance **notification callback**	327.1218	302.3398	279.735

Table 5.5: Bandwidth consumption for OMA RESTful location services

5.6.2.2 Server Capacity Analysis for a Specific OMA RESTful Location Service

Figure 5.7 shows the server capacity versus response time with respect to data of Table 5.6. If we increase the number of users, the average response time will also increases but it decreases the sever capacity. In order to validate this comparison, we collected average response time and server capacity result for a single terminal location of OMA RESTful location services with respect to different number of users shown in Table 5.6. It shows that initially server handles more requests because of a few numbers of users and less response time. When we increased the number of users the server performance degraded gradually. We can see that the average response time rises linearly up to 3000 users. Table 5.6 shows the success rate of 500 and 1000 users is 100% and error rate is 0%. In this case the server can handle average more than 200 requests per second. However, when we increased the number of users, the success rate decreased. For instance, the server can handle 13 requests per second for 3500 concurrent users.

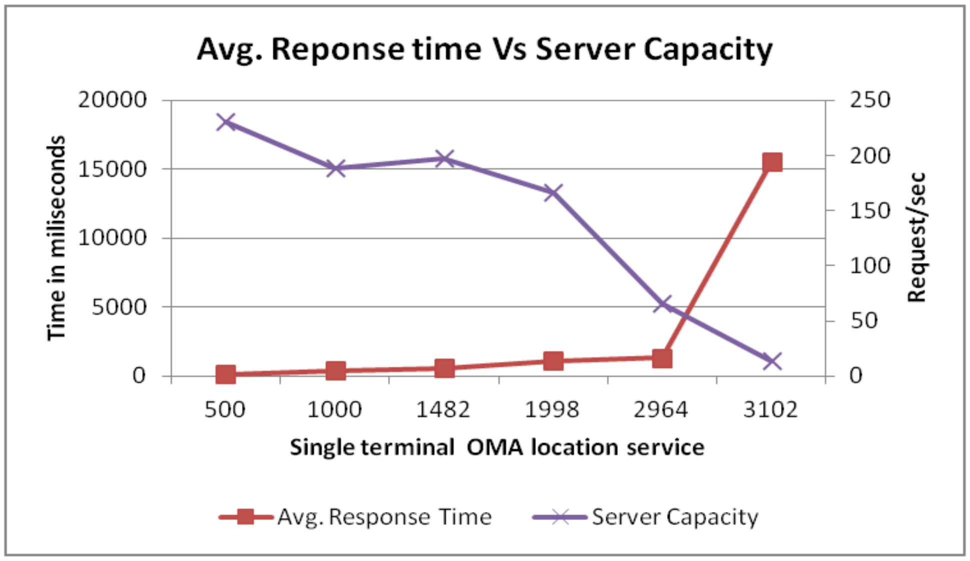


Figure 5.7: Average response time vs. Server capacity comparison

Number of users	Success Rate (%)	Error (%)	Avg. Response Time (ms)	Sever Capacity (req/sec)
500	100	0	120	230.7337
1000	100	0	398	188.253
1500	98.8	1.2	604	197.1269
2000	99.9	.1	1061	166.2368
3000	98.8	1.2	1322	65.5129
3500	90	10	15559	13.24

Table 5.6: Get single terminal location services data for different number of users

5.6.2.3 Performance Comparisons of OMA Query Location Services with XML and JSON Payloads

It is necessary to analysis the performance of our system with XML [70] and JSON [71] payloads as our system support both of them. First, we will focus on the performance analysis of our system according to four performance metrics i.e. average response time, average network load, server capacity and bandwidth consumption for all OMA query location services with XML and JSON payloads. After that we will show this performance analysis of our system for a single terminal location of OMA RESTful location services with XML and JSON payloads for different number of users.

Table 5.7 shows the experimental results of all OMA query location services with XML and JSON payloads according to our four performance metrics. It has been shown that the average response time with XML payloads is always higher than the average response time with JSON payloads and the server capacity with XML payloads is always less than the server capacity with JSON payloads. On the other hand, average network load and bandwidth consumption with JSON payloads are 50% faster than the average network load and bandwidth consumption with XML payloads. Figure 5.8 shows the performance comparison of all OMA query location services with XML and JSON payloads. We did the measurement for each service, when 300 users were accessing the service concurrently.

OMA Services	**Avg. Response Time**		**Avg. Network Load**		**Sever Capacity**		**Bandwidth Consumption**	
	XML	**JSON**	**XML**	**JSON**	**XML**	**JSON**	**XML**	**JSON**
Get current location of a terminal	110	102	420.42	225	241.3516	255.4844	99.09085	50.20409
Get current location of two terminal	155	108	739.97	432	172.644	223.2563	126.9257	70.99874
Get distance of a terminal from a given location	82	75	196.97	83.95667	264.0583	275.5172	51.75518	17.67003
Get distance between two terminals	88	82	197	83.98667	280.6361	295.5519	53.98956	18.25332

Table 5.7: Performance results of query location services with XML and JSON payloads for all OMA Query location services

According to our expectation, the average response time with an XML payload was higher than the average response time with a JSON payload. This is because XML requires more processing time for XML formatting and parsing. The bandwidth consumption and average network load for XML payload are almost two times higher than those for a JSON payload. This is due mainly to the fact that XML is more verbose

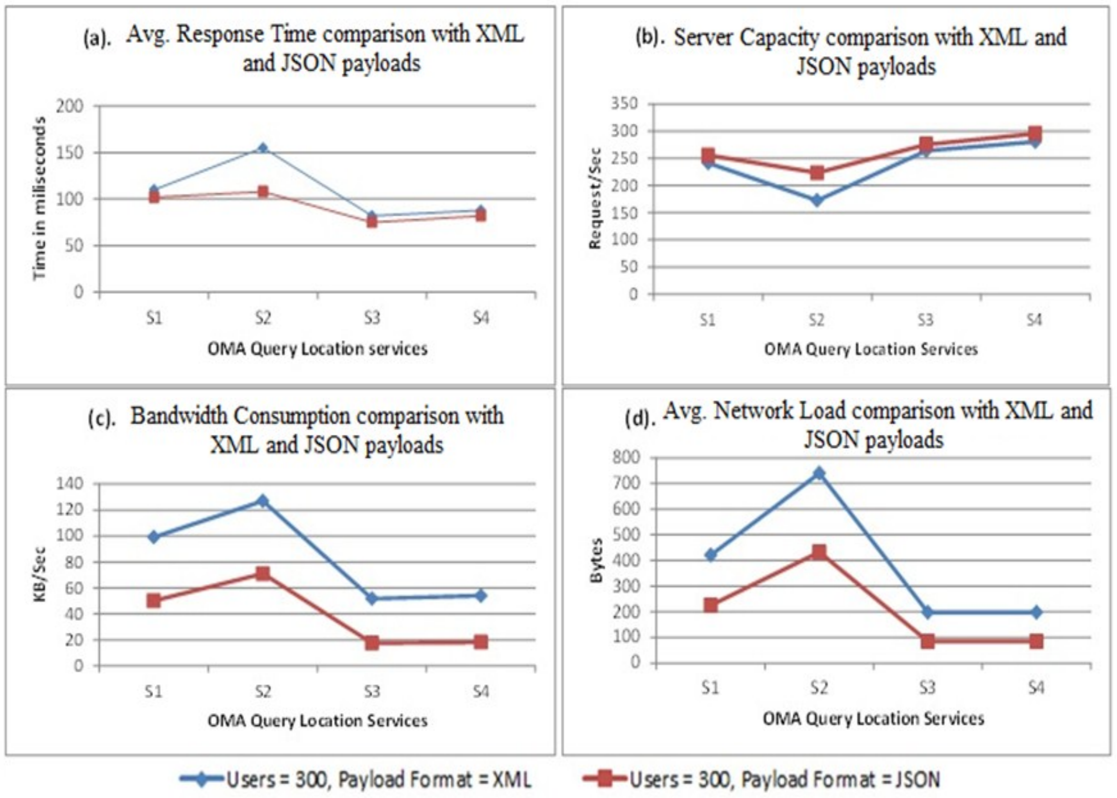


Figure 5.8: Comparison of (a) average response time, (b) server capacity, (c) bandwidth consumption, and (d) average network load of OMA query location services with XML and JSON payload formats

than JSON (e.g. XML needs more fields/tags to represent the location information). As a result, the server capacity with a JSON payload is higher than with an XML payload.

5.6.2.4 Performance Comparisons of a Specific OMA Service with XML and JSON Payloads

Table 5.8 shows the experimental result of our system for a single terminal location of OMA RESTful location services with XML and JSON payloads with different number of users. It has been shown that if we increase the number of users, the average response time will also increase. However, the average response time, average network load and bandwidth consumption with JSON payloads is always less than the average response time, average network load and bandwidth consumption with XML payloads.

The comparison of the performance analysis of single terminal location of OMA RESTful location services with XML and JSON payloads with respect to different number of users is shown in Figure 5.9. Figure 5.9 (a) shows the behavior of average response time for different number of users with XML and JSON payloads. We can conclude that if we increase the number of users, the average response time with JSON payload is still less than the average response time with XML payloads. Moreover, the server capacity with JSON payloads is higher than the server capacity with XML payloads shown in Figure 5.9 (c). The bandwidth consumption and average network load with JSON payloads are also two times less than the bandwidth consumption and average network load with XML payloads shown in Figure 5.9 (b) and Figure 5.9 (d).

Average Response Time			
Payloads	**Number of Samples**		
	100	**200**	**300**
XML	35	72	110
JSON	30	67	102
Average Network Load			
XML	420	420	420.42
JSON	225	225	225
Server capacity			
XML	255.7545	242.9245	241.3516
JSON	280.6436	263.1579	255.4844
Bandwidth consumption			
XML	104.8993	96.73496	99.09085
JSON	46.06427	57.82278	50.20409

Table 5.8: Experimental result for a single terminal location with XML and JSON payloads

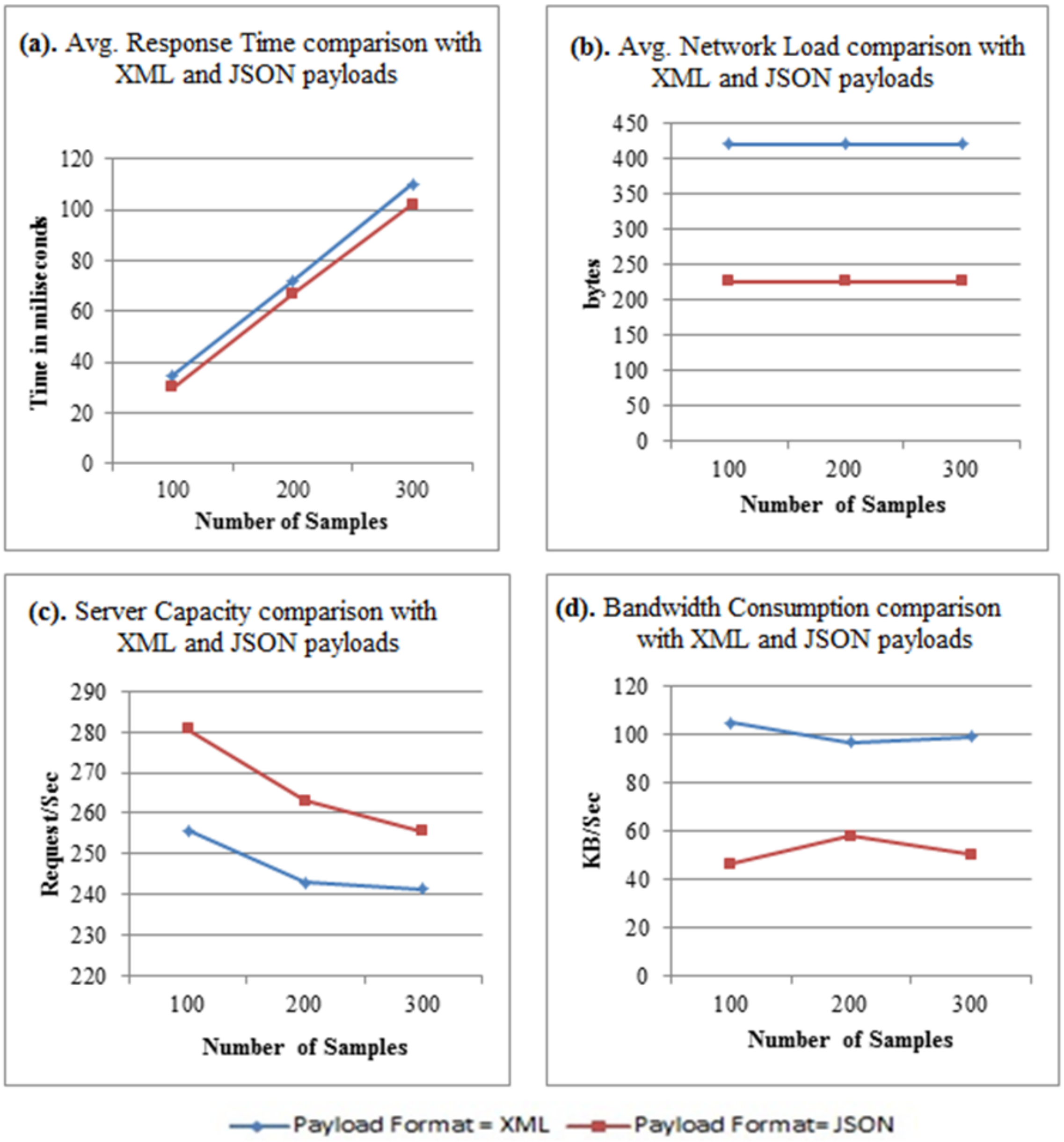


Figure 5.9: Comparison of (a) average response time, (b) average network load, (c) server capacity, and (d) bandwidth consumption get single terminal location service with XML and JSON payload formats for different number of users

Finally we can conclude that our system provides better performance with JSON payloads.

5.7 Challenges Faced and Lessons Learned

The first lesson we learned is that it is easy to develop a client application to access OMA query resources. The application can be either a webpage or a standalone application that sends HTTP requests (e.g. using Jersey client side APIs). It only took us a couple of hours to develop a client application since we already have some experience with HTTP, Java script [72] and Jersey Client APIs [73]. A second lesson is related to the notification resource. These notifications are sent to the client using HTTP POST requests, which requires the client device to run an HTTP server. This puts a stringent requirement on the device and excludes resource-constrained devices (e.g. cell phones). We therefore believe that it would be worthwhile to extend OMA location services to support asynchronous notification, which will free devices from running an HTTP server. Asynchronous communication can be implemented using Atmosphere Jersey [74].

A third lesson is that it is possible to map location information for each of the OMA services to the sensor-based location information. However, using geographic coordinates (i.e. geographic latitude, longitude and altitude) to describe terminal location does not match the fine-grained location accuracy provided by WSNs (e.g. it is not possible to know that a terminal is in room ‘a’ on the second floor of building ‘x’). Therefore, we believe that it would be valuable to extend OMA location services to provide more accurate location information. A fourth lesson is related to the implementation technologies/tools for both the server and client sides. For the implementation and deployment of the REST gateway, we needed an application server that supported RESTful web services. Moreover, it should be very simple and easy to use. We reviewed various existing development tools (e.g. Jersey [62], RESTlet [75], JBoss RESTEasy

[76], Apache CFX [77], etc.) and application servers, and discovered that Jersey with Glassfish Server 3.x [78] meets all our requirements. Jersey is an add-on of Glassfish server. Furthermore, since we are familiar with NetBeans [79]; we chose NetBeans as the development environment. Jersey provides a rich set of documentation for NetBeans. On the other hand, RESTlet is very convoluted and has less documentation, JBoss RESTEasy has no standardized client APIs, and Apache CFX is more suitable when the co-existence of SOAP and REST is required.

A fifth lesson concerns the processing of the requests' and the responses' payloads. We found that JAXB provides a fast and easy way to marshal and un-marshal both XML and JSON payloads. It easily maps XML/JSON documents into Java objects and vice-versa, without any requirement for extensive knowledge of XML and JSON programming. This simplifies and speeds up the development of RESTful web services. The last lesson is related to the testing tools for RESTful web services. NetBeans' 7.0 IDE provides a clear and easy-to-use testing environment. It automatically generates testing clients for the implemented services. We have used it to test both query and subscription resources. Apache JMeter is a free testing tool that can be used to test and asses the performance of a RESTful system, using various metrics and parameters (e.g. Measure the average response time of the system, and simulate a heavy load on a server).

5.8 Chapter Summary

In this chapter, we demonstrated the proof-of-concept of our architecture for OMA RESTful location services in wireless sensor environment by implementing a prototype application. Our proposed architecture provides an advantage to the location based service developers from whom we do not require any prior knowledge of WSN domain to implement new location based services and enables applications that require greater accuracy. The performance analysis was also conducted to show the efficiency of our proposed architecture. In the next and last chapter of this thesis, the conclusions with some future works will be presented.

Chapter 6 Conclusion and Future Work

In this chapter, we summarize the contributions of the thesis and discuss potential future work.

6.1 Summary of Contribution

Location services provide mobile device's location or position with other information to the end-users. They are gaining popularity in our everyday life. For instance, tracking people/person inside large building is one of the examples of location based services. However, accuracy and lack of standard REST interfaces are key issues in many of these applications. We addressed these issues in this thesis. OMA RESTful location services are standard RESTful web services for terminal location. They are location technology independent and enable applications portability and interoperability. In this thesis we provide a novel architecture for provisioning OMA RESTful location services using wireless sensor networks. It enables application developers to offer a wide range of new personalized and user centric applications with greater accuracy accessible via any mobile devices (e.g. PDAs, Laptops and smart phones etc.).

We designed and implemented all the architectural components of the REST and WSN gateways, and the OMA services. The REST gateway offers location services towards the end-users via a standard REST interface. The WSN gateway is responsible for collecting WSN raw data from individual sensors and providing this information to the REST gateway. We designed an interface called UDP-based interface between the REST gateway and the WSN gateway, and defined mappings. We chose UDP to implement this interface as it satisfies most of our requirements.

We developed a proof-of-concept prototype for people/person tracking application. We evaluated the performance of our prototype with respect to four different performance metrics, end-to-end delay, server capacity, network load, and bandwidth consumptions. With this evaluation, we concluded that the proposed architecture is efficient with respect to the four metrics. However, further and more thorough validation needs to be conducted.

6.2 Future Works

As future work we can consider the following directions:

- The aspect which was not included in our work is related to the notification resource. The notifications are sent to the client using HTTP POST requests, which requires the client device to run an HTTP server. This puts a stringent requirement on the device and excludes resource-constrained devices (e.g. cell phones). We therefore believe that it would be worthwhile to extend OMA location services to support asynchronous notification, which will free devices from running an HTTP server. Asynchronous communication can be implemented using Atmosphere Jersey [74].
- Another potential future work would be related to the fine-grained location accuracy. It has been shown that it is possible to map location information for each of the OMA services to the sensor-based location information. However, using geographic coordinates (i.e. geographic latitude, longitude and altitude) to describe terminal location does not match with the fine-grained location accuracy provided by WSNs (e.g. it is not possible to know that a terminal is in room ‘a’ on the second floor of building ‘x’). Therefore,

we believe that it would be valuable to extend OMA location services to provide more accurate location information.

References

[1] Nuru Yakub Othman, “Web Services as application enabler for Sink-less Wireless Sensor Networks”, Master’s Thesis, Electrical and Computer Engineering Dept., Concordia University, Montreal, February-2007.

[2] “MIT Cricket Sensor”, available online at: http://cricket.csail.mit.edu/ [February 17th 2012].

[3] R. T. Fielding, “Architectural styles and the design of network-based software architectures”, PhD thesis, 2000.

[4] J. Schiller, A. Voisard, “*Location-based Services*”, 1st edition, The Morgan Kaufmann Series in Data Management Systems, April 2004.

[5] Open Mobile Alliance: “RESTful bindings for Parlay X Web Services - Terminal Location”, Candidate Version-1.0, November 23rd 2010, available online at: http://www.openmobilealliance.org/Technical/release_program/parlayREST_v1_0.aspx [March 30th 2012].

[6] P. Enge, P. Misra, "Special Issue on GPS: The Global Positioning System", *Proceedings of the IEEE*, Vol. 87, No. 1, pp.3-15, January 1999.

[7] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, “A Survey on Sensor Networks”, *IEEE Communications Magazine*, Vol. 40, No. 8, pp 102-114, August 2002.

[8] “Apache JMeter”, avilable online at: http://jmeter.apache.org/ [February 17th 2012].

[9] C. Intanagonwiwat, R. Govindan, D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks", *MobiCom '00 Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 56–67, ACM New York, NY, USA, 2000.

[10] K. Sohraby, D. Minoli, T. Znati, "*Wireless Sensors Networks: Technology, Protocols and applications*", Wiley-InterScience, pp 1-31, 75-80, March 2007.

[11] "University of California-Berkeley Motes", available online at: http://www.xbow.com [January 11th 2012].

[12] "Tmoke-Sky sensors", available online at: http://www.snm.ethz.ch/Projects/TmoteSky, [January 11th 2012].

[13] "ScatterWeb sensor platform", available online at: http://cst.mi.fu-berlin.de/projects/ScatterWeb/ [January 11th 2012].

[14] "BTnodes sensor platform", available online at: http://www.btnode.ethz.ch/ [January 11th 2012].

[15] "TinyOS - sensor network Operating System", available online at: http://www.tinyos.net/ [January 11th 2012].

[16] "TinyDB - sensor network Database System", available online at: http://telegraph.cs.berkeley.edu/tinydb/ [January 11th 2012].

[17] T. Luckenbach, P. Gober, S. Arbanowsk, "TinyREST - a protocol for inte-grating sensor network into the internet", *in proc. REALWSN*, June 2005.

[18] "nesC", available online at: http://nescc.sourceforge.net/ [January 11th 2012].

[19] M. A. Islam, F. Belqasmi, R. H. Glitho, F. Khendek, "Implementation of OMA RESTful location services in wireless sensor environments", *Accepted in IEEE Symposium on Computers and Communication (ISCC'12)*, Cappadocia, Turkey, July-2012.

[20] A. Milenkovic, C. Otto, E. Jovanov, "Wireless sensor networks for personal health monitoring: Issues and an implementation", *Computer Communications (Special issue: Wireless Sensor Networks: Performance, Reliability, Security, and Beyond,* Vol. 29, pp. 2521–2533, 2006.

[21] T. Gao, T. Massey, L. Selavo, M. Welsh, M. Sarrafzadeh, "Participatory user centered design techniques for a large scale ad-hoc health information system", *in HealthNet '07: Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and networking support for healthcare and assisted living environments*, (NY, USA), pp. 43–48, ACM, 2007.

[22] A. V. Halteren, R. Bults, K. Wac, D. Konstantas, I. Widya, N. Dokovsky, G. Koprinkov, V. Jones, R. Herzog, "Mobile patient monitoring: The mobihealth system", *The Journal on Information Technology in Healthcare*, Vol. 2, No. 5, pp. 365–373, 2004.

[23] K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, S. Moulton, "Sensor networks for emergency response: Challenges and opportunities", *IEEE Pervasive Computing*, vol. 3, No. 4, pp. 16–23, Oct.-Dec. 2004.

[24] K. JeongGil, L. Chenyang, M. B. Srivastava, J. A. Stankovic, A. Terzis, M. Welsh, "Wireless Sensor Networks for Healthcare," *Proceedings of the IEEE*, Vol.98, No.11, pp.1947-1960, November 2010.

[25] A. Manjeshwar, D. P. Agrawal, “Teen: A routing protocol for enhanced efficiency in wireless sensor networks”, *Parallel and Distributed Processing Symposium, International*, Vol. 3, April 2001.

[26] D. Ganesan, R. Govindan, S. Shenker, D. Estrin, “Highly-resilient, energy efficient multipath routing in wireless sensor networks”, *CM SIGMOBILE Mobile Computing and Communications Review*, Vol. 5, No. 4, pp. 11–25, October 2001.

[27] K. Akkaya, M. Younis, “A survey on routing protocols for wireless sensor networks”, *Elsevier’s Ad-Hoc Network*, Vol. 3, No. 3, pp. 325–349, May 2005.

[28] A. Perrig, R. Szewczyk, J. D. Tygar, V. WEN, D. E. Culler, “Spins: security protocols for sensor networks”, *Wireless Networks Journal*, Vol. 8, No. 5, pp. 521–534, September 2002.

[29] C. Karlof, D. Wagner, “Secure routing in wireless sensor networks: Attacks and countermeasures,” *Elsevier’s Ad-Hoc Networks Journal*, Vol. 1, No. 2-3, pp. 293-315, 2003.

[30] F. Belqasmi, R. Glitho, C. Fu, "RESTful web services for service provisioning in next-generation networks: a survey", *IEEE Communications Magazine,* Vol.49, No.12, pp.66-73, December 2011.

[31] L. Richardson, S. Ruby, et al. “*Restful Web Services*”, 1st edition, O’Reilly Media, May 2007.

[32] L. Wirola, I. Halivaara, J. Syrjärinne, "Requirements for the Next Generation Standardized Location Technology Protocol for Location-Based Services", *Journal of Global Positioning Systems*, Vol. 7, No. 2 : 91-103, Nokia Inc., Finland, 2008.

[33] B. Burke, "*RESTful Java with JAX-RS*", 1st edition, O'Reilly Media, November 2009.

[34] Nadia Mohedano Troyano, "The design of a RESTful web-service", Master's Thesis, School of Electrical Engineering, KTH, Stockholm, Sweden, June 2010.

[35] R. Fielding et al., "Hypertext Transfer Protocol – HTTP/1.1", IETF RFC 2616, June 1999.

[36] Open Mobile Alliance: "RESTful bindings for Parlay X Web Services – Common", Candidate Version 1.0, November 23rd 2010, available online at: http://www.openmobilealliance.org/Technical/release_program/parlayREST_v1_0.aspx [March 30th 2012].

[37] C. Pautasso, O. Zimmermann, F. Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision", *In Proceedings of the 17th International World Wide Web Conference*, pages 805–814, Beijing, China, April 2008, ACM Press.

[38] "GSMA OneAPI", available online at: http://www.gsmworld.com/oneapi/index.html [February 26th 2012].

[39] Available online at: https://gsma.securespsite.com/access/Access%20API%20Wiki/Location%20RESTful%20API.aspx [February 26th 2012].

[40] AEPONA: “OneAPI reference implementation specification”, available online at: http://oneapi.aepona.com/ [February 26th 2012].

[41] Locatrix; available online at: http://locatrix.com/ [February 26th 2012].

[42] “Open Geospatial Consortium, Inc.”, available online at: http://www.opengeospatial.org/ [February 26th 2012].

[43] OpenGIS Location Services (OpenLS): “Core services, part-2 gateway service”, version 1.2, September 2008, available online at: http://www.opengeospatial.org/standards/ols [March 30th 2012].

[44] “Java Socket Programming’, available online at: http://docs.oracle.com/javase/1.4.2/docs/api/java/net/Socket.html [March 18th 2012].

[45] “Resource Oriented Architecture”, available online at: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#resource_oriented_model [March 18th 2012].

[46] M. J. Callaghan, P. Gormley, M. Mcbride, J. Harkin, T. M. Mcginnity, “Internal Location Based Services using Wireless Sensor Networks and RFID Technology”, *International Journal of Computer Science and Network Security,* Vol. 6 No.4, April 2006.

[47] S. Ahmad, R. Eskicioglu, P. Graham, "Design and Implementation of a Sensor Network Based Location Determination Service for use in Home Networks", *Mobile Adhoc and Sensor Systems (MASS), IEEE International Conference on* , Vol., No., pp.622-626, October 2006.

[48] W. Jiang, Y. Chen, Y. Shi, Y. Sun, "The Design and Implementation of the Cicada Wireless Sensor Network Indoor Localization System", *Artificial Reality and Telexistence—Workshops, ICAT '06, 16th International Conference on*, Vol., no., pp.536-541, November 29 2006-December 1 2006.

[49] "Millennial Net Wireless Sensor", available online at; http://www.millennial.net/ [January 27th 2012].

[50] "Mote-Track Sensor Network Platform", available online at: http://www.eecs.harvard.edu/~konrad/projects/motetrack/ [January 27th 2012].

[51] "OSGi: an Open source middleware", available online at: http://www.osgi.org/Main/HomePage [February 26th 2012].

[52] H. Pourreza, P. Graham, "On the fly service composition for local interaction environments", *Pervasive Computing and Communications Workshops, Fourth Annual IEEE International Conference on*, Vol., No., pp.6 pp.-399, 13-17 March 2006.

[53] Crossbow Technology Inc., "Mica2/Mic2Dot: Products and Specifications", available online at: http://www.xbow.com/ [February 26th 2012].

[54] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, D. Culler, "The Emergence of Networking Abstractions and Techniques in TinyOS", *In Proc. of the First Symposium on Networked Systems Design and Implementation*, pages 1–14, March 2004.

[55] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, D. Culler, "nesC: A Holistic Approach to Networked Embedded Systems", *In Proceedings of Programming Language Design and Implementation (PLDI)*, pages 1–11, June 2003.

[56] K. Lorincz, M. Welsh, "MoteTrack: A Robust, Decentralized Approach to RF-Based Location Tracking", *In Proc. of the Int'l Workshop on Location and Context-Awareness*, pages 63–82, May 2005.

[57] "Open Mobile Alliance", available online at: http://www.openmobilealliance.org/ [February 26th 2012].

[58] "GSM Association", available online at: http://www.gsm.org/ [February 26th 2012].

[59] M. A. Islam, F. Belqasmi, R. H. Glitho, F. Khendek, "The Design and Implementation of OMA RESTful location services in wireless sensor environments", *submitted to IEEE Communication Magazine*, 2012.

[60] Available online at: http://www.movable-type.co.uk/scripts/latlong.html [February 26th 2012].

[61] "Cricket v2 User Manual-Cricket Project", *MIT Computer Science and Artificial Intelligence Lab*, Cambridge, MA 02139; available online at: http://cricket.csail.mit.edu/ [January 25th 2012].

[62] "Jersey API", available online at: http://jersey.java.net/ [March 2nd 2012].

[63] "Altova UModel (Trial version) - 2012", available online at: http://www.altova.com/umodel.html [March 2nd 2012].

[64] "Enterprise Architecture software (Trial version)", available online at: http://www.sparxsystems.com/products/ea/index.html [March 2nd 2012].

[65] Arif Kadiwal, "Presence-based integration of Wireless Sensor Network and IP Multimedia Subsystem: architecture implementation and case studies", Master's Thesis, Electrical and Computer Engineering Dept., Concordia University, Montreal, November-2008.

[66] "JAXB: Java architecture for XML binding", available online at: http://jaxb.java.net/ [March 2nd 2012].

[67] "Java Commuity Process-Community development of Java specification technology", available online at: http://jcp.org/en/home/index [March 2nd 2012].

[68] "JAXB Architecture", available online at: http://docs.oracle.com/javaee/5/tutorial/doc/bnazg.html [March 2nd 2012].

[69] "JSRs Java Specification Request: JSR-311", available online at: http://jcp.org/en/jsr/detail?id=311 [March 10th 2012].

[70] "Extensible Markup Language (XML)", available online at: http://www.xml.com/ [March 10th 2012].

[71] "JavaScript Object Notation (JSON)", available online at: http://www.json.org/ [March 10th 2012].

[72] "Java Script tuitorial", available online at: http://www.w3schools.com/js/ [March 10th 2012].

[73] "Jsersey Client APIs", available online at: http://jersey.java.net/nonav/documentation/latest/client-api.html [March 10th 2012].

[74] "Atmosphere Jersey", available online at: http://atmosphere.java.net/ [March 10th 2012].

[75] "RESTlet", available online at: http://www.restlet.org/ [March 10th 2012].

[76] "JBoss REST EASY", available online at: http://www.jboss.org/resteasy [March 10th 2012].

[77] "Apache CXF", available online at: http://cxf.apache.org/ [March 10th 2012].

[78] "Glassfish Server 3.x", available online at: http://glassfish.java.net/ [March 10th 2012].

[79] "Netbean IDE v7.0", available online at: http://netbeans.org/ [March 10th 2012].

[80] "Web Application Description Language", available online at: http://wadl.java.net/ [March 18th 2012].

[81] "JSRs Java Specification Request: JSR-31", available online at: http://jcp.org/en/jsr/detail?id=031 [March 10th 2012].