Major Project –II Report
On

"Implementing home gateway using the CoAP protocol"

Submitted in Partial Fulfilment of the Requirement
For the award of Degree of

MASTER OF TECHNOLOGY

(Computer Science & Engineering)
Delhi Technological University, Delhi

SUBMITTED BY :

Ravish Malhotra

University Roll No. 2K12/CSE/16

UNDER THE GUIDANCE OF:

Dr. S.K Saxena
Department of Computer Engineering
Delhi Technological University

DEPARTMENT OF COMPUTER ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
BAWANA ROAD , DELHI -110042

2012-2014

ABSTRACT

This project aims at creating a home gateway / resource directory using the Constrained Application Protocol (CoAP) that would list as well provide various services to interact with the various sensors and actuators available in the household.

CoAP is an alternative to Hypertext Transfer Protocol (HTTP) for interconnected objects, exploiting a binary data representation and a subset of HTTP methods (GET, PUT, POST, DELETE). It follows the Representational State Transfer (REST) paradigm for making data and resources accessible. CoAP uses User Datagram Protocol (UDP) for transport, as Transmission Control Protocol (TCP) is considered too resource-consuming.

It is not feasible to interact with such constrained devices, since at a particular time these nodes may be in a sleeping state, be a part of disperse networks or networks with inefficient multicasting capability. Therefore in order to interact and observe multiple devices at a time it is required that a gateway be created which acts as a single point destination to which a client needs to connect and be able to communicate with the various smart devices in the household. The Resource Directory supports various services like register, maintain, lookup and remove resource description. Here we focus on implementing these interfaces using Java that could be used in Android platform.

Also, to provide useful information from the data captured from the sensors using CoAP protocol we have enhanced the gateway with a SPARQL endpoint which offers a uniform interface to access resources and its observation data in the domestic network, as well as retrieve data using SPARQL queries. Additional services have been created which provide observation data based on the time and location criteria.

In addition we have added the support of group management by which resources having a similar feature can create and join a group, as well as enable the client to interact with the group member resources by a single command. This has been achieved by creating a proxy resource on which the external client sends a request.

ACKNOWLEDGEMENT

Achieving a milestone for any person alone is extremely difficult. However, there are motivators, which come across the curvaceous path like twinkling star in the sky and make our work easier. I am fortunate enough to get immense help from my teachers, colleagues, family and friends by their valuable suggestions and constructive criticism. It becomes my humble and foremost duty to acknowledge all of them.

Words can hardly express my deep sense of gratitude and indebtness that I owe to my esteemed project supervisor, Dr S.K Saxena for providing me the opportunity of carrying out this project under his guidance. I express my sincere and deepest regards to him for the support, advice and encouragement he provided, without which the project would not have proceeded smoothly. The regular meetings and discussions were invaluable in the realization of this work.

Special thanks to Dr. Rajiv Kapoor (HOD), Department of Computer Engineering for providing me with an independent and conducive atmosphere to carry out the necessary research work.

This study would not have been possible without the constant support and efforts of the faculty and staff of Department of Computer Engineering.

Finally, I would like to thank my family and friends, for their undying love and faith in me, which motivated me to strive hard at every step of my life.

Ravish Malhotra
University Roll no: 2K12/CSE/16
M.Tech (CSE)
Department of Computer Engineering
Delhi Technological University

DELHI TECHNOLOGICAL UNIVERSITY
DELHI - 110042

DECLARATION

I hereby declare that the Major Project-II work entitled "Implementing home gateway using the CoAP protocol" which is being submitted to the Delhi Technological University, in partial fulfilment of requirements for the award of Master of Technology (Computer Science and Engineering) in the Department of Computer Engineering, is a bonafide report of the Major Project –II carried out by me. The material contained in this report has not been submitted to any University or Institution for the award of any Degree.

Date: _____

Ravish Malhotra
University Roll no: 2K12/CSE/16
M.Tech (CSE)
Department of Computer Engineering
Delhi Technological University

DELHI TECHNOLOGICAL UNIVERSITY

DELHI - 110042

CERTIFICATE

This is to certify that the Major Project –II Report entitled "Implementing home gateway using the CoAP protocol" is the work of Ravish Malhotra (Roll No: 2K12/CSE/16). This project was completed under my supervision and forms a part of Master of Technology (Computer Science and Engineering) course curriculum in the Department of Computer Engineering, Delhi Technological University, Delhi.

Date: _____

Dr S.K Saxena
Project Guide
Department of Computer Engineering
Delhi Technological University

CONTENTS

LIST OF FIGURES i
LIST OF ABBREVIATIONS ii

OVERVIEW 1

CHAPTER 1: UNDERSTANDING CoAP PROTOCOL 4
1.1 INTRODUCTION 5
1.2 CoAP MESSAGE FORMAT 5
1.2.1 CoAP Message Types 6
1.2.2 CoAP Methods 7
1.2.3 CoAP Options 8
1.2.4 CoAP Observe Option 9
1.3 CORE LINK FORMAT 9
1.4 CoAP UDP BINDING 10
1.5 CoAP IMPLEMENTATIONS 11

CHAPTER 2 : CoAP RESOURCE DIRECTORY 12
2.1 INTRODUCTION 13
2.2 SUPPORTED OPERATIONS 13
2.2.1 Discovery 13
2.2.2 Registration 13
2.2.3 Update 14
2.2.4 Validation 15
2.2.5 Removal 15

2.2.6 Lookup 15

CHAPTER 3 : CoAP MULTICAST SUPPORT 16

3.1 INTRODUCTION 17

3.2 GROUP MANAGEMENT 17

3.3 CoAP APPLICATION LAYER GROUP MANAGEMENT 18

3.3.1 Join And Leave Group 18

3.3.2 Reading all group memberships (GET) 20

3.3.3 Updating a group membership (PUT) 20

3.3.4 Deleting a single group membership (DELETE) 20

CHAPTER 4 : RESOURCE DESCRIPTION FORMAT 21

4.1 INTRODUCTION 22

4.2 JENA FRAMEWORK 23

4.3 SPARQL: A GRAPH-BASED QUERY LANGUAGE 24

4.4 SENSOR ONTOLOGY 25

CHAPTER 5 : ANDROID SENSORS 28

5.1 INTRODUCTION 29

5.2 SENSING & SENSOR MANAGER APIs IN ANDROID 29

CHAPTER 6 : SYSTEM ARCHITECTURE AND DEVELOPMENT 32

6.1 INTRODUCTION 33

6.2 SOCKET HANDLING USING JAVA.NIO 34

6.3 IMPLEMENTATION OF SERVICES 35

6.3.1 Registration 35

6.3.2 Send Updates 37

6.3.3 Get All resources registered 38
6.3.4 Delete a resource 39
6.4 GROUP MANAGEMENT SERVICES 39
6.4.1 Create a Group 40
6.4.2 Join a group 41
6.4.3 Leave a group 42
6.5 SPARQL IMPLEMENTATION 43

CHAPTER 7 : RESULTS AND CONCLUSION 44
7.1 RESULTS 45
7.2 CONCLUSION AND FUTURE WORK 49

BIBLIOGRAPHY: 51

LIST OF FIGURES

Figure 1 : CoAP Message Format....6
Figure 2 : The Observe Option 9
Figure 3 : Resource Discovery Architecture....13
Figure 4 : Resource Registration Flow 14
Figure 5 : Resource Update Flow 14
Figure 6 : Resource Deletion Flow....15
Figure 7 : Resource Lookup Flow 15
Figure 8 : CoAP Message for Group Management 18
Figure 9 : CoAP Multicast Support 19
Figure 10: RDF schema of a proximity sensor. 23
Figure 11: SPARQL Query Syntax 25
Figure 12: RDF Of Sensor based On SSN Schema....26
Figure 13: RDF of Observation based on SSN Schema....27
Figure 14: RDF of Sensor Location based on SSN Ontology....27
Figure 15: List of Android Sensors....30
Figure 16: Android Sensor Events....31
Figure 17: Connect to Resource Directory 46
Figure 18: Initial Page that allows Sensor Registration....46
Figure 19: After Registration with the RD 46
Figure 20: Send Updates on event change as well as on receive updates 46
Figure 21: Sensor is provided with an option to create, join , leave group 47
Figure 22: Sensor can create a group for ex:proximitysensor.group....47
Figure 23: Sensor can join the groups that are created in the RD 47
Figure 24: Sensor can leave the groups that it has joined earlier 47
Figure 25: Copper(Cu) Interface showing the registered resources 48
Figure 26: Copper(Cu) Interface showing a resource description....49

LIST OF ABBREVIATIONS

API	Application Programming Interface
Cf	Californium
CoAP	Constrained Application Protocol
CON	Confirmable Message
CoRE	Constrained RESTful Environments
Cu	Copper
DTLS	Datagram Transport Layer Security
Er	Erbium
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
NON	Non Confirmable Message
MLP	Multicast Listener Discovery Protocol
MTU	Maximum Transmission Unit
POS	Predicate-Object-Subject
RD	Resource Directory
RDF	Resource Description Framework
REST	Representational state transfer
RST	Reset
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
XML	Extensible Markup Language

OVERVIEW

The concept of Internet of Things (IoT) is in huge popularity and is a mean to make things smarter by allowing multiple devices to connect to the internet via wired/wireless network and share their data. These devices would essentially be comprised of sensors that can register various changes in environment like temperature, light, pressure, sound and motion . However, these devices have many limitations such as limited capabilities in terms of computation and memory and operate in constrained environments, such as low power and lossy networks. The existing protocols (HTTP over TCP/IP) do not comply with such constrained devices. Hypertext Transfer Protocol (HTTP) turns out to be heavy for such constrained devices. HTTP requires minimum of 9 packets, and has a big header that can get too verbose. Also HTTP does not guarantee message delivery and is dependent on lower layer protocols to manage this. It also suffers from packet loss and does not serve the purpose of bandwidth conservation, which becomes important due to current costs of connectivity.

As such new alternatives have been developed for communication among IoT devices. Internet Engineering Task Force (IETF) is currently working to develop a new protocol Constrained Application Protocol (CoAP) to be used as a generic protocol for constrained environments and can be seen in some ways as a compressed version of the HTTP, reducing the complexity of implementation as well as the size of packets exchanged. CoAP as compared to HTTP uses UDP instead of TCP, as a lightweight protocol.

CoAP provides a restful mechanism for allowing communication and making data and resource accessible. A resource state is referred by its namespace named as uri. The communication between the client and resource uses restful methods like GET, POST, PUSH, DELETE, OBSERVE. Client which is interested in knowing the state of a resource sends a request to the server which then responds with the current representation of the resource.

CoAP includes various features like request/response interaction model between application endpoints, discovery of resources, key concepts of the web such as Uniform Resource Identifiers (URI) and internet media types. Also it supports additional features like publish subscribe mechanism in order to get notifications. Caching has also been used to support fast transmission of data.

However, it is not feasible to interact with such constrained devices since at a particular time these nodes may be in a sleeping state. Also, in order to interact and observe multiple devices at a time it is required that a gateway be created which acts as a single point destination to which a client can connect and communicate with the various smart devices in the household.

As mentioned, the problem of direct discovery is resolved by using a gateway also known as a resource directory to which the smart devices register and keep on updating the data at different time intervals to the gateway. Also, it provides restful services to the client on the other side, to look up these resources. In a nutshell, the resource directory allows services to discover, register, maintain, lookup and remove resources.

In this work we implement a gateway that acts as a resource directory for the household smart devices, the gateway has been built by using the java implementation of CoAP namely jCoAP. jCoAP has been used as it is suitable for android devices which have various sensors installed on them that have been used in this project. jCoAP is still in its developing stage, and has limited features. We have augmented this implementation to support multicast support and allow group communication to take place through CoAP protocol. This is achieved by creating a proxy resource on which the external client sends a request.

Also, to store the various dataset collected from the devices, we have used semantic web paradigm that allows data from disparate sources having different conventions being managed by using Resource Description Format (RDF) scheme that are stored in the predicate –object – subject (pos) form in the triple data base.

An Android application has been created that allows the sensors (namely proximity sensor used in the project) inside the android device to interact with the gateway. The application initially connects to the gateway and on success, is provided the option to send its sensor data to the gateway. Along with this, the option to join and leave a group is also provided.

As a client, a separate host has been used that interacts with the gateway using Copper (Cu) implementation and can interact with the gateway using the restful services.

In the subsequent sections to follow, we go into a greater depth. The initial section introduces the CoAP protocol and understanding the various features provided by it as well as discusses the various implementation of CoAP inplace, in particular jCoAP, which has been used in the project. This is followed by an insight into the working of a Resource Directory (RD) that uses CoAP protocol, and discusses the various restful

services it provides. Next we introduce the concept of multicast and how the same can be achieved in the CoAP resource directory system. The next section provides an introduction to Semantic Web that has been used over relation database in the project to manage the sensor information. This is followed by a brief description of the Android Sensor APIs and sensor events. Next section covers the system architecture and implementation of the project. The last section discusses the results achieved and draws the conclusion.

CHAPTER 1

UNDERSTANDING CoAP PROTOCOL

1.1 INTRODUCTION

The Constrained Application Protocol (CoAP) is a specialised web transfer protocol for use with constrained nodes (e.g., low-power sensors, switches, or valves) and constrained (e.g., low-power, lossy) networks.
The Constrained RESTful Environments (CoRE) working group aims at realising the Representational state transfer (REST) architecture in constrained nodes and networks, keeping the message overhead small, thus limiting the use of fragmentation for expensive fragmentation of IPv6 packets.
CoAP is an application layer protocol that easily translates to HTTP for integration with the existing web while providing additional features as multicast support, very low overhead and simplicity for constrained environments, and machine-to-machine applications.
CoAP uses two message types, requests and responses, using a binary fixed-size header format followed by options in Type-Length-Value (TLV) format and a payload. The payload length is determined by the datagram length, which must fit inside a single UDP datagram. Message exchange is based on requests made to resource values accessed through Unique Resource Identifier.

CoAP provides the following options:

- Constrained Web protocol fulfilling machine-to-machine requirements.
- UDP binding with optional reliability supporting unicast and multicast requests.
- Low header overhead and parsing complexity.
- Push notifications through a publish/subscribe mechanism.
- Simple proxy and caching capabilities.
- Security binding to Datagram Transport Layer Security (DTLS).

1.2 CoAP MESSAGE FORMAT

The CoAP message format comprises of a fixed-length 4-byte header, followed by a series of options.

The header is constructed by:

- First two bits indicate the version of the protocol.
- Second two bits indicate the message type: Confirmable (CON), Non Confirmable (NON), Reset (RST) and Acknowledgement (ACK).

- 4 bits are kept to indicate the token length. So far, only a length between 0 and 8 bytes is allowed. Other values must be processed as message errors.
- 8-bit unsigned integer. This field indicates the Method or Response Code of a message. The value 0 indicates no code. The values 1-10 are used for Method Codes.
- 16 bits of the header contain the message id, to identify message duplication.
- Rest of the message can contain optional options and values. Options must appear in order of option type. A delta encoding is used between each option header, with the Type identifier for each Option calculated as the sum of its Option Delta field and the Type identifier of the preceding Option
- The rest of the packet is composed of the payload.

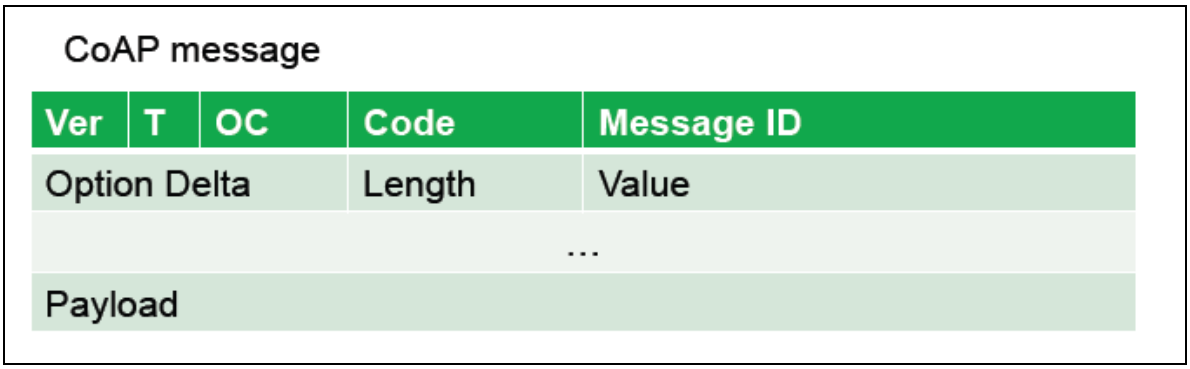

Figure 1 : CoAP Message Format

1.2.1 CoAP MESSAGE TYPES

As mentioned in the previous section, CoAP provides four types of messages:

Confirmable (CON): A confirmable message ensures that the sender will receive a acknowledgment from the receiver. In case the sender does not receive an acknowledgment within the stipulated time, retransmission will take place by using an exponential algorithm in which case the timeout period gets doubled. Thus confirmable message ensures reliability to unreliable UDP protocol.

Non-Confirmable (NON): A non-confirmable message is not acknowledged by a server. In this case, the sender may send many requests until he receives back an acknowledgement.

Reset (RST): A reset message indicates that a specific confirmable message was received, but some context is missing to properly process it. It tells the client that

something has gone wrong during the communication and its reasons are explained by the message code contained in the message.

Acknowledge (ACK): Acknowledgement have to be sent as a response of a CON message. The ACK can be piggybacked with the response and uses the same message id as that of the request to which the ACK corresponds. The client receiving the response has to acknowledge it using the new message id.

1.2.2 CoAP METHODS

CoAP supports the basic RESTful methods of GET, POST, PUT, DELETE, which are similar to HTTP. The GET, PUT and DELETE methods must be performed in such a way that they are idempotent.

GET

- The GET method retrieves the information of the resource identified by the request URI. Upon success a 200 (OK) response should be sent.
- The response to a GET method is cacheable if the resource whose information is gathered is less than as specified as the max age of the resource in its option tag. Cache refresh and versioning is handled using Etag option.

POST

- The POST method is used to request the server to create a new resource under the requested URI and get the resource registered on the server.

PUT

- The PUT method requests that the resource identified by the request URI be updated with the enclosed message body.

DELETE

- The DELETE method requests that the resource identified by the request URI be deleted. The response 200 (OK) should be sent on success.

1.2.3 CoAP OPTIONS

CoAP provides additional information to message exchange. It is composed of a numeric code, a format and a length. Options can be either critical (odd code value) and elective (even code value). The difference is in the way these are recognized by the server.
An unrecognized critical option makes the server to reset the connection by means of a RST message while an elective option gets ignored when it is not recognized properly.

Some of the CoAP options are:

Uri-Host, Uri-Port, Uri-Path, Uri-Query

Uri-Host, Uri-Port, Uri-Path and Uri-Query identify the targetted resource.

- Uri-Host option either represents the hostname or the ip address of the resource.
- Uri-Port represents the port of the resource.
- Uri-Path is a repeatable option containing in order all the components of the path identifying the resource in the device.
- Uri-Query specifies additional parameters to the resource query.

Content-Format

It indicates the content format of the message payload. So far, the acceptable values are a subset of the internet media types (also known as MIME).

Accept

It is a repeatable option used to specify which content format is acceptable in the response payload.

Max-Age

Max-Age indicates how long the resource can be cached before it is considered not fresh by the server.

ETag

The ETag identifies a particular representation of a resource. If the server supports it, it is able to mark every returned value. When the client uses it, the server is able to confirm if the retained resource is still valid without sending its value again.

Location-Path, Location-Query

These options contain the relative URI and a query string. It is used to indicate where the resource has been created in response to a POST request. While Location-Path is non-repeatable, Location-Query can be set multiple times to indicate all the queries parameterizing the resource.

1.2.4 CoAP OBSERVE OPTION

The Observe Option, when present, modifies the GET method. So it does not only retrieve a representation of the current state of the resource identified by the request URI, but also requests the server to add the client to the list of observers of the resource. The value of the option in a request must be zero on transmission and must be ignored on reception. In a response, the Observe Option identifies the message as a notification, which implies that the client has been added to the list of observers and that the server will notify the client of further changes to the resource state.

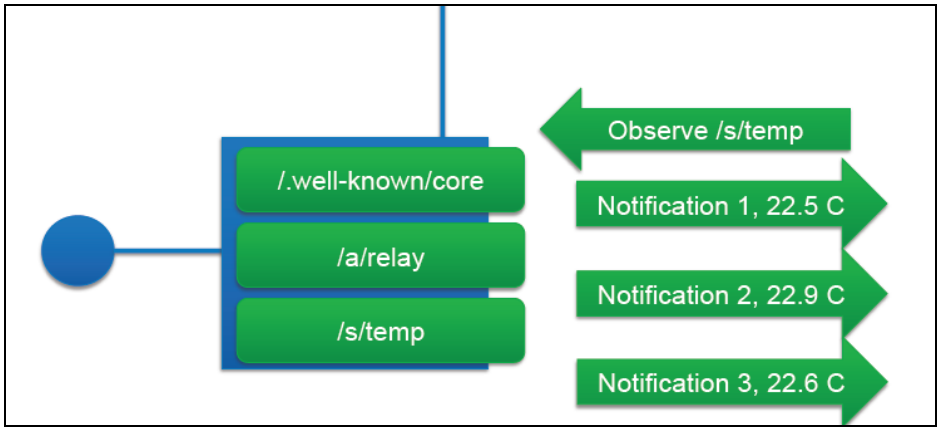

Figure 2: The Observe Option

1.3 CORE LINK FORMAT

A key feature for machine-to-machine interaction is resource discovery. Core Link Format has been defined to allow this feature in Constrained RESTful environments.

Resource discovery in Core Link Format makes the description of the resources available on the well-known interface ./well-known/core of each server. By this way, every server is provided with a default entry point meant to provide a description of its resources. Every resource is described by means of its Unique Resource Identifier, a set of attributes and the relations with the other resources.

Some of the parameters provided along with each resource:

- Title: This provides a human readable description of the resource.
- Type: This contains the media type of the returned resource.
- Resource Type (rt): This attribute contains a string used to assign an application specific semantic type to the resource.
- Interface Description (if): This indicates opaquely a specific interface definition.
- MTU: This attribute can be used to indicate approximately the expected size of the response.

1.4 CoAP UDP BINDING

CoAP over UDP has the following features:

- **Simple stop-and-wait retransmission** reliability with exponential back-off for Confirmable messages.
- **Transaction ID** for response matching
- **Multicast support** CoAP supports the use of multicast destination addresses. Multicast messages should be Non-Confirmable. If a Confirmable multicast message is sent, then retransmission must not be performed.
- **Retransmission**
  CoAP end-point keeps track of open confirmable messages it sent and are waiting for a response. Each entry includes at least
  - Destination IP address and port of the original message.
  - A retransmission counter.
  - A timeout.

  When a confirmable message is sent, an entry is made for that message with a default initial timeout of RESPONSE_TIMEOUT and the retransmission counter set to 0. When a matching acknowledgment is received for an entry, the entry is invalidated.

When a timeout is triggered for an entry and the retransmission counter is less than MAX_RETRANSMIT, the original message is retransmitted to the destination without modification, the retransmission counter is incremented, and the timeout is doubled.

1.5 CoAP IMPLEMENTATIONS

Californium
Cf is a CoAP framework written in Java and developed for the use in unconstrained environments. This allows an isolated implementation of different aspects such as message retransmission, transactions and block-wise transfer.

Erbium
Er is a low-power REST Engine for the Contiki operating system, which allows low-power systems to communicate with the Internet. This implementation is specialised for constrained environments as it is designed to run on small amounts of memory and low-power Central Processing Units or Microcontroller Units .

Copper
Cu is a CoAP user-agent for Firefox implemented in JavaScript, which can be added to firefox as an addon and enables the users to browse IoT devices in the same fashion in which they are used to explore the Web. It can render different types of response in the form of XML, JSON.

jCoAP
jCoAP is a java implementation of CoAP and it is still in its early-stages developed by University of Rostock. It is compatible with Java SE and Android. It currently supports limited functionality of CoAP but it is the most complete implementation in comparison to the other java implementations.

CHAPTER 2

CoAP RESOURCE DIRECTORY

2.1 INTRODUCTION

This section defines the REST interfaces between a Resource Directory (Gateway) and devices, and a lookup interface between the Gateway and the client.

The Gateway supports the discovery, registration, update, removal and lookup interfaces.

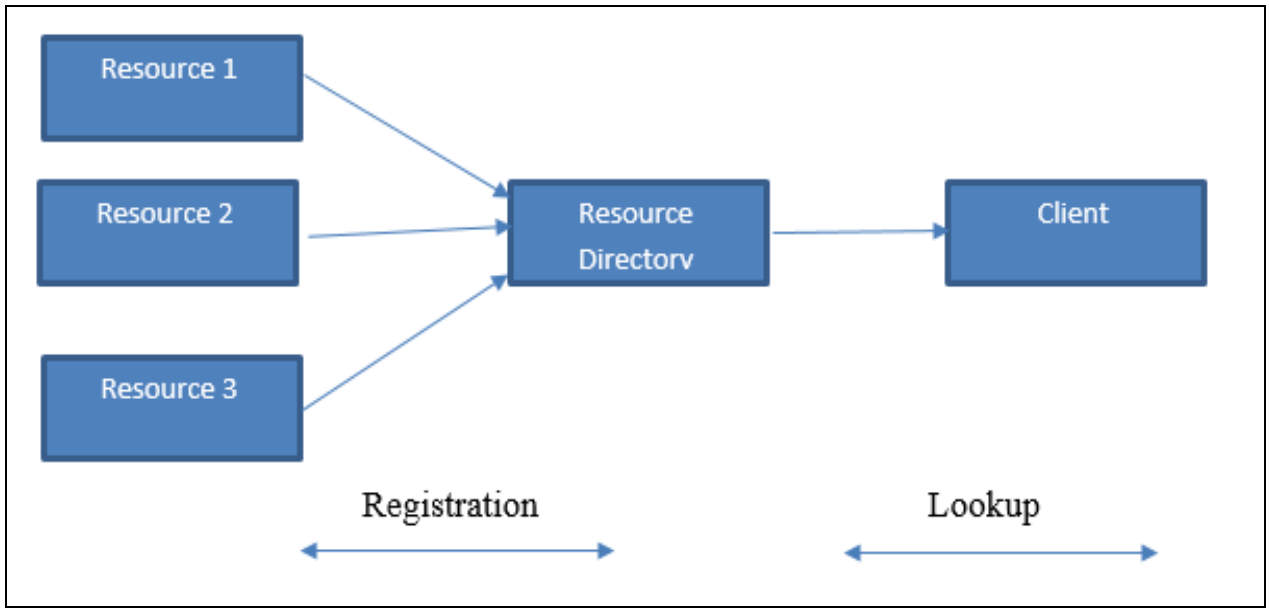

Figure 3: Resource Discovery Architecture

2.2 SUPPORTED OPERATIONS

2.2.1 DISCOVERY

A device can discover/know its gateway by several mechanisms such as a default location (as used by our application), by assigning an anycast address to the gateway, using DHCP, or by discovering the gateway using the CoRE Link Format.

2.2.2 REGISTRATION

After discovering the location of the gateway, the device can register its resources to the gateways registration interface. This is performed by using a POST method call along

with the list of resources in a core link format specifying the additional information like name of the end-point, an optional node identifier and the lifetime of the registration. The gateway then creates a new resource and returns its location. The resources remain active for the period-specified using max age and can be updated with new information using PUT services until its age exceeds the max age.

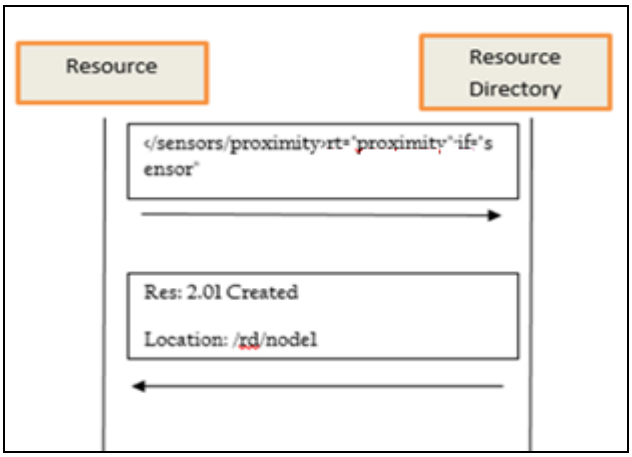

Figure 4: Resource Registration Flow

2.2.3 UPDATE

Update is performed by using the PUT service. Through this, the device can update its information in the gateway.

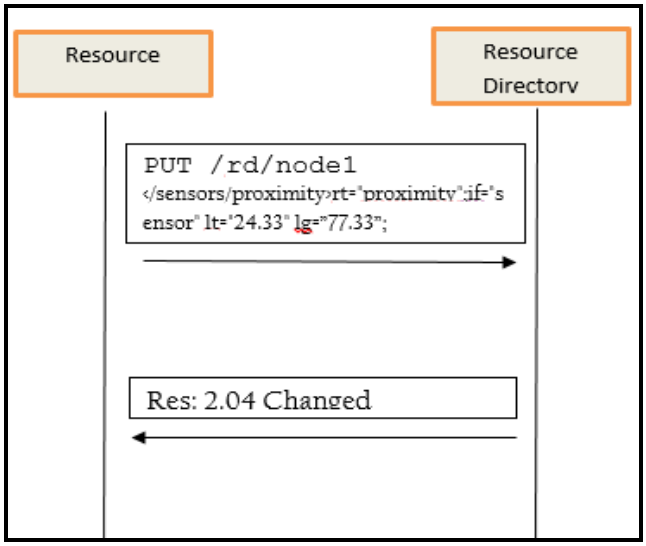

Figure 5: Resource Update Flow

2.2.4 VALIDATION

This is used to validate if the gateway has the latest version of the resource by issuing a GET method with the latest Etag.

2.2.5 REMOVAL

To remove a resource explicitly from a gateway a DELETE method request is used along with the of the resource to be removed. The removal interface is specified as follows:

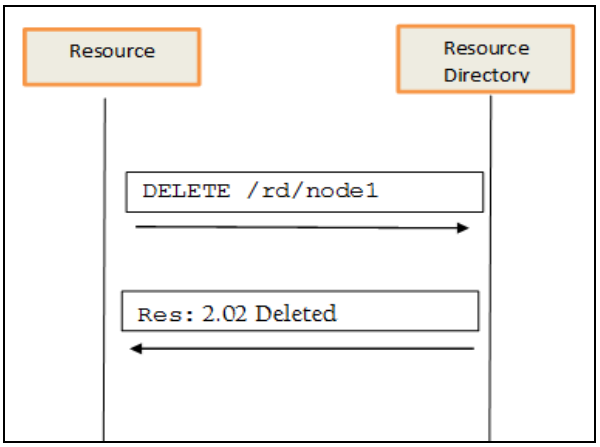

Figure 6: Resource Deletion Flow

2.2.6 LOOKUP

The client can look up the resource directory using the GET method on the core interface using the core format query to look up the resource.

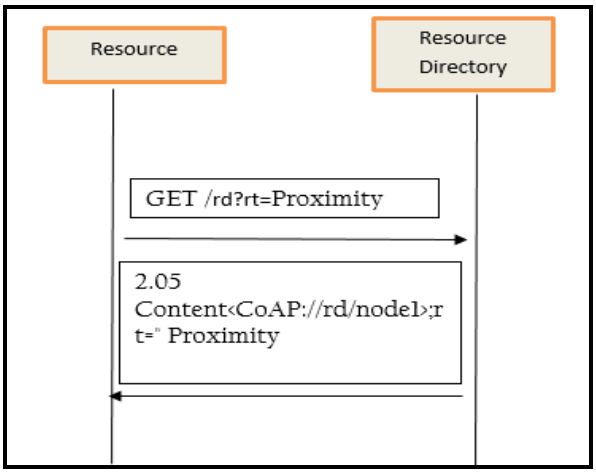

Figure 7: Resource Lookup Flow

CHAPTER 3

CoAP MULTICAST SUPPORT

3.1 INTRODUCTION

As a part of Internet of Things, it is required to support group management to better manage the devices.

For example : To close all the smart bulbs in a floor we can create groups for the various smart bulbs on different floors and give a single command (Close the bulb) which would be multicasted to members of the group.

All those group members which receive the notification , on receiving the message would take necessary action.

3.2 GROUP MANAGEMENT

IETF suggests two alternative approaches possible for CoAP group communications each with associated pros/cons:

- IP Multicast in which routers must support multicast protocols.
- CoAP Application level Group Management where application layer must support multicast functionality.

IP Multicast

In this, the CoAP sub-networks are directly connected to IP multicast enabled routers. Sending CoAP node can directly transmit group messages by setting IP address to selected multicast IP group address.

Receiver CoAP nodes use Multicast Listener Discovery Protocol (MLP) to subscribe and listen to any messages sent to selected IP multicast group.

Although it is the most efficient solution, since it is done at IP layer, however it cannot be deployed outside of corporate LANs and hence is practically unfeasible.

3.3 CoAP APPLICATION LAYER GROUP MANAGEMENT

CoAP can support group management features by either using IP layer multicasting or application layer support which does not require any underlying IP multicast support. CoAP allows the use of following group management features such as

- Create groups
- Discover groups
- Query group properties
- Remove from a group
- Add group members
- Remove a group member
- Provide security and access control primitives.

Multicast support can be handled by using a CoAP Proxy node which is responsible for group membership management. A constrained node joins (or leaves) a group by sending a CoAP request to the appropriate CoAP Group Proxy resource created. To join, the group name is included in the header field and sent using a PUT request to the Group Proxy Resource. Group names may be defined as arbitrary strings with a predefined maximum length or as URIs.

3.3.1 JOIN AND LEAVE GROUP

CoAP supports two elective Header Options for group management "Join" and "Leave" and hence assigned an even number. Packet for a node that can join or leave a group is represented using the given header format.

Ver	T	OC	Code	Message Id
delta	Length	Join Group A (URI)		
0	Length	Join Group B (URI)		

Figure 8: CoAP Message for Group Management

The join and leave group are provided as repeatable options in the header. Within the constrained network, CoAP runs over UDP for which IP multicast is supported. In a non-constrained network, HTTP over TCP is used for which IP multicast is not supported. A proxy node that supports group communication needs to have functionalities to support interworking of unicast and multicast.

Possible way of operation of the Proxy is presented:

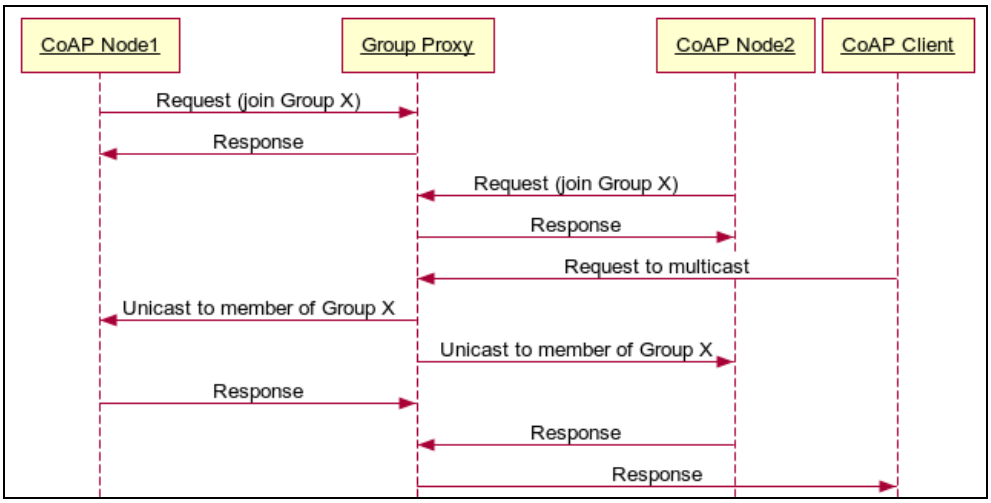

Figure 9: CoAP Multicast Support

The incoming request will carry a URI that resolves in the general internet to the proxy node. At the proxy node, the URI will then possibly be mapped and again resolved to an IP multicast destination. The proxy node will then multicast the CoAP Request to the appropriate nodes.

The resource includes zero or more group membership JSON objects. A group membership JSON object contains one or more key/value pairs. The key represents the index of the group. The OPTIONAL "n" key/value pair stands for "name" and identifies the group with a hostname, for example a FQDN. The OPTIONAL "a" key/ value pair specifies the IP multicast address

Examples of different group membership objects are: { "n": "sensors.proximity.group1" }

Following group management interfaces are supported:

3.3.2 READING ALL GROUP MEMBERSHIPS (GET)

A (unicast) GET on the CoAP-group resource returns a JSON object containing multiple keys and values, the keys being group indices and the values the corresponding group objects that indicates one multicast group membership.

Example:

Req: GET /CoAP-group

Res: 2.05 Content

Content-Format: application/CoAP-group+json

```
{ "1" :{ “n” : sensors.proximity.group1},
  "2":{ “n” : sensors.proximity.group2}
}
```

3.3.3 UPDATING A GROUP MEMBERSHIP (PUT)

A (unicast) PUT with a group configuration media type as payload will replace all current group memberships in the endpoint with the new ones defined in the PUT request.

3.3.4 DELETING A SINGLE GROUP MEMBERSHIP (DELETE)

A DELETE message is used to remove a group

Example:

Req: DELETE /CoAP-group /sensors.proximity.group1

Res: 2.02 Deleted

CHAPTER 4

RESOURCE DESCRIPTION FORMAT

4.1 INTRODUCTION

The home directory can be extended to form a network of such gateways which can then be part a part of middleware, from where such multiple home directory nodes can be accesssed. However to integrate such sensor data from disparate resource directory nodes, it is imperative that a common ontology is followed such that the sensor data is represented in a similar schema.
RDF is used as a data modelling tool for semantic web. It is used to represent graph like structure which are in the form of interconnected represented nodes representing linked data. The rdf scheme of graph representation is based on the SPO model. The Subject-Predicate-Object (SPO) model of rdf breaks up each edge connecting 2 nodes into 3 entities namely subject, predicate, object .
Each subject is represented as:

- blank node or
- URI, like http://samsung.core.sensor/proximity

Each predicate represents a relation/property between subject and object and is represented as:

- a URI, like http://samsung.core.sensor/proximity #maximumrange

Each object of a triple is represented as:

- a blank node, or
- a literal value

For example – 1.0 (maximum range)

Each SPO statement is termed as a triple.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:j.1="http://sensor/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:j.0="http://psensor/">
 - <rdf:Description rdf:about="http://sensor/xml">
    - <j.1:SensorName>
       - <rdf:Description rdf:about="http://psensor/the android open source project/goldfish proximity sensortest1">
            <j.0:power>20.0</j.0:power>
            <j.0:vendor>The Android Open Source Project</j.0:vendor>
            <j.0:version>1</j.0:version>
            <j.0:resolution>1.0</j.0:resolution>
            <j.0:maximumrange>1.0</j.0:maximumrange>
            <j.0:name>Goldfish Proximity sensor</j.0:name>
         </rdf:Description>
      </j.1:SensorName>
   </rdf:Description>
</rdf:RDF>
```

Figure 10: RDF schema of a proximity sensor.

4.2 JENA FRAMEWORK

RDF Graphs can be manipulated using Java based Jena framework that allows users to manipulate and query RDF graphs. Jena can be used to retrieve and parse a RDF file that contains a graph or a collection of graphs (graphset), store it in memory, examine each triple in turn using SPARQL queries or Jena APIs, write a serialized version of a graph to a file or STDOUT.

For example, to retrieve a resource uri having maximum range of 1.0 or retrieve list of all proximity sensors jena can be used.

An RDF graph in Jena is stored as a "model", which is obtained from its factory method:

Model m = ModelFactory.createDefaultModel();

Once a model has been defined, Jena can populate it by reading data from files specified in the rdf format, backend data bases, etc. in various formats, and once it has been populated, Jena can perform set operations on pairs of populated models and/or search models for specific values or combinations (patterns) of values.

One can access specific components using the Jena APIs

model.listSubjects(); // list subjects in the dataset of the model

model.listObjects();// list objects in the dataset of the model

Jena also provides APIs to compare a given component with a specified value:

model.listSubjectsWithProperty(Prop p, RDFNode object); //collection of subjects having property/predicate p .

Using selector to compare all components against specific values by defining a "selector" possessing specific values s, p and o, and then build the statement list.

Selector selector = new SimpleSelector(subject, predicate, object)

model.listStatements(selector);

4.3 SPARQL: GRAPH BASED QUERY LANGUAGE

Sparql is a language that is used to retrieve data from rdf graphs by specifying "templates" against which to compare graph components. Data that matches a template is returned from the query.

A triple template will contain variables that represent triplet

components (e.g., a subject, predicate, or object within a triplet).

For example the template:

?psensor <j:maximumrange> "1.0"^^xsd:float .

identifies a list of triplet subjects that have a maximum range of "1.0", and is analogous to asking "Which proximity sensor has a value 1.0?"

The basic syntax of a SPARQL query is of the form

BASE < some URI from which relative FROM and PREFIX entries will be offset >

```
PREFIX prefix_abbreviation: < some_URI >
SELECT
 some_variable_list
FROM
< some_RDF_source_URL >
WHERE
{
 { some_triple_pattern .
 another_triple_pattern . }.
}
```

Figure 11: SPARQL Query Syntax

Also there may be multiple FROM clauses, whose targets will be combined and treated as a single store. A "." separating multiple triple patterns is similar to "and" operator and is similar to the join operator between 2 triples.

The rdf data is essentially stored in dbms systems, sometimes called "triplestores," that have been customized to handle RDF. Two examples are Sesame and OpenLink's Virtuoso system.

4.4 SENSOR ONTOLOGY

Semantic technologies can assist in managing, querying, and combining sensors and observation data, allowing users to operate at an abstract level. The SSN-WSN group designed an OWL Ontology to describe properties and capabilities of sensors, act of sensing and resulting observation.

The SSN ontology is based on the Ontology Design Pattern (ODP) that describes the relationships between sensors, stimulus, and observations, the Stimulus-Sensor-Observation(SSO) pattern. The ontology can be seen from four main perspectives:

• A sensor perspective, with a focus on what senses, how it senses, and what is sensed.

For Example:

A sensor device having following attributes

- Name : proximitysensor_30
- SourceType :coapproject
- SensorType :proximity sensor
- Infor : Proximity sensor reading
- Source : "http://www.coap.rd/sensor/proximitysensor_30”
- Property : DISTANCE

can be represented in an ssn ontology in the given form:

```
<sensoruri><#type><#Sensor>.
<sensoruri><#PerformedAt>"201422T16:13:01.549+05:30"^<2001/XMLSchema#dateTime>
<sensoruri><#PerformedBy><http://www.coap.rd/sensor/proximitysensor_30>.
<sensoruri><hasSourceType> "coapproject ".
<sensoruri><#label> "proximitysensor_30".
<sensoruri><#hasLocation><locationuri>.
<sensoruri><#hasSensorType><sensorTypeuri>.
<sensoruri><#observes><observation1Uri>.
<sensoruri><#observes><observation2Uri>.
```

Figure 12: RDF Of Sensor based On SSN Schema

• An observation perspective, with a focus on observation data and related metadata. An Observation can have multiple observed properties.

Observation

- Time
- List of ObservedProperty
- Sensor which observes it

ObservedProperty

- ObservationId
- Type: DISTANCE
- Value :5
- Unit :"cm"

```
<ObservationUri><#type><http://purl.oclc.org/NET/ssnx/ssn#Observation>.
<ObservationUri><#observedBy><resourceUri>.
```

```
<ObservationUri><#featureOfInterest><resourceUri>.
<ObservationUri><#observationResultTime> "2014-05-22T17:11:02"^<#dateTime>.

<ObservationValueUri><#type><#ObservationValue>.
<ObservationValueUri><#isObservedPropertyOf><ObservationUri>.
<ObservationValueUri><#value> "5"^^<#double>.
<ObservationValueUri><#unit> "cm".
<ObservationValueUri><#label> "Distance".
<ObservationValueUri><#observedProperty><null>.
<ObservationValueUri><#observationResultTime>"2014-22T17:11:02"^^<#dateTime>.
```

Figure 13: RDF of Observation based on SSN Schema

• A system perspective, with a focus on systems of sensors and deployments.

- lat
- lng
- zipcode
- street
- city
- province
- country
- Linked Sensor

```
<sensoruri><#hasLocation><locationuri>.
<locationuri><#type><#Place>.
<locationuri><#type><#SpatialThing>.
<locationuri><pos#lat> "37.943267"^^<#decimal>.
<locationuri><pos#long> "23.870287"^^<#decimal>.
<locationuri><#label> ",".
<locationuri><#is_in_city><cityurl>.
<cityurl><#type><City>.
<cityurl><#label> ""..
```

Figure 14: RDF of Sensor Location based on SSN Ontology

• A feature and property perspective, focusing on what senses a particular property or what observations have been made about a property.

CHAPTER 5

ANDROID SENSORS

5.1 INTRODUCTION

A sensor measures a physical quantity and converts it into a signal, which can be read by an observer or by an instrument.

In Android a sensor is represented by a Sensor class which generates a Sensor Event when an event is generated. All sensors in a device are accessed using the Sensor Manager.

Some of Sensors provided in Android:

- **Sensor.TYPE_AMBIENT_TEMPERATURE**
  This sensor measures room temperature in degrees Celsius.

- **Sensor.TYPE_GRAVITY**
  This sensor measures gravity, in case the phone is at rest same as TYPE_ACCELEROMETER.

- **Sensor.TYPE_GYROSCOPE**
  This measure device's rate of rotation in radians / second around three axes.

- **Sensor.TYPE_LIGHT**
  This sensor measures light level in lux, lux is SI measure illuminance in luminous flux per unit area.

- **Sensor.TYPE_LINEAR_ACCELERATION**
  This measures acceleration force applied to device in three axes excluding the force of gravity.

- **Sensor.TYPE_MAGNETC_FIELD**
  This sensor measures ambient geomagnetic field in all three axes.

- **Sensor.TYPE_PRESSURE**
  This sensor measures ambient air pressure in hPa or mbar

- **Sensor.TYPE_PROXIMITY**
  This measures proximity of an object in cm relative to the view screen of a device. It is typically used to determine if handset is being held to person's ear during a call.

- **Sensor.TYPE_RELATIVE_HUMIDITY**
  It measures ambient humidity in percent (0 to 100).

- **Sensor.TYPE_TEMPERATURE**
  It measures temperature of the device in degrees Celsius.

Available Android Sensors

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a[1]	n/a[1]
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes[2]	Yes[2]	Yes[2]	Yes
TYPE_PRESSURE	Yes	Yes	n/a[1]	n/a[1]
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes[2]	Yes	Yes	Yes

Figure 15: List of Android Sensors

SENSOR-TYPE	VALUE COUNT	VALUE COMPOSITION	COMMENTARY
TYPE_ACCELEROMETER	3	value[0] : Lateral value[1] : Longitudinal value[2] : Vertical	Acceleration along three axes in m/s^2. The Sensor Manager includes a set of gravity constants of the form SensorManager.GRAVITY_*
TYPE_GYROSCOPE	3	value[0] : Azimuth value[1] : Pitch value[2] : Roll	Device orientation in degrees along three axes.
TYPE_ LIGHT	1	value[0] : Illumination	Measured in lux. The Sensor Manager includes a set of constants representing different standard illuminations of the form SensorManager.LIGHT_*
TYPE_MAGNETIC_FIELD	3	value[0] : Lateral value[1] : Longitudinal value[2] : Vertical	Ambient magnetic field measured in microteslas (μT).
TYPE_ORIENTATION	3	value[0] : Azimuth value[1] : Roll value[2] : Pitch	Device orientation in degrees along three axes.
TYPE_PRESSURE	1	value[0] : Pressure	Measured in kilopascals (KP).
TYPE_PROXIMITY	1	value[0] : Distance	Measured in meters.
TYPE_TEMPERATURE	1	value[0] : Temperature	Measured in degrees Celsius.

Figure 16: Android Sensor Events

5.2 SENSING & SENSOR MANAGER APIs IN ANDROID

ServiceManager provides access to the following services

- String service_name = Context.SENSOR_SERVICE;
- SensorManager sensorManager=(SensorManager)getSystemService(service_name)
- public int getMinDelay () //returns the minimum delay allowed between two events in microsecond .
- public float getMaximumRange () //returns the maximum range of the sensor in the sensor's unit
- public String getName () //returns the Sensor Name.
- public float getResolution () // returns the resolution of the sensor in the sensor's unit.
- public float getPower () //returns the power in mA consumed by sensor while in use.

CHAPTER 6

SYSTEM ARCHITECTURE AND DEVELOPMENT

6.1 INTRODUCTION

A java version of CoAP implementation namely jCoAP has been used to develop a CoAP Client as well as a CoAP resource server which can register, update, discover, validate or delete the resources published via the CoAP client using CoAP methods namely POST, PUT, GET. jCoAP also provides the feature of HTTP proxy that can be used where the underlying protocol used is HTTP. A corresponding mapping from CoAP to HTTP and HTTP to CoAP via proxy is thus provided.

Currently the stable version of jCoAP supports limited features and thus had to be customized accordingly for different methods GET, POST, PUT, DELETE.

Also, we have used a rdf scheme to represent each resource and store the resources into a dataset used by Fuseki (a Sparql endpoint). Each resource having a unique url is represented as a rdf and each property of the resource is associated with a timestamp .On further updates by the client the resource rdf is updated along with the timestamp. These rdf are presented to the fuseki which is a sparql endpoint and can be used to query and retrieve data from rdf dataset.

In our project, the CoAP client has been implemented on an Android Device which has various sensors namely Proximity Sensor, Orientation Sensor, Accelerometer, Temperature Sensor, Magnetic field Sensors.

The CoAP client is installed as an App on the Device which allows the user to provide the gateway Socket Details (IP Address, Port). Once connected, the various attributes of the sensor are posted to the gateway which registers the resource with /.well-known/core port, stores them in the form of RDF file and also a cache (implemented in the form of HashMap having the resource uri as the key).

Further updates due to events on sensors are provided to the server using PUT method. This data is also added to the resource rdf file along with the timestamp.

Also, we have used a firefox addon named Copper (Cu) Agent which provides a CoAP based Web Browser and provides various CoAP methods like GET, POST, PUT, DELETE ,OBSERVE. The Copper Agent can be used to discover the resources on the server/resource directory and further get information using restful calls built upon the GET method.

An observe option is also provided in which we can register for a resource and we would be notified whenever a change from the sensor is published. Copper also provides renderer for data of different type eg XML, JSON. The data is thus provided as a JSON to the Copper.

We have also provided group management support by allowing resources to form a group. For achieving the same instead of using multicast sockets which depend on the hardware of the lower layer network elements, we have supported the group management by handling the same at application layer. This has been achieved by creating a proxy resource for group management. This resource receives all the services related to group management and is responsible for handling all the group related concerns essentially creation of group, allowing joining or leaving from a group, sending message to group members.

We would now look into a greater depth of how the same has been achieved.

6.2 SOCKET HANDLING USING JAVA.NIO

The system compromises of three entities 1. The Endpoint 2.Resource Directory 3.Client.

The endpoint is a device that has sensors inside it which are to be registered to the resource directory.The endpoint sends its information via restful webservices to the resource directory.

The endpoint consists of a DatagramChannel (used for UDP message transfer) which initially binds to a local port once it is in open state and then performs a connect operation to the resource directory. The Datagram Channel is then configured to receive and send datagrams to a host (i.e RD) using the connect method. We then send the CoAP message initially added to a buffer by using the send method on the datagram channel and keep the message in a queue in case it is a CON message . The receive method of datagram channel waits for an acknowledgment for the corresponding message. In case no acknowledgment is received till timeout, the CoAP message is retransmitted and the retransmission counter incremented. The endpoint also creates a ClientChannel while sending the message which is kept as a map with the server IP address and port as key.

Similarly in the resource directory a channel is created that is bind to a port on which the server listens. On receiving a CoAP Message on the datagram channel, server checks

whether the message is either a CoAP request or response. In case the message is of a Request type, the Server creates a Map which holds the clients IP address and port as key and a new ServerChannel is created in case it does not exist. The server channel holds info regarding the socket, endpoint devices ip and port. This ServerChannel Object is attached to the CoAP message so that the response can be sent back to the endpoint using its information (ip and port) stored in the ServerChannel. Next this message is parsed according to CoAP format and depending upon the method (GET,PUT,POST,DELETE) and other CoAP Header options and payload, necessary APIs as detailed below (Section 6.3) are called. After parsing the message, an appropriate response is generated and sent to the endpoint using the serverchannel associated with the CoAP message. When the endpoint receives back the response, it checks the map and retrieves an existing clientChannel created. On receiving the response, necessary operation may be performed

The client uses the Copper Cu interface to view the resources registered on the resource directory as well as use post,get method to submit other queries on a resource.

6.3 IMPLEMENTATION OF SERVICES

6.3.1 REGISTRATION

The sensor device has been associated with a sensor context that contains basic information about the sensor for example sensor type, sensor name, version, manufacturer etc which are initially registered to the server using post restful service.

This type of request is essentially a CON confirmable request i.e this type of request expects a acknowledgment from the server. In case a request is not received and the retransmission timeout expires the request packet is sent again and the client tries in total for four times.

If the server receives the register request, it creates a resource at a given path having a unique uri which is returned back to the server along with the ACK at the location path option.The response code is 201.

Now the client initially keeps all open request in a hashmap with the message id as key and on receiving a response with the same message id endpoint stores the resource path provided in the Location-path option.

This location path is used for sending further request for the same sensor .

API at endpoint to register sensor as resource:

```
public void sendRegisterResourceRequest(String data,String name){
this.resourcename=name;
CoapRequest coapRequest=clientChannel.createRequest(true,CoapRequestCode.POST);
coapRequest.setContentType(CoapMediaType.text_plain);
coapRequest.setPayload(data);
coapRequest.setMessageID(counter++);
System.out.println("Sent Request");
clientChannel.sendMessage(coapRequest);
messageID2Resource.put(String.valueOf(coapRequest.getMessageID()),resourcename;
}
```

API at ResourceDirectory to register the sensor as resource:

```
public void onRequest(CoapServerChannel channel, CoapRequest request) {
CoapMessage response = null;
CoapRequestCode requestCode = request.getRequestCode();
String targetPath = request.getUriPath();
CoapResource resource=null;
if(targetPath!=null){
resource = (CoapResource) readResource(targetPath);
}
switch (requestCode) {
case POST:
//create resource
if(resource == null) {
/* if the resource does not exist, a new resource will be created */
resource=createResourceObject(request,channel.getRemoteAddress(),channel.getRemote
Port());
createResource(resource);
response = channel.createResponse(request, CoapResponseCode.Created_201);
response.setPath(resource.getPath());
resource.setResourceIp(request.getChannel().getRemoteAddress());
resource.setPort(request.getChannel().getRemotePort());
}
}
channel.sendMessage(response);
}
```

6.3.2 SEND UPDATES

In case an event occurs on a particular sensor, for example in case of proximity sensor an event is generated when the phone is placed near to the body. Such events along with associated data need to be continualy sent to the server where they are stored in rdf format.

The client sends these data as a CON message with the uri path set as the location path received at the time of registration.

API at endpoint to update sensor observation:

```
public void handleSensorChanged(SensorEvent event ,TextView ProximityReading,
String name) {
if(event.sensor.getType()==Sensor.TYPE_PROXIMITY)
{
CoapRequest coapRequest=clientChannel.createRequest(true,CoapRequestCode.PUT);
coapRequest.setUriPath(resource2Path.get(name));
coapRequest.setContentType(CoapMediaType.text_plain);
ProximityReading.append("ProximitySensorReading:"+String.valueOf(event.values[0]));
coapRequest.setPayload("v=" +event.values[0]);
clientChannel.sendMessage(coapRequest);
System.out.println("Sent Proximity Update");
}
}
```

API at resource directory to update sensor observation:

```
public void onRequest(CoapServerChannel channel, CoapRequest request) {
CoapMessage response = null;
CoapResource resource=null;
CoapRequestCode requestCode = request.getRequestCode();
String targetPath = request.getUriPath();
if(targetPath!=null){
resource = (CoapResource) readResource(targetPath);
}
switch (requestCode) {
case PUT:
//update the resource
if (resource != null){
parser.parsePayload(request,(BasicCoapResource)resource);
updateResource(resource);
```

```
response = channel.createResponse(request, CoapResponseCode.Changed_204);
resource.changed();
}
break;
channel.sendMessage(response);
}
```

6.3.3 GET ALL RESOURCES REGISTERED

On receiving a get request at the well known interface the resource directory provides information of all the resources registered on the well known interface.

API at resource directory to get sensor data:

```
public void onRequest(CoapServerChannel channel, CoapRequest request) {
CoapMessage response = null;
CoapRequestCode requestCode = request.getRequestCode();
String targetPath = request.getUriPath();
CoapResource resource=null;
if(targetPath!=null){
resource = (CoapResource) readResource(targetPath);
}
switch (requestCode) {
case GET:
// URI queries
Vector<String> uriQueries = request.getUriQuery();
finalbyte[] responseValue;
if (uriQueries != null) {
responseValue = resource.getValue(uriQueries);
} else {
responseValue = resource.getValue();
}
response = channel.createResponse(request, CoapResponseCode.Content_205,
resource.getCoapMediaType());
response.setPayload(responseValue);
if (request.getObserveOption() != null){
//client wants to observe this resource
if (resource.addObserver(request)){
// successfully added observer
response.setObserveOption(resource.getObserveSequenceNumber());
}
```

```
}
channel.sendMessage(response);
}
```

6.3.4 DELETE A RESOURCE

When a resource is to be deleted it must be removed from the resource directory from the well known interface as well as it should be removed from groups of which it was a member.

API at resource directory to delete sensor data:

```
public void onRequest(CoapServerChannel channel, CoapRequest request) {
CoapMessage response = null;
CoapRequestCode requestCode = request.getRequestCode();
String targetPath = request.getUriPath();
CoapResource resource=null;
if(targetPath!=null){
resource = (CoapResource) readResource(targetPath);
}
switch (requestCode) {
case DELETE:
List<MulticastGroup>  groups=((BasicCoapResource)resource).getGroups();
for(MulticastGroup group:  groups){
group.getMembers().remove((BasicCoapResource)resource);
}
deleteResource(targetPath);
response = channel.createResponse(request, CoapResponseCode.Deleted_202);
break;
}
channel.sendMessage(response);
}
```

6.4 GROUP MANAGEMENT SERVICES

As mentioned earlier, all the group related requests are handled by a proxy resource GroupProxy Resource.

6.4.1 CREATE A GROUP

To create a group a request is sent to the resource directory with the uri set to coap-grp so that the request is handled by groupproxy resource on the server. This is a post request with the payload of the form “c=groupname”, here c indicates that a group that needs to be created. Once the client receives the ACK back , it can then join the group.

API at endpoint to create a group:

```
public void sendCreateGroupMessageToServer(String payload){
CoapRequestcoapRequest=clientChannel.createRequest(true,CoapRequestCode.POST);
coapRequest.setContentType(CoapMediaType.text_plain);
coapRequest.setPayload(payload);
coapRequest.setUriPath("coap-grp");
clientChannel.sendMessage(coapRequest);
messageID2Resource.put(String.valueOf(coapRequest.getMessageID()),resourcename);
}
```

On receiving a request to create a group, the Group Proxy resource parses the payload to check if it contains, payload of the form c=groupname. On finding such pattern the GroupProxy Resource creates a new multicast group and sends a response of 201 for group creation.

API at resource directory to create a group:

```
public void onRequest(CoapServerChannel channel, CoapRequest request) {
CoapMessage response = null;
CoapRequestCode requestCode = request.getRequestCode();
String targetPath = request.getUriPath();
CoapResource resource=null;
if(targetPath!=null){
resource = (CoapResource) readResource(targetPath);
}
switch (requestCode) {
case POST:
if (resource != null){
if(resource instanceof GroupProxyResource){
String path=parser.parsePayload(request,(GroupProxyResource) resource);
response = channel.createResponse(request, CoapResponseCode.Created_201);
response.setPath(path);
}
```

```
channel.sendMessage(response);
}
```

6.4.2 JOIN A GROUP

When a client receives ACK from Group Proxy Resource of group creation, the client can then join the group. On selecting a particular group to join the, client sends a PUT request to the server to update itself as a member of the group of which it intends to join.

The join group is provided as an Option in the CoAP Header. The option specifies the group uri which it intends to join.

API at endpoint to join a group:

```
public void joingroup(String rname,String guri,String gname)  {
CoapRequest coapRequest=clientChannel.createRequest(true,CoapRequestCode.PUT);
coapRequest.setUriPath(resource2Path.get(rname));
coapRequest.setJoinGroup(guri);
coapRequest.setContentType(CoapMediaType.text_plain);
coapRequest.setMessageID(counter++);
clientChannel.sendMessage(coapRequest);
messageID2Resource.put(String.valueOf(coapRequest.getMessageID()),resourcename);
joinQueue.put(String.valueOf(coapRequest.getMessageID()), gname);
}
```

On receiving the request with Option set as join the reaource is updated as a member of the group (indicated by the group uri) already registered on the Group Proxy.

API at resource directory to create a group:

```
public void onRequest(CoapServerChannel channel, CoapRequest request) {
CoapMessage response = null;
CoapRequestCode requestCode = request.getRequestCode();
String targetPath = request.getUriPath();
CoapResource resource=null;
if(targetPath!=null){
resource = (CoapResource) readResource(targetPath);
}
switch (requestCode) {
case PUT:
//update the resource
```

```
//also join and leave group
if (resource != null){
if(request.getJoinGroup()!=null){
MulticastGroup multicastGroup=getGroup(request.getJoinGroup());
if(multicastGroup!=null){
resource.getGroups().add(multicastGroup);
multicastGroup.getMembers().add(resource);
}
}
updateResource(resource);
response = channel.createResponse(request, CoapResponseCode.Changed_204);
resource.changed();
}
break;
channel.sendMessage(response);
}
```

6.4.3 LEAVE A GROUP

A client can leave a group which it had joined earlier. On selecting a particular group to leave, the client sends a PUT request to the server to remove itself as a member of the group of which it intends to leave. The leave group is provided as an Option in the CoAP Header. The option specifies the group uri which it intends to leave.

API at endoint to leave a group:

```
public void leavegroup(String rname,String guri,String gname)  {
CoapRequest coapRequest=clientChannel.createRequest(true,CoapRequestCode.PUT);
coapRequest.setUriPath(resource2Path.get(rname));
coapRequest.setLeaveGroup(guri);
coapRequest.setContentType(CoapMediaType.text_plain);
coapRequest.setMessageID(counter++);
clientChannel.sendMessage(coapRequest);
messageID2Resource.put(String.valueOf(coapRequest.getMessageID()),resourcename;
leaveQueue.put(String.valueOf(coapRequest.getMessageID()), gname);
}
```

On receiving the request with Option set as join, the resource is updated as a member of the group (indicated by the group uri) already registered on the Group Proxy.

6.5 SPARQL IMPLEMENTATION

In order to provide a quick response back to the endpoint, instead of inserting the resource info into the virtuoso server using sparql queries, we have implemented a job which intself is a separate thread that runs and checks from a queue for pending resource creation/update methods which need to be inserted into the virtuose server. The resource info is converted to the SSN Sensor ontology schema which is then inseted into the dataspace inside virtuoso server.

Each Sensor record also includes the place related info of the sensor. The place of the sensor is retrieved using the latitude and longitude provided in the CoAP request and next Google's ReverseGeoCoding API has been used to get the location details in terms of street, countryand place.

The SPARQL has been used to perform the following operation

- Store the sensor details on resource registration using SSN ontology.
- Update the sensor triples in case an updation occurs on the resource.
- In case of a Sensor event, register the Observation linked with the referenced sensor.
- Retrieve sensor data/observation based on time/location criteria.

Provide API'such as

- **getSpecifiedSensorWithPlaceId,getSpecifiedSensorWithLatLng**: which return the Sensors belonging to a particular place
- **getNewestObservationForOneSensor**: returns the newest observation for a sensor
- **getObservationsWithTimeCriteria**: returns the observation between a given todate and fromdate
- **getSensorHistoricalData:** gives all previous Observations for a sensor

Also , a SPARQL endpoint has been provided to which other queries can be made on the virtuoso server which stores the resources information.

CHAPTER 7

RESULTS AND CONCLUSION

7.1 RESULTS

The CoAP represents the protocol of choice for the constrained networks. CoAP reduces the complexity of transferring messages and as compared to HTTP, reduces time needed to transfer a CoAP message. This is achieved by using a smaller message, a HTTP message being 1.2 times bigger than the CoAP message. Also using UDP instead of TCP hugely reduces time as well as memory complexity since TCP involves handshaking and other flow control techniques which does not fit into a constrained device having limited capabilities.

As such an implementation of CoAP in java has been implemented that enhances the existing jCoAP implementation with multiple features such as implementation of various restful services, allowing multicast communication among various sensors, use of semantic web to provide mining and persistence capability to the system.

Also, an android application has been created that allows the updates on sensor events to be passed to the resource directory which keeps all the observation taking into perspective, the location and time of the sensor device.

Also, services have been provided using SPARQL which can help us read the sensor observations collected over a period of time as well getting sensor information based on the location of sensor.

Some Snapshots of the system thus developed are presented.

ENDPOINT:

Android App serves as End Point and registers the proximity sensor. Once registered the sensor can send its update from the Updates Tab as well as perfrom group activities using Group Management Tab.

Figure 17: Connect to Resource Directory

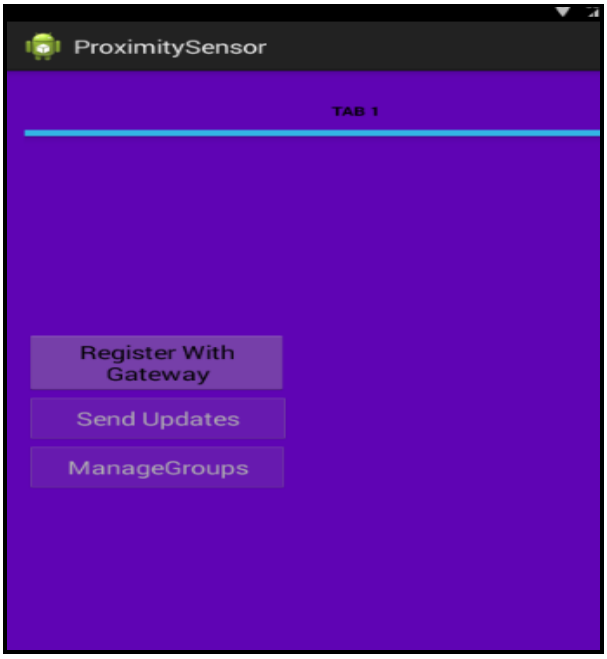

Figure 18: Initial Page that allows sensor Registration

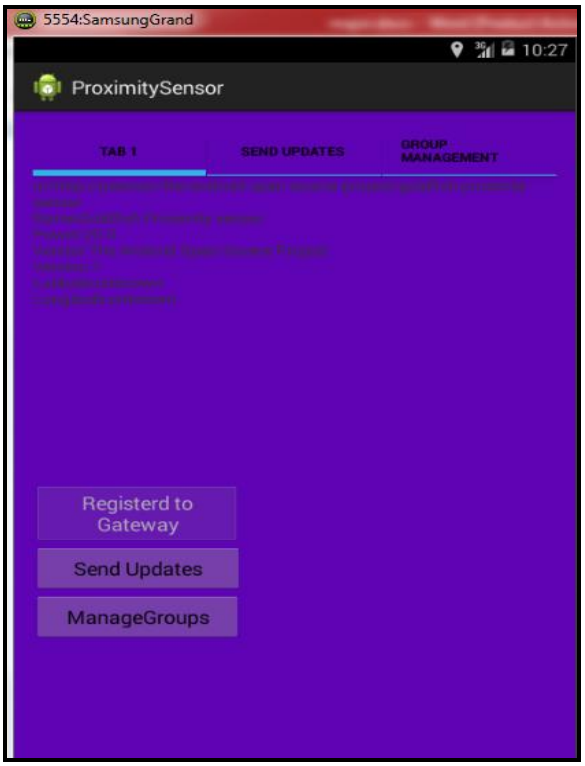

Figure 19: After Registration with the RD

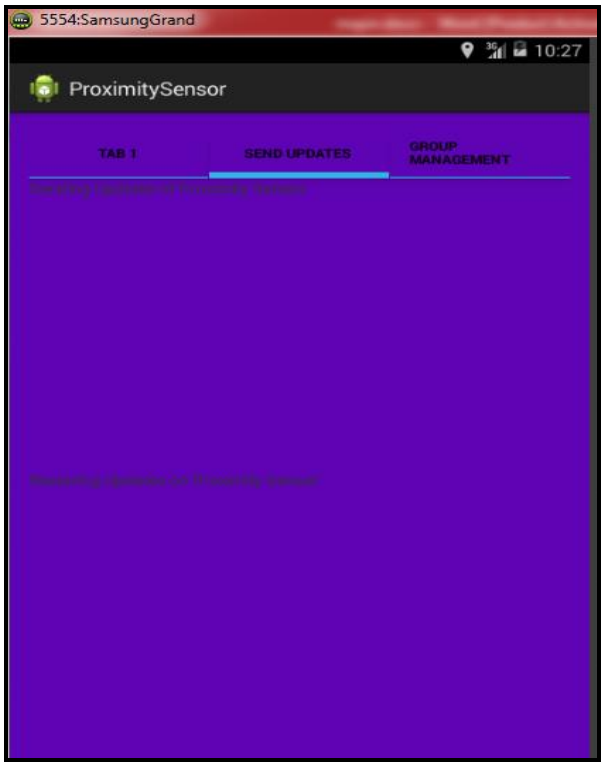

Figure 20: Send Updates on Event change as well as receive updates

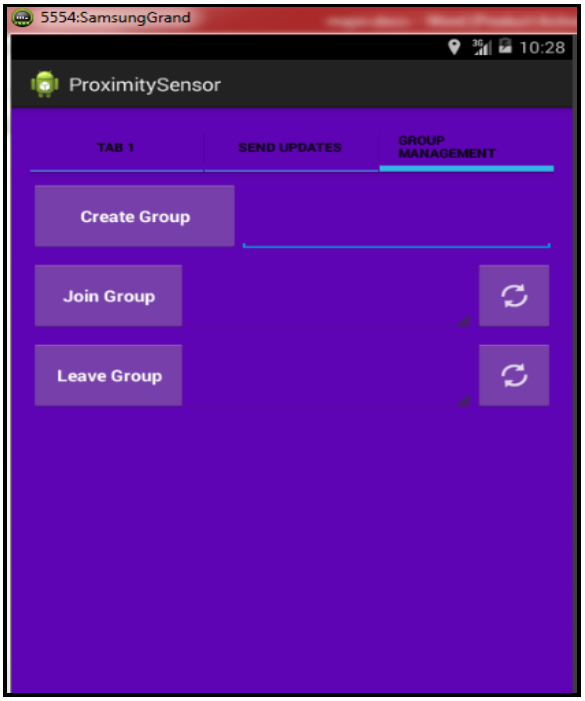

Figure 21: Sensor is provided with an option to create,join ,leave group

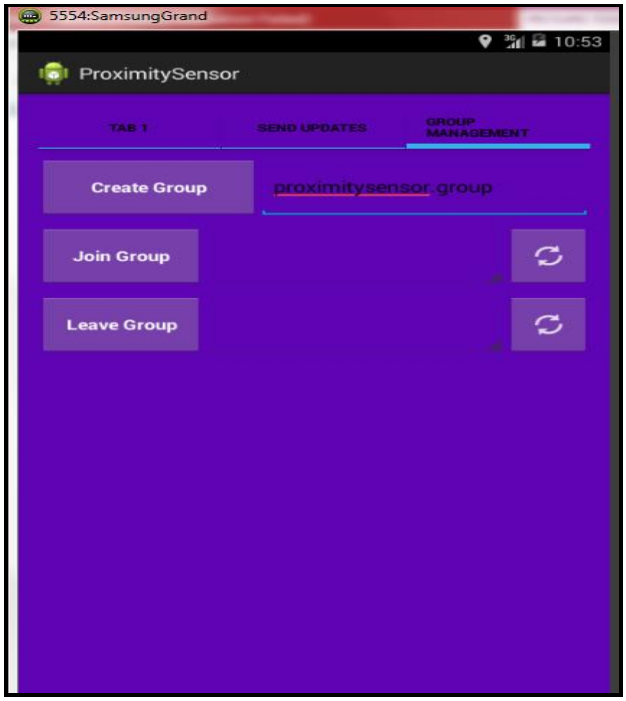

Figure 22: Sensor can create a group for ex: proximitysensor.group

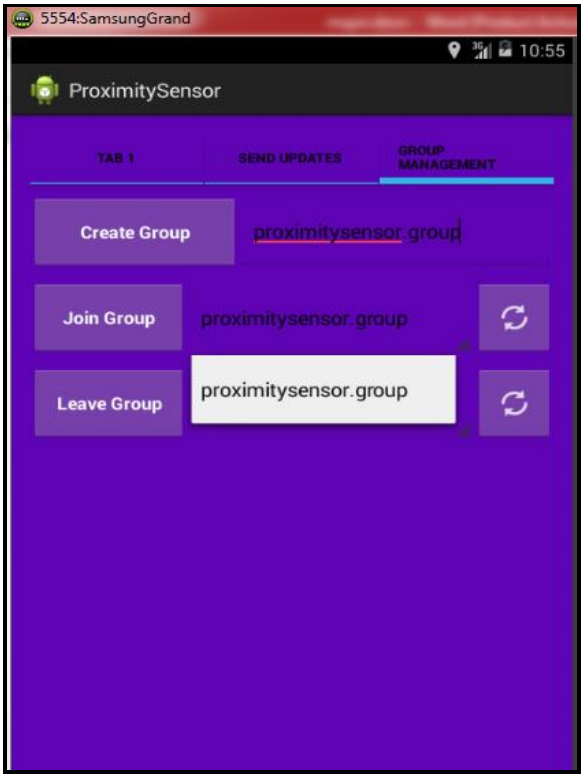

Figure 23: Sensor can join the groups that are created in the RD

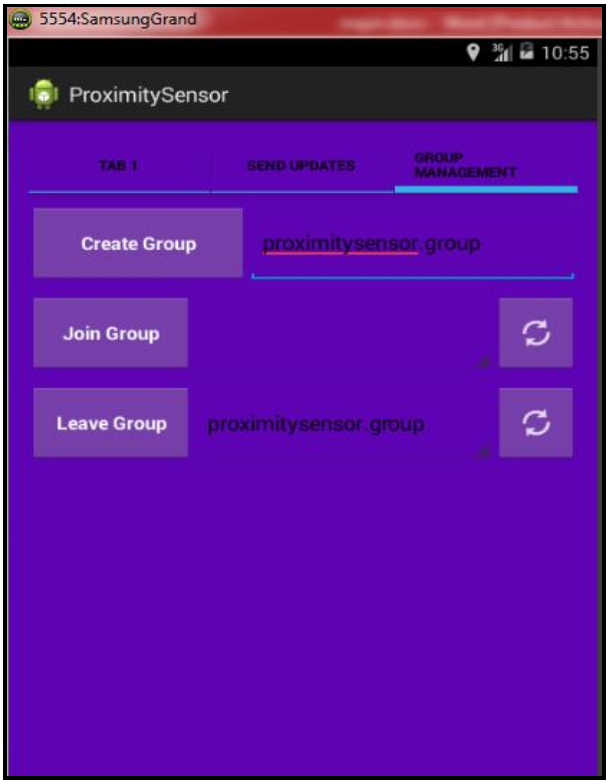

Figure 24: Sensor can leave the groups that it has joined earlier

RESOURCE DIRECTORY CLIENT:

Using Copper Agent, resources can be discovered on the resource directory and further get information about them using RESTful calls built upon the GET method.

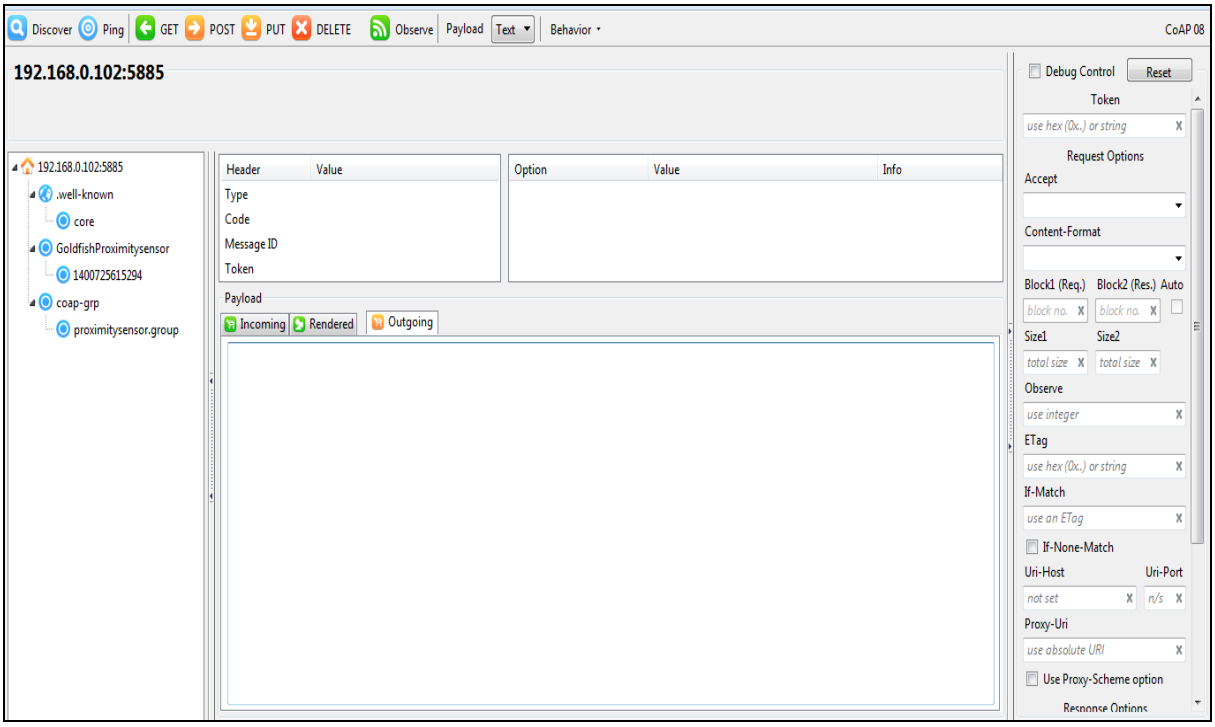

Figure 25: Copper (Cu) Interface showing the registered resources

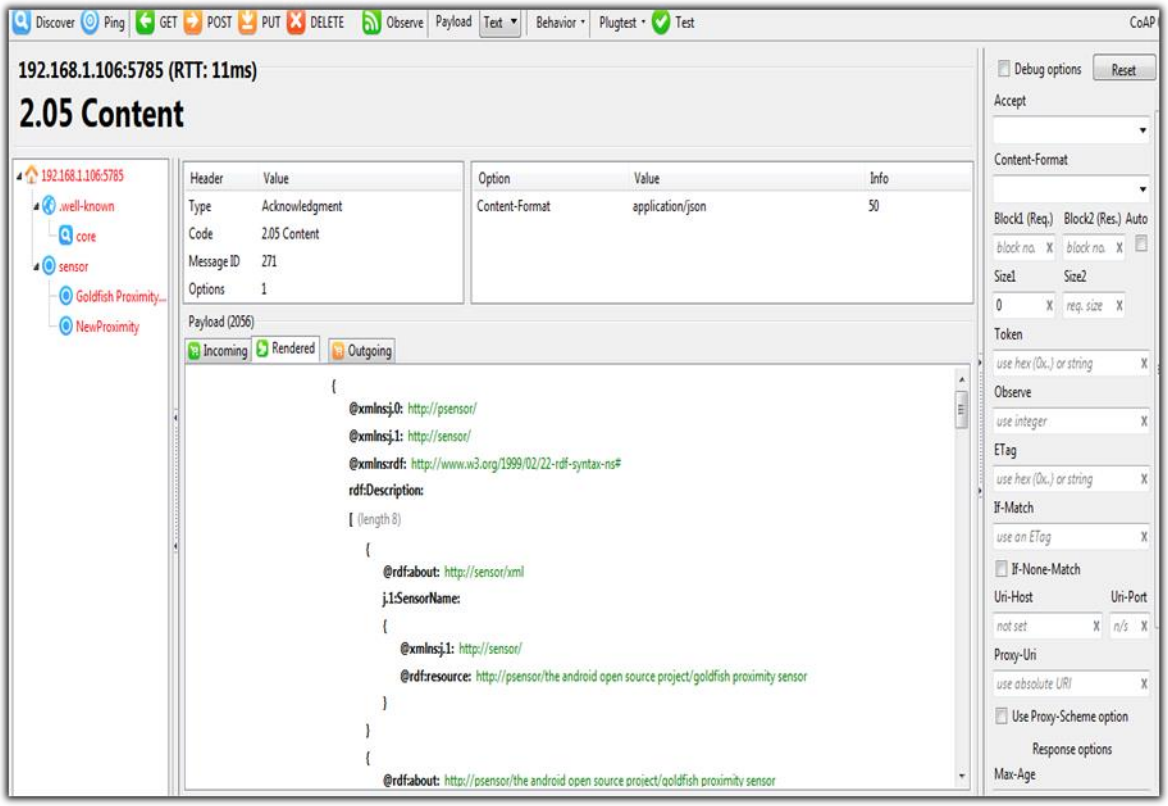

Figure 26: Copper (Cu) Interface showing a resource description

7.2 CONCLUSION AND FUTURE WORK

An Android App is created, through which the endpoint (android device) connects to the resource directory and registers the sensors to the resource directory. Also on further events, the updates are submitted to the resource directory. These events are posted to the clients which have issued a observe request on these resources. The feature of multicast support has been added which is used to manage the resources. Resources belonging to the group can also leave, join new group and update group details.

The resources description is stored into the triple database using semantic web framework Jena over fuseki .Queries can be issued over the fuseki sparql endpoint to retrieve data.

Copper Cu interface is used to query the resource directory and can be seen on the Cu interface. Further updates can also be observed on the Cu interface if the observe option is set.

This application can be used to monitor sensors running on android devices and provides interface to mine information from the dataset of information published by the sensors. Also, based upon the commands posted from the client to the resource directory which are then passed to the endpoint android device, the actuators in these android devices perform the specified operation.

FUTURE WORK

- Currently, we have created the application managing few sensor devices. We aim to extend this on more smart devices especially in smart metering. This would also involve creating ontologies for different type of sensors and actuators.
- Improving scalability of the system when connected to numerous devices by using various load balancing techniques needs to be taken care.
- Using a modified version of Datagram Transport Layer Security (DTLS) that is suited for constrained devices.

BIBLIOGRAPHY:

[1] Z. Shelby, B. Frank, D. Sturek, “Constrained Application Protocol (CoAP), Internet-Draft , draft-ietf-core-coap-07”, available online: http://tools.ietf.org/html/draft-ietf-core-coap-07, Jul. 2011.

[2] Z. Shelby, K. Hartke, C. Bormann, B. Frank, “Constrained Application Protocol (CoAP), draft-ietf-core-coap-13”, available online: http://tools.ietf.org/html/draft-ietf-core-coap-13, Jun 2013.

[3] T. Berners-Lee, J. Hendler, O. Lassila, “The Semantic Web”, Scientific American, May 2001.

[4] S. McIlraith, T. Son, H. Zeng, “Semantic Web Services”, IEEE Intelligent Systems, vol. 16, 2001.

[5] A. Bormann, A. Castellani, Z. Shelby, “CoAP: An application protocol for billions of tiny internet nodes”, Internet Computing, IEEE, vol. 16, 2012.

[6] M. Ruta, F. Scioscia, G. Loseto, F. Gramegna, A. Pinto, S. Ieva, “A logic-based CoAP extension for resource discovery in semantic sensor networks”, In: Fifth International Workshop on Semantic Sensor Networks, 2012.

[7] B.C Villaverde, D. Pesch, R. Alberola, S. Fedor, M. Boubekeur, "Constrained Application Protocol for Low Power Embedded Networks: A Survey", In: Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), Sixth International Conference, 2012.

[8] F. Gramegna, S. Ieva, G. Loseto, A.D Pinto, “Semantic-enhanced resource discovery for CoAP-based sensor networks”, In: Fifth International Workshop on Semantic Sensor Networks, 2012.

[9] M. Kovatsch, “Human-CoAP Interaction with Copper”, In: Pro-ceedings of the 7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), 2011.

[10] P. Barnaghi, S. Meissner, M. Presser, K. Moessner, “Sense and Sensability: Semantic Data Modelling for Sensor Networks”, Centre for Communication Systems Research (CCSR).

[11] M. Laine, “RESTful Web Services for the Internet of Things”, Aalto University School of Science, 2011.

[12] H.A Khattak, M. Ruta, E. DiSciascio, “CoAP-based healthcare sensor networks: A survey”, In: 11th International Bhurban Conference, 2014.

[13] N. Bressan, L. Bazzaco, N. Bui, P. Casari, L. Vangelista, M. Zorzi, “The Deployment of a Smart Monitoring System Using Wireless Sensor and Actuator Networks”, In: IEEE Smart Grid Communication, Gaithersburg, WA, USA, Oct. 2010.

[14] K. Konstantinos, K. Artem, “Semantic Interoperability on the Web of Things: The Semantic Smart Gateway”, In: Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2012..

[15] W. Wang, P. Barnaghi, G. Cassar, F. Ganz, P. Navaratnam, “Semantic sensor service networks”, Centre for Communication Systems Research, University of Surrey UK, 2012.

[16] R. Fielding, "Representational State Transfer (REST), Architectural Styles and the Design of Network-based Software Architectures”, University of California, Irvine, 2000.

[17] K. Kuladinithi , O. Bergmann, T. Potsch, M. Becker, C. Gorg, “Implementation of CoAP and its Application in Transport Logistics”, In :Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks, 2011.

[18] M. D'Aquin, A. Nikolov, E. Motta, “Building SPARQL-Enabled Applications with Android Devices”, In: 10th International Semantic Web Conference , ISWC 2011.

[19] Jena: A Semantic Web Framework for Java, http://jena.sourceforge.net/.

[20] D. Russomanno, C. Kothari, O. Thomas, “Sensor ontologies: from shallow to deep models”, In: Proceedings of the Thirty-Seventh Southeastern Symposium on, pp. 107, March 2005.

[21] M. Kovatsch, S. Duquennoy, A. Dunkels, “A Low-Power CoAP for Contiki”, In: Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems, MASS 2011.

[22] L. Christian, L. Nico, G. Frank, T. Dirk , “Connecting the Web with the Web of Things:Lessons Learned From Implementing a CoAP-HTTP Proxy”, University of Rostock, 2013.

[23] K. Taylor K, A. Ayyagari, D. Roure, “Demonstration: A RESTful SOS Proxy for Linked Sensor Data”, In : Proceedings of the 4th International Workshop on Semantic Sensor Networks, IWSSN 2011.

[24] G. Golatowski, D. Timmermann, “A Lightweight SOAP over CoAP Transport Binding for Resource Constraint Networks”, In: Eighth IEEE International Conference on Mobile Ad-Hoc and Sensor Systems, 2011.

[25] R. Chander, S. Elias, S. Shivashankar, “A REST Based Design for Web of Things In Smart Environments”, In : 2nd IEEE International Conference on Parallel, Distributed and Grid Computing, 2012.

[26] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, V. Terziyan, “Smart semantic middleware for the Internet of Things”, In: Fifth International Conference on Informatics in Control, Automation and Robotics, 2008.

[27] M. Iqbal, H. Lim, W. Wang, Y. Yao, “A Service-Oriented Model for Semantics-based Data Management in Wireless Sensor Networks”, In: International Conference on Advanced Information Networking and Applications Workshops, 2009.