A THESIS
FOR THE DEGREE OF MASTER OF PHILOSOPHY

A Study of CoAP Extension Based on ID for IoT Node Registration and Message Queuing
(IoT 노드 등록 및 메시지 큐잉을 위한 ID 기반 확장된 CoAP 프로토콜 연구)

Wenquan Jin

Department of Computer Engineering

GRADUATE SCHOOL

JEJU NATIONAL UNIVERSITY

February 2015

A Study of CoAP Extension Based on ID for IoT Node Registration and Message Queuing Wenquan Jin 2015

A THESIS
FOR THE DEGREE OF MASTER OF PHILOSOPHY

A Study of CoAP Extension Based on ID for IoT Node Registration and Message Queuing
(IoT 노드 등록 및 메시지 큐잉을 위한 ID 기반 확장된 CoAP 프로토콜 연구)

Wenquan Jin

Department of Computer Engineering

GRADUATE SCHOOL

JEJU NATIONAL UNIVERSITY

February 2015

A Study of CoAP Extension Based on ID for IoT Node Registration and Message Queuing
(IoT 노드 등록 및 메시지 큐잉을 위한 ID 기반 확장된 CoAP 프로토콜 연구)

Wenquan Jin
(Supervised by professor Do-Hyeun Kim)

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Computer Engineering

2015. 02.

This thesis has been examined and approved.

Department of Computer Engineering
GRADUATE SCHOOL
JEJU NATIONAL UNIVERSITY

Acknowledgements

First of all, I offer my humble thanks God for giving me strength to finish my master studies successfully. The LORD is my shepherd, I shall not be in want, He taught and emphasized the importance of learning and seeking knowledge.

I am extremely grateful to my advisor Prof. Do-Hyun Kim for his guidance and support during my studies in Jeju National University. Without his help, constant interaction, helpful discussions and keen interest, this thesis would not have been possible.

I wish to offer my humble gratitude to Prof. Wang-Cheol Song and Prof. Khi-Jung Ahn for their useful suggestions and extremely important comments during the process of my thesis evaluation. I would like to express my sincere thanks to Prof. Sang-Joon Lee and all the professors of my department for their king help and extended support in many ways to fulfill my course work successfully. I also want to extend my thanks to my friends, lab members and department secretaries for their help and cooperation.

Acronyms

API	Application Programming Interface
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environment
CRUD	Create, Retrieve, Update and Delete
GIS	Geographic Information System
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
M2M	Machine to Machine
MQ	Message Queue
RD	Resource Directory
REST	Representational State Transfer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier

Abstract

Internet Engineering Task Force (IETF) presented Constrained Application Protocol (CoAP) for the communication between sensor or actuator nodes by in a constrained environment such as small amount of memory and low power. CoAP and HTTP protocol can convert easily, and can use to monitor or control the infrastructure utility through low-power sensor and actuator networks in Internet of Things (IoT) and Machine-to-Machine (M2M) environment. In this thesis, we present CoAP extension based on ID for IoT service. The CoAP node is used for constrained environment which works in constrained network using CoAP, and it can be configured one or more units such as sensors and actuators. In order to support IoT service for this kind of CoAP node, we design and implement improved middleware and composite CoAP node, these elements interact with each other via CoAP in the IoT. The middleware includes Resource Directory (RD) and Message Queue (MQ) broker to interact with CoAP node for node registration and discovery, sleep scheduling and context data collecting. Through the interaction of the middleware and the CoAP node, we also propose an efficient sleepy and context data collecting mechanism building on RD and MQ functionalities.

요약문

인터넷 엔지니어링 태스크 포스 (IETF) 에서는 작은 메모리, 적은 에어지공급과 같은 제한된 자원을 가진 환경에서 작동하는 선서, 구동체 등 노드의 통신을 위한 Constrained Application Protocol (CoAP) 을 제안했다. CoAP 프로토콜은 HTTP 프로토콜과 서로 쉽게 변환 할 수 있고 사물인터넷 (IoT) 혹은 사물통신 (M2M) 환경에서의 저출력 센서 혹은 구동체 네트웍을 통하여 인프라 유틸리티를 모니터링 혹은 제어 할 수 있다. 본 논문에서 사물 인터넷 서비스를 위하여 ID 기반의 CoAP 프로토콜의 확장 기능을 제안한다. CoAP 노드는 CoAP 프로토콜을 통하여 제한된 자원을 갖는 환경에서 제한된 네트웍을 위하여 사용되며 이는 한개 혹은 여러개의 유닛을 포함 할 수 있다. 이런 복합노드에 관한 사물 인터넷 서비스를 제공하기 위하여 본 논문에서는 향상된 미들웨어와 복합 CoAP 노드를 설계하고 구현하여 CoAP 프로토콜을 통하여 사물인터넷상에서의 상호작용을 보여준다. 리소드 목록 (RD) 와 메시지 큐잉 (MQ) 브로커를 포함한 미들웨어는 CoAP 노드와의 상호작용을 통하여 노드 등록, 조회, 노드 수면 기능과 환경 데이터수집 등 기능을 갖고 있다. 미들웨어와 노드간의 상호작용을 통하여 우리는 RD 와 MQ 를 이용하여 효율적인 노드 수면 기능과 환경 데이터수집 기능을 제안한다.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

In the Internet of Things (IoT) paradigm, many of physical and virtual things are connected with each other through the internet to be a network infrastructure. The basic idea of this paradigm is the pervasive presence around us of a variety of things which enables the interconnections between people and machines, automobiles, mobile phones, sensors, actuators and computational elements. Unquestionably, the concept of IoT will have several aspects of everyday-life and behavior of potential users. For private users, the IoT will be visible in both working and domestic fields and also will be visible in business field such as industrial manufacturing, process management and automation [1]. All these things of internet had been installed to run their own software. Therefore, all types of real-world physical elements are able to interact with each other via own software in the IoT. Through the interaction of these elements in the environment around us, we can collect information and influence context from the environment. Accordingly, the IoT concept extends the Machined to Machine (M2M) communications concept via interaction with physical systems [2]. Machines exchange information and perform actions without any human interaction.

IETF (Internet Engineering Task Force) CoRE (Constrained RESTful environments) Working Group started global standardization for Constrained Application Protocol (CoAP) in 2010, and recently it announces RFC (Request for Comments) 7252. Therefore researchers have been studying and developing for realize CoAP in limited sensor or actuator network. CoAP is a specialized web transfer protocol for use with constrained nodes and constrained networks [3]. The protocol is designed for M2M applications such as smart energy and building automation. M2M interactions typically result in a CoAP implementation acting in both client and server roles. CoAP proxies which can cache and service requests for sleepy CoAP servers [4]. A client explicitly sends a CoAP request

(GET) to a forward proxy (identified by its IP address) while indicating the URI (of the resource of interest) associated to a sleepy CoAP origin server. If the proxy has a valid representation of the resource in its cache it can then respond directly to the client regardless of the current sleep state of the origin server. Otherwise the proxy has to attempt to retrieve (GET) the resource from the sleepy origin server. The attempt may or may not be successful depending on the sleep state of the origin server. [RFC6690] and [I-D.ietf-core-resource-directory] defines a Resource Directory (RD) mechanism where sleepy CoAP servers can register/update (POST/PUT to "/.well-known/core") their list of resources on a central (non-sleepy) RD server. This allows clients to discover the list of resources from the RD (GET /rd-lookup/...) for a sleepy server, regardless of its current sleep state. Unlike a proxy, the RD stores only the URIs for other nodes, and not the actual resource representation [RFC6690]. The client then may attempt to operate on (GET/PUT/POST/DELETE) the desired resource at the sleepy origin server. This attempt may or may not be successful depending on the sleep state of the origin server [5]. These objects in the M2M scenario, witch work in constrained environment and working alone without human in long time. It need efficient energy usage for being touched infrequently by administration [6]. The object may be a sensor or actuator, which has program works on the machine by processors. The machine is like a pc support interfaces to extend features. For example, there is a board which is Galileo board [7]. Using the Galileo board, we can configure several sensors and actuators in a board, and these units would be a part of the board.

1.2 Content of research

This thesis presents a conceptual architecture and design features of multiple unit IDs in a node for registration and discovery. The concept of node ID has been presented previously in [10]. We present the idea of nodes having multiple integrated sensing and/or actuating devices. Each of these devices is separately identifiable via a unit ID. The unit ID for a given resource must be unique among all the integrated resources in a single node while the same ID can represent a resource integrated in another node [11]. The integrated resources inside a node are separately identified by node IP and unit ID together. Every node has an IP address through which it can communicate with clients or other modules of the system. A detailed description of the purpose and features of Resource Directory has been presented [8]. We design and implement an improved IoT middleware to include Resource Directory (RD) and Message Queue (MQ) broker which interacts with IoT node for IoT node registration and discovery, sleep scheduling and context data collecting. IoT middleware and IoT node communicate via CoAP. Therefore, IoT middleware includes HTTP and CoAP both protocol library for supporting HTTP based RESTful API and CoAP based RESTful API. Using the system which we present, we test the performance of the message interaction for sleepy schemes.

1.3 Outline

The outline of this thesis is organized as follows: Chapter 1 describes the Background of the IoT elements and pervasive technologies. Chapter 2 discusses the IoT technologies in the constrained environment. And chapter 3 proposes unit ID based IoT architecture and RD and MQ based enhanced mechanisms. Chapter 4 presents design of IoT elements and mechanism for the proposed scenario. Chapter 5 presents implementation of the proposed scenario. In the chapter 6, we evaluate the performance of the system which are we present and chapter 7 summarizes the contents of the thesis.

2 Related work

2.1 CoAP

IETF CoRE (Constrained RESTful Environments) Working Group develop a framework for resource-oriented applications intended to run on constrained IP networks which support a RESTful architectural style and POST, GET, PUT, DELETE (so-called CRUD methods). It is stateless and exposes directory structure-like URIs and define mappings to compact binary forms and transport over UDP.

CoRE is providing a framework for resource-oriented applications intended to run on constrained IP networks. A constrained IP network has limited packet sizes, may exhibit a high degree of packet loss, and may have a substantial number of devices that may be powered off at any point in time but periodically "wake up" for brief periods of time. These networks and the nodes within them are characterized by severe limits on throughput, available power, and particularly on the complexity that can be supported with limited code size and limited RAM per node. More generally, we speak of constrained networks whenever at least some of the nodes and networks involved exhibit these characteristics. Low-Power Wireless Personal Area Networks (LoWPANs) are an example of this type of network. Constrained networks can occur as part of home and building automation, energy management, and the Internet of Things.

The CoRE working group will define a framework for a limited class of applications: those that deal with the manipulation of simple resources on constrained networks. This includes applications to monitor simple sensors (e.g. temperature sensors, light switches, and power meters), to control actuators (e.g. light switches, heating controllers, and door locks), and to manage devices.

The general architecture consists of nodes on the constrained network, called Devices that are responsible for one or more Resources that may represent sensors, actuators, combinations of values or other information. Devices send messages to change and query resources on other Devices. Devices can send notifications about changed resource values to Devices that have subscribed to receive notification about changes. A Device can also publish or be queried about its resources. (Typically a single physical host on the network would have just one Device but a host might represent multiple logical Devices. The specific terminology to be used here is to be decided by the WG.) As part of the framework for building these applications, the WG defined a Constrained Application Protocol (CoAP) for the manipulation of Resources on a Device.

CoAP is designed for use between Devices on the same constrained network, between Devices and general nodes on the Internet, and between Devices on different constrained networks both joined by an internet. CoAP targets the type of operating environments defined in the ROLL and 6LOWPAN working groups which have additional constraints compared to normal IP networks, but the CoAP protocol will also operate over traditional IP networks.

CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments.

CoAP has the following main features:

1) Web protocol fulfilling M2M requirements in constrained environments

2) UDP binding with optional reliability supporting unicast and multicast requests.

3) Asynchronous message exchanges.

4) Low header overhead and parsing complexity.

5) URI and Content-type support.

6) Simple proxy and caching capabilities.

A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.

Security binding to Datagram Transport Layer Security (DTLS) [3].

```
  0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.1. Message format.

There also may be proxies that interconnect between other Internet protocols and the Devices using the CoAP protocol. The WG will define a mapping from CoAP to an HTTP REST API; this mapping will not depend on a specific application. It is worth noting that proxy does not have to occur at the boundary between the constrained network and the more general network, but can be deployed at various locations in the unconstrained network.

CoAP will support various forms of "caching“. For example, if a temperature sensor is normally asleep but wakes up every five minutes and sends the current temperature to a proxy that as subscribed, when the proxy receives a request over HTTP for that temperature resource, it can respond with the last seen value instead of trying to query the Device which is currently asleep.

2.2 CoAP extensions

In many M2M applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resources descriptions. Furthermore, new link attributes useful in conjunction with an RD are defined [8].

A Resource Directory (RD) is used as a repository for Web Links about resources hosted on other web servers, which are called endpoints (EP). An endpoint is a web server associated with an IP address and port, thus a physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain sets of Web Links (called resource directory entries), for the RD to validate entries, and for clients to lookup resources from the RD. Endpoints themselves can also act as clients. An RD can be logically segmented by the use of Domains. The domain an endpoint is associated with can be defined by the RD or configured by an outside entity. Endpoints are assumed to proactively register and maintain resource directory entries on the RD, which are soft state and need to be periodically refreshed. An endpoint is provided with interfaces to register, update and remove a resource directory entry. Furthermore, a mechanism to discover a RD using the CoRE Link Format is defined. It is also possible for an RD to proactively discover Web Links from endpoints and add them as resource directory entries, or to validate existing resource directory entries. A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

```
          Registration       Lookup
+----+        |                |
| EP |----    |                |
+----+   ---- |                |
            --|-    +------+   |
+----+        | ----|      |   |     +--------+
| EP | -------|-----|  RD  |---|-----| Client |
+----+        | ----|      |   |     +--------+
            --|-    +------+   |
+----+   ---- |                |
| EP |----    |                |
+----+
```

Figure 2.2. The resource directory architecture.

When a node registers itself to the Resource Directory server, the registration request should contain its node identifier. This Option can be used to identify the node, either the client or the server [10]. This node identifier MAY be included in the NodeId option in the registration request, or MAY be included in the URI-Query option.

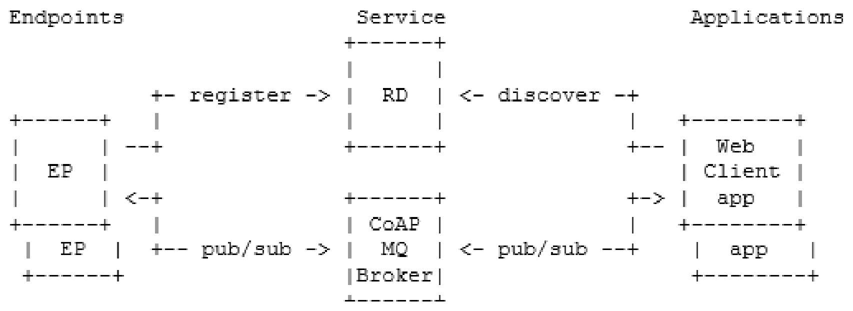


Figure 2.3. CoAP MQ architecture.

In the constrained environment, nodes with limited reachability to communicate using simple extensions to CoAP and the CoRE Resource Directory [8]. The extensions enable publish-subscribe communication using a Message Queue (MQ) broker node that enables store-and-forward messaging between two or more nodes. The MQ functionality for CoAP that extends the capabilities for supporting nodes with long breaks in connectivity and/or up-time.

The NodeId option is used to identify the node. The value SHOULD be unique for each node within a Resource Directory server. The value can be in the form of Binary bits, IMEI (International Mobile Equipment Identity number), IEEE 802 MAC Address, or other identifiers which can uniquely identify itself. Usually the value is pre-configured or pre-previsioned in the node [10]. NodeId Option specification adds a new option NodeId to CoAP. The main purpose is for a node to have a unique identity, named as NodeId. The NodeId is used by the node, as a sender, to identify itself to the recipient, during registration and communications.

Type	C	U	N	R	Name	Format	Length	Default
TBD	-	-	-	-	NodeId	string	1-255 B	(none)

Figure 2.4. NodeId option definition.

When a node registers itself to the Resource Directory server, the registration request SHOULD contain its node identifier. This node identifier MAY be included in the NodeId option in the registration request, or MAY be included in the URI-Query option [8].

This option MAY be used in a CoAP request or response. And it can be used to correlate the messages for a node in case of IP address change. As long as a node changes its IP address, the NodeId SHALL be included in the first request and response and sent in CON message. Whenever the node reboots or moves, the NodeId must not change. And the node SHOULD send the updated IP address with the NodeId to the RD server, using the update interface [8]. This informs the RD server a mapping relation between the new IP address and the NodeId identified node.

Endpoint is an entity participating in the CoAP protocol. Colloquially, an endpoint lives on a "Node", although "Host" would be more consistent with Internet standards usage, and is further identified by transport-layer multiplexing information that can include a UDP port number and a security association.

There is Endpoint ID due to the mobile nature of some devices. E.g. smartphones, they are often assigned new IP addresses because of a network change [12]. Thus, the IP address of a CoAP server might change during an ongoing conversion. The Endpoint ID scenario proposes a method to assign each communication partner with an identifier (endpoint ID) which replaces the IP address as (partial) key to relate requests and responses. Besides the common separated responses, the proposed method is also useful to handle IP address changes, e.g. during an ongoing observation or a block wise transfer.

3 Improved RD and MQ based on unit ID

The Constrained Application Protocol (CoAP) is a protocol intended towards devices which are constrained in terms of memory, processing and power i.e. small low power sensors, switches and valves etc. The CoAP allows such devices to interactively communicate over the Internet. The CoAP is a specialized web transfer protocol for constrained devices [12]. These devices typically have some combination of limited battery power, small memory footprint and low throughput links. It is expected that in CoAP networks there will be a certain portion of mode and temporarily suspend CoAP protocol communication [4]. In this chapter, we propose an interaction of the concept of composite IoT node and IoT middleware to illustrate a composited mechanism building on the IoT middleware functionality which includes Resource Directory (RD) and CoAP Message Queue (MQ) to enhance sleepy node support in CoAP networks.

As shown as figure 3.1, we present overall structure on this figure. In this structure, there are three IoT Components which includes service layer, IoT middleware and IoT node. IoT middleware links service layer and IoT nodes, It supports HTTP RESTful API to the service layer and supports CoAP RESTful API to IoT nodes in the constrained environment. We present composite CoAP node in the constrained environment in this structure. A CoAP node integrates multiple CoAP resources such as sensors and actuators. In the service layer, each service provider register to service registry to available in the internet. In the figure, App Server is a service provider which combines GIS, Sensor Web and Actuator Web to provider a new service. Service user can search IoT components from Service Registry to access through Service Provider in the service layer.

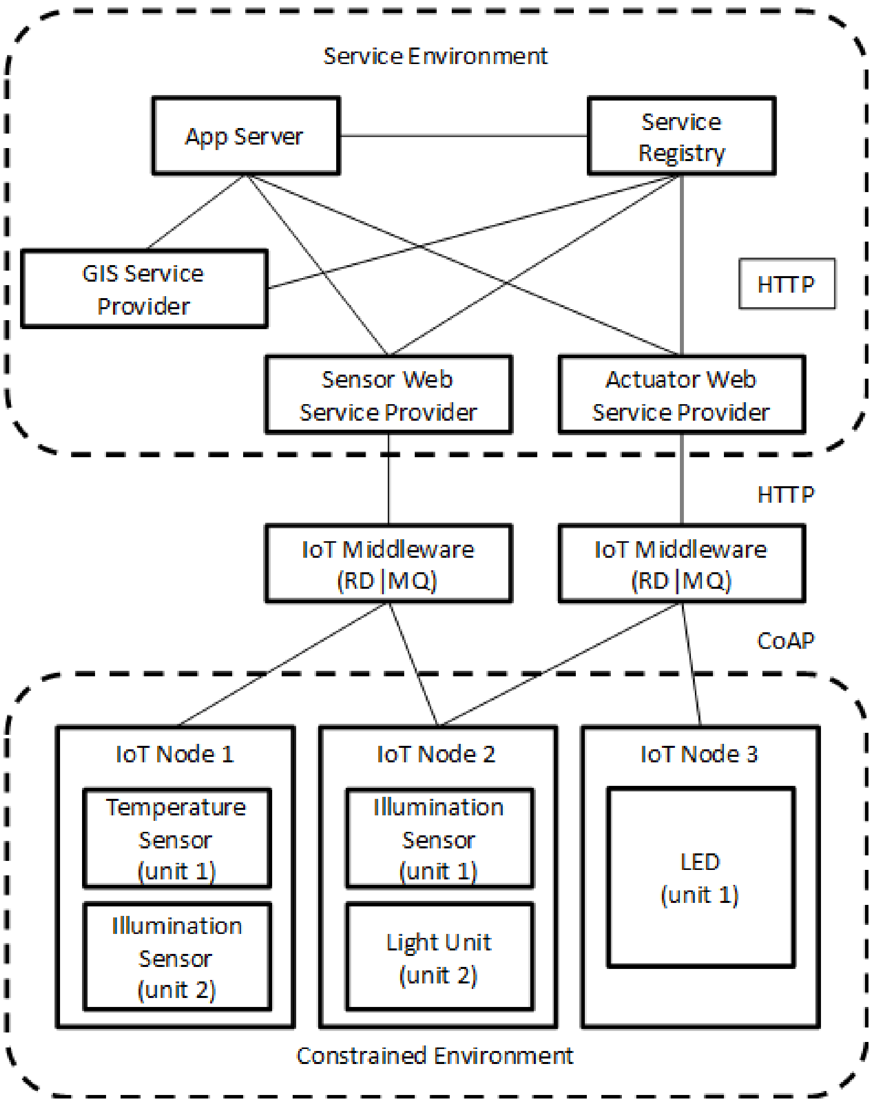


Figure 3.1. IoT system architecture based on unit ID.

3.1 IoT architecture based on unit ID

A single CoAP node integrates multiple CoAP resources to enable their own function such as sensor or actuator. The unit ID in the CoAP node, it enables the usage of composite nodes consisting of multiple sensors and actuators while having a single IP address for communication. The integrated resources can be individually or collectively communicated with and/or controlled using CoAP messages.

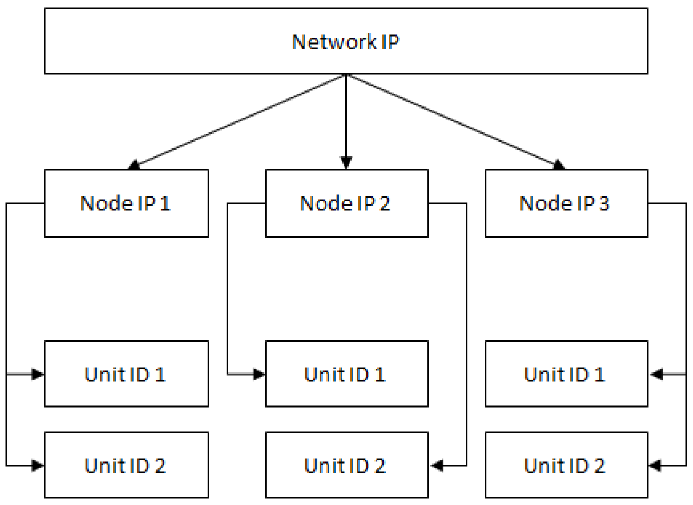


Figure 3.2. IP address and endpoint unit ID mapping architecture.

Figure 3.2 presents a generalized architecture for IP and ID mapping in the proposed Endpoint Unit ID scenario. The network IP and local IP addresses are used to access the network of the node and the physical node respectively. We proposed that a single node may have multiple integrated resources and each of these resources can be represented by multiple sub-identifiers (IDs). The sub-identifier for the integrated resource is called as the Unit ID and a node may have more than one Unit IDs.

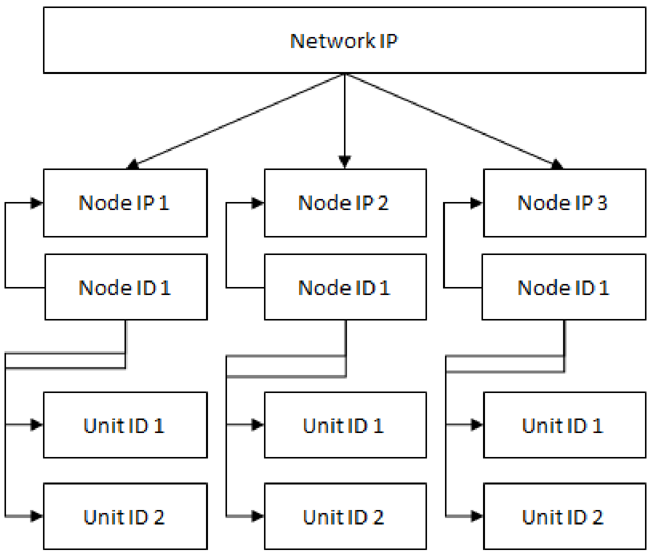
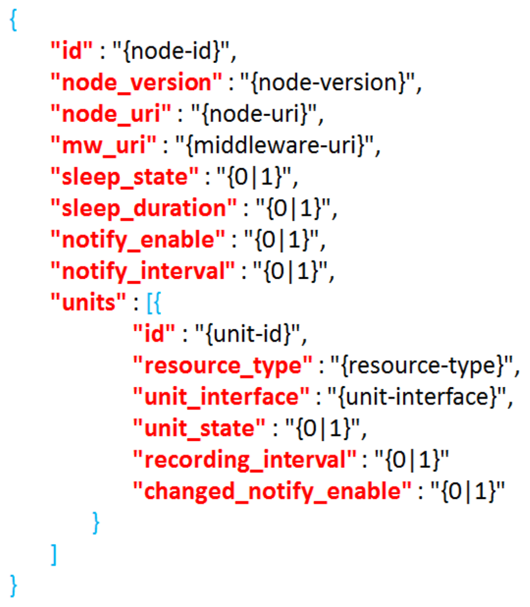


Figure 3.3. Node ID and endpoint unit ID mapping architecture.

In the mobile network, it is quite common for a node change its IP address. After the server or client changes its IP address, the peer of the other side will lose the connection. We propose an IoT node which includes node ID and unit ID for identify several resources in the IoT. Figure 3.3 shows the architecture of node ID and unit ID mapping for mobile network. The node ID is fix for identify the node. The changed IP should be retrieved through the node ID. And the resources can be controlled through the unit ID which are integrated in the node.



```
{
    "id" : "{node-id}",
    "node_version" : "{node-version}",
    "node_uri" : "{node-uri}",
    "mw_uri" : "{middleware-uri}",
    "sleep_state" : "{0|1}",
    "sleep_duration" : "{0|1}",
    "notify_enable" : "{0|1}",
    "notify_interval" : "{0|1}",
    "units" : [{
            "id" : "{unit-id}",
            "resource_type" : "{resource-type}",
            "unit_interface" : "{unit-interface}",
            "unit_state" : "{0|1}",
            "recording_interval" : "{0|1}"
            "changed_notify_enable" : "{0|1}"
        }
    ]
}
```

Figure 3.4. Example of IoT node property profile based on unit ID.

Figure 3.4 show an example for IoT node property profile in the IoT node. The information of the profile is formatted in JSON. It includes node’s information and unit’s information which are included in the IoT node application. Node version is used for synchronize the information between IoT node and IoT middleware. And another attributes is used for functionality of the IoT node such as sleep state, sleep duration, notify enable and notify interval. There can be multiple units of the node. The information of the units can be appropriately different without id of the unit. Unit’s information includes resource type and interface which are proposed in the RFC 6690 [5]. The resource type attribute is an opaque string used to assign an application-specific semantic type to a resource. And the interface description attribute is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource.

Each IoT node has own URI with IP address. But it is quite common for a node to change its IP address due to rebooting. We design the IoT node seamless synchronize its information with RD in the IoT middleware. Service Users of the IoT middleware, which retrieve Node URI via Node ID to find IoT node in the network. For example, a service provider need to send a command to actuate a unit of the IoT node, then the service provider will get a Node ID which are related the requirement. Service Provider request a command to the IoT middleware with Node ID and Unit ID which are parameters of query. And IoT middleware sends the command to the IoT node via Node URI through CoAP with unit ID, the Node URI is retrieved using node ID from database. Finally, IoT node receives the command and actuates the unit by the unit ID.

3.2 Extended CoRE RD for IoT node

Once a complete path is obtained for a register function set in the RD, the CoAP server may then register resources to the RD. The User then requests the RD to look up for registered resources. The RD then returns the access paths for the registered resources according to the request of the client. The returned resources may include simple or composite resources and the user can communicate with these resources. If a single CoAP node has multiple integrated sub devices, then the composite interaction with the resources is based on Unit-ID(s). The user can interact with individual sub units or collectively interact with all the sub units of a composite node. It is important to note that the description and discovery of resources hosted by a constrained web server is specified by the CoRE Link Format which is based on the web linking for the discovery of resources hosted by an HTTP Web Server [5][13].

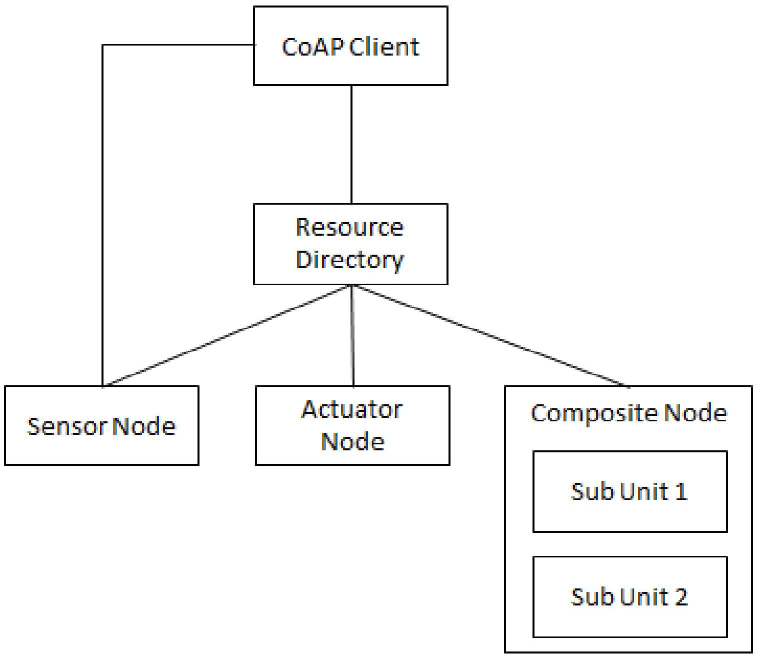


Figure 3.5. Endpoint unit ID and RD.

Figure 3.5 shows that a node may contain a single or multiple integrated resources i.e. multiple sensors, multiple actuators or sensors and actuators in a single node. The nodes register these resources with the RD in the IoT middleware. The etc. Once a node had registered all its integrated resources with the RD, the users may lookup single or multiple resources and may interact with them directly. The RD helps in the automated discovery and lookup of resources while the multi-unit IDs provide an efficient utilization of a single IP for interacting with multiple resources.

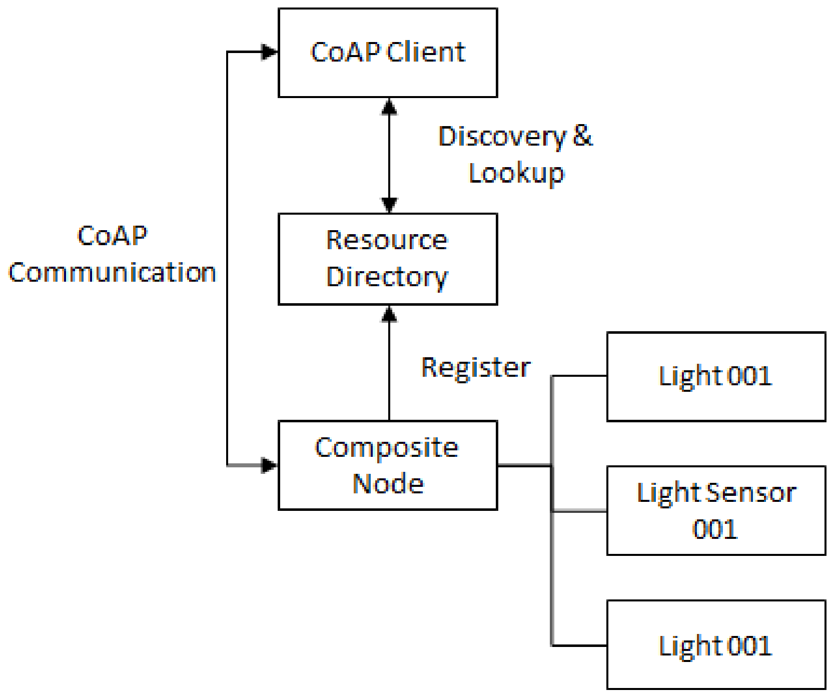


Figure 3.6. Use case of composite CoAP node based on multiple unit ID.

Figure 3.6 shows the use case scenario for a CoAP composite node which integrates a light sensor and two switches to control the lights in a room. The composite node is accessed via a single IP address assigned to it while the sub-resources of the composite node are accessed with unit IDs. The composite node like a normal CoAP Endpoint, registers its resources in the form of sub units with the RD. The RD, thus have a single IP address for the composite node and unit IDs for the sub units of the composite node.

As a part of discovering the services is offered by IoT middleware, a service provider has to learn about the IoT middleware. IoT middleware includes IoT node's information which is searched by client via service layer.

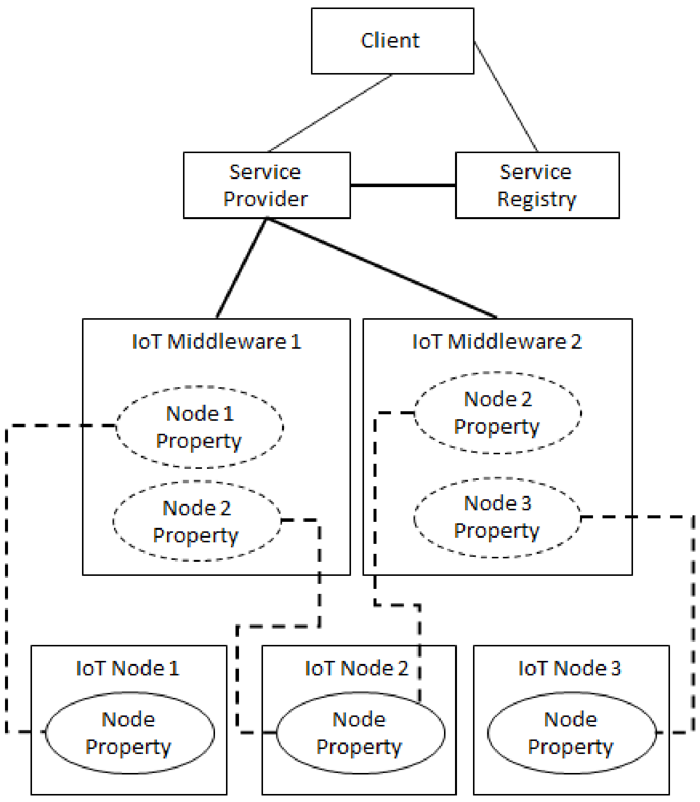


Figure 3.7. An example of IoT resource discovery architecture.

Figure 3.7 shows an example for discover an IoT resource by IoT service client. Client searches IoT node using the search service which are provided by Service Registry. Then the Service Registry retrieves service which are related, and using the service retrieves information of IoT node. The information of IoT node is registered by IoT node in the registration of IoT node. This information is synchronized in the time through the version attribute.

3.3 Sleepy mechanism based on CoAP MQ

The IoT middleware is used for in constrained networks for several reasons such as to improve performance, sleeping device scheduling. In CoAP networks there will be a certain portion of devices that are sleepy and which may occasionally go into a sleep mode and temporarily suspend CoAP protocol communication. We present a mechanism for looking up sleepy nodes through interaction with IoT middleware in the IoT. The functionality of RD and MQ are incorporated as part of the IoT middleware.

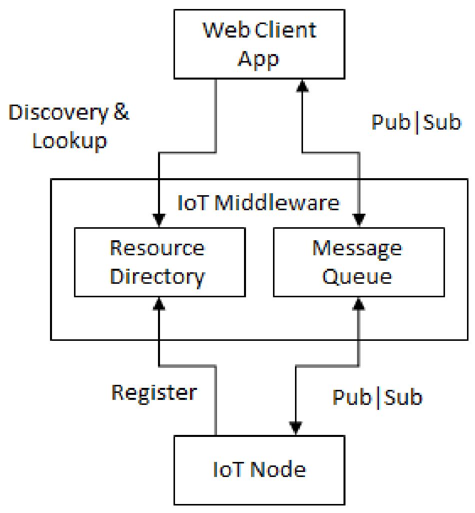


Figure 3.8. Functionality of RD and MQ in IoT middleware for sleep mode.

IoT middleware includes RD functionality to manage information of IoT node. RD supports HTTP service to Web Client APP for discovering and looking up information of IoT node which are registered by IoT node through CoAP service by RD. Functionality of MQ is used for performs store-and-forward messaging [14]. MQ enables IoT node publishes context data to the IoT middleware to be subscribed by Web Client App. In the same way, Web Client App publishes a command to the IoT middleware and forward to IoT node when the node is available.

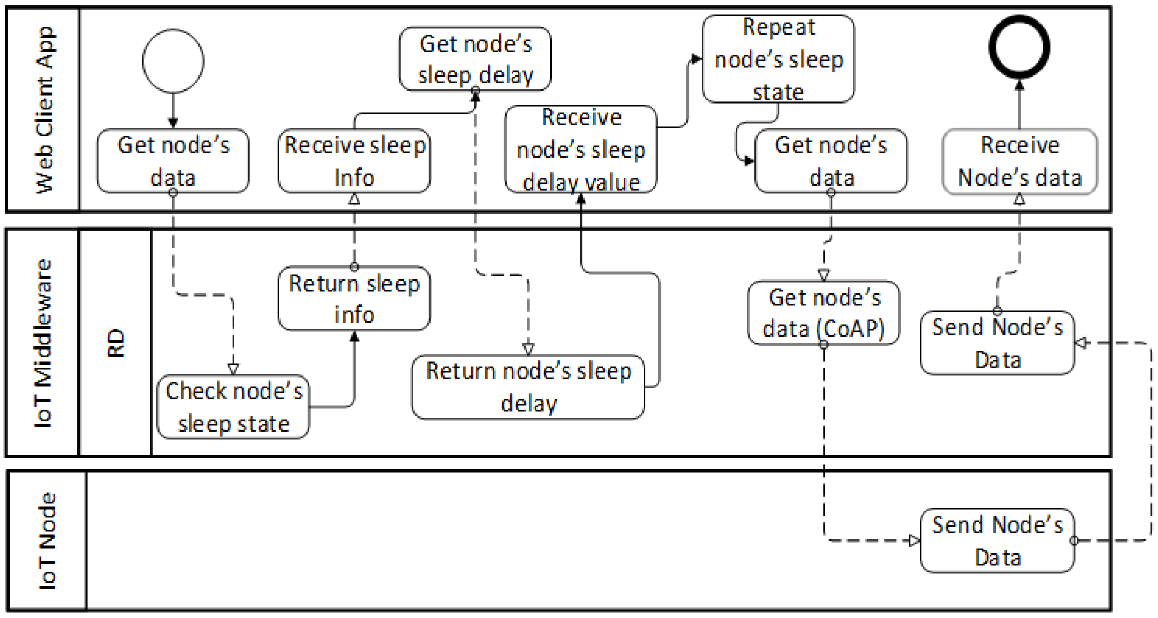


Figure 3.9. Business process model for synchronous sleepy scenario.

Figure 3.9 show three processes in the model which are Web Client App, RD and IoT node. This diagram show synchronous sleepy scenario using business process model. It shows actions work in separated process for sleepy scenario. The process begin from Web Client App part. It sends message to RD for get node's data, then RD check node's sleep state by node ID. And RD returns sleep information to Web Client App. Web Client App gets node's sleep delay data and synchronize the sleep state information with IoT middleware. When Web Client App knows node is awake, Web Client App can get the contextual data of node.

Figure 3.10 shows asynchronous Sleepy scenario via Business Process Model. It shows four processes in the model. There MQ part of the model which is used for save the command and forward to IoT node when it is awake from sleep. A part of the model is different with figure 3.8. Web Client App confirms sleep information of node from IoT middleware and subscribe node's contextual data. After IoT middleware receives the request, it will wait node to be awake and get node's contextual data to publish to Web Client App.

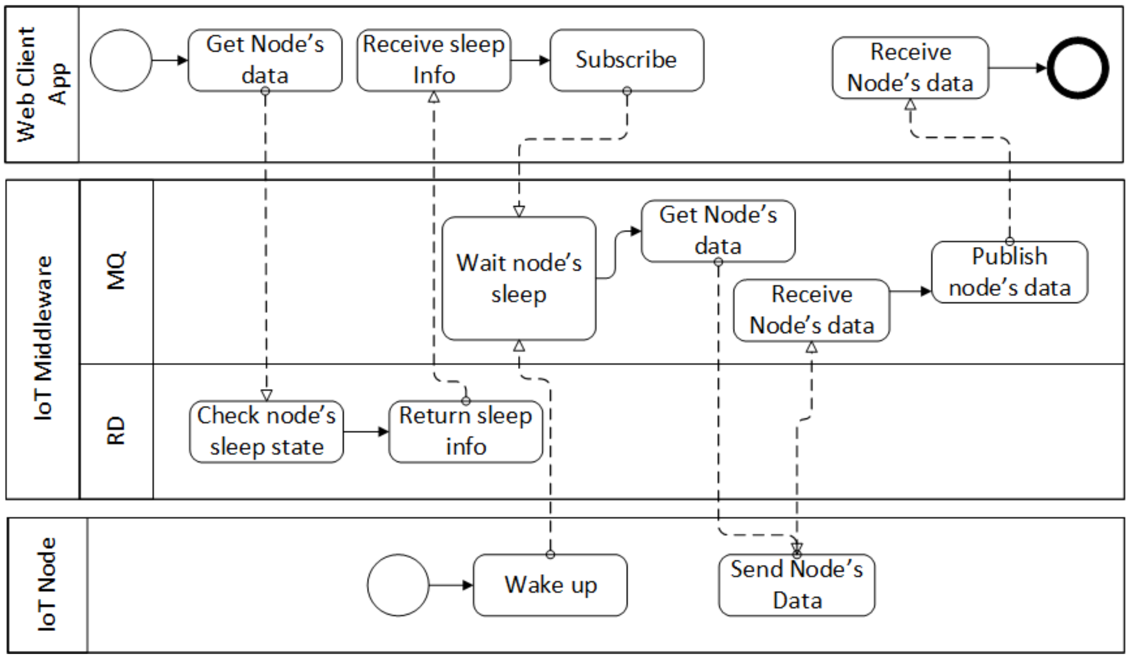


Figure 3.10. Business process model for asynchronous sleepy scenario.

3.4 Context data collecting mechanism based on CoAP MQ

IoT node collects context data from the environment in real time or periodically. We present tow way to collect the context data using the IoT middleware, which build on MQ mechanism.

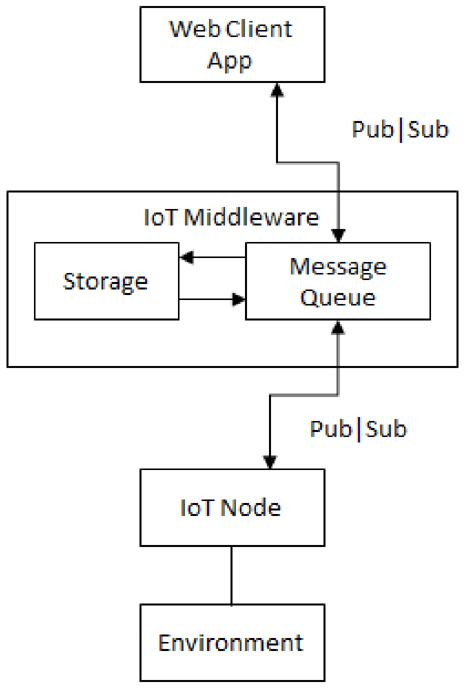


Figure 3.11. Functionality of MQ in IoT middleware for context data collection.

Figure 3.11 shows what the MQ do in the IoT middleware for collecting context data. There are Web Client App, IoT middleware and IoT node to interact in the context data collecting. Firstly, IoT node gets context data from environment around IoT node, and publish the data to IoT middleware which subscribed the function before. And the IoT middleware was subscribed by a Web Client App for context data, then the IoT middleware can publish the the saved context data.

In the case for the mechanism, this scenario is difference with real time collection of context data. In the case of the real time collection, each Web Client App need to send commend to IoT middleware in real time, and IoT middleware also do same way to the IoT node. It is not good for energy efficient. In the scenario that we presented, the context data is saved in the IoT middleware and forward each Web Client App. That means the IoT node don’s need to be request in real time. It is good for energy efficient for the CoAP node-IoT node.

4 Design of the IoT system based on unit ID

We design an IoT middleware to include RD and MQ broker which interacts with IoT node for IoT node registration, Sleepy mode and environment data collection. In the IoT, There are three IoT components which enables which are hardware such as made up of sensors, actuators and embedded communication hardware, middleware such as on demand storage and computing tools for data analytics and presentation such as novel easy to understand visualization and interpretation tools which can be widely accessed on different platforms and which can be designed for different applications. In this section, we present design of IoT elements and interaction of elements for functionalities.

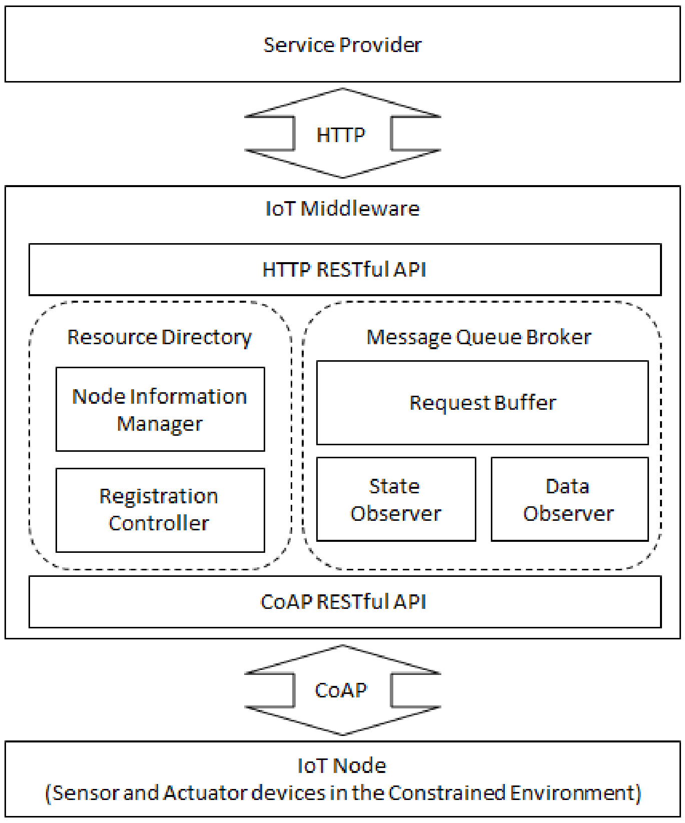


Figure 4.1. Overall functional architecture for interaction of IoT elements.

Figure 4.1 shows IoT elements that we proposed. There are IoT middleware and IoT node using CoAP to communicate. IoT middleware include RD part and MQ Broker part to implement Sleepy mode of IoT node. IoT node has CoAP library for working in constrained Environment and constrained network. IoT node may be a small board attaching several units such as sensor and actuator via the interface in the board. We design unit id for the IoT node controls functionality of unit.

4.1 IoT node based on unit ID

We present IoT node for constrained environment for the IoT. IoT node works in constrained network through CoAP. We design functionality of IoT node to support RESFful API. IoT node uses a link format, which is used by constrained web servers to describe hosted resources, their attributes, and other relationships between links. The CoRE Link Format is carried as a payload and is assigned an Internet media type [5].

The discovery of resources hosted by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. We refer to the discovery of resources hosted by a constrained web server, their attributes, and other resource relations as CoRE Resource Discovery in the IoT middleware.

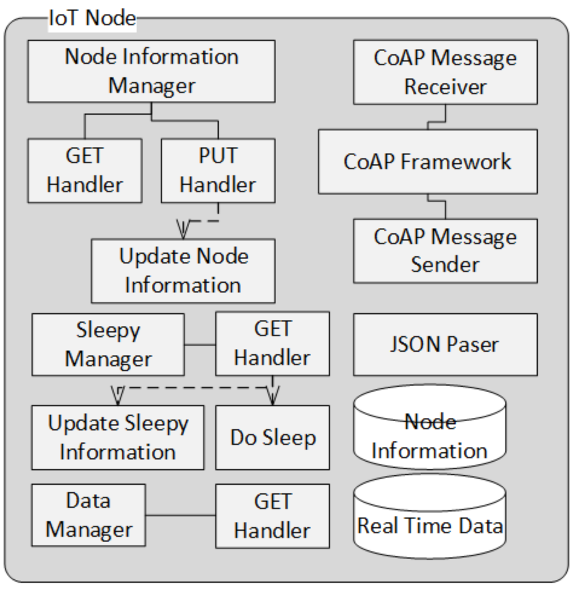


Figure 4.2. Functional structure of IoT node.

Figure 4.2 shows IoT node’s functions for working in Constrained RESTful Environment. There are 2 repository for save information. One is for saving Node Information and another one is for saving Real Time Data from collection of environment data. Node Information Manager has Get and PUT handler for control node’s information in the Node. Sleepy Manager is use for update sleepy information and control node’s sleepy state. When a request come to the part, it exist a method for falling asleep. Data Manager is use for sending collected data to client. We use the Libcoap CoAP Framework to implement the CoAP communication. Libcoap implements a lightweight application-protocol for devices that are constrained their resources such as computing power, RF range, memory, bandwith, or network packet sizes. This protocol, CoAP was standardized in the IETF as RFC 7252 [3]. Libcoap is published as open-source software without any warranty of any kind. Use is permitted under the terms of the GNU General Public License (GPL), Version 2 or higher, OR the revised BSD license [15]. In the process of the IoT node, all data formatted in JSON format. Incoming data and out-going data formatted in JSON in the interaction of IoT node and IoT middleware.

A CoAP resource provides a RESTful API to clients. IoT node includes 2 kind of resources and several sub resources for implement its functionality. These resources make itself accessible and modifiable by reacting to requests that carry one of the four request codes defined in CoAP: GET, POST, PUT, or DELETE. Each server holds a tree structure where each node is a resource. Each resource is identified by a URI that is composed of the URI of its parent plus its own name. When a request arrives at the server, it searches the resource tree for a resource that corresponds to the destination URI of the request. If the server finds the resource, the resource processes the request and responds with an adequate response code, options, and payload according to the CoAP protocol. If the server cannot find the destination resource, it responds with a 4.04 (Not Found) error code [16].

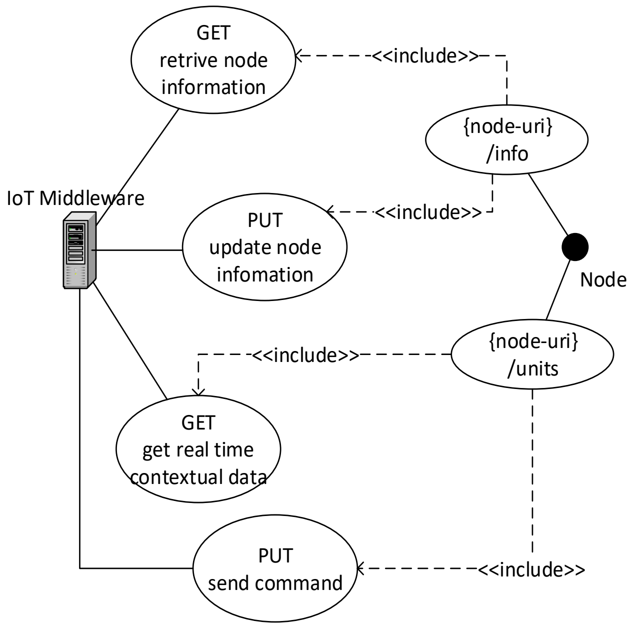


Figure 4.3. Use case for CoAP resources of IoT node.

Figure 4.3 shows resources of IoT node. IoT node has 2 resources in the initialized resources mainly. Resource “info” is use for manage information of the IoT node which includes GET method and PUT method. GET method is use for retrieve node’s information by search functionality of IoT middleware. PUT method is use for update node’s information. Resource “units” has GET method and PUT method for control units of IoT node. GET method is use for get real time context data and PUT method is use for control unit by CoAP client via command.

4.2 IoT middleware based on extended RD and MQ

In the IoT environment, a system is required to join the heterogeneous components together and to provide interoperability between them. This system should process, filter and route events in a scalable manner, given the other challenges such as volatility of network and massiveness of events. Given these requirements, a middleware is a very suitable solution for routing and delivering events. A middleware offers common services for the applications and eases application development [17]. In this thesis, we presented an IoT middleware to include RD and Message Queuing broker which interacts with IoT node for

IoT node registration, Sleepy mode and environment data collection. As devices become connected and the Internet of Things becomes ubiquitous, the multitude of devices will enable companies to improve customer service, offer newer products or streamline existing processes. Middleware plays a key role in acting as a bridge between such edge devices or things and enterprise applications. The role of middleware is to provide the infrastructure and IoT services which in turn help drive innovation, enable new revenue streams, and improve operational efficiencies [18]. In this part of IoT elements, there are some critical functionalities, such as aggregating and filtering the received data from the hardware devices, performing information discovery and providing access control to the devices for applications [19]. The IoT middleware to support device identification, device registration and look-up services for the IoT constrained environment. Different from all previous scenario, our design support the client interact with the composite node, the information regarding all its sub units is also provided to the client by the RD. We design IoT middleware to include RD and MQ broker to deal with Sleepy mode and node information management.

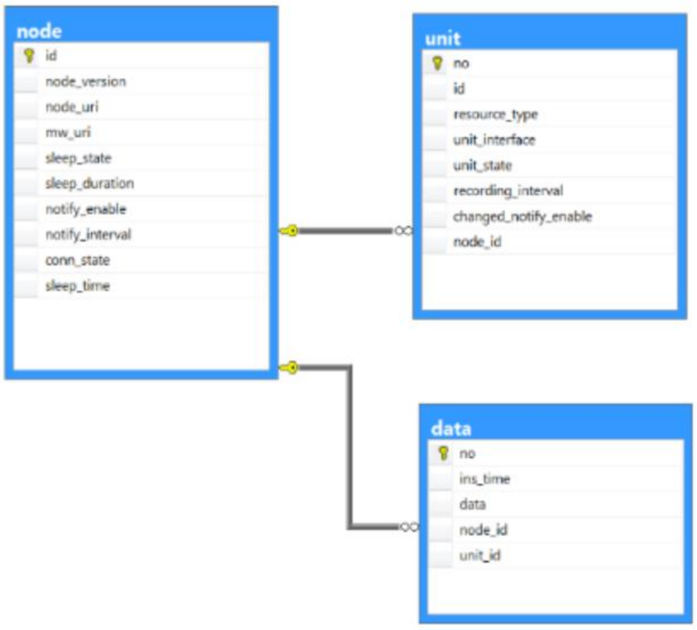


Figure 4.4. Database ER-Diagram for IoT middleware.

Figure 4.4 is database ER- diagram for database in the IoT middleware for save related information of node. In this ER-diagram, there are Node table, Unit table and Data table. Node table is use for save node information such as node ID, node’s URI and node’ state. Unit table includes column of unit information such as ID, resource type, interface and state. The unit Interface Description 'if' attribute is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource. One can think of this as describing verbs usable on a resource. The Interface Description attribute is meant to describe the generic REST interface to interact with a resource or a set of resources. It is expected that an Interface Description will be reused by different Resource Types. For example, the Resource Types "outdoor-temperature", "dew-point", and "rel-humidity" could all be accessible using the Interface Description "http://www.example.org/myapp.wadl# sensor". Multiple Interface Descriptions may be included in the value of this parameter, each separated by a space, similar to the relation attribute. The unit Resource Type 'rt' attribute is an opaque string used to assign an application-specific semantic type to a resource. One can think of this as a noun describing the resource [5].

Data table includes context data values and inserted time for recording collection of environment data by units of IoT node. These data from IoT node, which formatted in JSON type and parsed in the IoT middleware and save to the databased. The real time data save in the IoT node by collection of units such as sensors. The IoT middleware request to the IoT node to get the data discontinuously. And we design the resource interface of the IoT node to support respond period data. The IoT node save the real time environment data to the repository of IoT node, and the IoT middleware can request the period data from IoT node. In this case, the inserted time is same for all the data is inserted which from the request.

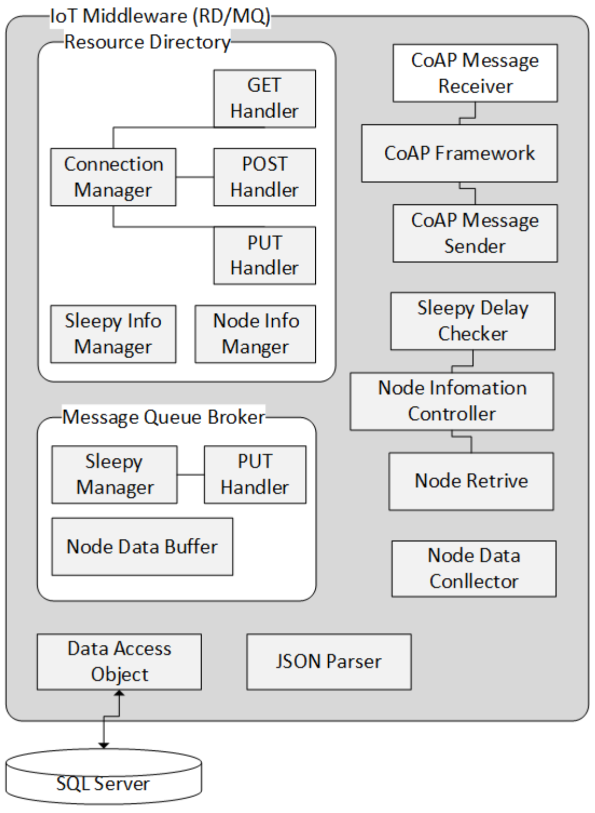


Figure 4.5. Functional structure of IoT middleware.

Figure 4.5 shows functional structure of IoT middleware. There are RD and MQ broker mainly. RD includes Connection Manager to control registration of IoT node and manages Sleepy information of IoT node and Node Information via Sleepy Information Manager and Node Information Manager. MQ broker has Sleepy Manager to receive request from IoT node. Sleepy Manager is a resource which has PUT method to handle request from IoT node. In the IoT middleware includes tow kind of communication functionalities for support service for HTTP and constrained environment. We use Californium (Cf) CoAP Framework to implement CoAP communication with IoT node in constrained network [23]. And another one is use for HTTP services. In the IoT middleware.

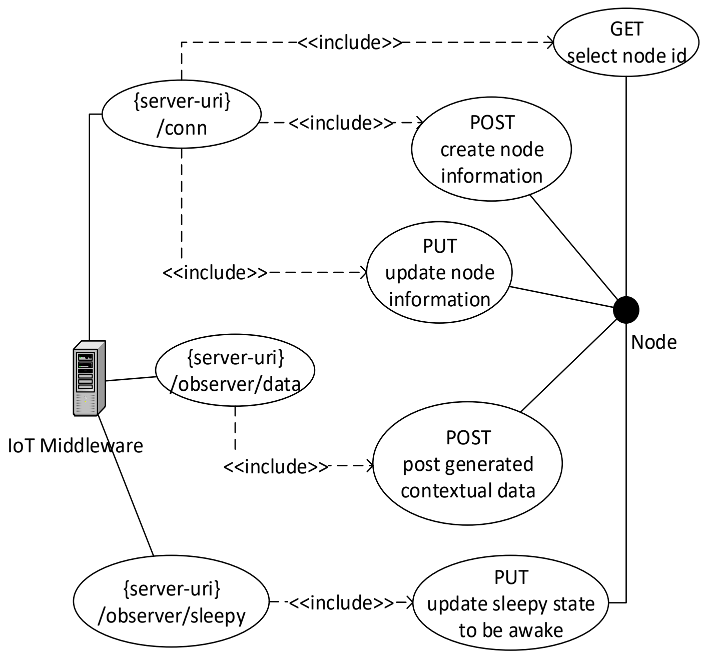


Figure 4.6. Use case for CoAP resources of IoT middleware.

Figure 4.6 shows resources of IoT middleware for CoRE. IoT middleware has three resources in the initialized resources mainly. "conn" resource is use for registration of IoT node. It support GET, POST and PUT method. GET method is use for select node' ID from database of IoT middleware by IoT node. POST method is use for create node information in the time of registration. PUT method is use for update node information which is not required. When node's information version is changed and difference with data in the IoT middleware, then IoT node use the PUT method access the resource. Resource "observer" has "data" and "sleepy" sub resources to be accessed. "data" resource is use for accepting context data from unit of IoT node such as collected environment data and actuator state data.

4.3 IoT node registration and discovery

IoT node has automatic registration process in the software. It connects to IoT middleware through configured IoT middleware URI to registry its information or update it. Information of IoT node in the database of IoT middleware, which has been inserted may be. In this case, the process is use for updating IoT node's information or do nothing. IoT node synchronizes information of node with IoT middleware via version attribute of IoT node profile (Section 3.1).

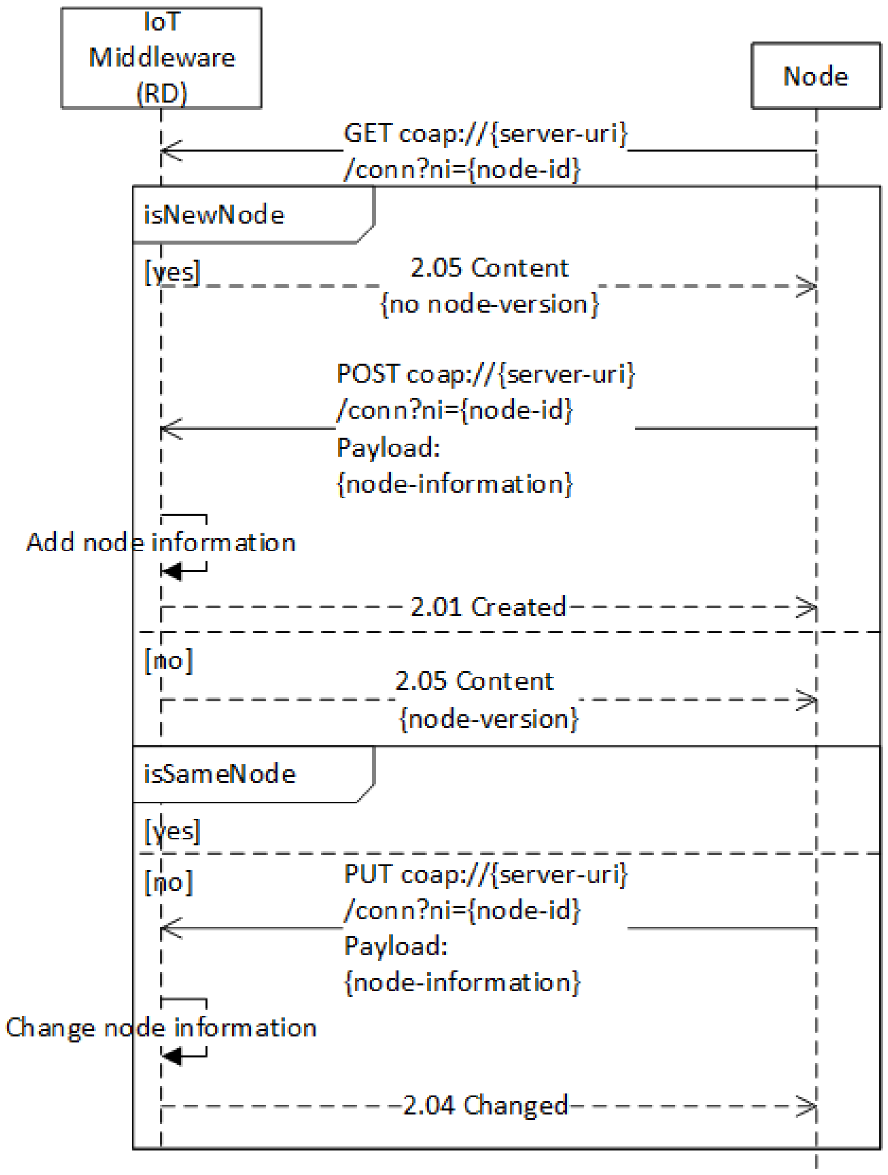

IoT Middleware (RD)
Node
GET coap://{server-uri} /conn?ni={node-id}
isNewNode
[yes]
2.05 Content {no node-version}
POST coap://{server-uri} /conn?ni={node-id} Payload: {node-information}
Add node information
2.01 Created
[no]
2.05 Content {node-version}
isSameNode
[yes]
[no]
PUT coap://{server-uri} /conn?ni={node-id} Payload: {node-information}
Change node information
2.04 Changed

Figure 4.7. Sequence diagram for IoT node registration.

Figure 4.7 shows the sequence diagram for process of IoT node registration. Firstly, IoT node sends a GET request with “ni” parameter to “conn” resource of IoT middleware. The parameter is Node ID which is used for retrieve the IoT node in the database of IoT middleware. If there is the information via the Node ID confirmed then IoT middleware responds Node Version to IoT node. Else IoT middleware responds a string to notify IoT node for there isn’t node information. After IoT node receives the response, it sends POST request to IoT middleware with “ni” and payload which includes node’s information for register a new IoT node. When there is same node information and difference version of the information then IoT node sends a PUT request to IoT middleware to update the information. And when there is same node and same version of the information then IoT node do nothing.

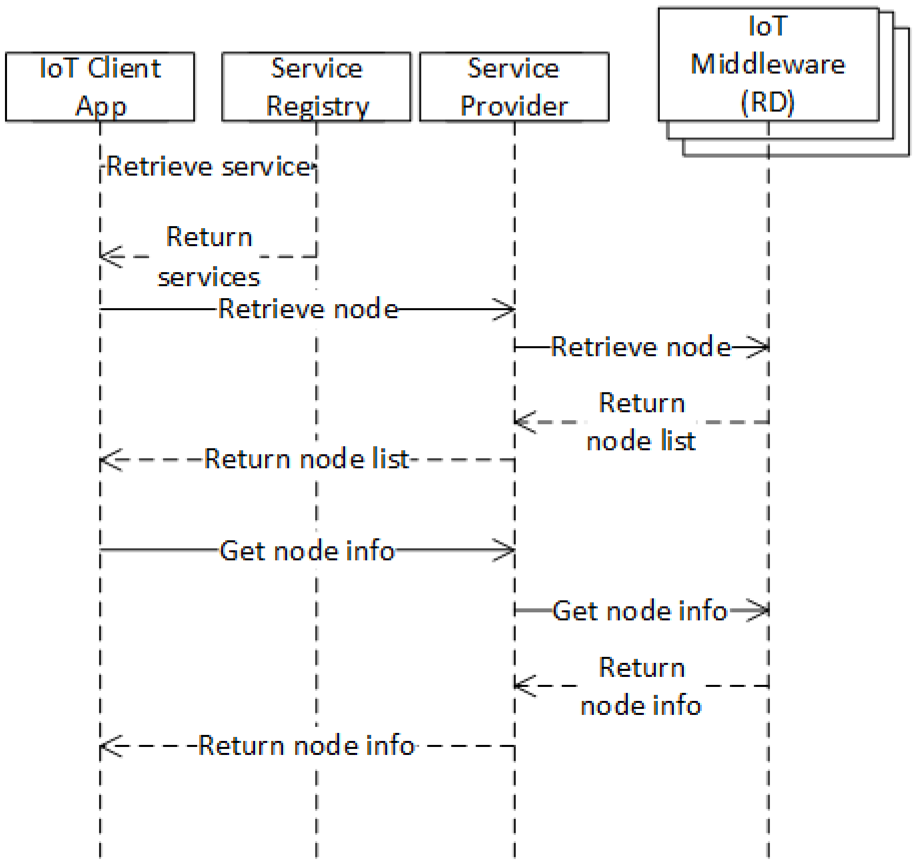


Figure 4.8. Sequence diagram for IoT node discovery.

Figure 4.8 shows sequence diagram for IoT node discovery by IoT Client App using IoT service. In the service layer of IoT, IoT Client App gets node discovering service from service provider. Firstly it gets web services from service registry which are registered. And he IoT Client App uses the web service to retrieve nodes that the IoT Client App needs. Then the service provider retrieves the nodes from IoT middlewares that enables to the service provider. After the node list arrives user of IoT Client App, the user can request a specific node to get detail information from RD of an IoT middleware.

4.4 Sleepy scheme based on MQ

The Sleepy information are changed in IoT middleware and IoT node both side. Sleep state indicates whether the node is currently in sleep mode or not. Sleep duration indicates the maximum duration of time that the node stays in sleep mode. There is notify process which includes synchronous sleepy mode and asynchronous sleepy mode. We design the mechanism using functionalities of RD and MQ for Web Client App is learned sleepy state information of IoT node. Synchronous process is used for Web Client App needs to synchronize the sleep time of the IoT node and asynchronous process is used for Web Client App needs to subscribe the IoT node which falls asleep.

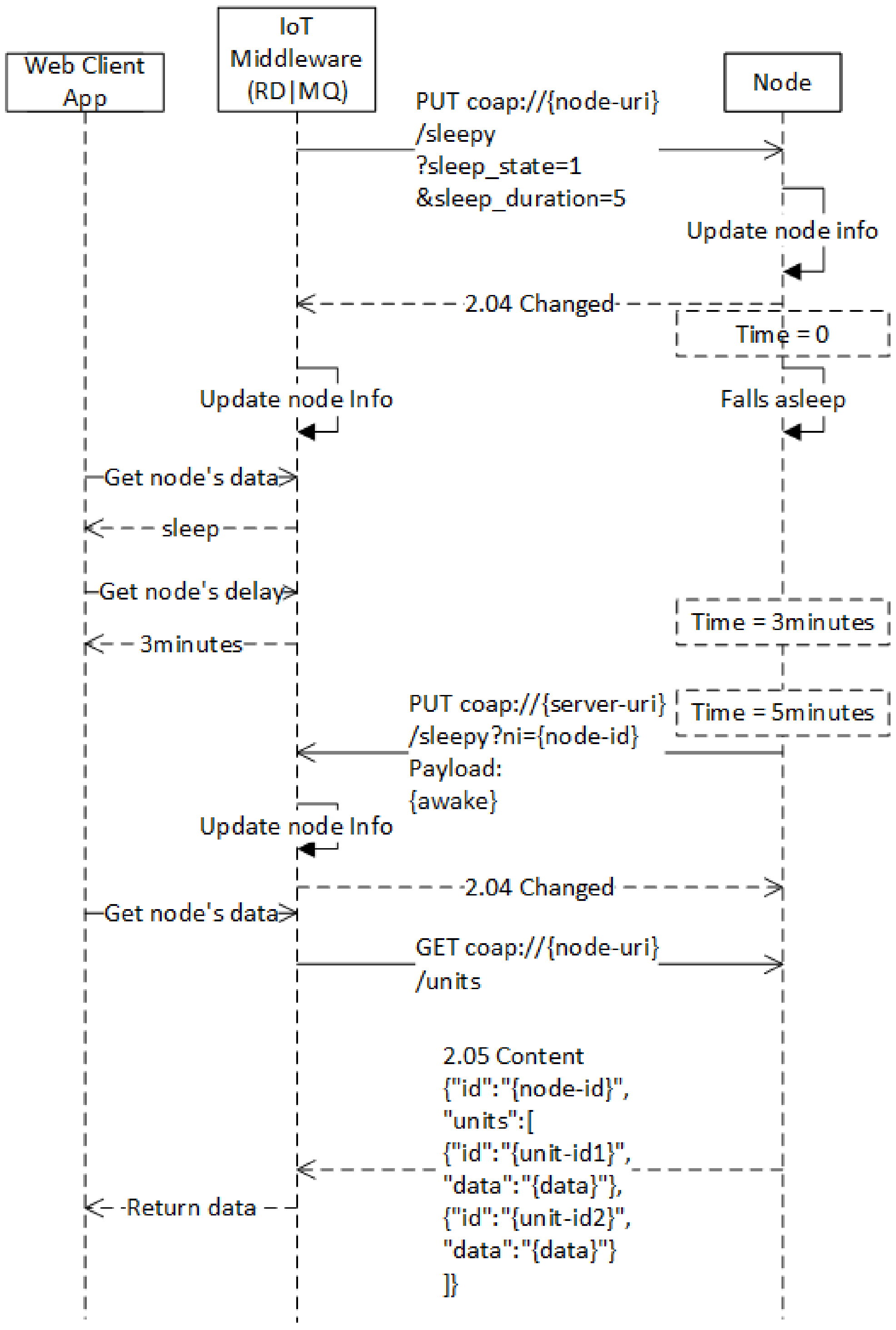

Web Client App
IoT Middleware (RD|MQ)
Node
PUT coap://{node-uri} /sleepy ?sleep_state=1 &sleep_duration=5
Update node info
2.04 Changed
Time = 0
Update node Info
Falls asleep
Get node's data
sleep
Get node's delay
3minutes
Time = 3minutes
PUT coap://{server-uri} /sleepy?ni={node-id} Payload: {awake}
Time = 5minutes
Update node Info
2.04 Changed
Get node's data
GET coap://{node-uri} /units
2.05 Content {"id":"{node-id}", "units":[{"id":"{unit-id1}", "data":"{data}"}, {"id":"{unit-id2}", "data":"{data}"}]}
Return data

Figure 4.9. Sequence diagram for synchronous sleepy scheme using HTTP.

Figure 4.9 shows IoT node receive a PUT request from IoT middleware for change sleepy state. When Web Client App requests for get node’s data then IoT middleware respond a message with sleep information. And Web Client App gets node's delay information from IoT middleware to synchronize the sleep state information.

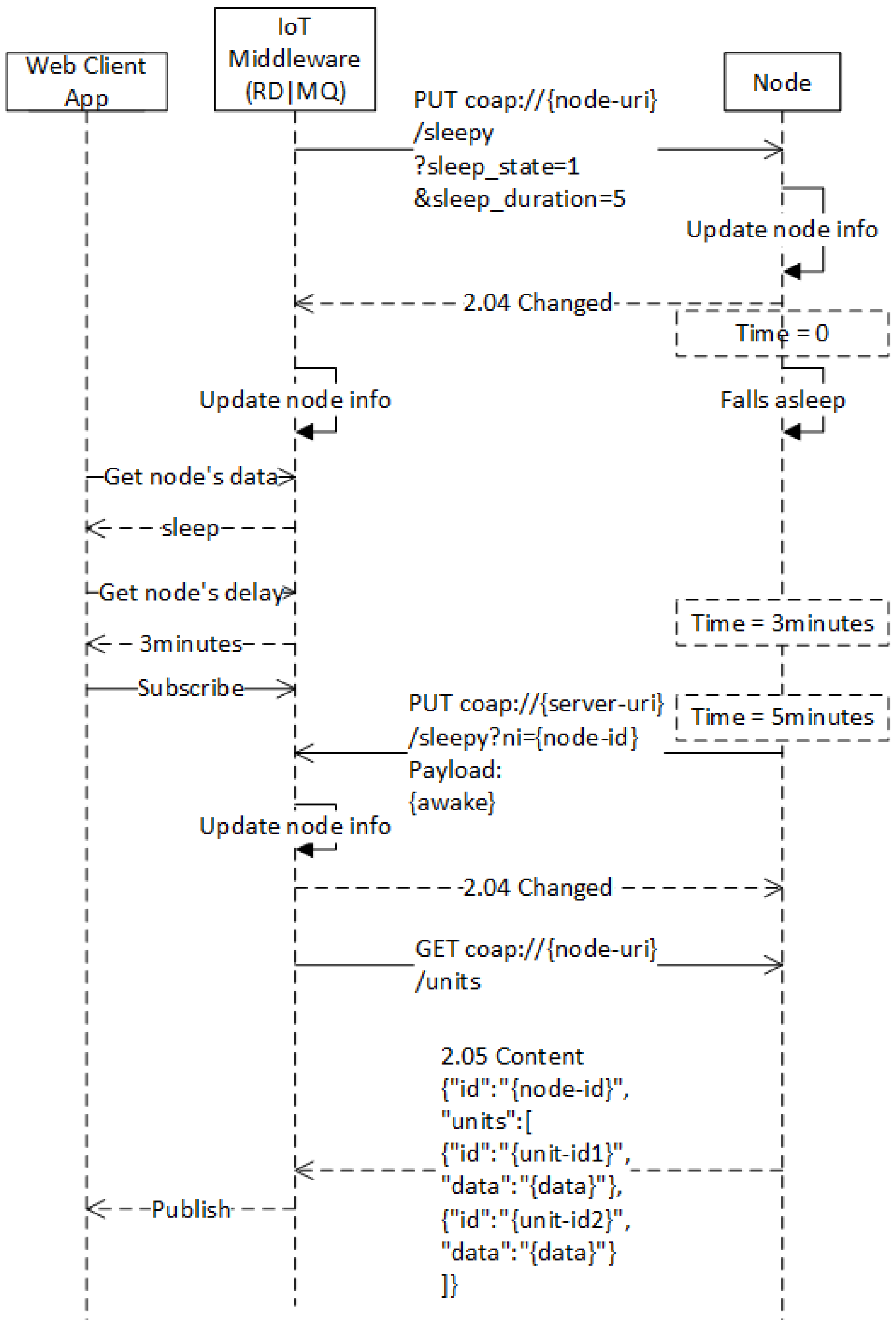

Web Client App
IoT Middleware (RD|MQ)
Node
PUT coap://{node-uri} /sleepy ?sleep_state=1 &sleep_duration=5
Update node info
2.04 Changed
Time = 0
Update node info
Falls asleep
Get node's data
sleep
Get node's delay
3minutes
Time = 3minutes
Subscribe
PUT coap://{server-uri} /sleepy?ni={node-id} Payload: {awake}
Time = 5minutes
Update node info
2.04 Changed
GET coap://{node-uri} /units
2.05 Content {"id":"{node-id}", "units":[{"id":"{unit-id1}", "data":"{data}"}, {"id":"{unit-id2}", "data":"{data}"}]}
Publish

Figure 4.10. Sequence diagram for asynchronous sleepy scheme based on MQ using HTTP.

Figure 4.10 shows a same process with figure 4.9 in first action for change sleep state and synchronize sleep state information with Web Client App. But this scenario present another way to notify the Web Client App. Web Client App can use the subscribe function to request for get node's data. When IoT middleware receive PUT request from IoT node for update sleep state to be awake, then IoT middleware sends a GET request to get context data from IoT node via CoAP. And IoT middleware sends the data to Web Client App via Publish functionality.

4.5 Context data collection using buffer based on MQ

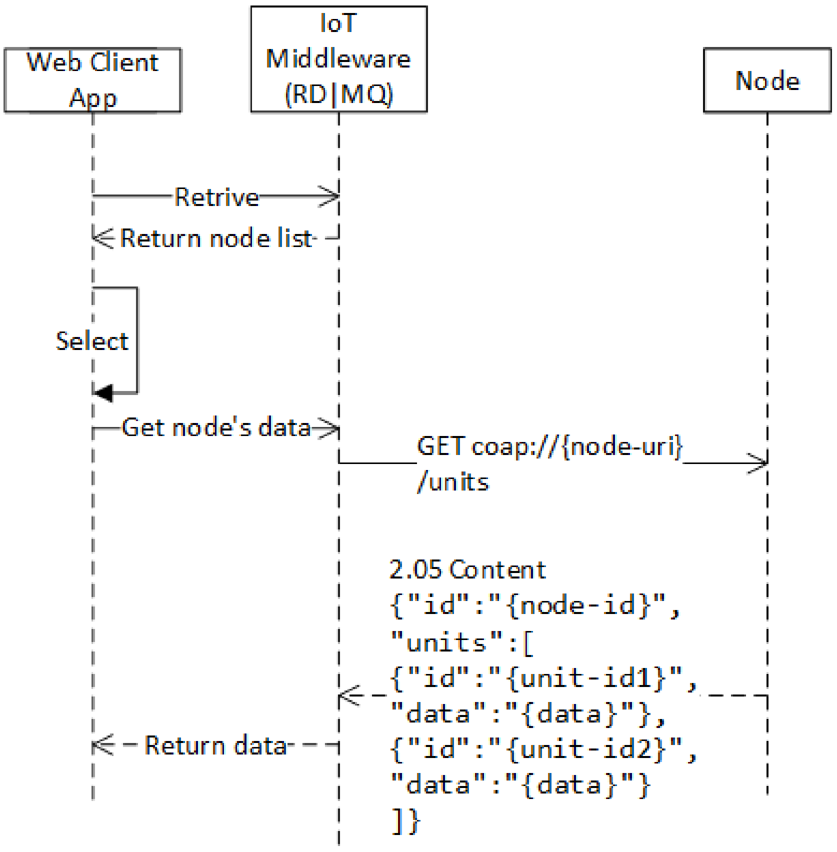

Web Client App
IoT Middleware (RD|MQ)
Node
Retrive
Return node list
Select
Get node's data
GET coap://{node-uri}/units
2.05 Content {"id":"{node-id}", "units":[{"id":"{unit-id1}", "data":"{data}"}, {"id":"{unit-id2}", "data":"{data}"}]}
Return data

Figure 4.11. Sequence diagram for context data acquiring.

Figure 4.11 shows the Web Client App acquire context data from IoT node through IoT middleware. Web Client App gets real time data from IoT node in this process. Firstly, Web Client App retrieve node information from IoT middleware and get node list. And a node is selected in the list to access for get context data of node. When IoT middleware receive a request with selected node id for get node' data then it send a GET request to IoT node via CoAP and receive context data from IoT node. Finally, IoT middleware forward the received data to Web Client App.

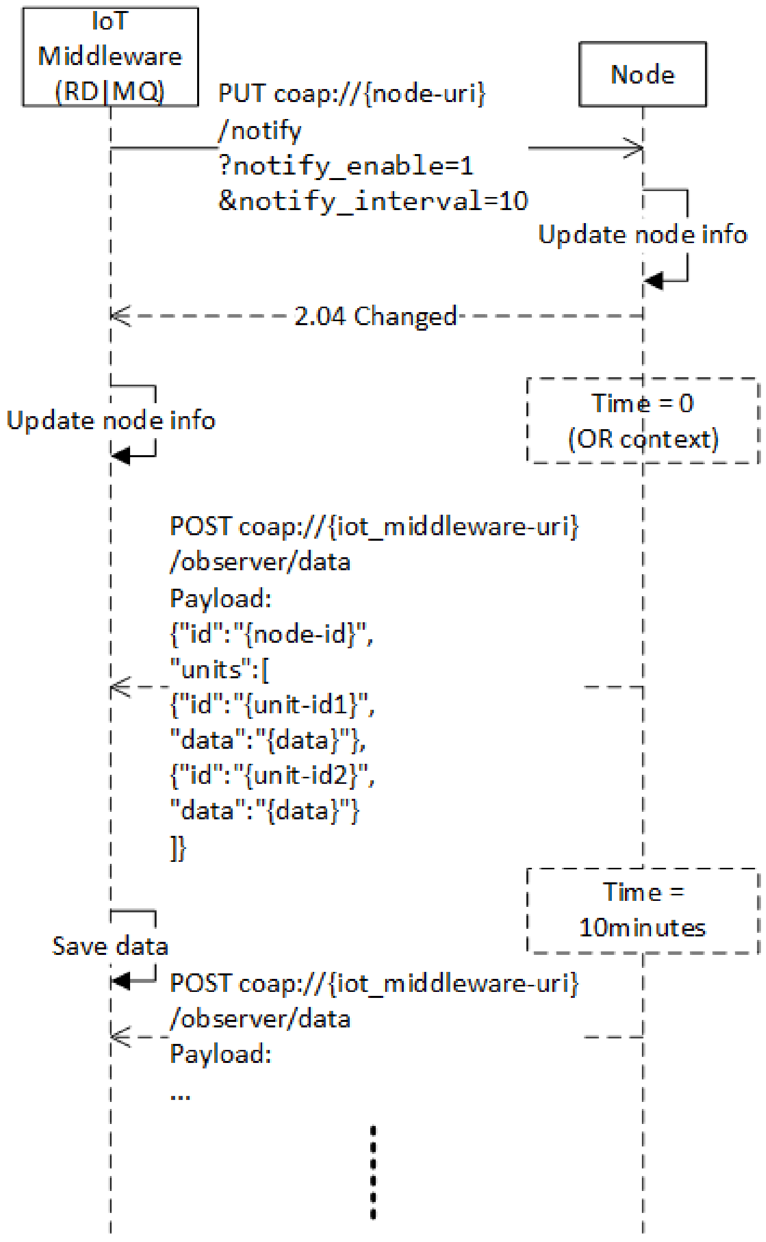


Figure 4.12. Sequence diagram for collecting context data from IoT node.

IoT middleware collects context data from IoT node for saving and support to Web Client App. Figure 4.12 shows a sequence for collecting data from IoT node. Firstly, an IoT node receive a command to change notify information from a IoT middleware. And the IoT node sends context data to the IoT middleware periodically. Or the context of the IoT node is changed, then the IoT node will sends the changed context date to the IoT middleware.

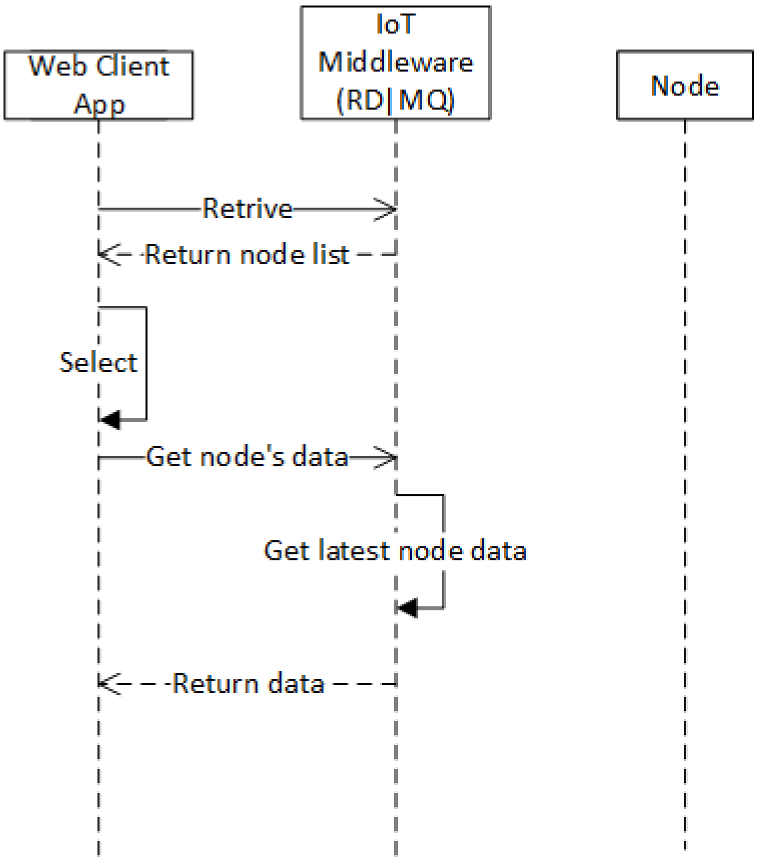


Figure 4.13. Sequence diagram for acquirement of context data from IoT middleware.

IoT middleware receive context data from IoT node periodically to save and support to Web Client App. Figure 4.13 show a sequence diagram for acquirement of context data from IoT middleware. Web Client App sends a request for get data to IoT middleware, and IoT middleware respond a latest node data which are saved in database.

5 Implementation of the IoT system

We use a Java framework to implement IoT middleware and using a c library to implement IoT node. IoT node is implemented in Linux C compile environment because of IoT node working in the constrained environment. IoT middleware is implemented in java compile environment. The IoT middleware is a web application which support web link to be accessed by user such as web client and HTTP client. In the service layer, our service provider supports SOAP web service which uses HTTP client to request the IoT middleware.

5.1 Basic IETF CoAP

We implement the CoAP protocol in the proposed different environment, and verify the interoperability. The table 5.1 is IDE for implementing the CoAP protocol. The client is made with Java Runtime Environment (JRE) in the Window OS. We used eclipse as a development tool. The server is made with C language in the Ubuntu Linux environment, and it is executed in GCC.

The CoAP protocol is application layer protocol and it can be executed to program. The CoAP client is implemented with Java, and the CoAP server is implemented with C. The CoAP client sends message to the CoAP protocol server through the CoAP protocol, and the server parsing the received message, and execute that function then send a result to the client. By using the CoAP protocol, the CoAP client request the context data and process the received data then send to the user via a web service. The CoAP client is implemented with the java language. The CoAP server is developed with C language to implement the CoAP protocol to the sensor node. We, parsing and analyze it in the CoAP server by using the CoAP request message format.

Table 5.1. Implementation environment for CoAP protocol.

Development environment		
Environment	Client	Server
OS	Windows 8.1	Ubuntu 12.4 (Linux)
Run environment	Java 7	gcc
Tool	eclipse	gedit
Language	Java	c

Figure 5.1 shows environment for running CoAP client and CoAP server program. The. CoAP client was written by Java, so it was executed in JRE environment. Only Java environment has influence in execution. The CoAP server uses Linux C library and can be executed where the Linux that GCC compiler is exist.

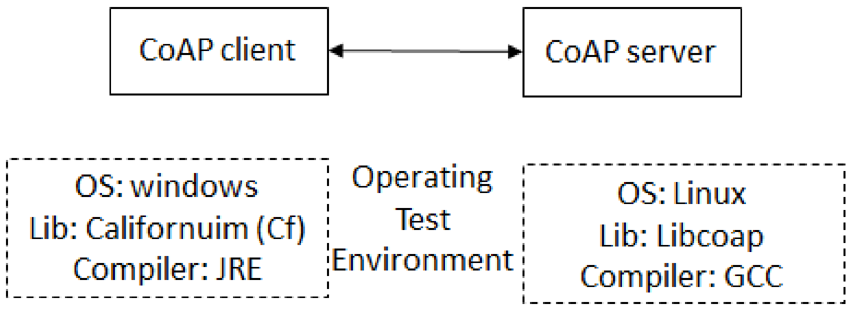
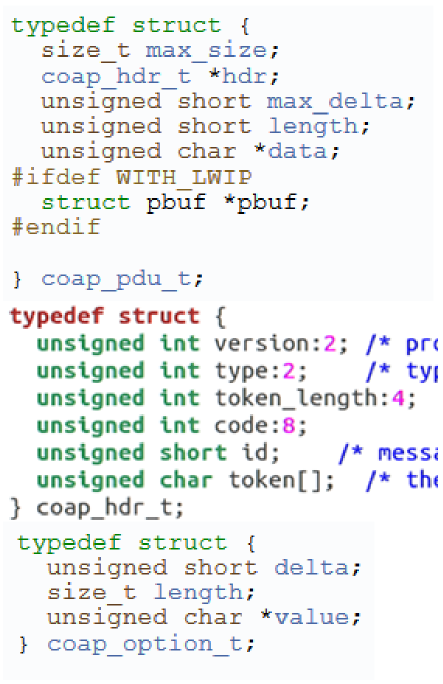


Figure 5.1. Experiment environment for CoAP protocol.

To implement the CoAP protocol in server and Linux environment, we use two libraries. The CoAP client server uses Californium which is based on the Java language, and the CoAP server uses Californium library which is based on the C language. The Californium library is ETSI IoT CoAP protocol and it uses “3-clause BSD” license [23]. The Libcoap protocol is implemented by using Linux C library, and it uses “BSD” license. Also, the Libcoap protocol is potting in TinyOS.



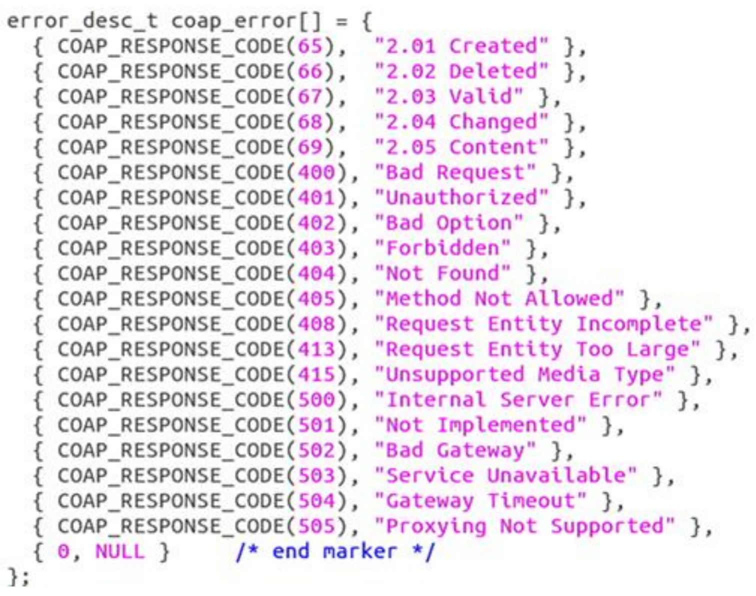
```
typedef struct {
  size_t max_size;
  coap_hdr_t *hdr;
  unsigned short max_delta;
  unsigned short length;
  unsigned char *data;
#ifdef WITH_LWIP
  struct pbuf *pbuf;
#endif

} coap_pdu_t;
```

```
typedef struct {
  unsigned int version:2; /* pr
  unsigned int type:2;    /* ty
  unsigned int token_length:4;
  unsigned int code:8;
  unsigned short id;     /* mess
  unsigned char token[];  /* th
} coap_hdr_t;
```

```
typedef struct {
  unsigned short delta;
  size_t length;
  unsigned char *value;
} coap_option_t;
```

Figure 5.2. Basic type and structures definition for CoAP protocol.

To implement the CoAP protocol stack, it is necessary to define of 8, 16, 32 bit data type and the structure for message header and option header. This definition is shown in figure 5.2. Similarly, the constant definition is necessary for getting value for method and response code in Code field [21]. For this purpose, the Libcoap Framework defines the filed name according to the familiar HTTP error code as shown as figure 5.3 [22].

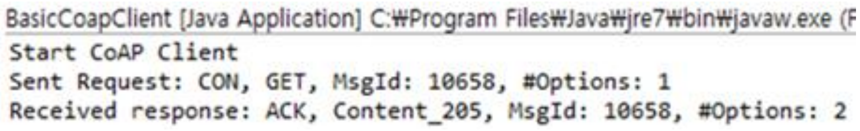


```
error_desc_t coap_error[] = {
  { COAP_RESPONSE_CODE(65),  "2.01 Created" },
  { COAP_RESPONSE_CODE(66),  "2.02 Deleted" },
  { COAP_RESPONSE_CODE(67),  "2.03 Valid" },
  { COAP_RESPONSE_CODE(68),  "2.04 Changed" },
  { COAP_RESPONSE_CODE(69),  "2.05 Content" },
  { COAP_RESPONSE_CODE(400), "Bad Request" },
  { COAP_RESPONSE_CODE(401), "Unauthorized" },
  { COAP_RESPONSE_CODE(402), "Bad Option" },
  { COAP_RESPONSE_CODE(403), "Forbidden" },
  { COAP_RESPONSE_CODE(404), "Not Found" },
  { COAP_RESPONSE_CODE(405), "Method Not Allowed" },
  { COAP_RESPONSE_CODE(408), "Request Entity Incomplete" },
  { COAP_RESPONSE_CODE(413), "Request Entity Too Large" },
  { COAP_RESPONSE_CODE(415), "Unsupported Media Type" },
  { COAP_RESPONSE_CODE(500), "Internal Server Error" },
  { COAP_RESPONSE_CODE(501), "Not Implemented" },
  { COAP_RESPONSE_CODE(502), "Bad Gateway" },
  { COAP_RESPONSE_CODE(503), "Service Unavailable" },
  { COAP_RESPONSE_CODE(504), "Gateway Timeout" },
  { COAP_RESPONSE_CODE(505), "Proxying Not Supported" },
  { 0, NULL }      /* end marker */
};
```

Figure 5.3 Method and responded codes definition.

When the client send a message to the server, the server send the response message. The CoAP protocol is UDP upper protocol, and when the response message is not reached to the server in the pre-defined time, it sends message again. Repeat this again 5 times and if it is not received response message, this means that the communication is failed.

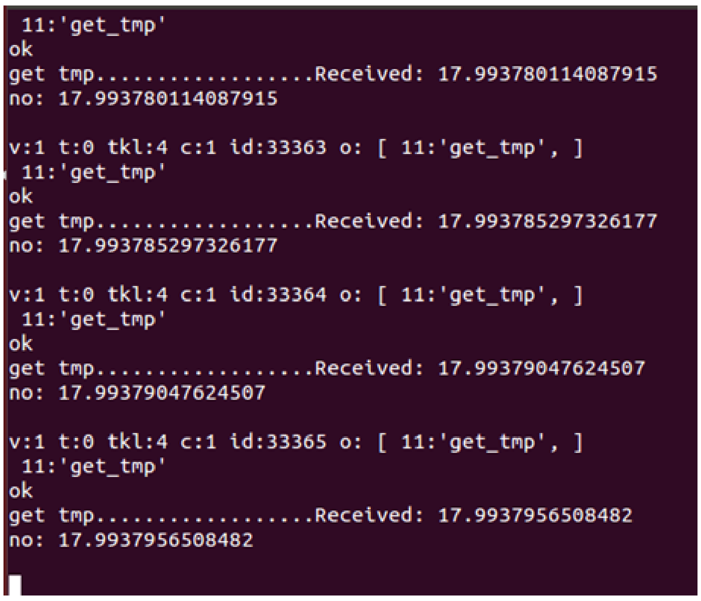
The figure 5.4 shows the result of CoAP client execution. When the CoAP protocol client send requirement message to the CoAP server via the CoAP protocol, the CoAP protocol server processes it and sends message to the CoAP protocol client. This figure shows message and response message. The value “CON” is the T (Type) value of the CoAP protocol message format, and the “GET” is the method type of the CoAP protocol, and the IETF defines 4 method type. In the “MsgId: 10658”, the “10658” means ID of message, and this value can be created automatically or developer can define it. The “#Options: 1” means the number of message attributes included in CoAP protocol message, and basically it is “MsgId” attribute.



```
BasicCoapClient [Java Application] C:₩Program Files₩Java₩jre7₩bin₩javaw.exe (F
Start CoAP Client
Sent Request: CON, GET, MsgId: 10658, #Options: 1
Received response: ACK, Content_205, MsgId: 10658, #Options: 2
```

Figure 5.4. Experiment results of CoAP client.

Figure 5.5 shows experiment screen of node emulator in CoAP server. This program is executed with Console in Linux environment, and receiving message from a CoAP protocol node and collecting fictive temperature is shown as form of Console screen.

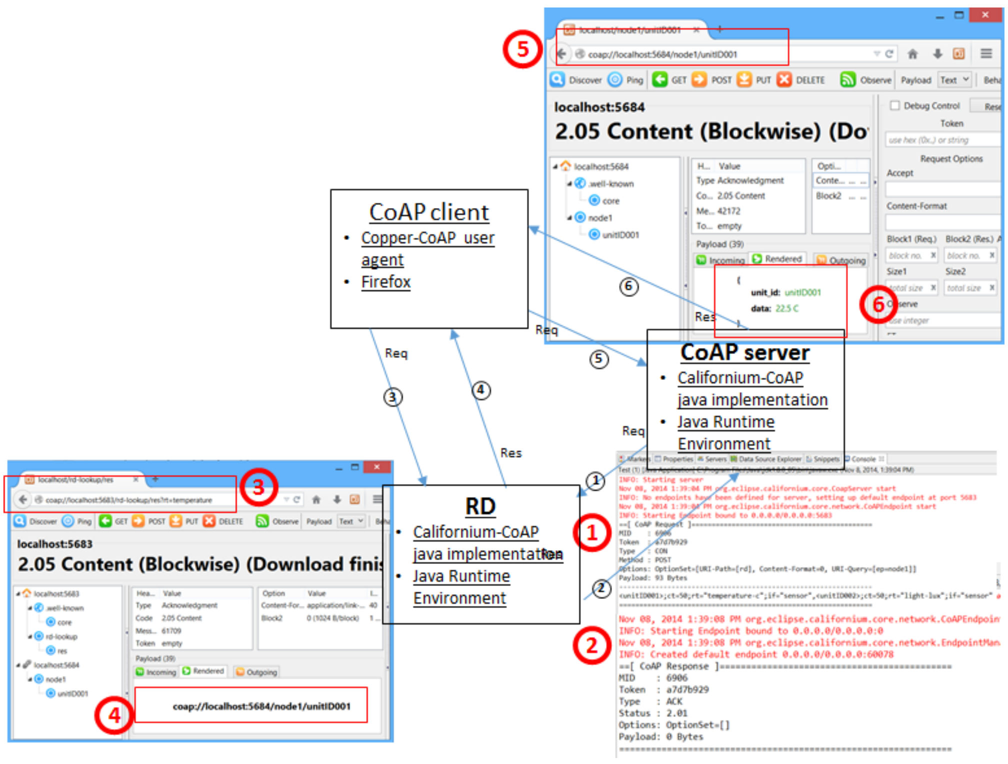


```
 11:'get_tmp'
ok
get tmp.................Received: 17.993780114087915
no: 17.993780114087915

v:1 t:0 tkl:4 c:1 id:33363 o: [ 11:'get_tmp', ]
 11:'get_tmp'
ok
get tmp.................Received: 17.993785297326177
no: 17.993785297326177

v:1 t:0 tkl:4 c:1 id:33364 o: [ 11:'get_tmp', ]
 11:'get_tmp'
ok
get tmp.................Received: 17.99379047624507
no: 17.99379047624507

v:1 t:0 tkl:4 c:1 id:33365 o: [ 11:'get_tmp', ]
 11:'get_tmp'
ok
get tmp.................Received: 17.9937956508482
no: 17.9937956508482
```

Figure 5.5. Experiment results of CoAP server.

5.2 Registration and discovery using extended RD based on unit ID

We designed unit ID based CoAP in the IoT. For this mechanism, we develop a prototype to show the CoAP interaction between each element of CoRE. In the interaction, there are RD, CoAP client and CoAP server which enable registration and discovery of CoAP Node. As shown as figure 5.6, the RD and CoAP server use Californium (Cf) framework to implements and running in Java Runtime Environment. And CoAP client uses Copper-CoAP user agent to interact with CoAP server and RD via Firefox (Web browser) [24]. According to the mechanism, The IoT middleware includes CoAP server and RD, and IoT node includes CoAP client.

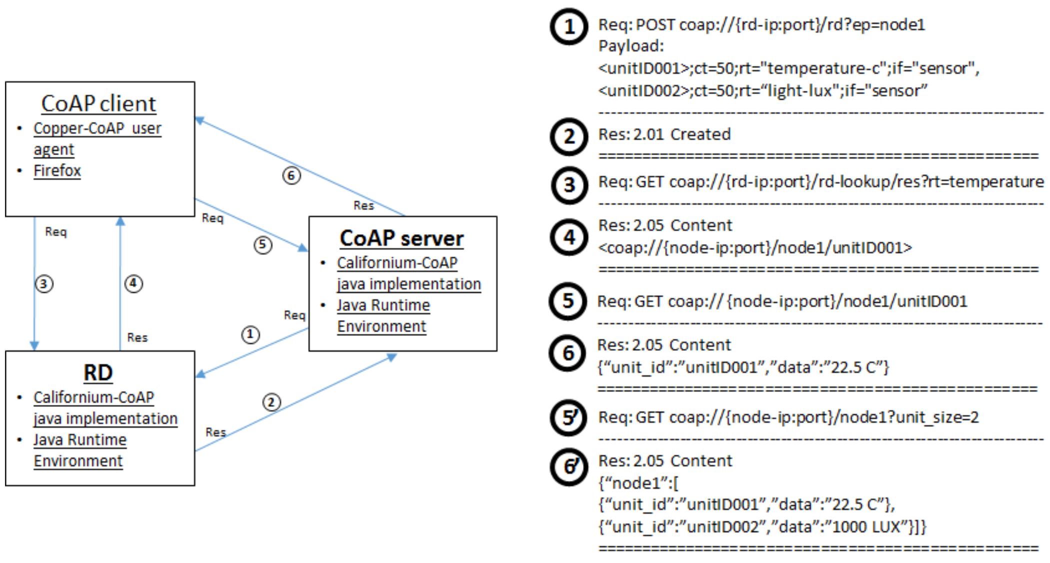

CoAP client
Copper-CoAP user agent
Firefox
CoAP server
Californium-CoAP java implementation
Java Runtime Environment
RD
Californium-CoAP java implementation
Java Runtime Environment
Req
Res
1
2
3
4
5
6
coap://localhost:5684/node1/unitID001
coap://localhost:5683/rd-lookup/res?rt=temperature
localhost:5684
2.05 Content (Blockwise)
localhost:5683
2.05 Content (Blockwise) (Download finis
unit_id: unitID001
data: 22.5 C
coap://localhost:5684/node1/unitID001

```
INFO: Starting server
Nov 08, 2014 1:39:04 PM org.eclipse.californium.core.CoapServer start
INFO: No endpoints have been defined for server, setting up default endpoint at port 5683
Nov 08, 2014 1:39:04 PM org.eclipse.californium.core.network.CoAPEndpoint start
INFO: Starting Endpoint bound to 0.0.0.0/0.0.0.0:5683
==[ CoAP Request ]============================================
MID     : 6906
Token   : a7d7b929
Type    : CON
Method  : POST
Options: OptionSet=[URI-Path=[rd], Content-Format=0, URI-Query=[ep=node1]]
Payload: 93 Bytes
---------------------------------------------------------------
<unitID001>;ct=50;rt="temperature-c";if="sensor",<unitID002>;ct=50;rt="light-lux";if="sensor"
===============================================================
Nov 08, 2014 1:39:08 PM org.eclipse.californium.core.network.CoAPEndpoin
INFO: Starting Endpoint bound to 0.0.0.0/0.0.0.0:0
Nov 08, 2014 1:39:08 PM org.eclipse.californium.core.network.EndpointMan
INFO: Created default endpoint 0.0.0.0/0.0.0.0:60078
==[ CoAP Response ]============================================
MID     : 6906
Token   : a7d7b929
Type    : ACK
Status  : 2.01
Options: OptionSet=[]
Payload: 0 Bytes
===============================================================
```

Figure 5.6. Prototype development for CoAP endpoint unit identification.

Figure 5.7 shows request and response interaction between RD, CoAP client and CoAP server. The process shows from registration to acquiring contextual data from CoAP server sequentially. Firstly, it shows request and response for a node uses the RD's registration function set and sends a CoAP POST message to the RD. The RD receives a valid request from the node, the source IP address and Port number from the CoAP request parameters or the message source address portion (default). The RD then extracts the Unit IDs from the message payload and save the information and returns a response message to the node. And it shows a client requesting for a specific type (Temperature) of resources registered with the RD. For this purpose, it sends GET request to the RD with the Resource Type (rt). The RD receives the message, checks if the message is a valid CoAP request and then gets the IDs for all the registered resources with the resource type value equivalent to the one requested by the CoAp client (Temperature). The RD then creates a response message with the list of node IP address and resource IDs and sends it to the client. The client may then choose a specific resource from this list and communicate with it directly using the CoAP protocol.

Finally, CoAP client sends request message to CoAP server for acquire contextual data through the URI which are chosen in the resource list. There is a different case for data acquisition. IoT middleware can response various data which includes multiple units.



```
① Req: POST coap://{rd-ip:port}/rd?ep=node1
  Payload:
  <unitID001>;ct=50;rt="temperature-c";if="sensor",
  <unitID002>;ct=50;rt="light-lux";if="sensor"
  -------------------------------------------------------------------
② Res: 2.01 Created
  ===========================================
③ Req: GET coap://{rd-ip:port}/rd-lookup/res?rt=temperature
  -------------------------------------------------------------------
④ Res: 2.05 Content
  <coap://{node-ip:port}/node1/unitID001>
  ===========================================
⑤ Req: GET coap:// {node-ip:port}/node1/unitID001
  -------------------------------------------------------------------
⑥ Res: 2.05 Content
  {"unit_id":"unitID001","data":"22.5 C"}
  ===========================================
⑤' Req: GET coap://{node-ip:port}/node1?unit_size=2
  -------------------------------------------------------------------
⑥' Res: 2.05 Content
  {"node1":[
  {"unit_id":"unitID001","data":"22.5 C"},
  {"unit_id":"unitID002","data":"1000 LUX"}]}
  ===========================================
```

Figure 5.7. Development of unit ID based CoAP node registration.

5.3 IoT middleware and IoT node based on extended RD and MQ

Figure 5.8 show software of IoT middleware and IoT node. IoT middleware interacts with Service Provider using HTTP and IoT middleware interacts with IoT node using CoAP. Figure 4.3 and figure 4.6 show CoAP RESTful API in IoT middleware and IoT node. In this section we present implementation scenario for interaction of IoT middleware and IoT node.

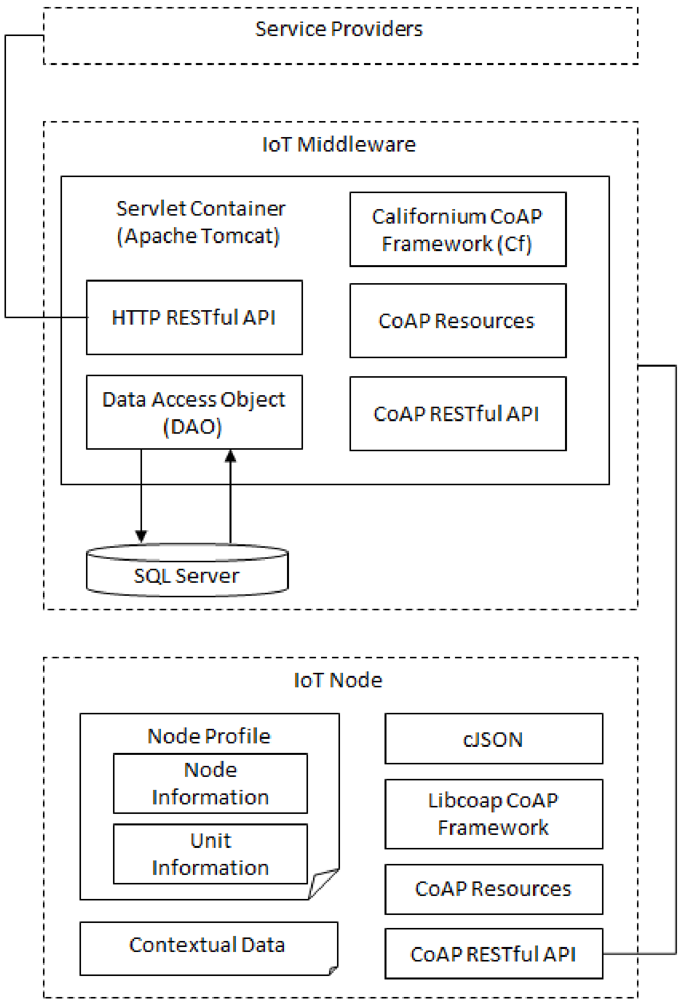
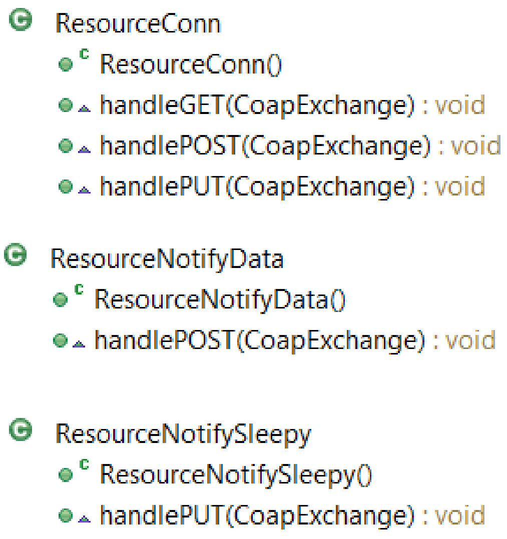


Figure 5.8. Implementation environment of IoT middleware and IoT node.

IoT middleware is a server which are provides computation, storing and communication. We implement IoT middleware using Java with Californium (Cf) CoAP framework. For supporting HTTP RESTful API, we use Apache Tomcat to run Servlet. When the servlets run in the Apache Tomcat, there is initialization function for initialize CoAP resources. Through CoAP resources, IoT middleware supports CoAP RESTful API to IoT node. IoT middleware includes SQL server for manage data via Data Access Object (DAO).

IoT node is used for constrained environment, therefore we implement it using Linux C library. IoT node includes Libcoap CoAP library and cJSON library for develop the software in IoT node [25]. We use Libcoap to implement CoAP resources for supporting CoAP

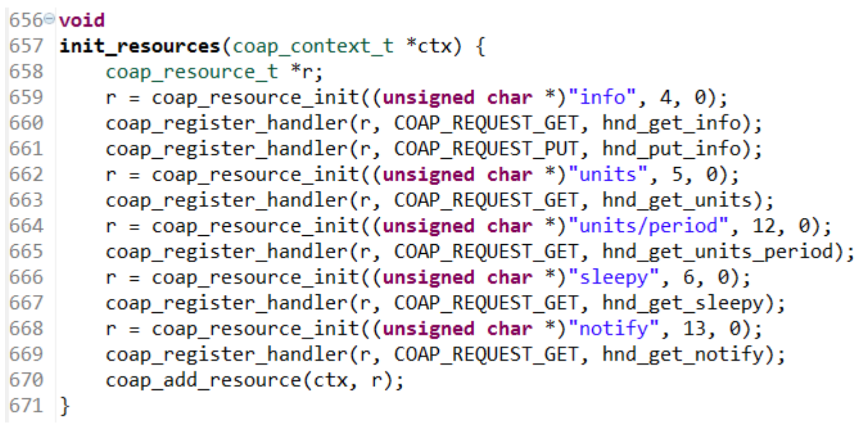
RESTful API, and cJSON is used for parsing JSON format. We use JSON in the communication of IoT middleware and IoT node. IoT node includes text files as storages. Node Profile is used for save information of node which includes state information and Unit Information etc. And another one is used for save Contextual Data of Node.



```
ResourceConn
    ResourceConn()
    handleGET(CoapExchange) : void
    handlePOST(CoapExchange) : void
    handlePUT(CoapExchange) : void

ResourceNotifyData
    ResourceNotifyData()
    handlePOST(CoapExchange) : void

ResourceNotifySleepy
    ResourceNotifySleepy()
    handlePUT(CoapExchange) : void
```

Figure 5.9. CoAP resource classes in IoT middleware.

Figure 5.9 shows class outline for CoAP Resources in IoT middleware. These resource classes extend CoapResouce in the Californium (Cf) CoAP framework. CoapResource is a basic implementation of a resource. Extend this class to write specific resources. Instances of type or subtype of CoapResource can be built up to a tree very easily. CoapResource uses four distinct methods to handle requests, that includes handleGET(), handlePOST(), handlePUT() and handleDELETE(). Each method has a default implementation that responds with a 4.05. That means the method is not allowed. Each method exists twice but with a different parameter, which are handleGET (Exchange) handleGET (CoAPExchange) for instance. The class is used internally in Cf to keep the state of an exchange of CoAP messages.

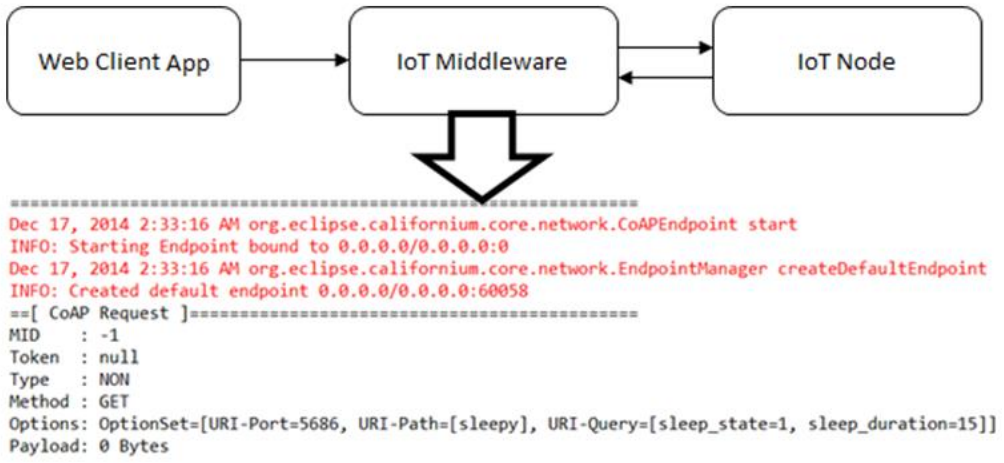


```c
void
init_resources(coap_context_t *ctx) {
    coap_resource_t *r;
    r = coap_resource_init((unsigned char *)"info", 4, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_info);
    coap_register_handler(r, COAP_REQUEST_PUT, hnd_put_info);
    r = coap_resource_init((unsigned char *)"units", 5, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_units);
    r = coap_resource_init((unsigned char *)"units/period", 12, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_units_period);
    r = coap_resource_init((unsigned char *)"sleepy", 6, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_sleepy);
    r = coap_resource_init((unsigned char *)"notify", 13, 0);
    coap_register_handler(r, COAP_REQUEST_GET, hnd_get_notify);
    coap_add_resource(ctx, r);
}
```

Figure 5.10. CoAP resource initialization in IoT node.

Figure 5.10 shows resource initialization function in IoT node. It invokes coap_resource_init() function in Libcoap CoAP library to initialize CoAP resource. And for the resource, using coap_register_handler() to register a handler function. For example, handler hnd_get_info is invoked then the resource is accessed by a request using GET method.

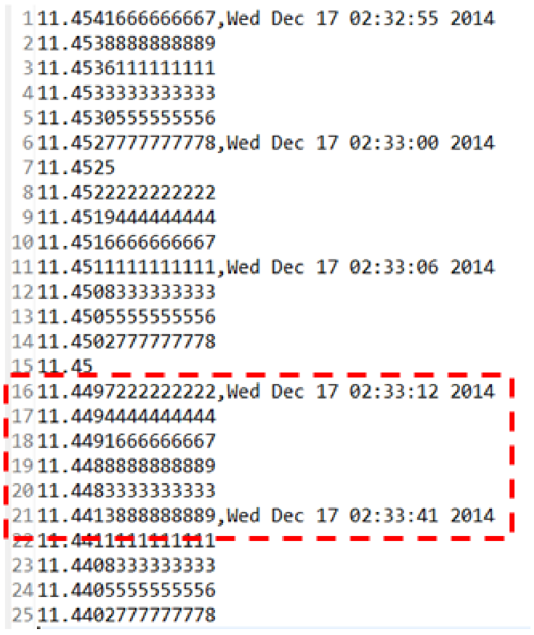
5.3.1 Extended RD and MQ for sleep scheme



```
==================================================================
Dec 17, 2014 2:33:16 AM org.eclipse.californium.core.network.CoAPEndpoint start
INFO: Starting Endpoint bound to 0.0.0.0/0.0.0.0:0
Dec 17, 2014 2:33:16 AM org.eclipse.californium.core.network.EndpointManager createDefaultEndpoint
INFO: Created default endpoint 0.0.0.0/0.0.0.0:60058
==[ CoAP Request ]================================================
MID    : -1
Token  : null
Type   : NON
Method : GET
Options: OptionSet=[URI-Port=5686, URI-Path=[sleepy], URI-Query=[sleep_state=1, sleep_duration=15]]
Payload: 0 Bytes
```

Figure 5.11. Implementation of request for sleep command.

Figure 5.11 shows a sleep command request from Web Client App to IoT node through IoT middleware. Web Client App send a request to the IoT middleware through HTTP with node ID, sleep_state and sleep_duration as parameters. And IoT middleware retrieve the IP address of IoT node via the node ID, and forward the sleep properties to IoT node through CoAP. Then IoT node invokes do_sleep() function for falling asleep.



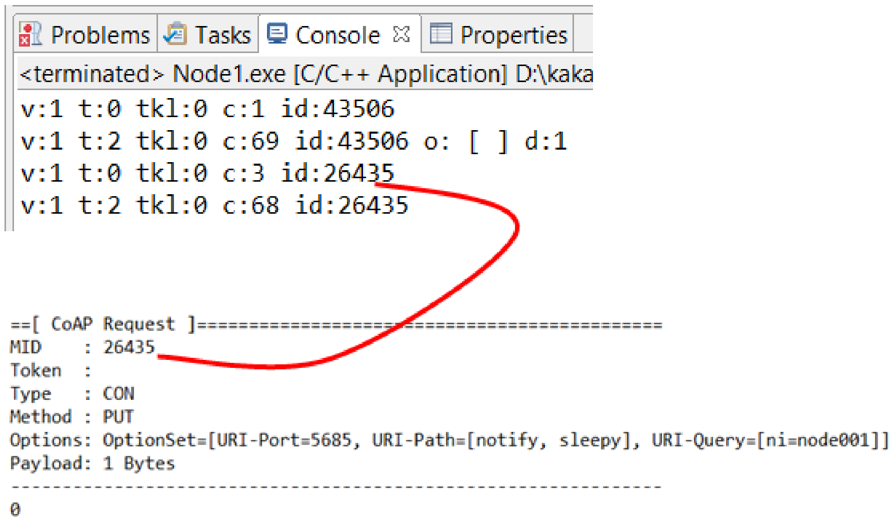
```
11.4541666666667,Wed Dec 17 02:32:55 2014
11.4538888888889
11.4536111111111
11.4533333333333
11.4530555555556
11.4527777777778,Wed Dec 17 02:33:00 2014
11.4525
11.4522222222222
11.4519444444444
11.4516666666667
11.4511111111111,Wed Dec 17 02:33:06 2014
11.4508333333333
11.4505555555556
11.4502777777778
11.45
11.4497222222222,Wed Dec 17 02:33:12 2014
11.4494444444444
11.4491666666667
11.4488888888889
11.4483333333333
11.4413888888889,Wed Dec 17 02:33:41 2014
11.4411111111111
11.4408333333333
11.4405555555556
11.4402777777778
```

Figure 5.12. Context data record list in the sleep of IoT node.

Figure 5.12 shows a context data record list when the IoT node falls asleep which are received a sleep command from IoT middleware. From figure 5.11, the IoT node received a CoAP Message with a uri-query that explains sleep_duration is 15 seconds. In this period, the IoT node will stop unit’s functions. The record list shows from “02:33:12” to “02:33:41”, in the period the IoT node had slept 15 seconds and waked up.

If Web Client App don’t want to subscribe the sleep state of a node, the Web Client App can request to IoT middleware for getting the sleep difference of the IoT node, and processing by itself. But the Web Client App needs the IoT middleware and IoT node deal with the task after the IoT node wakes up, then the Web Client App needs request to IoT middleware for subscribing sleep state of the IoT node.

When the IoT node wakes up after 15minuits as shown as fugure 5.13, then the IoT node will send a CoAP message to IoT middleware to update sleep information of the IoT node in the IoT middleware. Figure 5.13 shows the message of wake up command in the IoT node and IoT middleware. First window shows result of IoT node, which is sent by IoT node to IoT middleware with node ID, and in the second window shows result of IoT middleware it receive the message and update the sleep information of the IoT node. The CoAP resource of IoT middleware is “notify/sleepy”, that receives the message for update the information.



```
<terminated> Node1.exe [C/C++ Application] D:\kaka
v:1 t:0 tkl:0 c:1 id:43506
v:1 t:2 tkl:0 c:69 id:43506 o: [ ] d:1
v:1 t:0 tkl:0 c:3 id:26435
v:1 t:2 tkl:0 c:68 id:26435
```

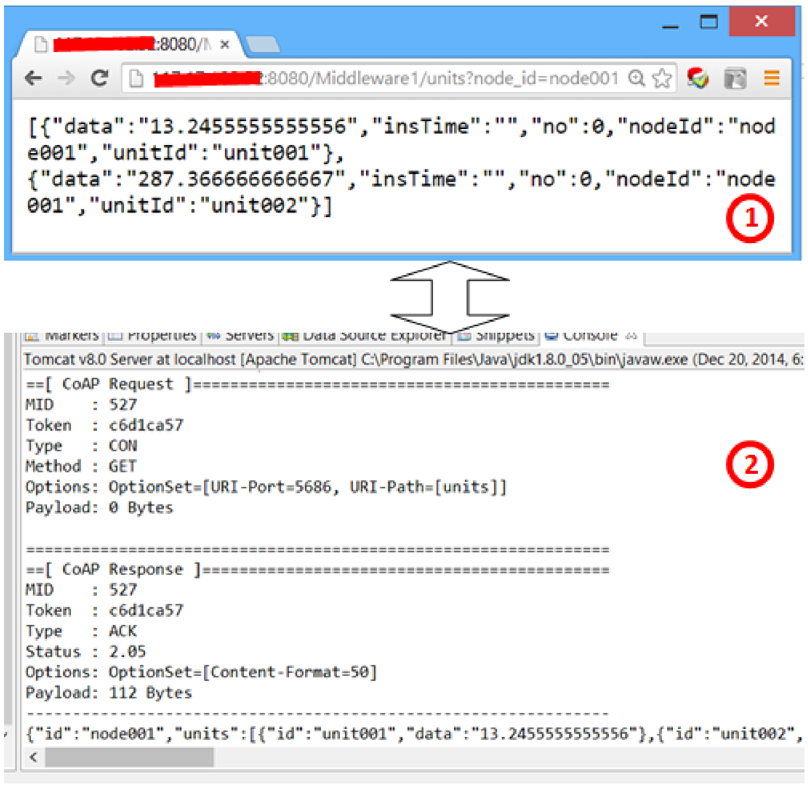
```
==[ CoAP Request ]============================================
MID     : 26435
Token   :
Type    : CON
Method  : PUT
Options: OptionSet=[URI-Port=5685, URI-Path=[notify, sleepy], URI-Query=[ni=node001]]
Payload: 1 Bytes
---------------------------------------------------------------
0
```

Figure 5.13. Wake-up message mapping for IoT node and IoT middleware.

5.3.2 Improved MQ for efficient context data collection

We presented tow kind of scheme for context data collection. Web Client App gets the context data from an IoT node through an IoT middleware. The Web Client App can gets real time data which are collected from an IoT node and gets saved data from an IoT middleware.

Figure 5.14 shows a result of real time context data collection. The first window shows a web browser as Web Client App that receives a real time context data from an IoT node through an IoT middleware. Second window shows a result of console in an IoT middleware. The IoT middleware receives a request from a Web Client App for collecting real time context data, and it requests to the IoT node which is acquiring by Web Client App.

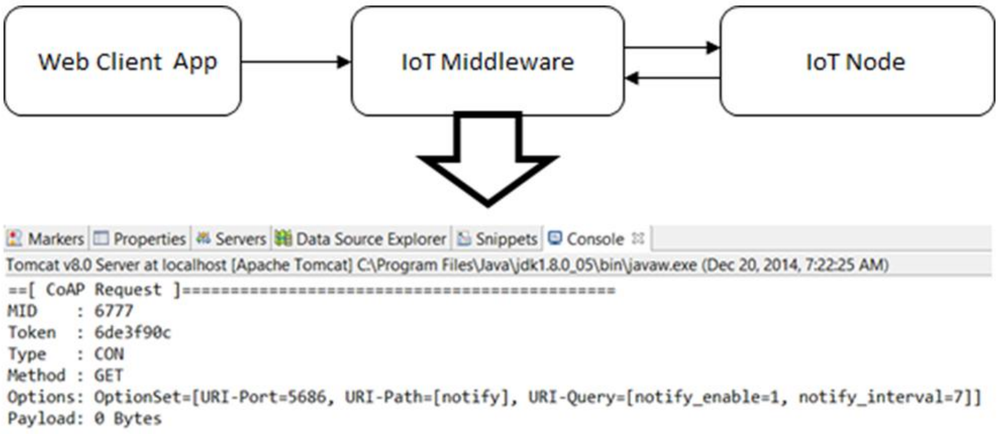


```
[redacted]:8080/M ×
[redacted]:8080/Middleware1/units?node_id=node001

[{"data":"13.2455555555556","insTime":"","no":0,"nodeId":"nod
e001","unitId":"unit001"},
{"data":"287.366666666667","insTime":"","no":0,"nodeId":"node
001","unitId":"unit002"}]
```

```
Markers | Properties | Servers | Data Source Explorer | Snippets | Console
Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_05\bin\javaw.exe (Dec 20, 2014, 6:
==[ CoAP Request ]=============================================
MID    : 527
Token  : c6d1ca57
Type   : CON
Method : GET
Options: OptionSet=[URI-Port=5686, URI-Path=[units]]
Payload: 0 Bytes

===============================================================
==[ CoAP Response ]============================================
MID    : 527
Token  : c6d1ca57
Type   : ACK
Status : 2.05
Options: OptionSet=[Content-Format=50]
Payload: 112 Bytes
---------------------------------------------------------------
{"id":"node001","units":[{"id":"unit001","data":"13.2455555555556"},{"id":"unit002",
```

Figure 5.14. Result of real time context data collection.

The other scheme for the context data collecting is getting saved data from an IoT middleware. The Web Client App needs send a command to change notify mode for the IoT middleware to collect context data from IoT node.

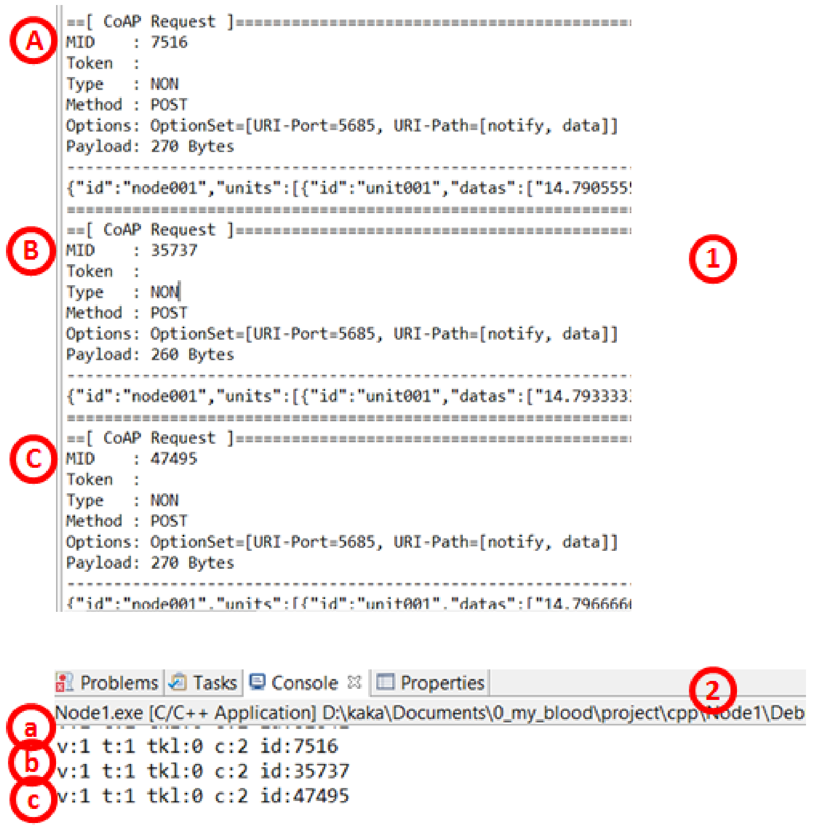


```
Markers | Properties | Servers | Data Source Explorer | Snippets | Console
Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_05\bin\javaw.exe (Dec 20, 2014, 7:22:25 AM)
==[ CoAP Request ]=============================================
MID    : 6777
Token  : 6de3f90c
Type   : CON
Method : GET
Options: OptionSet=[URI-Port=5686, URI-Path=[notify], URI-Query=[notify_enable=1, notify_interval=7]]
Payload: 0 Bytes
```

Figure 5.15. Implementation of request for notify command.

Figure 5.15 shows a notify command request from Web Client App to IoT node through IoT middleware. Web Client App send a request to the IoT middleware through HTTP with node ID, notify_enable and notify_interval as parameters. And IoT middleware retrieve the

IP address of IoT node via the node ID, and forward the notify properties to IoT node through CoAP. Then IoT node changes the information of notification and began to send context data to the IoT middleware.

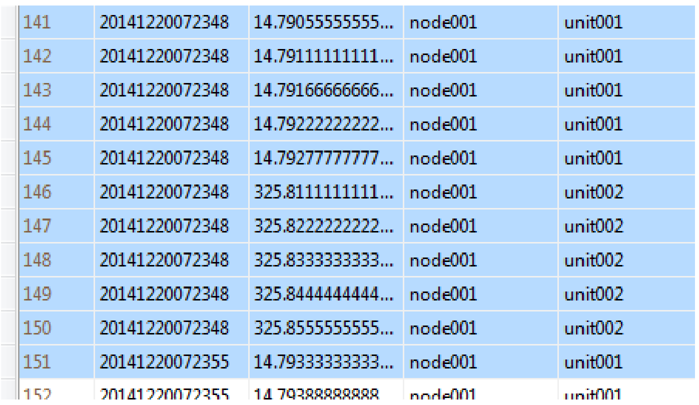


```
==[ CoAP Request ]=========================================
MID     : 7516
Token   :
Type    : NON
Method  : POST
Options: OptionSet=[URI-Port=5685, URI-Path=[notify, data]]
Payload: 270 Bytes
-----------------------------------------------------------
{"id":"node001","units":[{"id":"unit001","datas":["14.790555
===========================================================
==[ CoAP Request ]=========================================
MID     : 35737
Token   :
Type    : NON
Method  : POST
Options: OptionSet=[URI-Port=5685, URI-Path=[notify, data]]
Payload: 260 Bytes
-----------------------------------------------------------
{"id":"node001","units":[{"id":"unit001","datas":["14.793333
===========================================================
==[ CoAP Request ]=========================================
MID     : 47495
Token   :
Type    : NON
Method  : POST
Options: OptionSet=[URI-Port=5685, URI-Path=[notify, data]]
Payload: 270 Bytes
-----------------------------------------------------------
{"id":"node001","units":[{"id":"unit001","datas":["14.796666
```

```
Problems  Tasks  Console  Properties
Node1.exe [C/C++ Application] D:\kaka\Documents\0_my_blood\project\cpp\Node1\Deb
v:1 t:1 tkl:0 c:2 id:7516
v:1 t:1 tkl:0 c:2 id:35737
v:1 t:1 tkl:0 c:2 id:47495
```

Figure 5.16. Result of context data notification.

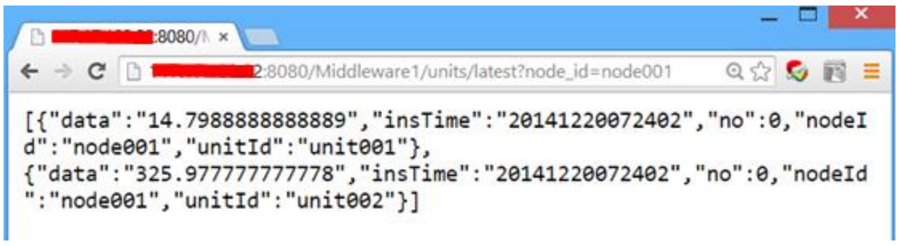
Figure 5.16 shows result of context data notification in the console of an IoT middleware and an IoT node. There are corresponding information in the both side in the figure such as “A” and ‘a’. The “A” means result of the IoT middleware that are requested by the IoT node using CoAP in NON-confirmable type.



141	20141220072348	14.79055555555...	node001	unit001
142	20141220072348	14.79111111111...	node001	unit001
143	20141220072348	14.79166666666...	node001	unit001
144	20141220072348	14.79222222222...	node001	unit001
145	20141220072348	14.79277777777...	node001	unit001
146	20141220072348	325.8111111111...	node001	unit002
147	20141220072348	325.8222222222...	node001	unit002
148	20141220072348	325.8333333333...	node001	unit002
149	20141220072348	325.8444444444...	node001	unit002
150	20141220072348	325.8555555555...	node001	unit002
151	20141220072355	14.79333333333...	node001	unit001
152	20141220072355	14.79388888888	node001	unit001

Figure 5.17. Result of saved context data in database.

Figure 5.17 shows data table in the SQL Server. These data are context data from IoT node. The Web Client App can retrieve these data from the IoT middleware.



```
[{"data":"14.7988888888889","insTime":"20141220072402","no":0,"nodeI
d":"node001","unitId":"unit001"},
{"data":"325.977777777778","insTime":"20141220072402","no":0,"nodeId
":"node001","unitId":"unit002"}]
```

Figure 5.18. Result of requesting notified context data.

Figure 5.18 shows result of context data which are requested by a Web Client App such as web browser. These data from an IoT middleware which are saved before.

5.4 IoT service for client application

In the service layer, we deploy sensor web provider, actuator web provider, app server and GIS provider [20]. Service Registry includes information of deployed element in the service layer. App server invokes service of sensor web provider, actuator provider and GIS provider to combine for providing a new service. Sensor web provides service about sensor web, Actuator web provides service about sensor web and GIS provides service about GIS. These applications fit to their own application environment before. The IoT middleware supports HTTP RESTful API to the Service Layer. We changed communication function to HTTP Web Client from Socket Client in Service Provider which are interacted with IoT middleware.

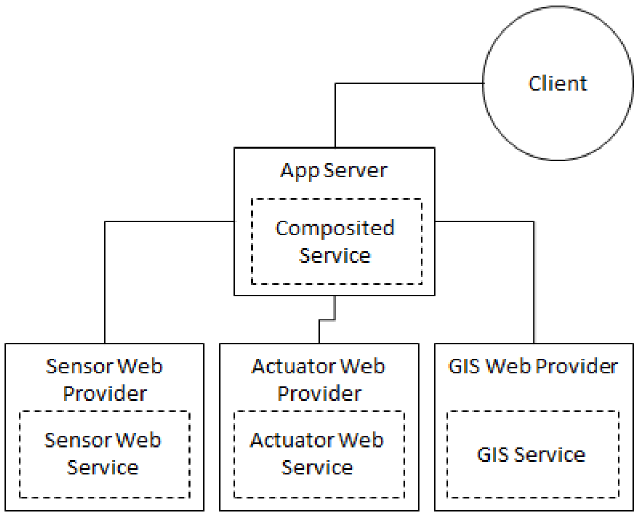


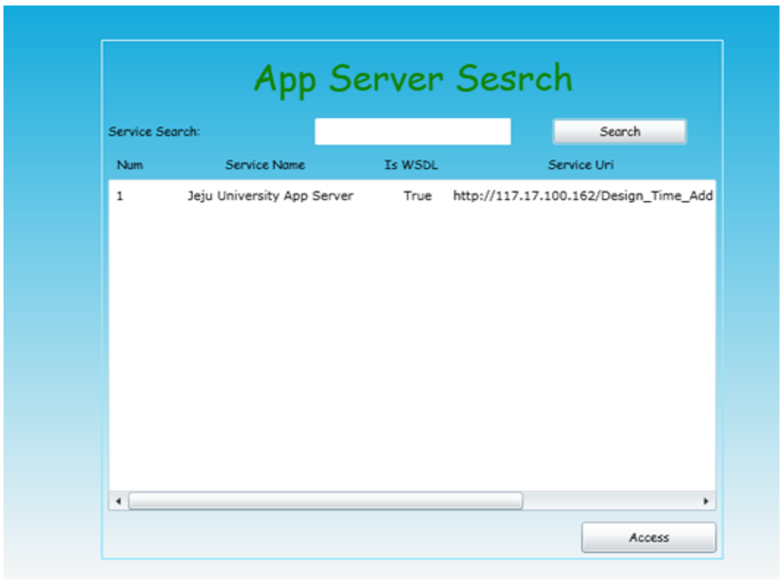
Figure 5.19. Service layer architecture for composited service.

In the sensor web provider, service Interface provides access interface to outside service. The main services offered by Sensor-Web module are Sensor-Web Content Service, Sensor-Web Provider Service and Sensor-Web Sensing Service.

Sensor-Web Content Service is used for middleware configuration Management and Sensor Information Management. Middleware Configuration Management involves the activities of saving and managing middleware ID, IP addresses and service access privilege information. Sensor Information Management involves the creation and management of sensor information such as node name, ID and sensing type. Sensor-Web Provider is utilized for Sensor Searches, Sensor Information supply and Sensing Data supply. Sensor Searches returns sensor ID list for a specific search keyword. Sensor Information supply provides related information such as sensor node name, sensing type for a sensor ID. Sensing Data supply for a sensor ID, provides the corresponding real time sensing data, the client referred here means all objects that use supply service. Sensor-Web Sensing Service is used for Sensor State management and as Sensing Data receiver. Sensor State management involves keeping record of the sensing state of sensors. Sensing Data Receiver provides the facility of receiving sensing data collected from sensor middleware and saving it. Service Control involves the launching and closing of Provider Service, Sensing Service and Content Service. Service State Viewer manages a service's state of execution. XML Configuration provides WCF Service state. Database saves sensor related information such as name, ID and sensing data.

Actuator Web provider consists of Actuator Web Content Service, Actuator Web Provider Service and Actuator Middleware Service. Actuator Web Content Service is composed of Actuator Middleware configuration management and Actuator information management. Middleware configuration management involves the creation of middleware ID, IP address and service access right information and management. Actuator information management involves the creation of actuator node name, ID, sensing type information and management. Actuator Web Provider Service offers actuator information, information search and control service. Actuator Web Provider Service is composed of Actuator searcher, Actuator info supply and Actuator control. Actuator Searcher returns actuator ID list in response to the search keyword. Actuator Info Supply offers actuator node name, actuator state information for a given actuator ID. Actuator control offers actuator remote control interface. Actuator middleware service manages message parsing and sending the middleware mapping table. Mapping Table Management saves actuator ID in memory and manages actuator middleware address information. Message Parser converts the received actuator middleware messages to pre-defined format. Service Control is composed of Provider Service Control, Middleware Service Control and Content Service Control. It also provides the start and stop control for Provider Service, Middleware Service and Content Service. Service State Viewer shows service state (service interface for on / off information). XML Configuration arranges WCF Service configuration. Database saves actuator node information (actuator name, ID, model, Attribute) that the service offers. Actuator Mapping Table saves actuator connect actuator ID and the video relation information between actuator middleware address.

Service client Application is used for service registered user search service what the user want and access the service. The client can see building information in GIS by means of accessing service. In this view, the user can look over room status information and object state.



Num	Service Name	Is WSDL	Service Uri
1	Jeju University App Server	True	http://117.17.100.162/Design_Time_Add

Figure 5.20. Client main page.

Figure 5.20 shows the client application service search execution screen. When client Web page is accessed, the service search screen like in the above figure is shown. The user enters the search keyword and hits the search button. Service registry responds with a list of application server provider services. The user then selects any of the services and can access that. There is object binding service list and user select one of the list and click the “access” button then user can use this service. There is a search function for user input the service keyword and search services.

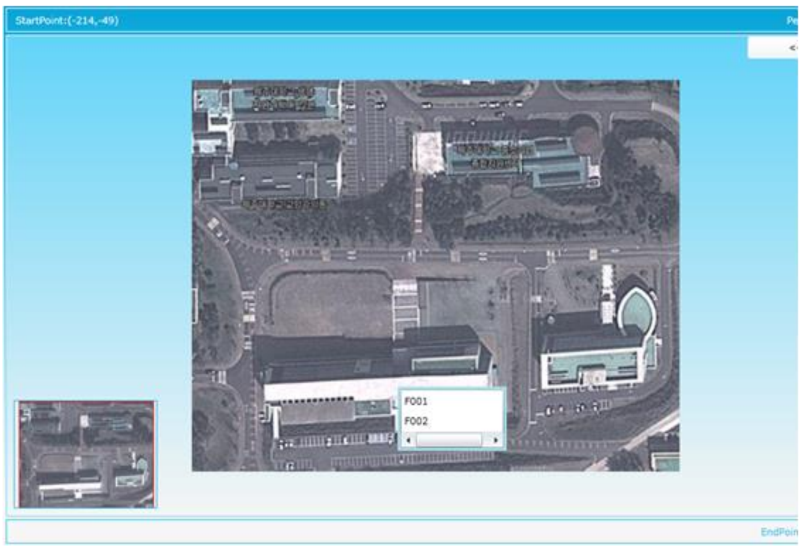
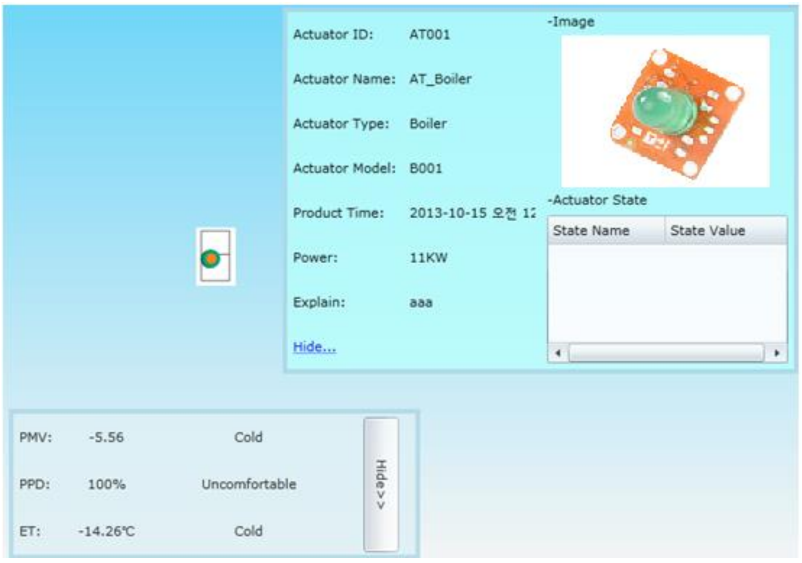


Figure 5.21. Client total map page.

Figure 5.21 is outdoor map viewer execution screen. The client accesses application server using the service searcher which shows the total map information first. This view shows the registered building information on the map. It shows floor list when we click a building which are registered in database by user. This service is provided by GIS Service Provider. It get the GIS information through the service from database of GIS Service Provider. After this screen, we can see the object such as actuators and sensors.



Actuator ID:	AT001
Actuator Name:	AT_Boiler
Actuator Type:	Boiler
Actuator Model:	B001
Product Time:	2013-10-15 오전 12
Power:	11KW
Explain:	aaa

Hide...

-Image

-Actuator State

State Name	State Value

PMV:	-5.56	Cold
PPD:	100%	Uncomfortable
ET:	-14.26℃	Cold

Figure 5.22. Client room page.

Figure 5.22 is indoor map viewer execution screen. The indoor (floor) map viewer shows room, sensor node and actuator node information for a specific floor. The map of the floor shows indoor registered sensor and actuator nodes. The bottom of figure shows the indoor comfort index and environment state for a selected (right-clicked) room. The right part of figure shows the detailed information about a selected (right-clicked) actuator node. In the part of screen, it also shows the screen which appears when a sensor node on the floor map is selected (right-clicked). The window shows sensor node detailed information and sensing data.

6 Performance evaluation

We use a Java framework to implement IoT middleware and using a c library to implement IoT node. IoT node is implemented in Linux C compile environment because of IoT node working in the constrained environment. IoT middleware is implemented in java compile environment. The IoT middleware is a web application which support web link to be accessed by user such as web client and HTTP client.

Using the system which we implemented, we test the performance of the message interaction for sleepy schemes. In this experiment, when an IoT node wakes up from sleep, then a Web Client App will get context data of IoT node. There is tow kind of scheme for sleep scheme (section 4.4).

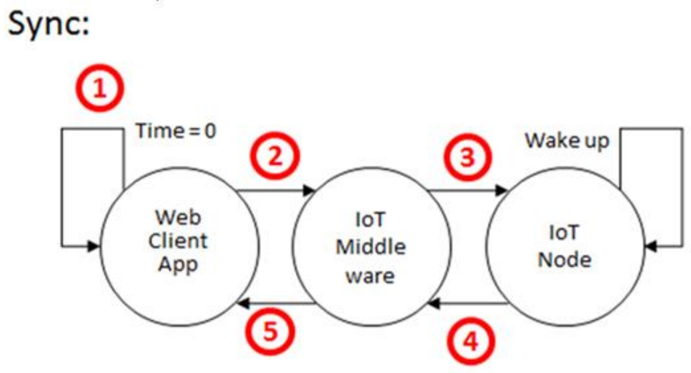


Figure 6.1. Message transfer process for synchronous sleep scheme.

Figure 6.1 shows message transfer process for sleep scheme of synchronous. In the synchronous scheme for sleepy IoT node, the state information of sleepy time is known by the Web Client App. When the time of sleep is up in the Web Client App, then it sends a request to the IoT middleware for acquiring context data of IoT node. Then the IoT middleware sends a CoAP request message to the IoT node to get context data. And the IoT middleware responds the context data to the Web Client App.

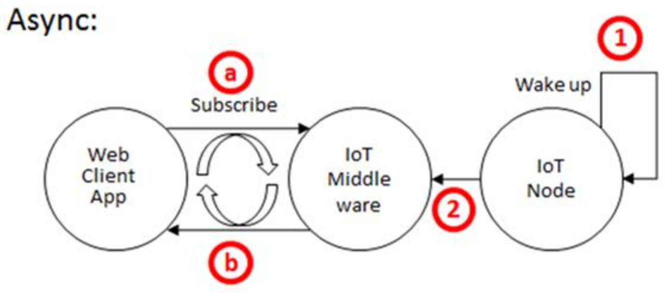
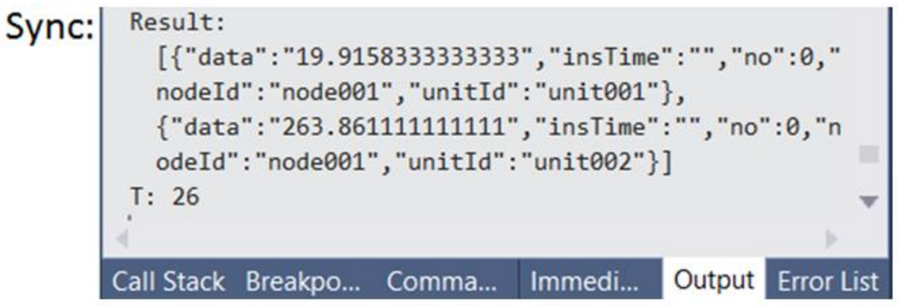


Figure 6.2. Message transfer process for asynchronous sleep scheme.

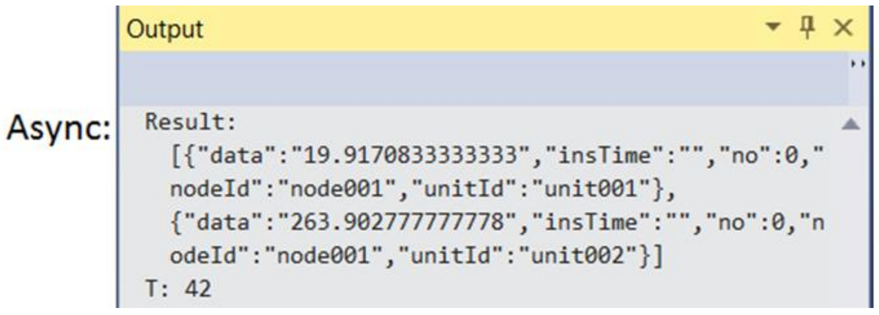
Figure 6.2 shows message transfer process for sleep scheme of asynchronous. In the asynchronous scheme for sleepy IoT node, the Web Client App sends a request message to IoT middleware for subscribe the IoT node. After the IoT node wakes up from sleep, it will sends a request to notify the IoT middleware for the node state awake. And the IoT middleware gets the context data from IoT node to publish the Web Client App. The subscription and publication process that we use loop to implement.



```
Result:
  [{"data":"19.9158333333333","insTime":"","no":0,"
  nodeId":"node001","unitId":"unit001"},
  {"data":"263.861111111111","insTime":"","no":0,"n
  odeId":"node001","unitId":"unit002"}]
T: 26
```

Figure 6.3. Result of context data and time estimation for synchronous sleep scheme.

Figure 6.3 shows result of the test for sleep scheme of asynchronous. In this result, there are result of context data of IoT node and time estimation for the process. The unit of time is millisecond. From the result, the timestamp of the process is 26 ms.



```
Result:
  [{"data":"19.9170833333333","insTime":"","no":0,"
  nodeId":"node001","unitId":"unit001"},
  {"data":"263.902777777778","insTime":"","no":0,"n
  odeId":"node001","unitId":"unit002"}]
T: 42
```

Figure 6.4. Result of context data and time estimation for asynchronous sleep scheme.

Figure 6.4 shows result of the test for sleep scheme of asynchronous. In this result, there are result of context data of IoT node and time estimation for the process. From the result, the timestamp of the process is 42 ms. Therefore synchronous and asynchronous scheme for sleep node takes same timestamp. But, from the figure 6.1, synchronous scheme has request and response process in 3 and 4. In the asynchronous scheme, the IoT node pushes the data to IoT middleware to simplify the process in the constrained environment.

7 Conclusion

Internet Engineering Task Force (IETF) presented CoAP for constrained environment in the Internet of Things (IoT) and there are several extensions for CoAP. For CoAP node, there is identification of node and solution for dynamic IP of node [10] [12]. We also presented identification for multiple units in CoAP node [11]. In this study, we have presented IoT node based on registration and discovery of multiple units, sleep scheduling and context data collecting functionalities which are built on IoT middleware for interaction of IoT elements. IoT node is a CoAP node, which works in constrained environment using low power and limited RAM and ROM. So, we designed and implemented the IoT node to fit the requirement. IoT middleware works with IoT node via CoAP in the Constrained RESTful Environment (CoRE).

We proposed registration and discovery for unit ID based composite node in the constrained environment using CoAP, and using Resource Directory (RD) and CoAP Message Queue (MQ) mechanism to present an enhanced mechanism for management of sleep node and context data collection. This mechanism is tested with IoT node and IoT middleware based on CoAP. With service providers in the service layer, we used part of implemented IoT system from previous work which included several service providers to communicate with middleware using specific message format through TCP [20]. We also presented IoT middleware, which supports HTTP web API for providers to implement services to be used by client. Service provider can be an application in several platforms. The middleware can be accessed heterogeneous applications to implement their own user service.

Using the system which we presented, we tested the performance of the message interaction for sleepy schemes. From the result of the test, synchronous and asynchronous scheme for sleep node takes almost same timestamp. However, the asynchronous scheme based on MQ which simplifies the process in the constrained environment.

References

[1] Luigi Atzori, Antonio Iera, Giacomo Morabito, "The Internet of Things: A survey", Computer Networks, Vol. 54, Iss. 15, pp. 2787-2850, 2010.

[2] Jamal N. Al-Karaki, Kwang-Cheng Chen, Giacomo Morabito, Jaudelice de Oliveira, "From M2M communications to the Internet of Things:Opportunities and challenges", Ad Hoc Networks, Vol. 18, pp. 1-2, 2014.

[3] Z. Shelby, K. Hartke, C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[4] A. Rahman, "Enhanced Sleepy Node Support for CoAP", Internet-Draft, draft-rahman-core-sleepy-05, February 2014.

[5] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[6] M. Meddeba, M. Ben Alayaa, T. Monteila, A. Dhraiefc, K. Driraa, "M2M Platform with Autonomic Device Management Service", Procedia Computer Science, Vol. 32, pp. 1063-1070, 2014.

[7] Intel Galileo Board, <http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html>.

[8] Z. Shelby, C. Bormann, "CoRE Resource Directory", Internet-Draft, draft-ietf-core-resource-directory-01, December 2013.

[9] M. Koster, A. Keranen, J. Jimenez, "Message Queueing in the Constrained Application Protocol (CoAP)", Internet-Draft, draft-koster-core-coapmq-00, July 2014.

[10] K. Li, G. Wei, "CoAP Option Extension: NodeId", Internet-Draft, draft-li-core-coap-node-id-option-01, June 2014.

[11] Y. Hong, Y. Choi, D. Kim, M. Khan, W. Jin, "CoAP Endpoint Unit Identification for Multiple Sensor and Actuator in a Node", Internet-Draft, draft-hong-core-coap-endpoint-unit-id-00, July 2014.

[12] O. Kleine, "CoAP Endpoint Identification", Internet-Draft, draft-kleine-core-coap-endpoint-id-00, April 2014.

[13] M. Nottingham, "Web Linking", RFC 4287, October 2010.

[14] M. Koster, A. Keranen, J. Jimenez, "Message Queueing in the Constrained Application Protocol (CoAP)", Internet-Draft, draft-koster-core-coapmq-00, July 2014.

[15] libcoap: C-Implementation of CoAP, <http://libcoap.sourceforge.net/>.

[16] Martin Lanter, "Scalability for IoT Cloud Services", Master Thesis, Department of Computer Science, ETH Zurich, 2013.

[17] K. Paridel, E. Bainomugisha, Y. Vanrompay, Y. Berbers, and W.D. Meuter, "Middleware for the Internet of Things, Design Goals and Challenges", ECEASST Journal, Vol. 28, ISSN 1863-2122, 2010.

[18] An Oracle White Paper, "Internet of Things: Role of Oracle Fusion Middleware", 2014.

[19] Shirin Zarghami, "MIDDLEWARE FOR INTERNET OF THINGS", Master Thesis, University of Twente, 2013.

[20] Chen Nan, "Design and Implementation of Actuator Web and Middleware for Controlling Indoor Environment", Master Thesis, Jeju national university, Republic of Korea, 2013.

[21] Jin Wen-Quan, Kim Do-Hyeun, "Implementation and Experiment of CoAP Protocol Based on IoT for Verification of Interoperability", The Journal of the Institute of Webcasting, Internet and Telecommunication, Vol. 14, Iss. 4, pp. 7-12, 2014.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L.Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616, June 1999.

[23] Californium (Cf) CoAP framework in Java, <http://people.inf.ethz.ch/mkovatsc/californium.php>.

[24] Copper (Cu), <https://addons.mozilla.org/en-US/firefox/addon/copper-270430/>.

[25] cJSON, <http://sourceforge.net/projects/cjson/>.