Journal of Theoretical and Applied Information Technology
31st January 2018. Vol.96. No 2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TOWARD IMPLEMENTATION OF ONEM2M BASED IOT PLATFORM

[1]VINCENTIUS CHRISTIAN ANDRIANTO, [2]JUNHUY LAM, [3]RYAN NATHANAEL SOENJOTO WIDODO, [4]SANG-GON LEE*, [5]HOON-JAE LEE, [6]HYO-TAEK LIM

Department of Ubiquitous IT, Division of Computer & Information Engineering, Dongseo University,

Busan, South Korea

E-mail: [1]vincent.ch93@gmail.com, [2]timljh@msn.com, [3]ryannswidodo@gmail.com,
[4]nok60@dongseo.ac.kr, [5]hjlee@dongseo.ac.kr, [6]htlim@dongseo.ac.kr
*corresponding author

ABSTRACT

A Many organizations have its own proprietary Internet of Things (IoT) architecture. This creates additional processes such as system extension, integrating new data, managing device, and adding security become difficult, especially when it involves multiple platforms. In order to solve this problem, the standardized oneM2M architecture was initiated. OCEAN is an IoT platform based on oneM2M standards. This work presents an implementation of oneM2M based IoT platform using OCEAN, in which the system is built from the proposed smart home scenario.

Keywords: *Internet of Things, M2M, MQTT, OCEAN, oneM2M.*

1. INTRODUCTION

Internet of Things was first declared in 1999 by Kevin Ashton in the context of supply chain management [1], the definition of things has changed as technology evolved. However, the main goal remains the same, which is to provide a solution that involves communication between the machines and automates without human intervention [2].

Task of the IoT device management is very challenging due to the heterogeneity of things in terms of communication, types of data generated, access control for users, ease of adding and deleting IoT device description, and etc. [3].

These issues make it difficult to extend systems to support new services, integrate new device or data, and interoperate with other Machine-to-Machine (M2M) systems. The "oneM2M Global Initiative" proceed to standardize a common M2M service layer platform for globally applicable and access-independent M2M services [4].

One of the IoT platform based on oneM2M standards is OCEAN (Open Alliance for IoT Standard). OCEAN aims to develop IoT platforms, products, and services by the widespread adoption of IoT standards-compliant. OCEAN has IoT server and device platform; namely, Mobius and &Cube respectively [5].

In this paper, we demonstrated a reference exploitation of OCEAN, to implement a simple Smart Home IoT service scenario. For the first step of implementation, we modeled physical things into virtual things based on oneM2M entity modeling. The second step is to design a communication and message architecture between each entity. The third step is to build the system architecture based on the existing entity. The final step is to test the whole system using a use case scenario.

This paper is organized as follows. In Section II, we explain the background and related works of our approach. In Section III, we address design and implementation. In Section IV, we describe our proposed system architecture. Finally, Section V concludes the paper and discusses future work.

2. BACKGROUND AND RELATED WORK

2.1 Message Queue Telemetry Transport (MQTT)

MQTT was invented by Dr. Andy Stanford-Clark from IBM, and Arlen Nipper from Arcom (now Eurotech) in 1999. It is a connectivity protocol for M2M or IoT. MQTT was designed as an extremely lightweight publish/subscribe messaging transport. It has low power usage, small header, and efficient distribution to one or many clients [6].

The principle of publish/subscribe messaging transport is entities which are interested in certain

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

information will register their interest. The process of registering an interest is called subscription, and the interested entity is called subscriber. Entities, which want to produce certain information, called publishers. There is a broker between these two entities, which ensures data delivery from the publishers to the subscribers [7].

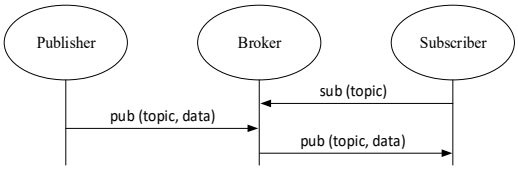

Figure 1:Topic-based pub/sub system.

There are three kinds of pub/sub systems: topic based, type based, and content-based [8]. This paper is using a topic-based as the pub/sub systems. With topic-based systems, the list of topics is known in advance, e.g., during the design phase of an application. The publishers and subscribers both know which topic they want to publish or subscribe to.

Topic-based pub/sub system is the simplest system compared with the other. The communication model of a topic-based pub/sub system is shown in Figure 1. A subscriber sends a sub(topic) message to the broker, whereas a publisher sends a pub(topic, data) message which contains the data to be published to its related topic. The broker will transfer the pub(topic, data) message to the subscriber if the topic is matched. A single message from the publisher may be distributed to multiple subscribers, as long as the topic is matched [7].

MQTT supports basic end-to-end Quality of Service (QoS) [9]. MQTT has three QoS levels, depending on how reliable messages should be delivered to its receiver. QoS level 0 is the simplest one, where messages are delivered either once or not at all, no retransmission or acknowledgment is defined. QoS level 1 provides a more reliable transport, messages will be retransmitted until acknowledged by the receiver. The highest QoS, level 2, ensures not only the reception of the messages but also that they are delivered only once to the receiver. It's up to the application to select which QoS should they use for the systems [7].

2.2 OneM2M: An M2M Standards

Currently, many industries rely on vertically specified machine-to-machine (M2M) solutions that typically designed for that industry's service. More industries are developing their own M2M system. Thus, there is a need to develop a horizontal common platform across of industry, which can increase the efficiency of M2M deployment. A common M2M service platform also needed to handle heterogeneous M2M systems.

oneM2M is an international partnership project, established in order to maintain the M2M service layer specifications related to M2M solutions. This common service layer is used for various hardware and software to ensure M2M devices can communicate on a global scale, this is the main objective of the oneM2M.

The oneM2M system is formed by functional entities called nodes. These are known as application dedicated node (ADN), application service node (ASN), and infrastructure node (IN). Each node consists of one oneM2M common service entity (CSE) or one oneM2M application entity (AE). The CSE is an entity that has a set of service functions called common services functions (CSFs). The AE is an entity that provides application logic for end-to-end M2M solutions [1].

oneM2M used a resource-based data model [11]. All of the oneM2M services are represented as resources. A resource is a data structure that can be uniquely addressed by a uniform resource identifier (URI). Figure 2 shows the resources structure of oneM2M services. oneM2M use a tree-based resources structure to explain the connection between each resource. <CSE> can have multiple <AE>, but one <AE> has to be under one <CSE>. Each <AE> can have multiple <container> indicating the data inside that <AE>. Each <container> has one <contentInstance> which represent a data instance in the container. The operations of the resources can be achieved by the (CREATE, RETRIEVE, UPDATE, and DELETE) CRUD commands [10].

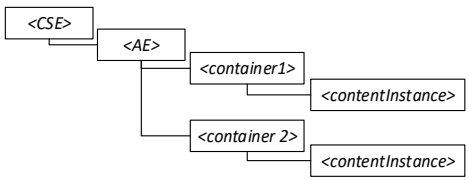

Figure 2: An example of oneM2M resource structure.

oneM2M has adapted some of the standardized communication protocol, such as MQTT, REpresentational State Transfer (REST), and Constrained Application Protocol (CoAP). These

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

three protocols are mentioned in oneM2M technical specifications [11]. In this paper, we used MQTT and REST as our communication protocol.

MQTT connection in oneM2M uses parameters that are serialized into the payload of an MQTT publish packet. It has request and response primitives. When an MQTT client sends a request message to the broker, the MQTT receiver response it with a response message. The request shall contain the mandatory parameters such as operation, to, from, and request identifier [11].

An example of an MQTT request message serialized using JSON is: {"op": 1, "to": "//xxxxx/2345", "fr": "//xxxxx/99", "rqi": "A1234", "ty": 18, "pc":{"m2m:sch": {"rn":"schedule1", "se": "* 0-5 2,6,10 * * * *"}}, "ot": 20150910T062032}.

An example of an MQTT response message is: {"rsc": 2000, "rqi": "A1234", "pc": {"m2m:sch":{"se": "* 0-5 2,6,10 * * * *"}}, "to": "//xxxxx/2345", "fr":"//xxxxx/99"}. Where op is a short name of operation parameter, rsc is a short name for response status code parameter, to is a short name of to parameter, fr is a short name of from parameter, rqi is a request identifier, ty is a resource type, pc is a content, and ot is an originating timestamp [11].

2.3 OCEAN: An IoT Platform Based on oneM2M Standards

OCEAN is an IoT platform based on oneM2M standards initiated by Korea Electronics Technology Institute (KETI) in 2010. OCEAN aims to develop and commercialize oneM2M standards-compliant platforms, products, and services. OCEAN has a server and gateway platform namely Mobius and &Cube. There are several variants of each platform provided by OCEAN.

Mobius server platform manages information of various IoT devices installed. Mobius controls access of devices, manages authentications and users, and provides many IoT services such as MQTT server, HTTP server, and database connector. Mobius platform acts as a bridge between devices and applications in order for them to communicate with each other. The device sends the data to the server platform, then the platform will store the data and forwards it to the application with the matching access rights. The server contains four main parts, which are Mobius MQTT proxy, MySQL database, Mobius REST Server, and MQTT broker.

&Cube provides common service functions for IoT systems, working together with the Mobius platform, including sending, managing, and receiving control commands for end-node devices. &Cube supports various protocol bindings in order to communicate with the Mobius, including REST, MQTT, and CoAP [5].

3 DESIGN AND IMPLEMENTATION SMART HOME SCENARIO

Our IoT Smart Home scenario consists of IoT server, IoT gateway, IoT end-user application, end-node device, one sensor and one actuator. We use basic temperature and humidity sensor, and Light Emitting Diode (LED) for the actuator. These sensor and actuator are connected directly to the end-node device. When there is a 1.0 point change in temperature or humidity, the end-node device will publish the data to the IoT server through IoT gateway. Our IoT end-user application can add, view, edit, or remove resources inside the server. For example, it can view the current value of the temperature and humidity sensor or turn on or off the LED. Registration of the resources is done by the IoT gateway.

3.2 Resource-based Information Modelling

From our scenario, we model our system to the resource-based information. In oneM2M system, the information inside the network will be stored in the system as resources as mentioned in II. In our case, the resource-based information will be stored inside the MySQL database reside at the server. In order to manipulate the resource information, actions such as CRUD can be applied.

Defined resource types in our system are the system resource and application data resource. System resources are Common Service Entity (<CSE>) the Mobius server and Application Entity (<AE>) the &Cube gateway. Application data resources are sensors and actuators. We model our sensor and actuator as a <container>. We named our temperature sensor (<temp1>), humidity sensor (<hum1>), and the LED (<led1>) respectively, their value will be stored inside the <contentInstance> of the respective container. Resource tree structure of smart home scenario is shown in Figure 3.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

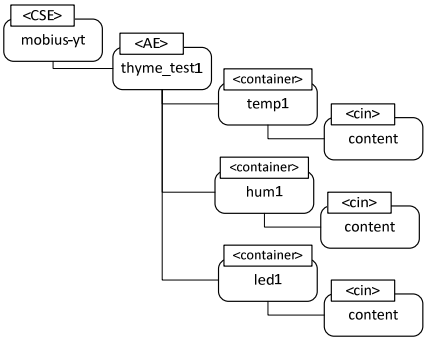

Figure 3: Resource tree structure of smart home scenario.

When resources are created in the server's database, attributes and child resources of the respective resource also created at the same time. Square boxes with round corners are used for attributes and square boxes are used for resources as shown in Figure 4 [11].

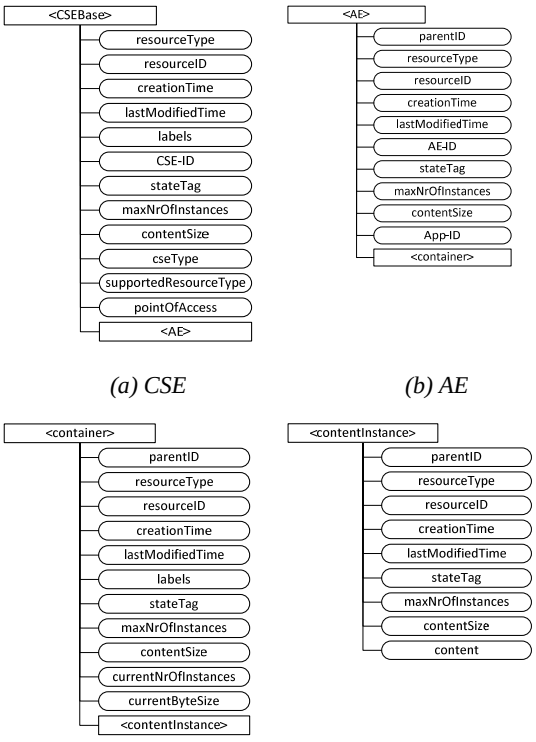

Figure 4: Diagram of respective resources.

3.3 oneM2M Communication Protocol

In order to connect each resource reside in the system, it requires service layer function in oneM2M architecture. The oneM2M protocol specification will cover resource details, procedures among oneM2M entities, and message sequences that cross the reference points. After we have our resource-based structure, the next step is to design how these system resources are connected with each other and processes application data resources.

OCEAN has several variants of server and gateway platform. Each of them has their own specifications. In this paper, we used Mobius: Yellow Turtle version 2.0.5 for our server and &Cube: Thyme version 1.5 for the gateway. These two platforms use MQTT for their communication channel. We chose MQTT over CoAP because MQTT runs over Transmission Control Protocol (TCP) so that it become more reliable compared to CoAP. For the security aspect, we can enhance the security of the transport level of TCP using Transport Layer Security (TLS).

Communication between gateway and server runs through MQTT connection. As mentioned in section II, MQTT uses pub/sub system as their communication scheme. End-node device and gateway are connected directly through a Universal Serial Bus (USB) cable. This can be replaced with another connection such as Ethernet or Wi-Fi. End-node device connected directly to the sensors and actuators. The sensors will publish the data to the gateway only when there is a change in the sensor's value.

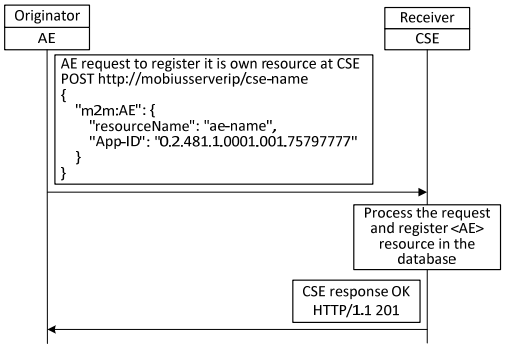

Figure 5: AE resource registration to CSE.

Mobius also provides a REST API through the HTTP server in order to communicate with the end-user application. The end-user application will request or provide data to the Mobius server by using CRUDN commands. It means that end-user

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

application can view and modify the resource on the server. These two communication channels operate independently.

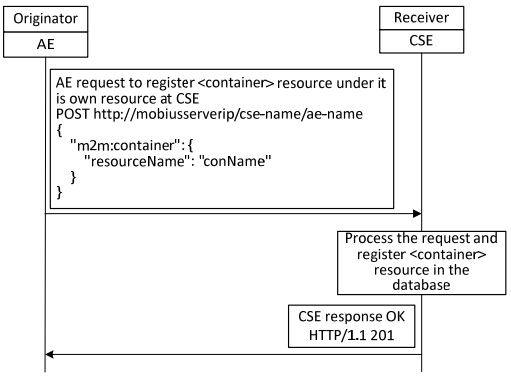

Figure 6: Container resource registration to CSE.

Resource registration must be done first before the system resources can communicate through MQTT connection. The registration can only be done through the REST API by using CREATE command. Resources created are stored inside the server's database as shown in Figure 5 and 6. After all resources registered on the server, the gateway can start to pub/sub to the respective <container>.

In the MQTT protocol, each MQTT client shall subscribe to the MQTT broker to receive messages [11]. After each MQTT client connects to the broker, it can start to communicate through MQTT connection. Topics that are used between MQTT client are "/oneM2M/req/<originator>/ <receiver>" for the request messages and "/oneM2M/resp/<originator>/<receiver>" for response messages. The message transferred between these two clients is serialized using JSON as mentioned in II. For example, the request message published by <AE> in Figure 8 will look like this: {"op": 1, "to": "//cse-name/ ae-name/conName", "fr": "//AE-ID", "rqi": "A1234", "ty": 4, "pc":{"m2m:cin":{"con": "content"}}, "ot": 20160910T073423}. It means that the request is from <AE> with ID "AE-ID" to the <container> named "conName" located under "/cse-name/ae-name/conName" and the message is "content". Thus, the receiver shall response with a response message, which will be look like this: {"rsc": 2001, "rqi": "A1234", "pc": {"m2m:cin":{"con":"content"}}, "to": "//cse-name/ae-name/conName","fr":"//CSE-ID"}, which means the response code is "2001" for the request message from originator.

The end-user application can do the CRUD command through the HTTP connection. It can READ the sensor's data using HTTP GET command or update the actuator's data using HTTP POST command to the respective container as shown in Figure 7.

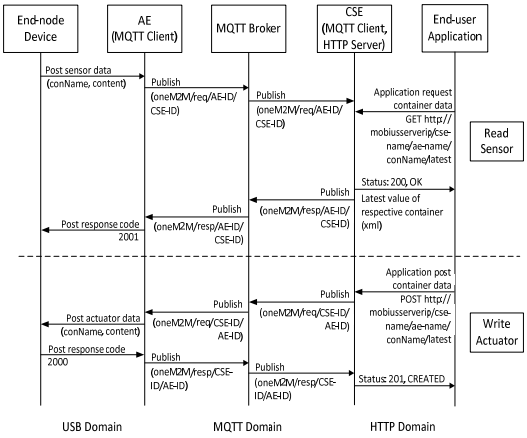

Figure 7: Message communication diagrams of IoT system.

4 SYSTEM ARCHITECTURE

We designed our architecture based on the service and communication diagram as shown in Figure 7. Our device architecture has four main parts, which are an IoT application, IoT server, IoT gateway, and end-node device.**IoT Server**

The Mobius server contains Mobius MQTT Proxy, MySQL database, and Mobius REST Server. Mobius: Yellow Turtle version 2.0.5 was setup on a computer running Linux OS Ubuntu Server 14.04.4. The database used is MySQL version 5.6.30. The requirement of Mobius's web server that provides the REST API and the Mobius's MQTT proxy is node.js version 4.4.3. The last component is the MQTT broker that provides the communication channel for the gateway to transfer the information to the server.

In Figure 8, it shows that there are two Linux Container Server with two Internet Protocol (IP) addresses, 10.0.3.20 (Container 1) and 10.0.3.21 (Container 2) respectively. Container 1 is hosting the Mobius Webserver, MySQL database, and MQTT proxy, while Container 2 is hosting only the MQTT broker.

The Mobius web server is mainly used to perform three tasks. The first one will be providing the REST API to the mobile/web applications (user interface) to perform CRUDN commands. The

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

second task will be retrieving the data from the database and converting it to the XML/JSON format when the mobile/web applications request for a particular information. The third task is receiving the data in XML/JSON format from the application and converting it to the database format then storing it there. The Mobius webserver is running on port 7579. In MySQL database, all of the tables were created in accordance with the oneM2M specification.

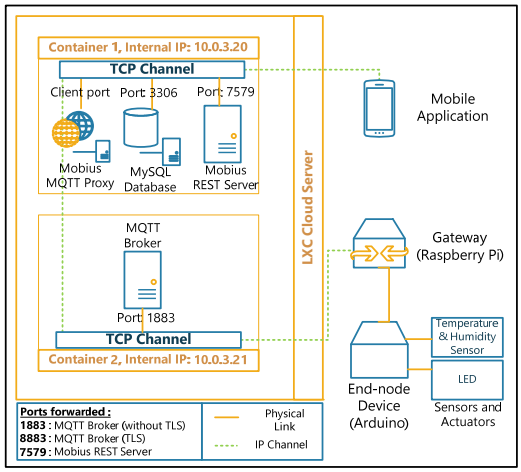

Figure 8: Device architecture of IoT system.

On the MQTT side of the Mobius server, it requires MQTT broker and MQTT proxy, and both of these services have to be on two different servers. In Figure 8, it shows that both MQTT proxy and gateway can connect to MQTT broker through port 1883. When the gateway sends data to the MQTT broker, the MQTT broker then forwards these message to the MQTT proxy. MQTT proxy then will translate the messages and store them in the database. This communication also works in the opposite direction. MQTT broker will also reply to the message from the gateway using a oneM2M response status codes as mentioned in [11].

4.2 IoT Gateway

We used Raspberry Pi 2 Model B as our gateway device. For the platform, we are using &Cube: Thyme version 1.5. This Thyme variant uses the node.js framework. &Cube: Thyme connects to the MQTT broker through port 1883 of the IoT server. The gateway can modify the value of the <contentInstance> as well as getting data from it. The gateway will publish a sensor data to its corresponding resource, or send the information to the actuator based on subscribed resource.

Gateway will publish the message to a topic, and then MQTT proxy will subscribe to that topic and then store/retrieve the message from the database. The message transferred in this connections is in XML/JSON format. MQTT proxy will do the translation of the XML/JSON files in order to modify the resource in the database.

4.3 End-node Device

We used Arduino Mega 2560 for our end-node device in this paper. Sensor and actuator are connected directly with Arduino's input/output (I/O) pin. The Arduino will read the data from the sensor and then send it via USB serial to the gateway (Raspberry Pi). The gateway will send the sensor's data to the server, and the server will reply with the acknowledgment status as mentioned in oneM2M technical specification [11] as shown in Figure 9. This is opposite for the actuators, where Arduino will write to the actuator based on the data received from the gateway. The gateway and end-node device connected with USB cable, but it can be replaced with other communication channels such as Ethernet or Wi-Fi.

```
vincentius@K43SD-ubuntu:~/Downloads/ThymeS sudo
[sudo] password for vincentius:
Connecting...  /dev/ttyACM0
Message: {"ctname":"temp1","con":"24"}
received data: {"ctname":"temp1","con":"2001"}
Message: {"ctname":"hum1","con":"37"}
received data: {"ctname":"led1","con":"2001"}
Message: {"ctname":"hum1","con":"40"}
received data: {"ctname":"hum1","con":"2001"}
Message: {"ctname":"hum1","con":"38"}
received data: {"ctname":"hum1","con":"2001"}
Message: {"ctname":"hum1","con":"37"}
received data: {"ctname":"hum1","con":"2001"}
```

Figure 9: Screenshot of message communication between the gateway and end-node device.

End-node device will publish data to the gateway if there is any change in the sensor's value. End-node device will compare the current value with the previous one, if there is no change, then it will not publish the data to the gateway. The data transferred between the end-node device and the gateway has a format: "ctname, con", where ctname is the container's name (sensor or actuator's name), and con is the content of that container as shown in Figure 9.

4.4 IoT End-user Application

We made an IoT mobile application runs on Android Operating System (OS). There are two main parts of the application. The first one is the user interface, which contains buttons to edit sensors or actuators, or change settings of the IoT system. This user interface allows the user to read

Journal of Theoretical and Applied Information Technology
31st January 2018. Vol.96. No 2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

sensor's value or send actuator's value and even add, edit, or delete sensors or actuators resources as shown in Figure 10. The second part is object management, serialization, and deserialization. Mobius supports XML and JSON format for data exchange and we choose JSON for the application. To serialize and deserialize the object we used FasterXML Jackson. The JSON data received from the server is deserialized into a custom Java object. Once it is deserialized, the content can be processed by the application and displayed to the user. When sending a data to the server, the application serializes the content into JSON format and sends it to the server.

Figure 10: Screenshot of application's main menu.

Mobius uses Uniform Resource Locator (URL) to address a resource. For example, to GET a resource at server.com, with <CSE> name, cse1, <AE> name, ae1, and <container> name, temperature1, we will use "http:// server.com/cse1/ae1/temperature1/latest" as the URL. The string latest at the end of the URL defines that we want the latest data. Contrary, if we want to retrieve the first data created for that resource, we can use "oldest" instead of "latest".

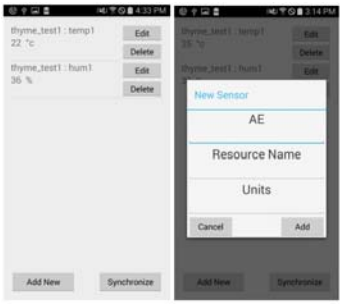

Figure 11: Screenshot of application's sensor menu.

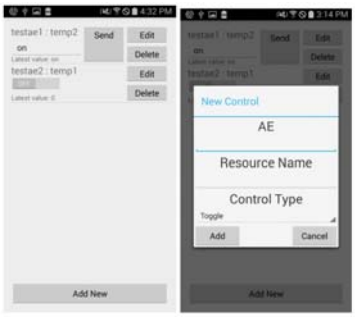

Figure 12: Screenshot of application's control menu.

4. CONCLUSIONS

In this paper, we built a complete IoT system based on oneM2M specification with OCEAN platform. A oneM2M architecture incorporating the entire components prototype implementation. We built our system from the mobile applications for end user interface, IoT server setup, and the gateway with the end-node device.

We put the server on the cloud so that the user can access and manage the server from anywhere. As for the gateway, we used the Raspberry Pi 2, so that we can connect it to the cloud server as well. We chose the MQTT publish/subscribe protocol as the messaging between the gateway and server. The user can access the data on the server through the easy to use mobile applications. The user can easily manage, view, and edit the resource inside the system.

oneM2M as the standardized protocol for the IoT system, proven to facilitate the system to achieve a better IoT implementation.

5. FUTURE WORK

Our simple smart home scenario is the starting step for the bigger IoT system ahead. As for future work, we are exploring other more use cases, and adding several gateways, sensors and actuators, or end-node devices so that our system can be extended to a larger system.

ACKNOWLEDGMENT

This research was supported by Basic Research Program through National Research Foundation of Korea funded by the Ministry of Education, Science and Technology (Grand no. : NRF-

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2014R1A1A2060021). The fifth Author (Hoon-Jae lee) was supported by Basic Research Program through National Research Foundation of Korea funded by the Ministry of Education, Science and Technology (Grand no. : NRF-2016R1D1A1B01011908).

REFRENCES:

[1] Ashton, Kevin. "That 'internet of things' thing." RFiD Journal 22.7 (2009): 97-114.

[2] Gubbi, Jayavardhana, et al. "Internet of Things (IoT): A vision, architectural elements, and future directions." Future Generation Computer Systems 29.7 (2013): 1645-1660.

[3] Datta, Soumya Kanti, and Christian Bonnet. "A lightweight framework for efficient M2M device management in oneM2M architecture." Recent Advances in Internet of Things (RIoT), 2015 International Conference on. IEEE, 2015.

[4] Swetina, Jorg, et al. "Toward a standardized common M2M service layer platform: Introduction to oneM2M." IEEE Wireless Communications 21.3 (2014): 20-26.

[5] OCEAN: A global alliance based on open source and IoT standards, http://www.iotocean.org

[6] MQ Telemetry Transport, http://mqtt.org

[7] Hunkeler, Urs, Hong Linh Truong, and Andy Stanford-Clark. "MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks." Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on. IEEE, 2008.

[8] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kernmarrec, "The many faces of publish/subscribe," ACM Computing Surveys, vol. 35, no. 2, pp. 114–131, June 2003.

[9] D. Chen and P. K. Varshney, "QoS support in wireless sensor networks: A survey." in International Conference on Wireless Networks, 2004, pp. 227–233.

[10] Richardson, Leonard, and Sam Ruby. RESTful web services. " O'Reilly Media, Inc.", 2008.

[11] oneM2M Technical Specifications, http://www.onem2m.org/technical/published-documents