Received 16 November 2022, accepted 5 January 2023, date of publication 12 January 2023, date of current version 22 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3236412

RESEARCH ARTICLE

Enhanced Interoperating Mechanism Between OneM2M and OCF Platform Based on Rules Engine and Interworking Proxy in Heterogeneous IoT Networks

NGUYEN ANH TUAN[1], **RONGXU XU**[2], **AND DOHYEUN KIM**[1]
[1]Department of Computer Engineering, Jeju National University, Jeju 63243, South Korea
[2]Big Data Research Center, Jeju National University, Jeju 63243, Republic of Korea

Corresponding author: Dohyeun Kim (kimdh@jejunu.ac.kr)

This work was supported by the Institute for Information & Communications Technology Pro-motion (IITP) (NO. 2022-0-00980, Cooperative Intelligence Framework of Scene Perception for Autonomous IoT Devices), and this work was supported by the Institute for Information & Communications Technology Promotion (IITP) (2021-0-00188, Open-source development and standardization for AI-enabled IoT platforms and interworking). Any correspondence related to this paper should be addressed to Dohyeun Kim.

ABSTRACT In recent years, the Internet of Things (IoT) is growing rapidly and is being applied in a variety of industries including healthcare, smart homes, and smart cities. Many standard IoT platforms are proposed to connect and communicate with IoT devices easily and securely such as oneM2M, Google Weave and Apple HomeKit. However, this makes IoT application development difficult as it requires IoT devices and applications to support multiple protocols to connect different IoT platforms. Therefore, it is necessary to provide a consistent schema to support interoperability in heterogeneous IoT networks. In this paper, we propose how to design and implement interoperating schema between two edge servers oneM2M and Open Connectivity Foundation (OCF) with heterogeneous IoT devices. Specifically, we build proxies for bridging oneM2M Mobius edge server and OCF IoTivity edge server. The sensor data is collected from various IoT devices and sent to an edge server with a compatible platform. Next, the data stored in each server will be exchanged with the other server through a proposed interworking proxy. We also use a rules engine to automatically identify registered devices to support edge server interaction within the same domain and across domains. In addition, we build a web application in each edge server to provide friendly IoT services (data visualization) to clients from different environments. In order to evaluate our system, we collect the delay time of each process in the edge servers. The results show that our proposal is completely applicable in practice.

INDEX TERMS Internet of Things, interoperability, oneM2M standard, open connectivity foundation, rules engine, interworking proxy, hypertext transfer protocol, constrained application protocol.

I. INTRODUCTION

The term ''Internet of Things'', which Ashton first introduced in 1999, refers to a technical innovation that will change how computers and communications operate in the future [1]. With the advancements in wireless networking (Wifi, Bluetooth, etc), cellular networks, and the increase in the number of electronic devices (smartphones, etc), objects all over the world can now interact with one another and share information about their data, status, and environment information and allowing the objects to collaborate and carry out common tasks without the need for human intervention [2]. Recently, IoT technology has garnered a lot of social attention on a global scale and is the trend in ICT

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

(Information and Communications Technology) [3]. Many proposed IoT platforms enable developers and producers to build innovative products and intelligent services to improve human life [4]. IoT Platforms are software based on standards from organizations to connect various devices (e.g. sensors, access points, data networks) [5]. They enable easy management, development, and operation of IoT applications. Some commonly used IoT platforms are oneM2M [6], OCF [7], and AllSeen Alliance [8], etc.

In 2012, a standards collaboration project called oneM2M was established by standards development organizations from many countries [9]. The main objective of oneM2M is to provide a standardized framework to connect and interact with different IoT devices and provide intelligent services to users, such as Healthcare [10], Smart Home [11], [12], and Smart Car [13]. In order to provide interoperability, oneM2M plays as a distributed Operating System for IoT. It proposes common service functions between applications and connectivity transport [14]. Applications and IoT devices can access oneM2M's common service functions through RESTful APIs [15]. Besides, semantic technologies on the Web (Semantic Web of Things) are developed to describe and analyze the oneM2M resources [16]. Currently, popular platform software resources that developers can access to try out and build end-to-end IoT systems using oneM2M standards such as OCEAN [17], ACME [18], OASIS SI [19].

The OCF standard is developed by Open Connectivity Foundation which is one of the largest IoT standard organizations with over 500 members [20]. Similarly to oneM2M, the target of the OCF standard is to build a common service platform to make it easier for developers to deal with the complexity of IoT connectivity. It ensures interoperability trust and secure communication between IP-connected IoT devices and services. Besides, the OCF standard also supports various transport protocols (e.g. Wi-Fi, Bluetooth, ZigBee) to interconnect different IoT devices and provides a common resource model that developers can use to interface with all IoT devices and data [21]. Additionally, OCF supports both IPv4 and IPv6. In the core functions in OCF, it uses CoAP [22] protocol to message and discovery in combination with RESTful design and API expose [23]. CoAP is an application protocol created by Internet Engineering Task Force (IETF) that is used for supporting constrained devices. Some popular open source platforms implement OCF standard such as IoTivity [24], plgd [25].

As described before, there are now many platforms with various standards to support interaction with devices in IoT networks. However, each platform has its own way of managing and communicating with IoT devices. Furthermore, with the increasing number of devices in different environments, some devices only conform to certain standards. The connection and interaction with heterogeneous devices and platforms become a challenge in IoT systems [26]. For example, sensors are often placed in the edge environment (usually embedded inside low-power, wireless nodes). They gather data from their immediate environment and produce usable information for IoT services. However, because most IoT businesses use proprietary systems, it can be challenging for an IoT system in a certain industry (like smart homes) to gain sensor data from other fields (e.g. smart healthcare). Therefore, it is important to identify each sensor device and communicate and share data between different IoT platforms [27]. To solve this, it is important to provide a consistent schema to support interoperability in heterogeneous IoT networks.

In this paper, we propose a schema for interoperating between two edge servers (with oneM2M and OCF platforms). Specifically, we build an IoT proxy with the rules engine in each edge server as a bridge to recognize different IoT platforms and exchange data between platforms and devices. Edge servers are built on top of Mobius and IoTivity which are frameworks developed according to oneM2M and OCF standards respectively. Edge servers are connected to IoT devices with compatible platforms to collect data from sensors (temperature, humidity, and dust sensors). Then, the proxy in each server will process, and create a message to deliver data to the other server. In addition, we also build a Django-based web application that allows users to view the data collected from sensors in edge servers. Our proposed system's contribution can be summarized as follows:

1) Developing a mechanism to connect edge servers with IoT devices based on oneM2M and OCF standards.
2) Designing and implementing interoperating system between oneM2M Mobius and OCF IoTivity edge servers.
3) Deploying a web application in each edge server that allows users to monitor the historical data of sensors through visualization.

The remainder of this paper is structured as follows. In Section II, we introduce related works including oneM2M Mobius and OCF IoTivity platform, rules engine, proxy, and studies of interoperability in heterogeneous IoT networks. Section III describes our proposed system and detail the components in each edge server. In Section IV, we introduce the environment and the implementation of our proposed system. The performance evaluation for delay time in each process and comparing them in two edge servers are presented in Section V. Finally, Section VI concludes this paper and presents our future works.

II. RELATED WORK

IoT Platforms based on standards enable developers to distribute applications, connect and manage securely sensor devices from the environment in many industries [28]. OCEAN is one of the IoT platforms based on oneM2M standards. It provides two versions for supporting servers and devices, Mobius and nCube respectively [4]. As oneM2M specifies, Mobius acts as a middleware server platform that provides common services functions and connects diverse IoT devices including oneM2M and non-oneM2M devices [29]. nCube is a device platform based on oneM2M standard for supporting interworking between device and server. Implementing Java language, it can run on most embedded devices through Java runtime virtual machine. Besides, nCube is suitable for resource-limited devices or devices in heterogeneous environments [30]. Mobius and nCube are popular oneM2M platforms for servers and devices that have been extensively researched, developed, and applied in many industries. In [31], Yun et al. developed middleware software based on the nCube platform to support different sensors and actuators. Sensor values and actuator commands are transformed into oneM2M resources by the nCube platform. Thereby, the authors developed an Android application that can access and control sensors and actuators through REST APIs based on the oneM2M standard. Lee et al. [32] proposed an architecture to integrate Hololens, a Mixed Reality device into the Mobius platform. In this way, it gives users an intuitive and simple experience without major configuration changes in different domains. In [33], the authors implemented an IoT system based on Mobius for interacting with old consumer products (Nest thermostat, Withings blood pressure monitor, etc). By mapping device information to oneM2M resource tree, an IoT application can access and control legacy devices even if they are non-oneM2M devices. In [34], Choi et al. proposed a real-time data distribution system with oneM2M platform using Mobius to be implemented in the management system of UAVs. According to experimental results, this system enhances delay performance across a range of network environments. In [35], the authors developed wearable IoT Android software based on Mobius to gather biosignal data. The resulting biosignal data will be used to analyze and predict emotions (recognize two emotions of joy and sadness). Um et al. [36] proposed a container-based smart factory system with oneM2M standard. It provides a common standard environment that helps manufacturers reduce errors between different devices. Besides, it offers the benefit of easy version control and is simple to distribute and apply via the Docker registry.

As for the OCF standard, IoTivity [24] is the most popular platform developed by the OCF organization itself. As an open-source project, it provides intelligent functions to support device-to-device and device-to-server connectivity. It supports many different connections (e.g. Ethernet, NFC, Wifi) and can run on various OS (e.g. Windows, Linux, Arduino, and so on). In order to design, deploy, and control IoT systems based on the IoTivity platform, a lot of research was conducted. In [37], Lee et al. developed an IoTivity application to collect data from healthcare sensors on the human body. Data is sent to the server via Bluetooth Low Energy (BLE) and organized to the resource model of the OCF standard. Mandza and Raji [38] designed a home energy management system using the IoTivity-Lite platform (a lighter version of IoTivity for constrained devices). It is suitable for older devices and it is cost-effective and easy to scalability. In [39], Doan et al. developed a software called RES-Hub based on the IoTivity framework to maintain services to IoT home devices when the connection between cloud and home devices is unavailable. With IoTivity, RES-Hub ensures secure access and control of devices. In [40], the authors designed a smart farm system using the IoTivity framework. Sensor data and actuator commands are aggregated to the edge server via CoAP. The edge server will analyze sensor data and give the actuators the next action.

The rules engine in IoT is a system to analyze collected data and takes specific action if they satisfy a necessary condition. It first receives data from IoT devices. It then runs logical analyzes formatted as an ‘‘if-then’’ statement and takes further action based on the results of the analysis. It can make it easy to model the logic of complex systems and help users to adjust the logic of the system without any coding experience [41]. The rules engine helps IoT systems automatically make decisions and control the flow when receiving enormous amounts of data from sensors or actuators. Nowadays, there are many tools that support the rules engine in IoT platforms: Drools [42], eKuiper [43], Jess [44], etc. Drools is a rules engine tool that implements knowledge-based systems using the rule-based methodology. The knowledge-based systems are systems that use knowledge representation to perform complex tasks in different environments. Users can build rules through DRL (Drools Rule Languages) files. DRL files contain rules defined by conditions (with the ‘‘when’’ keyword) and actions (with the ‘‘then’’ keyword). In [45], Luo et al. designed a Drools-based rules engine system that can perform large-scale trigger-action tasks in IoT environments. Besides, they designed an algorithm to find and delete conflicted rules registered by users. The results demonstrated that their method is optimal in terms of energy consumption and system delay time. In [46], the authors implemented a tool called TrigGen to optimize the number of rules in Drools. Experiments showed that TrigGen has optimized more than 80% of the rules written by users.

According to the previous section, it is challenging to recognize each heterogeneous IoT device or communicate and share data between different IoT platforms. Tao et al. [47] proposed to manage data from different IoT platforms using the public cloud. They design a data management system for heterogeneous devices by utilizing ontology representation in the public cloud. However, not most platforms can use the cloud, and constant retrieval of the cloud is also a challenge for constrained devices. Koo and Kim [48] focused on device ID interoperability in IoT platforms. They proposed a device ID translator to translate diverse types of ID into oneM2M format. It can easily translate the IDs of the platforms that have a similar structure to the oneM2M ID format. However, with other formats like OCF IoTivity, it will cause a lack of necessary parts in the oneM2M format. In this [49], the authors proposed an IoT Smart Hub to aggregate data from heterogenous devices and map data format into a predefined common resource format. Thus, applications can access and interact with data through a common resource format.

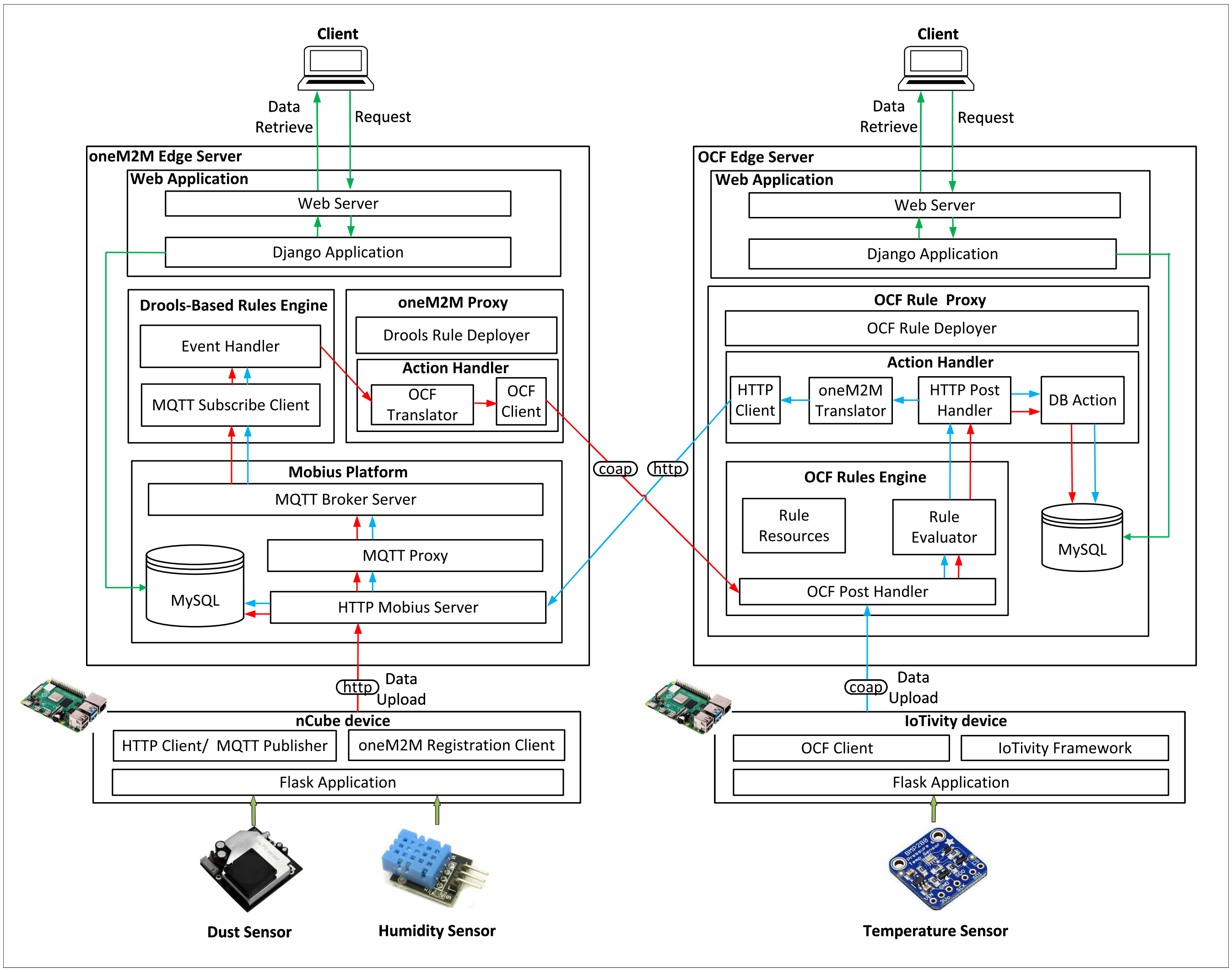


FIGURE 1. Proposed architecture for interworking between oneM2M edge server and OCF edge server.

However, it requires a predefined format suitable for all resource formats of different platforms.

Previous related works have often studied interactions between heterogeneous devices and servers or end-user applications. In our research, we focus on studying the interworking between two heterogeneous edge servers (oneM2M and OCF servers) as well as designing the interaction mechanism between the server and the IoT devices. The details of our proposed method are described in section III.

III. PROPOSED INTEROPERATING MECHANISM BETWEEN oneM2M AND OCF PLATFORM BASED ON RULES ENGINE AND PROXY

In this section, we introduce our proposed architecture based on the interworking proxy and rules engine and discuss the details components of each edge server.

A. PROPOSAL ARCHITECTURE FOR IoT SYSTEM

Figure 1 depicts our proposed architecture to provide interoperability between IoT devices, edge servers, and clients. The proposal comprises three sensors (dust sensor SDS011, humidity sensor DHT11, and temperature sensor BMP280). They are connected to two IoT devices to measure in two different places. IoT devices are deployed in the local environment and based on nCube and IoTivity platforms. IoT devices collect data from sensors through web services. Specifically, we build a Flask application in each device to collect data from sensors. Each device is linked to each compatible edge server. The oneM2M edge server connects to nCube device whereas the OCF edge server connects to IoTivity device.

The main components of edge servers include an IoT platform (Mobius for oneM2M server and IoTivity for OCF server), interworking proxy, rules engine, and web services. The nCube device sends data to Mobius in oneM2M edge

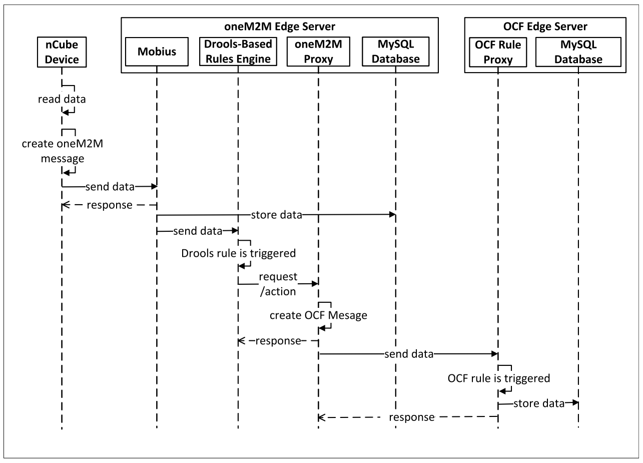


FIGURE 2. Sequence diagram of collecting and sending dust and humidity data.

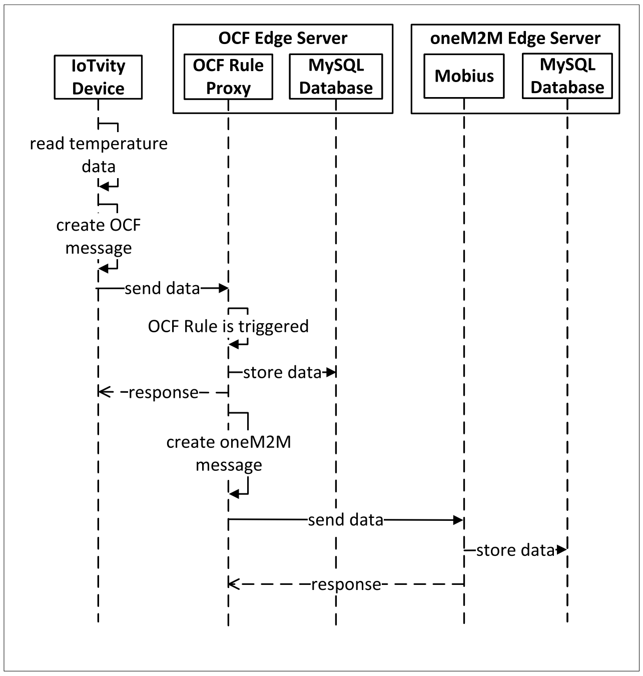


FIGURE 3. Sequence diagram of collecting and sending temperature data.

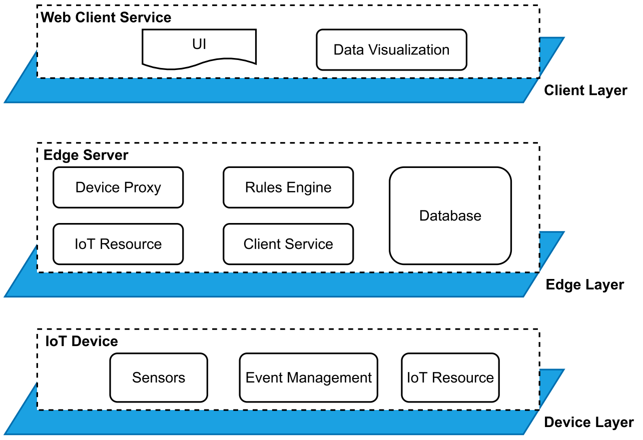


FIGURE 4. Proposed IoT system hierarchical architecture.

server through HTTP protocol and IoTivity device sends data to OCF edge server through CoAP protocol. The rules engine in each edge server provides inference ability based on registered rules to perform different actions when receiving data from different sources. Proxy support to translate messages and deliver them to the other server via HTTP and CoAP protocol. As shown in figure 1, the oneM2M edge server uses a proxy to translate messages containing data collected from the nCube device and send them to the OCF edge server via CoAP. Proxy in OCF edge server works with similar functions. Finally, web services in each edge server allow clients to access and track the environmental data.

From figure 1, we have two directions of data flow for each edge server. Figure 2 shows a sequence diagram for getting dust and humidity data from nCube device and distributing them to edge servers. Firstly, nCube device gets data from the dust sensor and humidity sensor through the Flask application. Then, the nCube device acts as MQTT Publisher and sends data to Mobius by HTTP Protocol. Sensors data will be stored in MySQL database and the message will be sent to the rules engine in oneM2M edge server. Based on registered rules, action will be triggered and it will send a request to oneM2M proxy. OneM2M proxy will create an OCF message with sensor data and send it to the OCF edge server via CoAP protocol. With registered rules, the OCF edge server will verify data and store it in the MySQL database.

Similarly, figure 3 depicts a sequence diagram for getting temperature data from the IoTivity device and distributing them to oneM2M and OCF edge servers. Firstly, the IoTivity device will get data from the temperature sensor and send it to the OCF edge server through the CoAP protocol. Next, the OCF edge server will store it in the MySQL database. Then, it will create an oneM2M message and send temperature data to Mobius in the oneM2M edge server via HTTP protocol. Mobius will store data to MySQL database in oneM2M edge server.

Figure 4 presents the layer diagram of our proposed IoT system's functions including device, edge, and client layers. In the device layer, IoT devices collect sensor data from the local environment. Event management is used to create events (including data and device information) and publish them to the edge server. Built from IoT platforms, IoT resources provide services for communicating to edge servers secure, easy to manage, and track in real-time.

The edge layer contains oneM2M and OCF edge servers that are used for receiving and distributing sensor data. Each edge server includes a database for data storage and retrieval. An interworking proxy is in the edge server for connecting and sharing sensor data with other edge servers. The rules engine in the edge server is used to make automatic rules for recognizing IoT devices and handling sensor data.

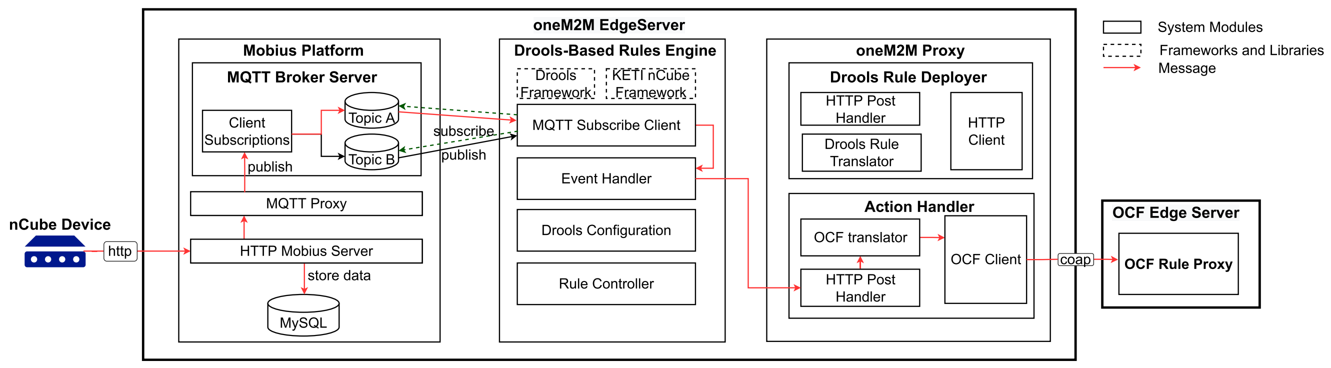


FIGURE 5. Detail of Mobius, Rules Engine, and Proxy components in oneM2M edge server.

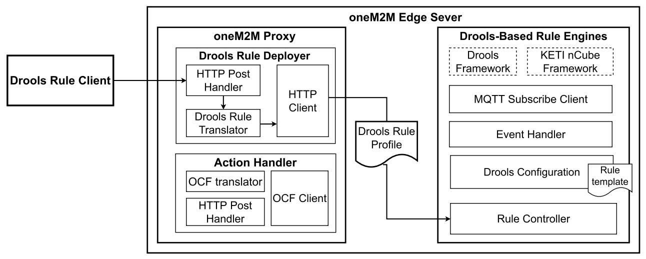


FIGURE 6. Client registers rules to Drools Rules Engine.

Besides, each edge server provides client service to allow clients can access and track data.

The client service in the client layer is used to provide historical data to the clients through web applications. Web client service includes User Interface (UI) and data visualization. Using UI, data visualization is provided to display updated and historical values from the sensor.

B. ONEM2M EDGE SERVER ARCHITECTURE ANALYSIS

The oneM2M edge server includes Mobius platform, Drools-based rules engine, oneM2M proxy, and web application. Figure 5 depicts in detail the main components of Mobius, rules engine, and proxy in oneM2M edge server and data distribution from nCube device.

Mobius provides internal servers to support various data transfer protocols including HTTP, CoAP, MQTT, and WebSocket. In order to communicate with nCube device, we use an HTTP Mobius server with the default HTTP protocol in our design. Moreover, the MQTT proxy is used to convert HTTP to MQTT protocol for Publish/Subscribe (Pub/Sub) messaging. After receiving the message from nCube device, HTTP Mobius server will handle and store sensor data to the MySQL database. At the same time, the MQTT proxy will translate the message to MQTT protocol. After that, MQTT message will be published to MQTT Broker server. MQTT Broker will filter messages based on topic and then distribute them to subscribers. With Pub/Sub messaging, it announces event notifications quickly for distributed applications.

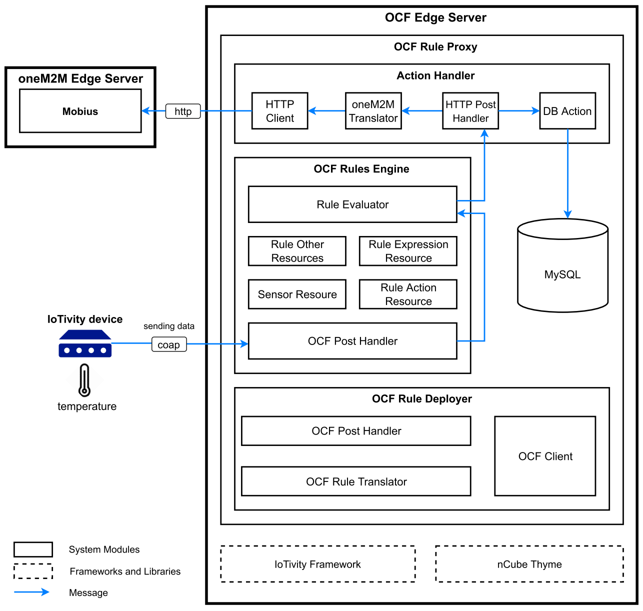


FIGURE 7. Detail components in OCF edge server.

The rules engine in oneM2M edge server is developed by Drools framework. In our system, the rules engine subscribes to all topics in MQTT Broker. Therefore, it receives all event messages whenever MQTT Broker publishes them. The Drools rules engine uses rules to take action when data is delivered from different sources. If data comes from nCube device, it needs to be shared with OCF edge server. After confirming the data is coming from the nCube device, the event handler component in the rules engine will handle the MQTT message and send it to the action handler in oneM2M proxy.

OneM2M proxy consists of two main components rule deployer and action handler. The action handler receives the message from the rules engine and translates message from HTTP to CoAP protocol.

Drools rule deployer in proxy is used for registering and updating rules from client. Figure 6 presents the rules registration process from client. When the event message is

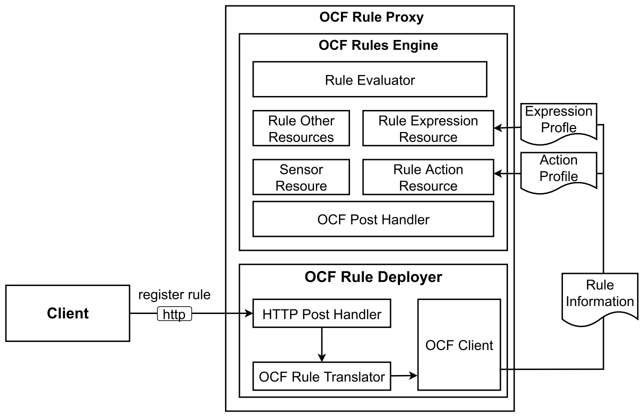


FIGURE 8. Client registers rules to OCF Rules Engine.

delivered from client, the rule deployer will handle, transfer message to Drools format and send it to rule engines. Rule controller will get profile of new rules and extract object ID and command ID based on rule template in Drools configuration. Then, the rule controller updates objects and command IDs for condition and action. In our system, the rule will check if the message is coming from nCube device (humidity and dust data) or coming from OCF server (temperature data).

C. OCF EDGE SERVER ARCHITECTURE ANALYSIS

Figure 7 depicts detailed components in the OCF edge server and data distribution from the IoTivity device. The main parts of the OCF rule proxy include:

- OCF Rule Deployer: Updates rules from the client.
- OCF Rule Server: Stores rules, receives OCF requests and checks requests with the updated rule.
- Action Handler: Do action if the rule is satisfied.

Different from oneM2M edge server, we use the rules engine that is integrated with the IoTivity platform. In the OCF rules engine, a rule is divided into components and stored in different resources (i.e. Expression, Action resources). They are linked together to become a complete rule. Similar to oneM2M edge server, the rules in the OCF edge server will check if the message is coming from IoTivity device (temperature data) or coming from oneM2M server (humidity and dust data).

When the OCF edge server receives any messages through the CoAP protocol, OCF Post Handler will handle this message to get the main factors. After that, based on the registered rules in rule resources, Rule Evaluator will evaluate and decide the next action. If the message comes from oneM2M server, it will be stored database directly. Otherwise, if the IoTivity device sends the message, rules will be triggered and Action Handler will handle, convert the message to oneM2M format and send it to the oneM2M edge server.

Figure 8 presents the rules registration process from the client to OCF rules engine. Rule information will be split and stored in Rule Expression and Rule Action resources. Rule Expression Resource (with resource type id

TABLE 1. Development environment for edge servers.

Hardware			Software	
PC	OS	Ubuntu 20.04 64-bit	Language	Python3.8, Java
	CPU	Intel core i5-9500 3GHz		HTML5, CSS3, Javascript
	Memory	8 GB	Application	Eclipse
	Hard Disk	100 GB	Framework	Django
				jQuery, Bootstrap
				Drools framework (for oneM2M server)
				IoTivity, nCube–Thyme (for OCF server)
				Mobius (for oneM2M server)
				Spring Boot
			Database	MySQL

"oic.r.rule.expression") contains logical expressions which evaluate a boolean value. Rule Action Resource (with resource type id "oic.r.rule.action") contains the action to be taken. When a message with sensor data arrives, the Rule Evaluator will trigger actions in the Rule Action Resource if the logical expressions in the Rule Expression Resource return true.

IV. IMPLEMENTATION OF PROPOSED SYSTEM

A. DEVELOP ENVIRONMENT FOR EDGE SERVERS AND IoT DEVICES

Table 1 shows the detailed development specification of the environment in edge servers. In our proposed solution, both edge servers (oneM2M and OCF servers) use the same hardware configuration. We use the Ubuntu 20.04 with 100GB hard disk, 8GB memory, and an Intel core i5-9500 CPU desktop for each edge server. To build the web application, we use the Django framework with Python, HTML5, CSS3, and JavaScript. JQuery and Bootstrap libraries are used to design the front-end of the website. We use Java with Spring Boot framework to implement proxy and rules engine. The oneM2M server uses Mobius based on the oneM2M standard and the OCF server uses IoTivity based on the OCF standard and both of them use the MySQL database. Drools framework is used in oneM2M server to provide rules-based service scenarios whereas the OCF server uses rules resources in IoTivity.

Table 2 shows the development environment in IoT devices. We use Raspberry Pi 4 with Ubuntu 20.04 64-bit, Quad Core 15GHz 64-bit CPU, 4GB Memory, and 32GB MicroSD Card as the IoT device to collect data. Both devices use Python with Flask framework to connect to sensors.

TABLE 2. Development environment for IoT devices.

Hardware			Software	
Raspberry Pi 4 Model B	OS	Ubuntu 20.04 64-bit	Language	Python 3.8, Java
			Application	Visual Studio Code
	CPU	Quad Core 15GHz 64-bit	Framework	Flask
	Memory	4 GB		IoTivity (for IoTivity device)
	MicroSD Card	32 GB		nCube-Thyme (for nCube device)

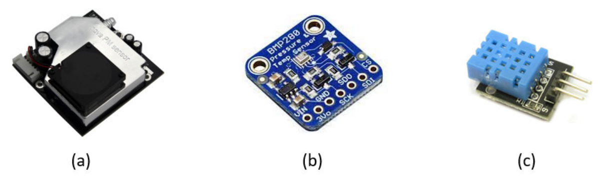


FIGURE 9. Photos of sensors deployed in our experiment; (a) Dust sensor, (b) Temperature sensor, (c) Humidity sensor.

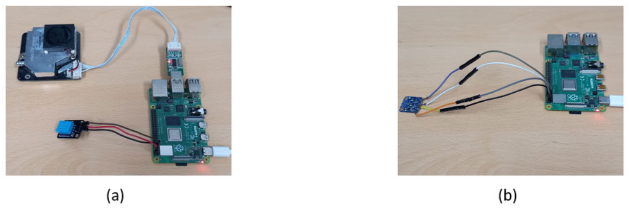


FIGURE 10. IoT devices with sensors; (a) nCube device with dust and humidity sensors, (b) IoTivity device with temperature sensor.

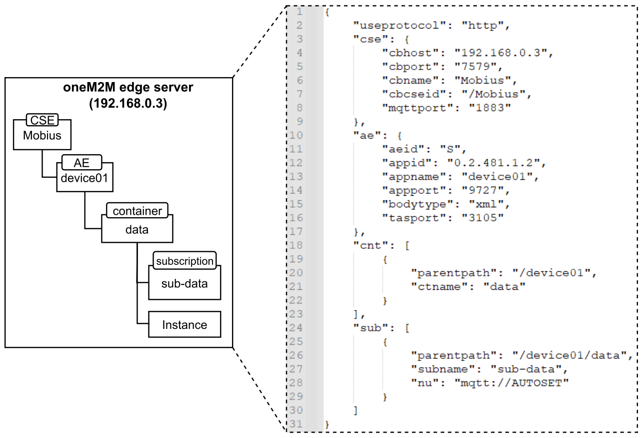
TABLE 3. Sensors output in our experiment.

Sensors	Output Description
Dust sensor	PM2.5: Fine particulate matter with a diameter of 2.5 microns or less ($\mu g/m^3$)
	PM10: Fine particulate matter with a diameter of 10 microns or less ($\mu g/m^3$)
Temperature sensor	Environment temperature (degree Celcius)
Humidity sensor	Environment humidity (%)

We use Visual Studio Code to develop applications. One device uses the IoTivity framework (called IoTivity device) to connect to the OCF edge server and the other uses the nCube framework (called nCube device) to connect to the oneM2M edge server.

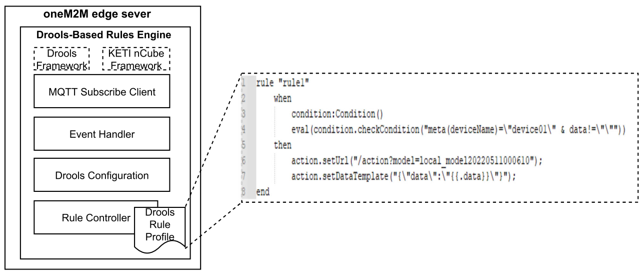
Figure 9 presents the photos of sensors; (a) dust sensor (SDS011), (b) temperature/pressure sensor (BMP280) and (c) temperature/humidity sensor (DHT11). Table 3 shows output of sensors in our implementation.

Figure 10 describes IoT devices with sensors in our implementation work; (a) nCube device connects to humidity and dust sensors while the IoTivity device connects to the temperature sensor.



```
{
    "useprotocol": "http",
    "cse": {
        "cbhost": "192.168.0.3",
        "cbport": "7579",
        "cbname": "Mobius",
        "cbcseid": "/Mobius",
        "mqttport": "1883"
    },
    "ae": {
        "aeid": "S",
        "appid": "0.2.481.1.2",
        "appname": "device01",
        "appport": "9727",
        "bodytype": "xml",
        "tasport": "3105"
    },
    "cnt": [
        {
            "parentpath": "/device01",
            "ctname": "data"
        }
    ],
    "sub": [
        {
            "parentpath": "/device01/data",
            "subname": "sub-data",
            "nu": "mqtt://AUTOSET"
        }
    ]
}
```

FIGURE 11. Information registration from nCube device ("device01") to oneM2M server.

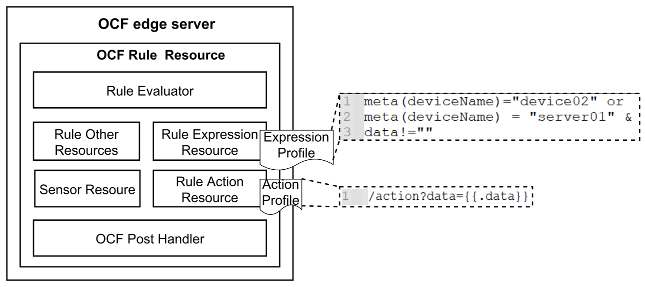


```
rule "rule1"
    when
        condition:Condition()
        eval(condition.checkCondition("meta(deviceName)=\"device01\" & data!=\"\""))
    then
        action.setUrl("/action?model=local_model20220511000610");
        action.setDataTemplate("{\"data\":\"{{.data}}\"}");
end
```

FIGURE 12. Registered rule in oneM2M server.

B. IMPLEMENTATION RESULTS OF THE PROPOSED SYSTEM

Figure 11 shows the device information config in nCube device to register to oneM2M server. The oneM2M standard APIs used a resource-based data model to perform CRUD request operations on the target resource. This figure shows the resource structure of oneM2M services. Common Service Entity (<CSE>) has multiple <AE> (application entity), but one <AE> has to be under one. A <CSEBase> resource represents a CSE and serves as the root resource for all resources that are residing in the CSE. An <AE> resource represents an Application Entity that is registered to a CSE. An <AE> resource supports attributes such as identifiers, contact information, status, and capabilities of the Application Entity. Each <AE> can have multiple <container> indicating the data inside that <AE>. Containers describe attributes of the data and child resources. Each <container> has one <contentInstance> which represents a data instance and one <subscription> contains subscription information for its "subscribed-to" resource. In our work, we use Common Service Entity (<CSE>) as the Mobius server. Application object (<AE>) is the name of nCube device is "device01". The application data resource is collected from sensors.



```
meta(deviceName)="device02" or
meta(deviceName) = "server01" &
data!=""
```

```
/action?data={{.data}}
```

FIGURE 13. Registered rule in OCF server.

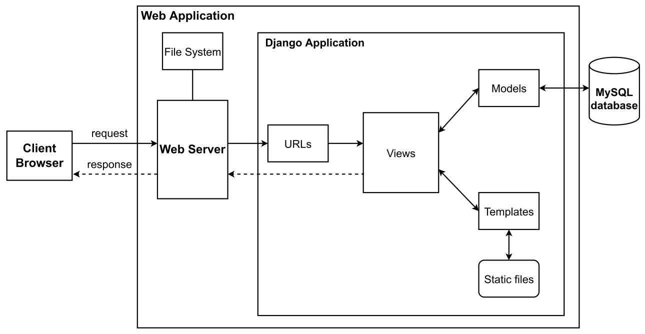


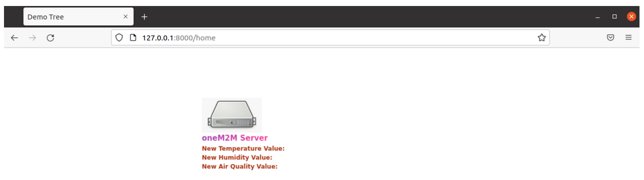
FIGURE 14. Web application in the edge server.

Figure 12 shows the registered rule in the oneM2M server from the client. The Drools rule profile is stored in a rule controller in the rules engine. When receiving data from nCube device (device name is "device01"), send data to the Action method. The Action method will create an OCF message and send data to OCF edge server. If the rule is not satisfied (device name is not "device01") and the device is registered, data will be stored in the MySQL database and not trigger the Action method.

Figure 13 presents rule registration in the OCF edge server. Rule Evaluator evaluates received messages, and identifies key elements in the message (i.e. device name, data). The logical expression in the Rule Expression Resource includes checking whether the message is coming from the IoTivity device ("device02") or coming from the oneM2M edge server ("server01") and checking if the data is available. Action in the Rule Action Resource includes the address to send data to Action Handler to perform the next step.

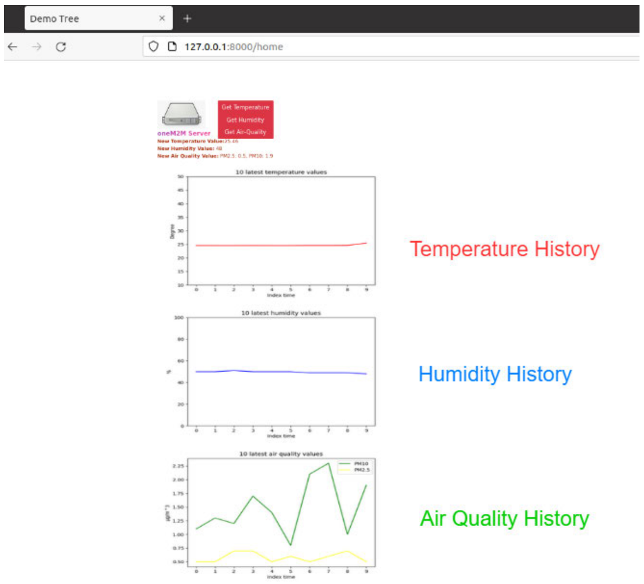
Figure 14 depicts the simplified flow of a web request from a visitor's browser to the Django service in the edge server and back. Both edge servers use the same template and design for web services. Clients send a request to the web server through the interface in the web browser. Then, the web server will access to MySQL database and respond to client history sensor data.

Figure 15 presents the interface of web service in the oneM2M edge server (the same interface in the OCF edge server). It provides the data visualization function when customers want to monitor the data collected from sensors.



New Temperature Value:
New Humidity Value:
New Air Quality Value:

FIGURE 15. Web service interface in edge sever.



New Temperature Value:25.46
New Humidity Value: 48
New Air Quality Value: PM2.5: 0.5, PM10: 1.9

FIGURE 16. Data visualization of Web services in the edge server.

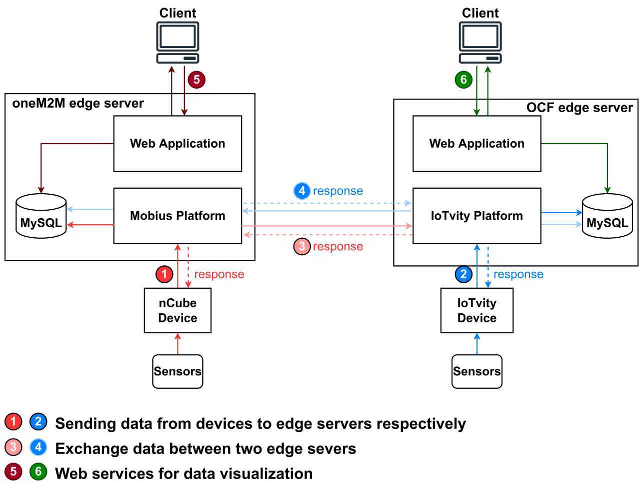


FIGURE 17. Testing scenario in our implementation.

Figure 16 depicts the implementation result of the edge client for data visualization. When clients click to button, it provides new sensor values collected and history data of dust, temperature, and humidity sensors.

V. PERFORMANCE EVALUATION

In order to evaluate the performance of our proposed system, we collected the time delay of all processes between devices,

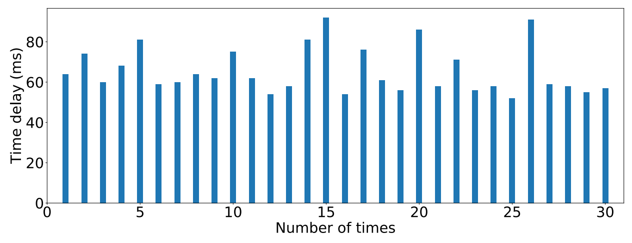


FIGURE 18. Time delay of sending data from nCube device to oneM2M edge server.

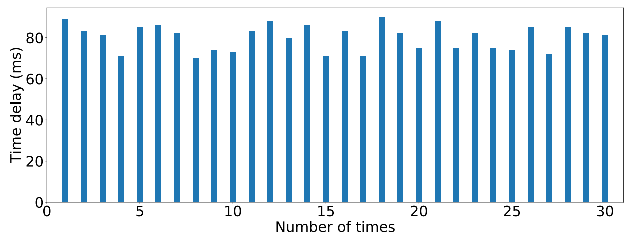


FIGURE 19. Time delay of sending data from IoTivity device to OCF edge server.

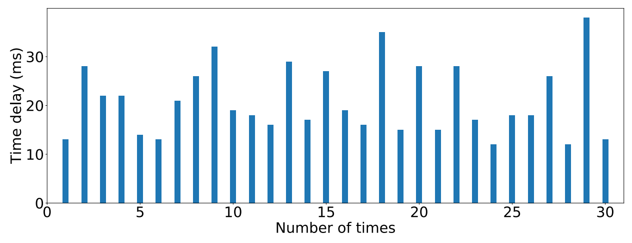


FIGURE 20. Time delay of sharing data from oneM2M edge server to OCF edge server.

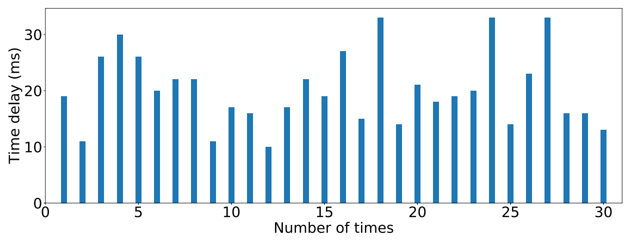


FIGURE 21. Time delay of sharing data from OCF edge server to oneM2M edge server.

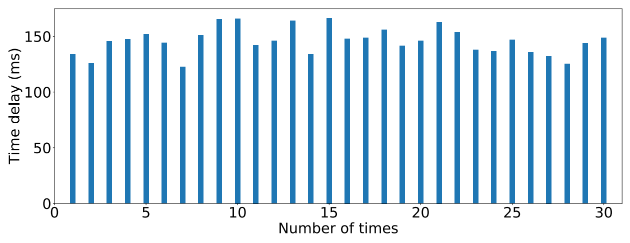


FIGURE 22. Time delay of accessing data visualization through oneM2M Web service.

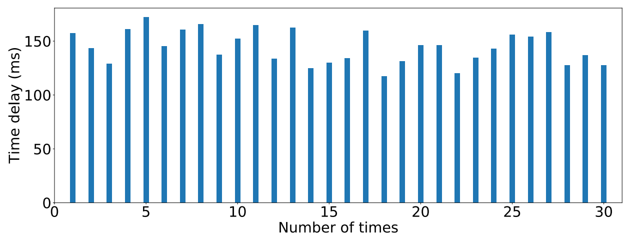


FIGURE 23. Time delay of accessing data visualization through OCF Web service.

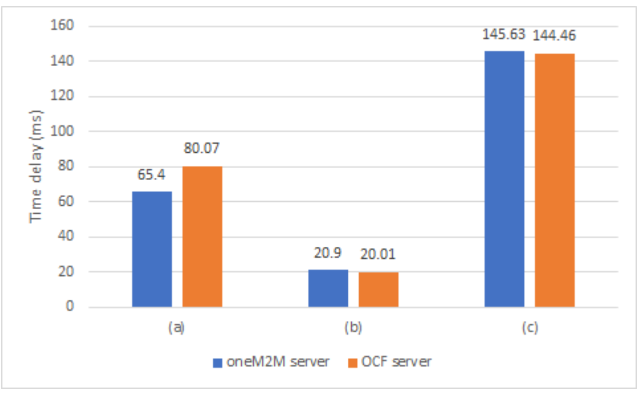


FIGURE 24. Average time delay comparisons between oneM2M and OCF server; (a) getting data from devices, (b) exchanging data to other edge servers, (c) data visualization from client web services.

edge servers, and clients including sending data from IoT devices, exchanging data between two servers, and accessing data visualization through the Web application. Figure 17 presents 6 testing scenarios in our experiment (3 testing processes in each server). Each process is tested 30 times and recorded.

Figures 18 and 19 show the results of evaluating the network communication time delay of sending data from devices to edge servers. In Figure 18, the time delay when sending sensor data from the nCube device to the oneM2M edge server range from 52 ms to 65.4 ms. Whereas, figure 19 shows that the connection time delays between the IoTivity device and the OCF edge server are between 70 ms to 90 ms.

Figures 20 and 21 present the results of evaluating the time delays of exchanging data between two edge servers. When sharing data from the oneM2M edge server to the OCF edge server, the time delays are between 12 ms and 38 ms as shown in figure 20. In figure 21, the delays range from 10 ms to 33 ms in sharing data from OCF edge server to oneM2M edge server.

Figures 22 and 23 present the results of evaluating the network communication time delays for accessing data visualization through the oneM2M Web service and OCF Web service. Figure 22 shows the delays in accessing oneM2M Web service are between 122.43 ms to 166.42 ms. Whereas, the time delays of accessing OCF Web service are between 117.39 ms and 172.08 ms in figure 23.

Figure 24 describes the average time delay comparisons between the oneM2M edge server and the OCF edge server. The average time delays of oneM2M in each process are

65.4 ms, 20.04 ms, and 145.63 ms while the results of OCF sever are 80.07 ms, 20.01 ms, and 144.46 ms respectively. We can see that the time delays in the processes of the two edge servers are almost the same.

VI. CONCLUSION

Nowadays, with the development of IoT technologies, there are numerous IoT platforms in development. However, each platform has its own way of managing and communicating with IoT devices. In this paper, we propose a schema for interoperating between oneM2M Mobius and OCF IoTivity platform. The goal is that they can connect and exchange sensor data with each other. To solve this, we use a proxy with the rules engine to operate the process. Edge servers are connected to IoT devices with compatible platforms to collect data from sensors. Then, the proxy in each server will process, and create a message to exchange data with the other server. Besides, we also built a Django web application to provide services that allow clients can view sensor data (dust, temperature, and humidity sensors). Our experimental results show proposed system is suitable for real-world applications.

In the future, we plan to extend our research to integrate other IoT platforms such as AllJoyn, Google Weave, and Apple HomeKit. In addition, we will research to improve delay time when exchanging data between platforms.

REFERENCES

[1] K. Ashton, “That ‘Internet of Things’ thing,” *RFID J.*, vol. 22, no. 7, pp. 97–114, Jun. 2009.

[2] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, “A survey on 5G networks for the Internet of Things: Communication technologies and challenges,” *IEEE Access*, vol. 6, pp. 3619–3647, 2018.

[3] A. Paul and R. Jeyaraj, “Internet of Things: A primer,” *Hum. Behav. Emerg. Technol.*, vol. 1, no. 1, pp. 37–47, Jan. 2019.

[4] J. Kim, S.-C. Choi, J. Yun, and J.-W. Lee, “Towards the oneM2M standards for building IoT ecosystem: Analysis, implementation and lessons,” *Peer-Peer Netw. Appl.*, vol. 11, no. 1, pp. 139–151, Jan. 2018.

[5] J. Koo, S.-R. Oh, and Y.-G. Kim, “Device identification interoperability in heterogeneous IoT platforms,” *Sensors*, vol. 19, no. 6, p. 1433, Mar. 2019.

[6] *OneM2M Specification*. Accessed: Sep. 28, 2022. [Online]. Available: https://www.onem2m.org/technical/published-specifications

[7] *OCF Specification 2.2.5*. Accessed: Sep. 28, 2022. [Online]. Available: https://openconnectivity.org/developer/specifications

[8] AllSeen Alliance. *AllJoyn Open Source Project*. Accessed: Sep. 28, 2022. [Online]. Available: https://github.com/alljoyn/alljoyn.github.com/wiki

[9] X. Costa-Pérez, A. Festag, H.-J. Kolbe, J. Quittek, S. Schmid, M. Stiemerling, J. Swetina, and H. van der Veen, “Latest trends in telecommunication standards,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 2, pp. 64–71, Apr. 2013.

[10] J. N. S. Rubí and P. R. L. Gondim, “IoMT platform for pervasive healthcare data aggregation, processing, and sharing based on oneM2M and OpenEHR,” *Sensors*, vol. 19, no. 19, p. 4283, 2019.

[11] H. Sun, C. Zhang, J. Si, and W. Xu, “Smart home system design and implementation based on oneM2M,” in *Proc. 2nd Int. Conf. E-Commerce Internet Technol. (ECIT)*, Mar. 2021, pp. 344–347.

[12] A. Muhammad, B. Afzal, B. Imran, A. Tanwir, A. H. Akbar, and G. Shah, “OneM2M architecture based secure MQTT binding in Mbed OS,” in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops*, Jun. 2019, pp. 48–56.

[13] M. B. Alaya, S. Medjiah, T. Monteil, and K. Drira, “Toward semantic interoperability in oneM2M architecture,” *IEEE Commun. Mag.*, vol. 53, no. 12, pp. 35–41, Dec. 2015.

[14] H. Park, H. Kim, H. Joo, and J. Song, “Recent advancements in the Internet-of-Things related standards: A oneM2M perspective,” *ICT Exp.*, vol. 2, no. 3, pp. 126–129, Sep. 2016.

[15] L. Richardson and S. Ruby, *RESTful Web Services*. Sebastopol, CA, USA: O’Reilly Media, 2008.

[16] A. J. Jara, A. C. Olivieri, Y. Bocchi, M. Jung, W. Kastner, and A. F. Skarmeta, “Semantic web of things: An analysis of the application semantics for the IoT moving towards the IoT convergence,” *Int. J. Web Grid Services*, vol. 10, nos. 2–3, pp. 244–272, 2014.

[17] *OCEAN: A Global Alliance Based on Open Source and IoT Standards*. Accessed: Sep. 28, 2022. [Online]. Available: http://developers.iotocean.org

[18] *ACME OneM2M CSE*. Accessed: Sep. 28, 2022. [Online]. Available: https://github.com/ankraft/ACME-oneM2M-CSE

[19] *Oasis SI OneM2M Server*. Accessed: Sep. 28, 2022. [Online]. Available: https://github.com/iotoasis/SI

[20] *Open Connectivity Foundation*. Accessed: Sep. 28, 2022. [Online]. Available: https://openconnectivity.org

[21] H. S. Oh, S. B. Seo, G. T. Lee, W. S. Jeon, and M. G. Lee, “OCF-to-ZigBee (O2Z) bridging technique and IoTivity-based implementation,” *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16418–16426, Nov. 2021.

[22] C. Bormann, A. P. Castellani, and Z. Shelby, “CoAP: An application protocol for billions of tiny internet nodes,” *IEEE Internet Comput.*, vol. 16, no. 2, pp. 62–67, Mar. 2012.

[23] S. Park, “OCF: A new open IoT consortium,” in *Proc. 31st Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, Mar. 2017, pp. 356–359.

[24] *IoTvity: Open Source for IoT*. Accessed: Sep. 28, 2022. [Online]. Available: http://iotivity.org/

[25] *PLGD*. Accessed: Sep. 28, 2022. [Online]. Available: http://iotivity.org/

[26] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A survey on enabling technologies, protocols, and applications,” *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[27] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, and K. Wasielewska, “Semantic interoperability in the Internet of Things: An overview from the INTER-IoT perspective,” *J. Netw. Comput. Appl.*, vol. 81, pp. 111–124, Mar. 2017.

[28] J. Swetina, G. Lu, P. Jacobs, F. Ennesser, and J. Song, “Toward a standardized common M2M service layer platform: Introduction to oneM2M,” *IEEE Wireless Commun.*, vol. 21, no. 3, pp. 20–26, Jun. 2014.

[29] A. Garg and N. Mittal, “A security and confidentiality survey in wireless Internet of Things (IoT),” in *Internet of Things and Big Data Applications*. Cham, Switzerland: Springer, 2020, pp. 65–88.

[30] J. Yun, I.-Y. Ahn, N.-M. Sung, and J. Kim, “A device software platform for consumer electronics based on the Internet of Things,” *IEEE Trans. Consum. Electron.*, vol. 61, no. 4, pp. 564–571, Nov. 2015.

[31] J. Yun, I.-Y. Ahn, J. Song, and J. Kim, “Implementation of sensing and actuation capabilities for IoT devices using oneM2M platforms,” *Sensors*, vol. 19, no. 20, p. 4567, Oct. 2019.

[32] S. Lee, G. Lee, G. Choi, B.-H. Roh, and J. Kang, “Integration of oneM2M-based IoT service platform and mixed reality device,” in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2019, pp. 1–4.

[33] J. Yun, R. C. Teja, N. Chen, N.-M. Sung, and J. Kim, “Interworking of oneM2M-based IoT systems and legacy systems for consumer products,” in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2016, pp. 423–428.

[34] S.-C. Choi, I.-Y. Ahn, J.-H. Park, and J. Kim, “Towards real-time data delivery in oneM2M platform for UAV management system,” in *Proc. Int. Conf. Electron., Inf., Commun. (ICEIC)*, Jan. 2019, pp. 1–3.

[35] I. Y. Ahn, N.-M. Sung, J.-H. Lim, J. Seo, and I. D. Yun, “Development of an oneM2M-compliant IoT platform for wearable data collection,” *KSII Trans. Internet Inf. Syst.)*, vol. 13, no. 1, pp. 1–15, 2019.

[36] C. Um, J. Lee, and J. Jeong, “Virtualized oneM2M system architecture in smart factory environments,” in *Proc. 28th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2018, pp. 1–6.

[37] J.-C. Lee, J.-H. Jeon, and S.-H. Kim, “Design and implementation of healthcare resource model on IoTivity platform,” in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2016, pp. 887–891.

[38] Y. S. Mandza and A. Raji, “IoTivity cloud-enabled platform for energy management applications,” *IoT*, vol. 3, no. 1, pp. 73–90, Dec. 2021.

[39] T. T. Doan, R. Safavi-Naini, S. Li, S. Avizheh, and P. W. L. Fong, “Towards a resilient smart home,” in *Proc. Workshop IoT Secur. Privacy*, Aug. 2018, pp. 15–21.

[40] S. Yoon, M.-H. Choi, and J. Park, “Implementation of smart farm devices using open source software,” in *Proc. 23rd Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2021, pp. 205–209.

[41] C. E. Kaed, I. Khan, A. Van Den Berg, H. Hossayni, and C. Saint-Marcel, "SRE: Semantic rules engine for the industrial Internet-of-Things gateways," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 715–724, Feb. 2018.

[42] P. Browne and P. Johnson, *JBoss Drools Business Rules*. Birmingham, U.K.: Packt Publishing, 2009.

[43] *Kuiper—An Edge Lightweight IoT Data Analytics Software*. Accessed: Sep. 28, 2022. [Online]. Available: https://ekuiper.org/docs/en/latest/

[44] E. Friedman-Hill, "Jess, the rule engine for the Java platform," Sandia Nat. Laboratories, 2008. [Online]. Available: https://www.sandia.gov/

[45] X. Luo, Y. Fu, L. Yin, H. Xun, and Y. Li, "A scalable rule engine system for trigger-action application in large-scale IoT environment," *Comput. Commun.*, vol. 177, pp. 220–229, Sep. 2021.

[46] C. Nandi and M. D. Ernst, "Automatic trigger generation for rule-based smart Homes," in *Proc. ACM Workshop Program. Lang. Anal. Secur.*, Oct. 2016, pp. 97–102.

[47] M. Tao, K. Ota, and M. Dong, "Ontology-based data semantic management and application in IoT- and cloud-enabled smart Homes," *Future Gener. Comput. Syst.*, vol. 76, pp. 528–539, Nov. 2017.

[48] J. Koo and Y.-G. Kim, "Interoperability of device identification in heterogeneous IoT platforms," in *Proc. 13th Int. Comput. Eng. Conf. (ICENCO)*, Dec. 2017, pp. 26–29.

[49] A. Ahmed, M. Kleiner, and L. Roucoules, "Model-based interoperability IoT hub for the supervision of smart gas distribution networks," *IEEE Syst. J.*, vol. 13, no. 2, pp. 1526–1533, Jun. 2019.

NGUYEN ANH TUAN received the B.S. degree in applied mathematics and informatics from the Hanoi University of Science and Technology, in 2021. He is currently pursuing the integrated master's and Ph.D. degree in computer engineering with Jeju National University, South Korea.

His research interests include the development of the Internet of Things, optimization, and machine learning.

RONGXU XU received the B.S. degree in computer science from the Yanbian University of Science and Technology, China, in 2014, the M.S. degree in computer engineering from Konkuk University, South Korea, in 2017, and the Ph.D. degree from the Mobile Computing Laboratory (MCL), Department of Computer Engineering, Jeju National University, South Korea, in 2022.

He has been a Postdoctoral Researcher with the Big Data Research Center, Jeju National University, since March 2022. He is currently working as a Project Manager for a research project with the National Research Foundation of Korea. His research interest includes integrating edge computing with intelligent mechanisms for smart services. In addition, he studies optimization and machine learning for smart homes, deep neural networks, and federated learning for edge intelligent practice.

DOHYEUN KIM received the B.S. degree in electronics engineering and the M.S. and Ph.D. degrees in information telecommunication from Kyungpook National University, South Korea, in 1988, 1990, and 2000, respectively.

From 2008 to 2009, he was a Visiting Researcher with the Queensland University of Technology, Australia. He joined the Agency of Defense Development (ADD), from March 1990 to April 1995. Since 2004, he has been with Jeju National University, South Korea, where he is currently a Professor with the Department of Computer Engineering. His research interests include sensor networks, M2M/IoT, energy optimization and prediction, intelligent service, and mobile computing.