MQTT-based Translation System for IoT Interoperability in oneM2M Architecture

Jiwoo Park, Geonwoo Kim and Kwangsue Chung
Department of Electronics and Communications Engineering
Kwangwoon University, Seoul, Korea
e-mail: {jwpark, gwkim}@cclab.kw.ac.kr, kchung@kw.ac.kr

***Abstract*—The key challenge for the future Internet of Things (IoT) is interoperability between IoT systems and platforms. To support the interconnection and interoperability between heterogeneous IoT systems and services, open-source Application Programming Interfaces (APIs) is one of the key features of common software platforms for IoT devices, gateways, and servers. The oneM2M standard is a global initiative led jointly by major standards organizations in order to standardize a common platform for globally-applicable and access-independent IoT services. In this paper, we present the design and open-source implementation of an IoT translator that enables heterogeneous IoT devices to be interoperable in oneM2M architecture. The translation system abstracts basic functionalities from IoT devices and interconnects them within oneM2M platform through MQTT, which is a publish/subscribe messaging protocol for the lightweight M2M communication. The implementation has been validated in a real test case and proved the interoperability by testing tools.**

Keywords-Internet of Things; interoperability; oneM2M; MQTT.

I. INTRODUCTION

The Internet of Things (IoT) is a large and heterogeneous collection of networks, protocols, devices, systems, services, solutions, and users. Advances in low cost processors have been a key enabler of intelligent automation devices. IoT takes the next step of networking these devices, resulting in intelligent environments. With the heterogeneity of independent platforms, a numerous of protocols have been developed. Many of the protocols will never be known as they are proprietary. But even within standardized protocols, there is a large variety to choose from. They are the result of evolving requirements and technology, leading to a highly dynamic ecosystem of co-existing protocols unable to work with each other. Interoperability in such an ecosystem is a major challenge, and yet it is a crucial aspect of successful IoT. However, many issues are still open in this domain, and the interest has constantly increased in the recent years both in the research and industrial communities [1].

One of the critical points for IoT deployment is the interoperability between devices and applications across multiple architectures, platforms and networking technologies. As a matter of facts, the proliferation of competing communication protocols and data representations across the device ecosystem makes it difficult for smart things to be easily integrated and cooperate with each other in a common IoT network. Several IoT horizontal platforms are being developed to overcome this issue; such platforms aim at abstracting from the complexity of the hardware and the networking sub-systems, so as to give smart things the ability to automatically discover and communicate with each other, and dynamically join and leave IoT proximal networks. Examples of open-source frameworks currently being developed by industries are the AllJoyn platform developed by the AllSeen Alliance [2], IoTivity, sponsored by the Open Connectivity Foundation (OCF) [3], and Google's Thread [4] and Weave [5]. Standard platforms are also being specified like, e.g., oneM2M [6], while many others have been developed as a result of research projects (e.g., EU FIWARE [7] and BETaaS [8]).

There exist many relatively mature IoT communication protocols, such as HTTP, Constrained Application Protocol (CoAP) [9] and Message Queueing Telemetry Transport (MQTT) [10], that may be already applied to pre-existing successful IoT implementations and are not supported for the future IoT framework. Replacing or updating IoT devices to integrate them in such frameworks is not always a feasible option due to device cost and other technical limitations. In these cases, a middleware that behaves like a translator between the pre-existing IoT platforms is therefore more appropriate in order to address interoperability challenges.

In this paper, we present the design and open-source implementation of a translation system that enables non-oneM2M devices to be compatible with oneM2M systems. The contribution of this paper covers the mapping of both IoT resources into oneM2M entities, and MQTT messages into oneM2M interface. The implementation of the translation system is validated in a real test case, and proved to properly work in a transparent manner.

The rest of the paper is organized as follows. Some relevant related work is described in Section II. In Section III, the design and architecture of the proposed system is presented while Section IV illustrates some implementation results. Finally, Section V offers concluding remarks.

II. RELATED WORK

Without standards, IoT systems and services would be developed independently for different vertical domains, causing high fragmentation problems and increasing the overall cost for development and maintenance. In order to mitigate the fragmented IoT ecosystem, several industry/international standards organizations have come together and published standard specifications on IoT systems.

 ISBN: 978-1-61208-598-2

A. oneM2M Standard

The oneM2M global initiative has made an effort to standardize a common service layer platform for globally-applicable and access-independent M2M/IoT services. Swetina et al. summarized well the oneM2M standardization activities [11]. The oneM2M first collected various compelling use cases from a wide range of vertical business domains. After that, it formulated requirements for the oneM2M common service layer and then designed the system architecture. The main goal of oneM2M is to define a globally agreed M2M service platform by consolidating currently isolated M2M service layer standards activities. The oneM2M standard is organized into five technical working groups focusing on M2M requirements, system architecture, protocols, security, and management, abstraction and semantics. The oneM2M standard adopted a RESTful architecture, thus all services are represented as resources to provide the defined functions.

Fig. 1 presents the oneM2M reference architecture model. Considering a configuration scenario where oneM2M systems are deployed, the oneM2M architecture divides M2M/IoT environments into two domains (infrastructure and field domain) and defines four types of nodes, which reside in each domain: Infrastructure Node (IN), Middle Node (MN), Application Service Node (ASN), and Application Dedicated Node (ADN). Furthermore, the oneM2M architecture is based on a layered model, which comprises the application layer, the common service layer, and the underlying network service layer, each of which is represented as an entity in the oneM2M system. The Application Entity (AE) represents application services located in a device, gateway, or server. The Common Service Entity (CSE) stands for an instantiation of a set of Common Service Functions (CSFs) that can be used by applications and other CSEs. CSFs includes registration, security, application, service, data and device management, etc.

B. AllJoyn Framework

AllJoyn is an open source IoT software framework developed under the guide of the AllSeen Alliance consortium [12]. Its role is to handle the complexities of discovering nearby IoT devices, creating sessions between them, and communicating securely. AllJoyn can run on multiple platforms and it supports multiple language bindings and transports, so that devices and applications from different manufacturers, running on different operating systems, written with different language bindings have a common way to interact to each other. The basic element of the framework is the AllJoyn bus, which enables the exchange of marshaled messages around the distributed system. AllJoyn provides its own bus based on the D-Bus Wire protocol, an Inter-Process Communication (IPC) and Remote Procedure Call (RPC) mechanism, and extends it to support distributed devices.

The AllJoyn network comprises AllJoyn Routers, which deliver messages within the network, and AllJoyn Applications, which include the application code and the AllJoyn Core Library. Applications implement and advertise one or more service objects, each of which exposes its functionality through AllJoyn interfaces. The framework

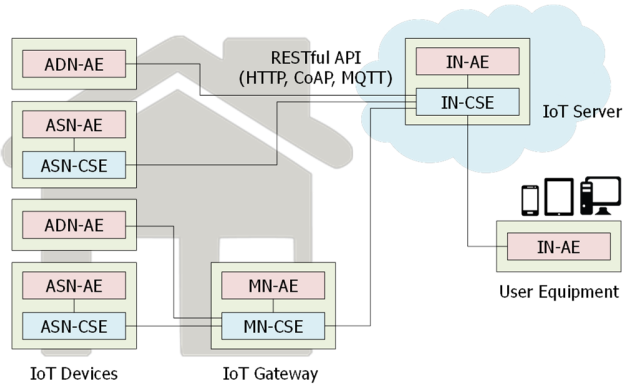

Figure 1. oneM2M reference architecture.

enables client applications to discover service objects advertised on the network, and then to invoke methods and properties, or to receive signals, provided by the interfaces. A consumer application invokes methods through a proxy object, which is a local representation of a remote service object.

C. MQTT

MQTT stands for message queuing telemetry transport. Unlike CoAP, TCP, UDP, it is used because MQTT specializes in low-bandwidth, high-latency environments; it is an ideal protocol for M2M communication. Basically, there are three components in MQTT: publisher, subscriber and broker. Here, the process of receiving and publishing the data is very much secure and accurate. Whenever the user wants to check or go through any data it sends the request to broker and upon receiving the request it sends to the publisher, it responds to the requests and sends the data that is requested by the subscriber and hence publishes the data, in overall process the communication is secure and up to the topic of interest. MQTT broker acts like a filter allowing only those data, which are requested thereby saving the flow of ambiguous data.

III. PROPOSED MQTT-BASED TRANSLATION SYSTEM

Middleware is a common approach to addressing interoperability. We developed a translation system as a middleware in the oneM2M platform in order to interwork with the non-oneM2M devices.

A. System Overview

The interaction between heterogeneous IoT systems ensured by a translation middleware should be done in a transparent manner, so that an application can communicate with any devices in the foreign system as if those devices were based on the same technology. Therefore, the design of the translation system primarily involves how to map the communication interface to a resource-oriented system based on oneM2M. The translation middleware also entails mapping both IoT resources into oneM2M entities, and MQTT messages into oneM2M interface.

Fig. 2 shows the concept of the translation system. The proposed system consists of things, a translation middleware, a server, a validation tool and a user device. Things include

 ISBN: 978-1-61208-598-2

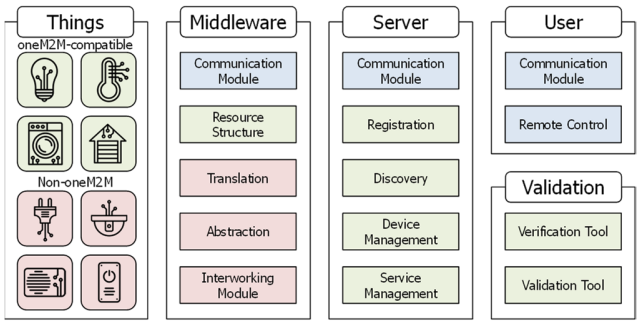

Figure 2. Concept of the proposed translation system.

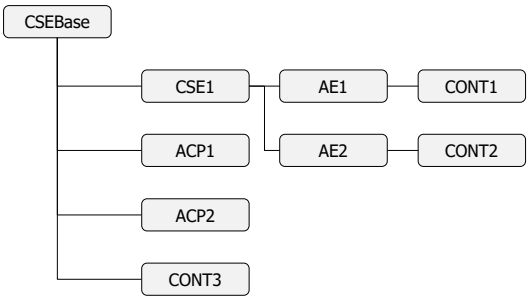

Figure 3. oneM2M-based resource structure.

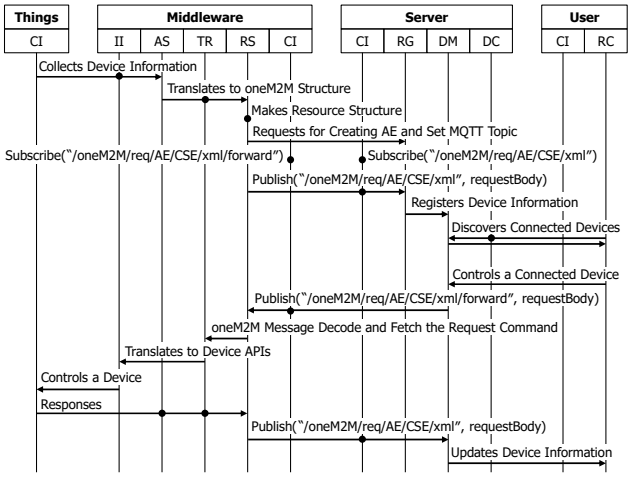

Figure 4. Sequence diagram of the MQTT-based translation system.

oneM2M-compatible devices and commercial IoT products that are not compatible with oneM2M but support open-source APIs and libraries. The translation middleware is basically working as a MN-CSE to communicate with oneM2M devices. For supporting non-oneM2M devices, the interworking module is developed in the middleware using open-source libraries supported by each IoT service platform. The middleware abstracts data from connected non-oneM2M things and translates their functionality to the oneM2M-based resource structure. The server provides high-level features to users; for example, registration and device management. To verify and validate the translation function, the proposed system runs several test tools for interoperability.

B. Design of MQTT-based Translation System

Since the oneM2M architecture adopted the Resource-Oriented Architecture (ROA) model, the services and data that oneM2M system supports are managed as a resource information model. With the ROA concept, resources in the ROA are uniquely addressed by the Uniform Resource Identifier (URI), and the interactions with the resources are supported by the basic four operations: create, retrieve, update, and delete. The oneM2M system manages its resources as a hierarchical structure as shown in Fig. 3. Starting from the root of CSEBase, resources are created as child resources, which represent services and data in the oneM2M system. When accessing the resource, the address of the resource should be represented as a hierarchical address that looks like the resource structure. For example, considering the CONT1 resource, its address with which it can be accessed is CSEBase/CSE1/AE1/CONT1. Additionally, oneM2M specifies a service layer protocol and its protocol binding with underlying delivery protocols including HTTP, CoAP and MQTT.

The oneM2M standards are developed for creating globally-applicable, access-independent IoT applications, but there exists a huge number of non-oneM2M systems already deployed across multiple domains. To interwork with the non-oneM2M systems, the proposed translation system provides non-oneM2M reference points and remapping the related data model into the oneM2M-defined data model, which are eventually exposed to other oneM2M systems. When translating data models, a full semantic interworking between two data models would be possible with the help of the related protocol interworking, but otherwise, the encoded non-oneM2M data and command messages will be packaged into a list of oneM2M containers. Consequently, the oneM2M applications need to know the protocol rules of the non-oneM2M systems to decode and understand the content within the containers.

Fig. 4 shows a sequence diagram of the MQTT-based translation system. First, the middleware searches for IoT devices through the Interworking Interface (II) module. After searching for devices, the Abstraction (AS) module collects device information and extracts features and functions from it. The Translation (TR) module translates its functionality to the oneM2M Resource Structure (RS). The Communication Interface (CI) modules in the middleware and the server subscribe a specific topic for the MQTT communication. When the server receives a resource structure from the RS module, the Device Management (DM) module first registers newly added devices through a Registration (RG) module. Users find IoT devices connected to the home network by requesting a device list from the Discovery (DC) module. Once users get the device list from the server, users can control IoT devices by using a Remote Control (RC) module. The control messages are translated to the corresponding device APIs in the TR module. After performing users' command, it needs to update the device status. Updating process is performed in a similar way to the registration step as we described above.

 ISBN: 978-1-61208-598-2

Figure 5. Configuration of the test environment.

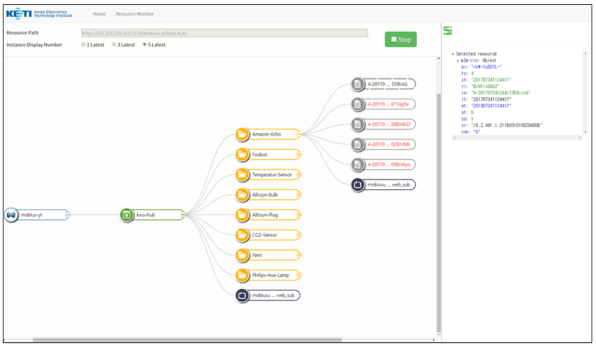

Figure 6. oneM2M resource discovery using a monitoring tool.

IV. Implementation Results

We conduct the implementation of the proposed system in a real environment. The test case is set up with a translation middleware, an IoT server and commercial IoT products, as shown in Fig. 5. The proposed system is built on oneM2M platforms, Mobius [13] and &Cube [14]. The translation middleware is implemented on a Raspberry Pi 3, which is s a credit-card sized single-board computer. The server runs an open-source MQTT broker, Mosquitto [15], on a Linux PC to interconnect with the middleware and users. We start with interworking AllJoyn-enabled devices, such as smart light bulbs and smart plugs. Then, we support various types of IoT products, e.g., thermostat, air quality monitor, smart home appliances.

To validate interoperability between IoT devices, we use a web-based resource monitoring tool supported from the oneM2M organization. When we access the server using the monitoring tool, the oneM2M-based resource structure is presented in a graphical form as shown in Fig. 6. If the translation system works correctly, the resource of non-oneM2M devices will be listed with oneM2M-compatible devices. We have shown that all the devices connected to the middleware works well in a real test case and can access their resources by using the oneM2M monitoring tool.

V. Conclusion

In this paper, we presented the design and implementation of a translation system that allows non-oneM2M devices to be accessible in oneM2M systems. On the device side, each device is connected to the middleware in order to advertise its resources and functions. The translation middleware provides an oneM2M-based hierarchical structure for each of these resources, which is exchanged with the IoT server. The IoT server is implemented as part of the translation system to provide high-level IoT functions for users. The implementation has been extensively tested in a real test case, and is built on open-source libraries. Currently, the proposed system supports a few commercial IoT services but the interoperability with oneM2M systems is fully tested.

For future work, we plan to support other commercial IoT solutions, especially sensor/actuation-type devices and voice-activated services. We also plan to modify the proposed system in order to satisfy the oneM2M release 2 specification.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT). (No.2017-0-00167, Development of Human Implicit/Explicit Intention Recognition Technologies for Autonomous Human-Things Interaction)

References

[1] J. A. Stankovic, “Research directions for the Internet of Things,” *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, Feb. 2014.

[2] The AllSeen Alliance. [Online]. Available: https://allseenalliance.org

[3] IoTivity. [Online]. Available: https://www.iotivity.org

[4] Thread. [Online]. Available: http://threadgroup.org

[5] Weave. [Online]. Available: https://developers.google.com/weave

[6] oneM2M - Standards for M2M and the Internet of Things. [Online]. Available: http://www.onem2m.org

[7] The FIWARE Catalogue. [Online]. Available: https://catalogue.fiware.org

[8] C. Vallati et al., “BETaaS: A platform for development and execution of machine-to-machine applications in the internet of things,” *Wireless Personal Communications*, vol. 87, no. 3, pp. 1071–1091, May 2015.

[9] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” *RFC 7252*, Jun. 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252

[10] “Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1,” *ISO/IEC 20922:2016*, Jun. 2016.

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

[12] The AllJoyn Core Framework. [Online]. Available: https://allseenalliance.org/framework/documentation

[13] J. Kim, S. Choi, J. Yun, and J. Lee, “Towards the oneM2M standards for building IoT ecosystem: Analysis, implementation and lessons,” *Peer-to-Peer Networking and Applications*, pp. 1-13, Sep. 2016.

[14] J. Yun, I. Ahn, N. Sung, and J. Kim, “A device software platform for consumer electronics based on the Internet of Things,” *IEEE Transactions on Consumer Electronics*, vol. 61, no. 4, pp. 564-571, Nov. 2015.

[15] Mosquitto. [Online]. Available: https://mosquitto.org

 ISBN: 978-1-61208-598-2