Interworking of oneM2M-based IoT Systems and Legacy Systems for Consumer Products

Jaeseok Yun
Dept of Internet of Things, SCH Media Labs
Soonchunhyang University
Asan, S. Korea 31538
yun@sch.ac.kr

Ramnath Chekka Teja, Nan Chen, Nak-Myoung Sung, Jaeho Kim
IoT Platform Research Center
Korea Electronics Technology Institute
Seongnam, S. Korea 13509
{ramnathteja, xuehu0000, nmsung, jhkim}@keti.re.kr

***Abstract*—The emerging Internet of Things (IoT) market and ecosystem will be fragmented due to heterogeneous systems and standards, which is often considered as one of the main challenges in the IoT research. The oneM2M initiative established in 2012 has been working on tackling such a fragmentation problem of IoT standards, and finally released its first specifications in January 2015. In particular, it defines a way of interworking oneM2M and non-oneM2M systems based on interworking proxy entity (IPEs). In this paper, we illustrate the implementation of oneM2M IPE for interworking with legacy systems of consumer products, including Nest thermostat, Jawbone fitness tracker, and Withings blood pressure monitor. The implemented IPE works well with the chosen products, and several interworking scenarios were successfully demonstrated at the oneM2M Showcase hosted by TIA (Telecommunications Industry Association) in Dallas, TX, U.S. in June 2015.**

I. Introduction

The Internet of Things (IoT), coined by Ashton in 1999 [1], represents a technological revolution that represents the future of computing and communications. Technological advances in identification systems (RFID), short-range wireless communication (ZigBee, Bluetooth, WiFi), cellular networks and widespread use of smartphones allow objects in the world to be connected with each other, sharing their status and surroundings, enabling the objects to work together and perform daily tasks without human intervention [2].

However, most IoT industries make use of proprietary systems composed of heterogeneous hardware and software components to provide their IoT services, which will probably limit their interoperability with each other. In particular, sensing is a foundational technology for IoT systems. Sensors are located at the edge of IoT systems (often embedded into low-power, wireless nodes), and play a major role for aggregating data from their surrounding environment to generate useful information for IoT services. However, due to the proprietary systems deployed in most IoT industries, it is difficult for an IoT solution of a given industry (e.g., smart homes) to take advantage of sensing data collected from other industries (e.g., smart cities or healthcare).

To mitigate the IoT market and standard fragmentation, the oneM2M standards initiative was founded by seven global standards development organizations (SDOs) in 2012[1]. Recently, TSDSI (India's telecom SDO) has joined the oneM2M initiative (eight SDOs in total). The objective of the oneM2M initiative is to develop globally-applicable specifications for machine-to-machine (M2M) and IoT service layer platforms based on close cooperation of the eight SDOs and six ICT fora including Broadband Forum and Open Mobile Alliance. Based on the global specifications, M2M/IoT applications can be efficiently developed and readily embedded within a wide range of hardware and software systems, finally being able to interoperate with each other on a global scale.

One of the Key issues of oneM2M standards is making the huge number of non-oneM2M systems that are already installed and those that will be deployed in future to be accessible via oneM2M systems. The first specification (oneM2M Release 1) defines a way of interworking oneM2M and non-oneM2M systems based on interworking proxy entities (IPEs), which support two main functionalities for interworking: protocol/message translation, and remapping non-oneM2M data model into oneM2M resource structure. By employing these interworking proxy entity's functionalities appropriately for a given non-oneM2M system we can achieve interworking between oneM2M and non-oneM2M systems, e.g., interworking with consumer products.

In this paper, we present the implementation of an oneM2M IPE for interworking with legacy consumer products. We have chosen three well-known products including Nest thermostat, Jawbone fitness tracker, and Withings blood pressure monitor. The main objective of the oneM2M IPE implementation is to enable IoT applications developed complying with oneM2M to access consumer product-related resources (collected data or control commands) through oneM2M service platforms. To this end, we have developed an IPE for our IoT platforms, which performs message translation, resource mapping, and authentication process handling.

Our work is particularly exciting because to the best of our knowledge this is the first effort to employ oneM2M standards for interworking with legacy consumer products. In spite of standardization activities such as oneM2M, many IoT companies including tech giants like Google are developing IoT products relying on their own proprietary systems, which is likely to cause a fragmented IoT ecosystem. We hope that this work will be the first step towards an interoperable IoT ecosystem with the widespread diffusion of oneM2M and IPEs in most industry verticals.

[1] http://www.onem2m.org/

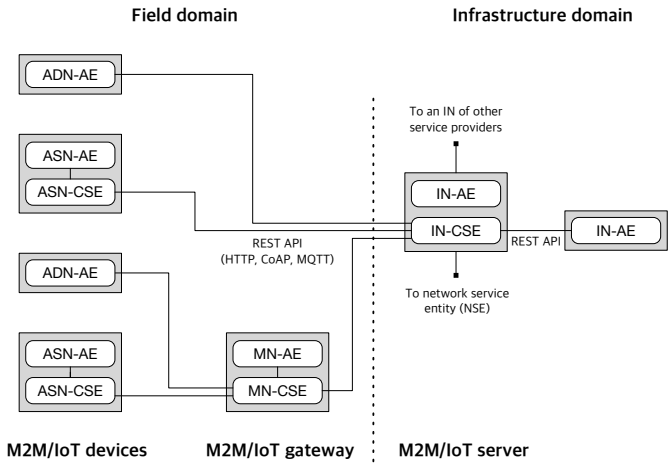

Fig. 1. oneM2M Reference Architecture.

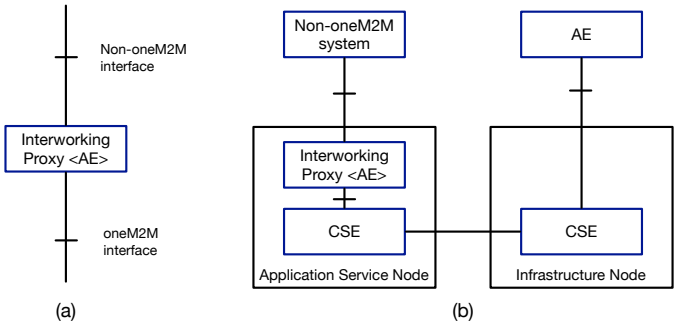

Fig. 2. (a) oneM2M IPE architecture and (b) its example.

II. oneM2M Standards

The oneM2M standards Release 1 was published in 2015. A thorough explanation of the oneM2M standards is out of scope, and its detailed description is available in the official oneM2M Release 1 specification documents [3]. Instead, Swetina *et al*. summarized well the oneM2M working groups (WGs) and standardization activities [4]. There are six working groups WGs for oneM2M: WG1-REQ for use cases and requirements ([5], [6]); WG2-ARC for architecture ([7]); WG3-PRO for protocols ([8], [9], [10], [11]); WG4-SEC for security ([12]); WG5-MAS ([13], [14]) for management, abstraction and semantics; WG6-TST for testing. Here, we briefly introduce the oneM2M reference architecture and IPE structure for better understanding of our implementation.

A. oneM2M Standards and Reference Model

As shown in Figure 1, the oneM2M architecture is based on a layered model comprising application layer, common service layer, and underlying network service layer. Each of these three layers are represented as an entity in the oneM2M system [6], [7]. The application entity (AE) represents application services located in device, gateway, or server. The common service entity (CSE) stands for an instantiation of a set of common service functions (CSFs) with which the oneM2M platform provides common services for the M2M/IoT service environments. The network service entity (NSE) implies network services from the underlying network to be utilized by the CSEs. In particular, CSFs provide common service functions shared by different applications, including registration, data management and repository, device management, security, discovery, and subscription/notification, etc. Considering a configuration scenario when 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 resides in each domain: infrastructure node (IN), middle node (MN), application service node (ASN), and application dedicated node (ADN).

B. Interworking Proxy Entities

The oneM2M standards define a specialized AE, named interworking proxy entity (IPE) for interworking with non-oneM2M systems, as shown in Figure 2a. Its functionalities are summarized as two main parts: supporting a non-oneM2M interface to translate non-oneM2M protocols and messages to oneM2M-specific ones, and mapping non-oneM2M data models into oneM2M resources, which are eventually exposed to oneM2M-based services. Figure 2b illustrates an example of interworking scenario with an IPE. In the example, an ASN consisting of an IPE and a CSE is connected with a non-oneM2M system (e.g., ZigBee-based temperature sensors) and an IN with an CSE to which an AE is registered. Here, the IPE needs to translate ZigBee-based protocols, and then create appropriate resources within the ASN-CSE. With this interworking capabilities, the AE (e.g., a weather app for smartphones) registered with the IN-CSE can access the temperature data collected from non-oneM2M systems (i.e., the ZigBee temperature sensors). There are currently a few ongoing work items (WIs) in oneM2M-based interworking, including WI-18 (AllSeen Alliance), WI-24 (lightweight M2M), and WI-44 (Open Interconnect Consortium).

III. Our Interworking Architecture

For oneM2M-based interworking with legacy IoT systems, we chose three consumer products (see Figure 3): Nest thermostat[2], Jawbone fitness tracker[3], and Withings blood pressure monitor (BPM)[4]. They are widely used as consumer products, but also provide easy-to-use development toolkits[5,6,7] such as a suite of representational state transfer (REST) application programmable interfaces (APIs) that enable developers to create third-party applications for the products.

Figure 4 illustrates our overall interworking architecture, where we assumed that an IPE (e.g., IPE 1) is created for a user (e.g., User 1) who owns the consumer devices.

We also assumed that the user interacts with his or her IPE and CSE via smartphone apps, named onePass and ConnecThing, respectively. As described above, the main roles of an IPE are protocol/message translation, and resource mapping into oneM2M-specific ones. Besides the oneM2M standard aspects, we also need to consider handling access tokens for a given consumer device, which will allow its corresponding

[2]https://nest.com/thermostat/meet-nest-thermostat/
[3]https://jawbone.com/up
[4]https://www.withings.com/us/en/products/blood-pressure-monitor
[5]https://developers.nest.com/
[6]https://jawbone.com/up/developer
[7]http://oauth.withings.com/api

Fig. 3. Three consumer products for oneM2M-based interworking: (a) Nest thermostat, (b) Jawbone fitness tracker, (c) Withings blood pressure monitor

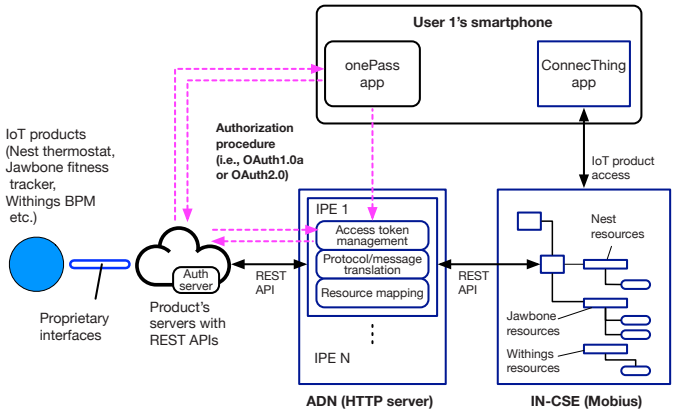

Fig. 4. Overall interworking architecture for consumer products.

IPE to access the device server to retrieve data or send control commands, as demonstrated with dashed lines in Figure 4. In Section 3-C, we will explain the full details on the authentication procedure for handling access tokens that allow access to the server systems for consumer products.

A. Protocol and Message Translation

Legacy IoT consumer products are usually designed to interwork with their servers via their own proprietary systems and interfaces. For example, Jawbone wrist tracker and Withings BPM are linked with smartphones via Bluetooth whose apps will send gathered data to servers. Thus, instead of trying to directly connect to the devices, we designed our IPE to be interworked with their servers via REST APIs provided, as shown in Figure 4. Accordingly, the device servers and IPE both use Internet protocols (i.e., HTTP), and all we need to do is to translate messages encapsulated within the HTTP protocol, which absolutely depends on the data model of each device.

B. Resource Mapping

After translating messages, IPEs need to map the encapsulated data into appropriate oneM2M resources which can be accessed through the ConnecThing app, as shown in Figure 4. However, a huge variety of data models (i.e., non-standard data structure) for consumer products make it difficult to perform fine-grained mapping strategy for interworking. Thus, we decided to create a `<container>` oneM2M resource type for a given device, and then map a set of resources of

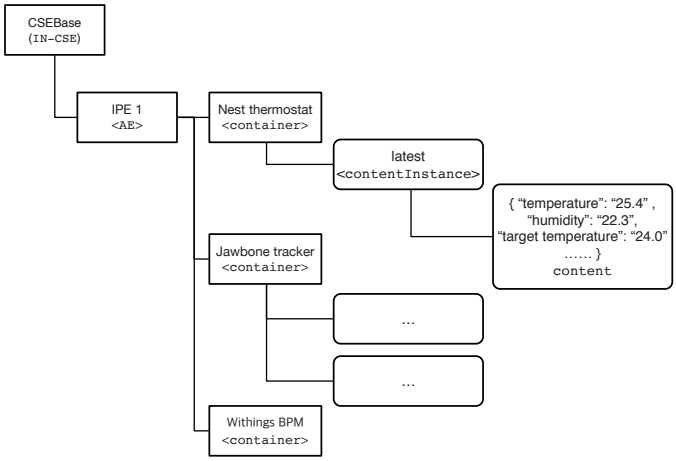

Fig. 5. An example of resource mapping for a Nest thermostat.

the device at a given time all into a `<contentInstance>` oneM2M resource type, which represents a data instance in a `container`. Figure 5 shows an example of resource mapping for the Nest thermostat. In this example, we can see that the Nest thermostat-specific data such as temperature (25.4), humidity (22.3), target temperature (24.0) all are mapped into a `contentInstance` as a `content`, an actual content of `contentInstance`. Finally, this resource mapping procedure is scheduled to perform at a regular interval to sync data between oneM2M and non-oneM2M systems.

C. Access Token Management

Besides the above two functions of IPE, we should consider security for accessing device data. Security is one of key issues in IoT systems, and much works for securing them are actively underway [15]. Both Nest and Jawbone servers provide developers with OAuth 2.0 authorization, while Withings server provide OAuth 1.0a. Thus, we have created two authorization process for interworking with OAuth 1.0a and OAuth 2.0 for non-oneM2M systems, respectively.

OAuth 1.0a [16] and OAuth 2.0 [17], developed by the Internet Engineering Task Force (IETF), are an authorization framework that enables third-party clients to access resources stored in a data server with the resource owner provided access token. Although it has been announced that OAuth 2.0 would replace and obsolete OAuth 1.0, there exist a myriad of IoT systems still using OAuth 1.0 as in Withings servers. The main difference between OAuth 1.0a and OAuth 2.0 can be described as follows (summarized from [18]):

- OAuth 2.0 adds six new flows that better support applications running on a desktop or mobile device, including user-agent flow, Web server flow, device flow, username and password flow, client credentials flow, assertion flow;
- provides a cryptography-free option for authentication;
- simplifies signature support;
- issues short-lived access tokens with long-lived authorizations;

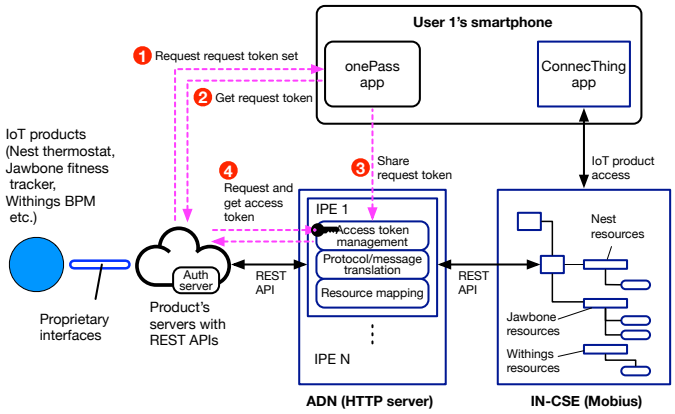

Fig. 6. Authorization procedure for OAuth 1.0a-based systems (e.g., Withings servers).

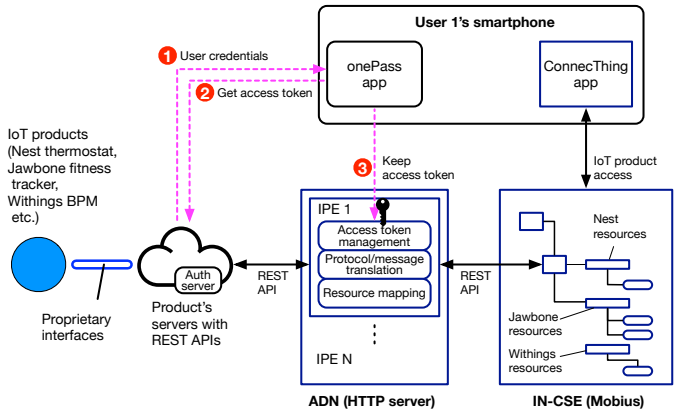

Fig. 8. Authorization procedure for OAuth 2.0-based systems (e.g., Nest and Jawbone servers).

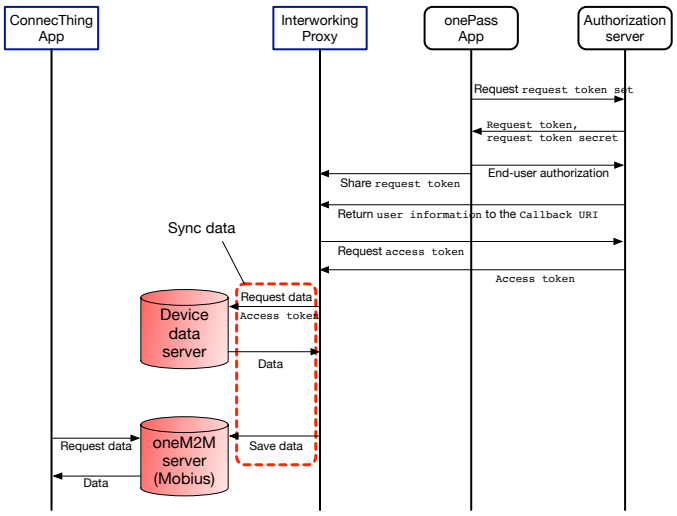

Fig. 7. Overall message flow of OAuth 1.0a-based systems (e.g., Withings servers) for data authorization and mirroring.

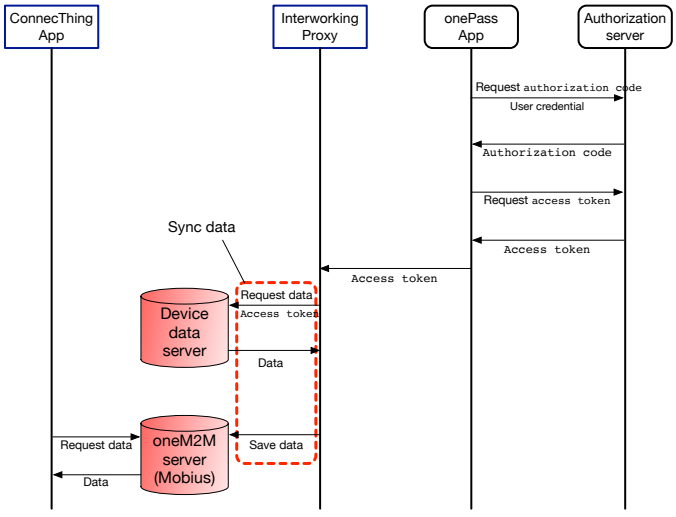

Fig. 9. Overall message flow of OAuth 2.0-based systems (e.g., Nest and Jawbone servers) for data authorization and mirroring.

- separates the role of the authorization server responsible for dealing with user authorization and the server responsible for dealing with OAuth requests and issuing tokens.

First, we explain the OAuth 1.0a-based authorization procedure for Withings servers. Figure 6 and Figure 7 present the authorization procedure and overall message flow for interworking of OAuth 1.0a-based systems, respectively. In Figure 7 authorization starts with a 'request token set' request. This token set which contains token and token secret will expire in two minutes after acquisition. The request token is used as one of the parameter in End-user authorization request whereas the request token secret is used in the applying hmac-sha1 to base string (see Withing developer site for more details). The request token set 'token' is needed to be shared with IPE so that when the Withings server call backs IPE, the IPE can match which user's request the call back is from. Since the token set expires in two minutes and the IPE does not have any knowledge on token secret, there is no security compromise from IPE.

Next, we explain the OAuth 2.0-based authorization procedure for Nest and Jawbone servers. Figure 8 and Figure 9 present the authorization procedure and overall message flow for interworking of OAuth 2.0-based systems, respectively. The OAuth 2.0 provides a relatively simpler way compared to the OAuth 1.0a. In Figure 9, an 'authorization code' is first requested, and then the corresponding access token is obtained from the authorization server.

Both Figure 7 and 9 follow same authorization process described in the OAuth 1.0a [16] and OAuth 2.0 [17] respectively, which means that user credentials (e.g., ID and password) do not need to be shared with clients (e.g., onePass or IPE) for authorization. With the issued access token passed from the Auth server (i.e., OAuth 1.0a) or from the onePass app (i.e., OAuth 2.0), the IPE can access resources in the device servers, and sync them with the oneM2M data server. Finally, the ConnecThing app can retrieve the synced data, or send control commands to the device servers.

IV. Implementation

Based on the above architecture, we have implemented the IPE on an HTTP server, onePass app, and ConnecThing app for interworking with Nest, Jawbone, and Withings servers.

A. IN-CSE

We have implemented the IN-CSE using 'Mobius' [19], an oneM2M-compliant IoT server platform, provided by the OCEAN (Open allianCE for iot stANdards)[8]. The OCEAN also provides an oneM2M platform for devices (e.g., IoT gateway), called &Cube [20]. It has already been shown that using open sources for the Mobius and &Cube can facilitate development and deployment of smart IoT services [21], [22].

B. IPE and onePass App

The IPE is implemented as an AE registered with the IN-CSE (i.e., Mobius). Although the AE can be resided in the IN, we implemented it in a different oneM2M node, ADN (i.e., HTTP server), as shown in Figure 6 and 8.

We implemented the onePass app for authorization process of OAuth 1.0a (e.g., Withings) and OAuth 2.0 (e.g., Nest, Jawbone), as shown in Figure 10 (captured from the onePass for the Withings) and 11 (captured from the onePass for the Nest). The onePass app for OAuth 2.0 directs the user to the device servers (i.e., authorization servers trusted by the device servers) and get the authorization code presenting the user's grant (see Figure 11a-d). Subsequently, by submitting the authorization code to the authorization server, the access token is issued and passed to the IPE (see Figure 11e-f). Almost similar steps are required for Jawbone server authorization.

Since OAuth 1.0a follows a different authorization process from OAuth 2.0, the onePass app for OAuth 1.0a works a bit different from the onePass app for OAuth 2.0. The Nest and Jawbone onePass app will acquire the access token and pass the token to IPE which is then used to provide services for the user, whereas the Withings onePass app will just request the Withings server to issue credentials and token required for access token request to a call back server. In this case, the call back server is the IPE itself (see Figure 10c). Once the IPE acquires the required credentials and token, the IPE itself will perform access token request and let the onePass know the status of the request (see Figure 10d).

With the issued access token, the IPE can perform its genuine functions for interworking: message translation and resource mapping. To this end, we investigated the REST APIs and JSON data provided from the legacy IoT product servers. The device data encapsulated in a JSON object is translated into a string, and mapped to a `contentInstance`, as described in our interworking architecture (see Figure 5). Thus, the IoT device data can be exposed to other oneM2M applications via REST APIs provided. For example, `http://open.iotmobius.com/Mobius/AE-0.2.481.2.0001.001.7591/container-nest_thermostat/latest` is a REST API to obtain the latest data of the Nest thermostat with the assumption that the CSE's base URL is `http://open.iotmobius.com/Mobius/`, and the

[8]http://iotocean.org/

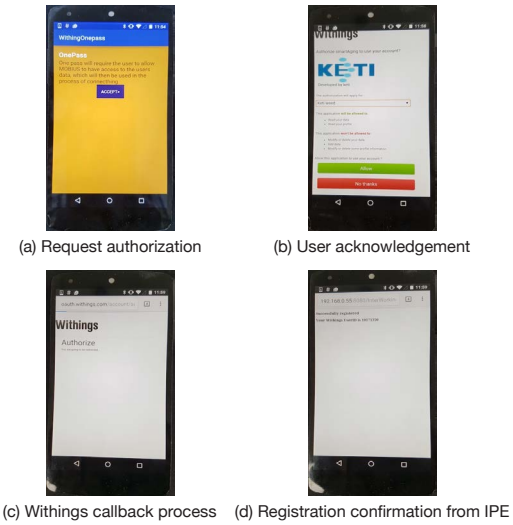

Fig. 10. Obtaining the access token with the onePass app by interacting with the Withings server (i.e., its OAuth 1.0a authorization server).

Fig. 11. Obtaining the access token with the onePass app by interacting with the Nest server (i.e., its OAuth 2.0 authorization server).

IPE's URI is `http://open.iotmobius.com/Mobius/AE-0.2.481.2.0001.001.7591`. Finally, by performing this procedure at a regular interval, the Mobius will be able to keep the device data synced with its server.

The IPE is also implemented to get a notification message as long as the device resources within the Mobius it subscribes to are changed (i.e., subscription and notification function defined in oneM2M CSEs). For example, when the user sets the new target temperature for Nest thermostats via the ConnecThing app (explained below), the IPE will immediately receive a notification message from the Mobius, and pass the command to the Nest server.

C. ConnecThing App

Finally, we created a smartphone application (i.e., IN-AE), called ConnecThing, which enables users to interact with the

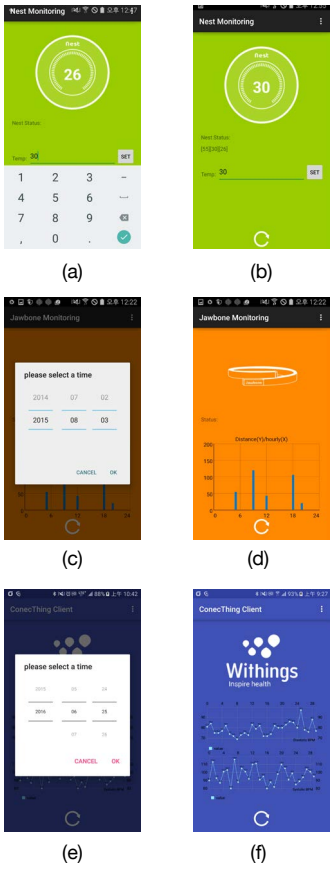

Fig. 12. Examples of interacting with the Mobius via the ConnecThing app, (a) set the Nest thermostat to a new temperature (30 degree), (b) update the Nest thermostat, (c) retrieve the Jawbone data on a given day, (d) plot the Jawbone data, (e) retrieve the Withings BPM data on a given day, (f) plot the Withings BPM data (spread out due to multiple subjects and trials).

Mobius, that is, retrieve data collected in IoT device servers, or send the servers control commands. The ConnecThing app is implemented using REST APIs the Mobius provides, so any developers who are familiar with the APIs will be able to create their new applications. Figure 12 shows examples of interacting with the Mobius for retrieving data from the Jawbone server (a-b), sending a control command to the Nest server (c-d), and retrieving data from the Withings server (e-f).

V. Discussion and Conclusion

Although we implemented the authorization process for the IPE to gain access to data in OAuth-based servers, there is now no secure connection between the onePass and ConnecThing apps to share IPE information. This implies that the ConnecThing app has no way to securely discover the user's IPE ID or name within the Mobius, which is the future work we are investigating. We also consider standardization activities in the oneM2M initiative based on the lessons learned from the work.

We have presented an implementation work for interworking oneM2M systems with legacy consumer products from Nest, Jawbone, and Withings. We have implemented an interworking proxy entity (IPE) defined in the oneM2M standards, enabling users to interact with legacy systems via the oneM2M platform (i.e., Mobius). We have also implemented smartphone applications, onePass and ConnecThing, each for a common authorization for legacy IoT product access and GUI-based user interaction via the Mobius, respectively. We expect that the implementation work will help developers who are interested in building oneM2M-based interworking scenarios with non-oneM2M systems.

Acknowledgment

This work was supported by Institute for Info. & comm. Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.B0184-15-1003, The Development of oneM2M Conformance Testing Tool and QoS Technology).

References

[1] K. Ashton, "That 'Internet of Things' Thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, July 2009.

[2] ITU-T, "ITU Internet Reports 2005: The Internet of Things – Executive Summary," 2005.

[3] oneM2M. http://www.onem2m.org/technical/published-documents (2016).

[4] 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, June 2014.

[5] oneM2M. TS-0002-Requirements-V-1.0.1. Technical Specification (2015)

[6] oneM2M. TS-0011-Common-Terminology-V-1.2.1. Technical Specification (2015)

[7] oneM2M. TS-0001-Functional-Architecture-V-1.6.1. Technical Specification (2015)

[8] oneM2M. TS-0004-Service-Layer-Core-Protocol-Specification-V-1.0.1. Technical Specification (2015)

[9] oneM2M. TS-0008-CoAP-Protocol-Binding-V-1.0.1. Technical Specification (2015)

[10] oneM2M. TS-0009-HTTP-Protocol-Binding-V-1.0.1. Technical Specification (2015)

[11] oneM2M. TS-0010-MQTT-Protocol-Binding-V-1.0.1. Technical Specification (2015)

[12] oneM2M. TS-0003-Security-Solutions-V-1.0.1. Technical Specification (2015)

[13] oneM2M. TS-0005-Management-Enablement(OMA)-V-1.0.1. Technical Specification (2015)

[14] oneM2M. TS-0006-Management-Enablement(BBF)-V-1.0.1. Technical Specification (2015)

[15] S. L. Keoh, S. S. Kumar, and H. Tschofenig, "Securing the Internet of Things: A Standardization Perspective," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 265–275, June 2014.

[16] E. Hammer-Lahav, "The OAuth 1.0 Protocol," Internet Engineering Task Force, RFC 5849, April 2010.

[17] D. Hardt, "The OAuth 2.0 Authorization Framework," Internet Engineering Task Force, RFC 6749, October 2012.

[18] E. Hammer-Lahav, "Introducing OAuth 2.0," Hueniverse, 2010.

[19] J. Kim, S.-C. Choi, J. Yun, and J.-W. Lee, "Towards the oneM2M Standards for Building IoT Ecosystem: Analysis, Implementation and Lessons," *Peer-to-Peer Networking and Applications*, (revision).

[20] 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 Transactions on Consumer Electronics*, vol. 51, no. 4, pp. 564–571, December 2015.

[21] M. Ryu, J. Kim, and J. Yun, "Integrated Semantics Service Platform for the Internet of Things: A Case Study of a Smart Office," *Sensors*, vol. 15, no. 1, pp. 2137–2160, January 2015.

[22] J. Yun, I.-Y. Ahn, S.-C. Choi, and J. Kim, "TTEO (Things Talk to Each Other): Programming Smart Spaces Based on IoT Systems," *Sensors*, vol. 16, no. 4, pp. 467, April 2016.