International Journal of Control and Automation
Vol. 10, No. 10 (2017), pp.101-112
http//dx.doi.org/10.14257/ijca.2017.10.10.09

Design and Implementation of IoT Interworking of Anchor Service Provider and Mobius Platform Using RESTful API

Lei Hang[1] and Do-Hyeun Kim[2]

[1]*Computer Engineering Department,*
Jeju National University, South Korea
[1]*hanglei112233@hotmail.com,* [2]*kimdh@jejunu.ac.kr*

Abstract

Recently, the number of connected smart objects will radically increase in IoT(Internet of things) landscape. The emerging need for cross-domain IoT applications and services highlights the necessity of interoperability across IoT platforms is a big challenge for the development trend of the internet of things. This paper present IoT interworking architecture based on anchor service provider for connecting heterogeneous IoT platforms. Also, we design interworking between an anchor IoT service provider and Mobius platform based on proposed interworking architecture to support interoperability of heterogeneous IoT environment. And, we implement the interworking of anchor service provider and Mobius platform using a RESTful API. All client can access IoT resources of the Mobius platform thought the anchor service provider using HTTP protocol.

Keywords*: Interworking, Internet of things, Mobius, HTTP, RESTful*

1. Introduction

In the recent years the current internet has been given a whole new dimension by interconnecting all the physical objects, devices and their virtual representation. Internet of things (IoT) enables a global connectivity between the real world and a virtual world of entities or things [1]. These IoT devices detect and interact with their environment to take and give operation commands. These devices allow a growing number of applications to provide people with digital aids for their daily activities. In this ever-changing landscape of IoT, recent researches imply that the number of intelligent connected objects will increase dramatically over the next few years [2]. This flourishing of smart devices will absolutely promote the scale of IoT business [3], bring new services and technologies to pave the way for revolutionary applications.

Growth will also have side effects that will challenge the continuity of present and prospective IoT solutions. The gap between the sensors of the device and the data networks is filled by an IoT platform and there are many IoT platforms available now that provide the option of deploying Internet applications to applications [4]. As a result, there is a pressing need for multi-domain IoT applications that can cover multiple fields of daily routine is increasingly apparent today. Mobius [5] is an operating system for IoT devices that enables a broad range of value-added activities and services. It is a server platform of IoT services complying with globally-accepted, widely-used IoT standards oneM2M specifications.

oneM2M is the global standards initiative for Machine to Machine Communications and the Internet of Things. The oneM2M is intended to develop a standardized service layer that can be easily integrated into different hardware and software in the field with M2M application around the world [6], [7]. The Mobius has recently published its first open source release that enables developers to construct their own IoT applications based on oneM2M standard specification, we utilized the Mobius source code and test the feasibility of interworking with other IoT service platforms.

ISSN: 2005-4297 IJCA

In this paper, we propose IoT interworking architecture based on anchor service provider for connecting heterogeneous IoT platforms with Non-oneM2M and oneM2M. IoT platforms support to collect and store for provision environmental context information. And, we design and implement interworking using a RESTful API based on HTTP protocol to exchange a context data between Mobius platform and anchor service provider. The experiment demonstrates successful interworking between these two platforms and show the effectiveness of the proposed interworking architecture. In the remainder of this paper, we overview the related work to service provider and Mobius as well as the oneM2M based interworking technologies in section 2. In section 3, we present the approach of IoT interworking architecture based on anchor service provider. Section 4 describes the design of IoT interworking between an anchor service provider and Mobius platform. Section 5 illustrates the implementation of the proposed interworking architecture and experiment result. Finally, section 6 discusses the conclusion and future works.

2. Related Works

Mobius platform is an operating system for IoT devices that enables a broad range of value-added activities and services. Mobius platform is a server platform of IoT services. It complies with oneM2M, the global standards initiative for M2M and the Internet of Things. As depicted in Figure 1, Mobius platform connects with IoT devices and applications. IoT devices support HTTP, CoAP and MQTT protocol, on the other hand, the application only supports HTTP. MySQL DBMS is used for storing resources in Mobius platform. Many IoT projects and research contributions are geared towards development of prototypes, standards and process automation.

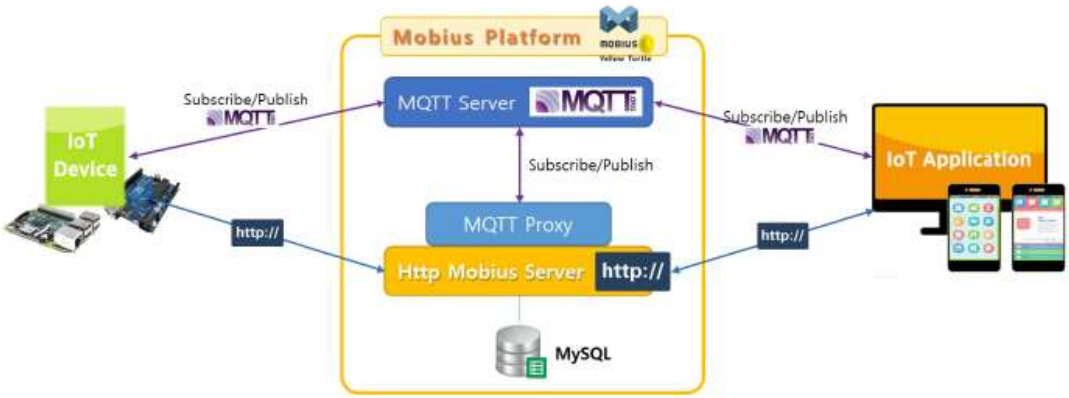

Figure 1. Mobius Platform Architecture

The current version of Mobius platform is developed based on Node JS using Java Script. It uses MySQL DBMS and support HTTP/MQTT protocols. Mobius platform uses HTTP as a default protocol. There's also a proxy in the platform which can transfer MQTT to HTTP. All of the HTTP request must be aggregated at the router and after going through parser – actor, data can be stored in the DB. Finally the response is handled in the responder.

The goal of oneM2M is to interoperate with heterogeneous hardware and software components world widely. oneM2M through its global membership to share considerations and issues for supporting better features of the standard. For example, IoT-A [11] is standard IoT reference model project designed in EU that has many resemblances with the oneM2M architecture.

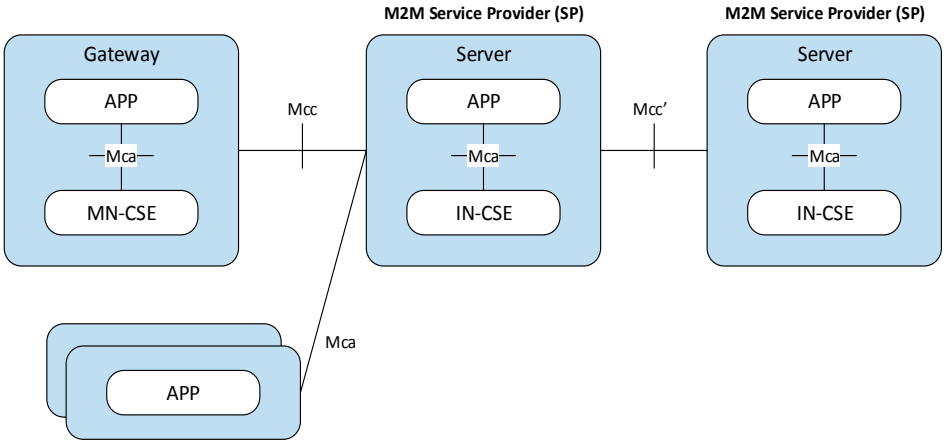

Figure 2. oneM2M Interworking Architecture

In order to interact with other standards, oneM2M is able to link several protocols for communications such as HTTP, CoAP [12], HTTP protocol, *etc*. The protocol working group has recently released a protocol analysis [13], which analyzes the existing communication protocols such as HTTP, CoAP, and XMPP in order to provide the system with the capable of interacting with different protocols A semantic interworking solution described in a technical report that introduces interworking proxy functions in order to allow an M2M application to use devices from other technologies such as 6LoWPAN and Zig-Bee. On this occasion, the M2M application just needs to know the oneM2M information model and semantics of the local interface.

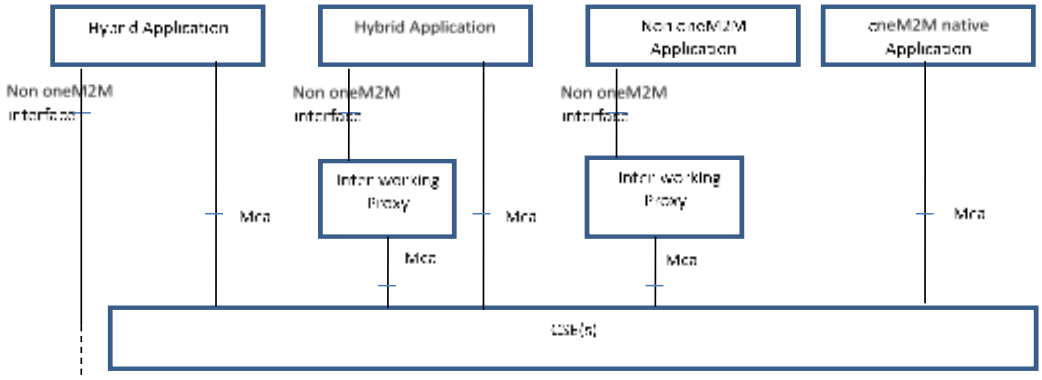

Figure 3. oneM2M Interworking with Non oneM2M Platform

oneM2M is required to interact with the existing M2M standard and oneM2M based services and infraconfiguration. In oneM2M communication, there are three defined reference points (*i.e.*, Mca, Mcc, and Mcc') for communication between oneM2M entities as indicated in Figure 2. For example, the communication between AEs and CSEs are enabled the Mca reference point; Mcc is responsible for communication between CSEs in the identical domain; Mcc' provides functions similar to Mcc, but communication is taken into account on different M2M service provider domains. Mcc' extends the service accessibility of one service provider to other service provider domains. To permit interworking between applications with other components (device/gateway/server) in oneM2M, the Mca must follow the interoperability of oneM2M between AE and CSE. The Mcc should be interoperated under oneM2M under the circumstance of interworking between device and gateway or between gateway and server. The Mcc 'handles communication between IN-CSEs in the same domain for service layer communication

and provide a common and essential solution for the integration of multi-vendor methods in the IoT domain.

In the case of collaboration with other standard, OIC Interworking [14], oneM2M and AllJoyn Interworking [15], LWM2M Interworking [16] and 3GPP_Rel13_IWK [17] are published as documents. Interworking between OIC, LWM2M and AllJoyn are implemented by Interworking Proxy Entity (IPE) which is deployed between application entities and CSEs. IPE [18] enables the integration between oneM2M systems and non-oneM2M systems. The task of the interworking proxy shown in Figure 3.

3. Proposed IoT Interworking based on Anchor Service Provider

We present IoT interworking architecture based on anchor service provider. The proposed IoT interworking architecture comprises three parts: app client, anchor service provider, other service providers. App client is a stand-alone application running on the client machine which provides a visual view to ease the interaction with the service domain and offers a high-level API to access different service providers and support service discovery and management of IoT resources. Anchor service provider offers interworking APIs in accordance with other service providers to enable cooperation and cross-platform discovery, which acts as an intermediary module between app client and other service providers. Other service providers provide organization with communications, storage, processing and many other services in their own domains. The App client is a HTTP client which offers an API requestor for requesting the anchor service provider.

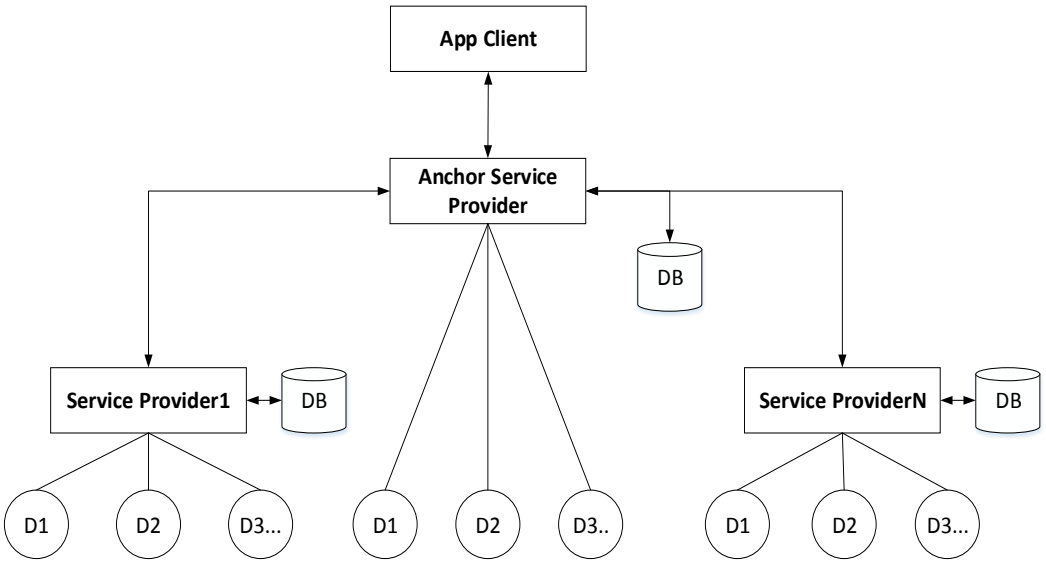

Figure 4. Anchor Service Provider based IoT Interworking Architecture

The anchor service provider acts as a HTTP client as well as a HTTP server. It providers an API receiver that receives the request from the app client and perform tasks according to the request body. The Anchor Service Provider also offers interworking requesters in accordance with other service providers for accessing other services. For example, the anchor service provider request the service provider1 using the interworking API requestor for querying the IoT resources. The interworking receiver of the service provider1 receive and send command to the related database to get resource information according to the request body then the service provider1 responds the retrieved resource info to the anchor service provider and finally the resource info is sent to the App Client.

Figure 5 represents the proposed interworking architecture between the service provider and Mobius platform. The app client implements the client-side functionality offering an API requestor for requesting the resource in the service provider. The service provider acts as a HTTP client as well as a HTTP server. To enable the connection with Mobius platform, we implement a RESTful API which receives the request from app client and perform tasks according to the request body. The service provider also offers a unified view on different IoT middlewares enabling management of middlewares which monitor and control the IoT devices. Mobius platform offers interworking API receiver which receives the request from Semantic IoT Platform and perform tasks according to the request body. IoT resource information of Mobius is stored in the Mobius DB which is related to the Mobius platform and can be accessed through HTTP protocol. Mobius platform retrieves the resource information from DB and respond back the service provider. The service provider receives the response and send to the app client to display resource information to user.

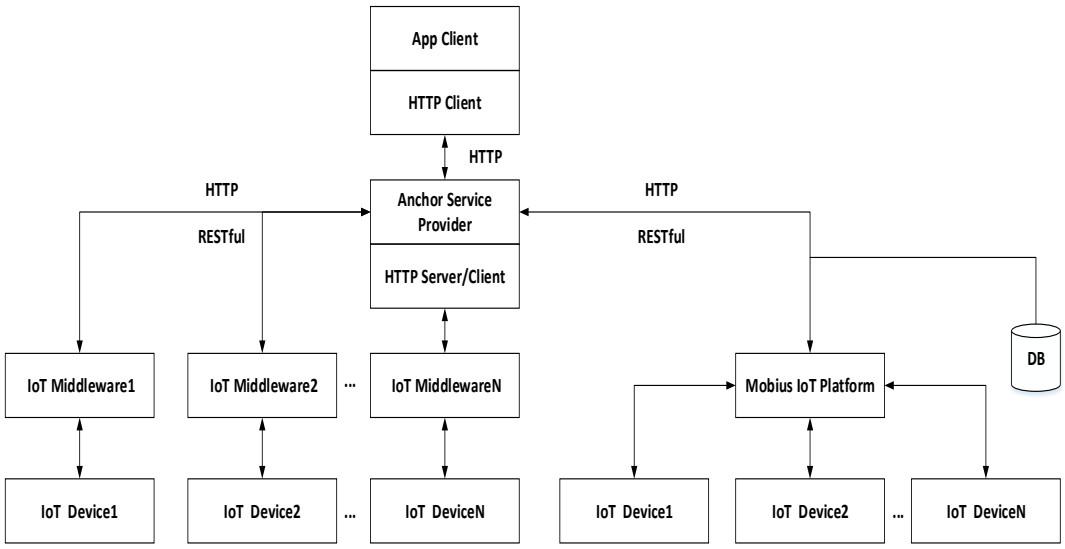

Figure 5. Proposed Interworking Architecture between Anchor Service Provider and Mobius platform

4. Design of IoT Interworking of Anchor Service Provider and Mobius Platform

In the proposed use case, the app client sends a HTTP request to the service provider. The service provider supports HTTP based server and client functionalities. The service provider receives the request from the client and forward to the REST Requester which invokes the web API to access the Mobius Server by the specified URI. The Mobius Server then calls the REST Receiver to connect the database to get the resource information and send response to the service provider. Similarly the service provider receives the response and transmit to the App Client. In order to provide the communication capability, we implemented a HTTP resource in the service provider for the app client to access and the Sensor service provider directly calls the API whenever receives the request from the App Client. We implemented the JSON Converter in the service provider in order to convert the response format from XML to JSON. The app client has a JSON Deserializer module which is used to parse the response from JSON to string format that is shown to the user in Figure 6.

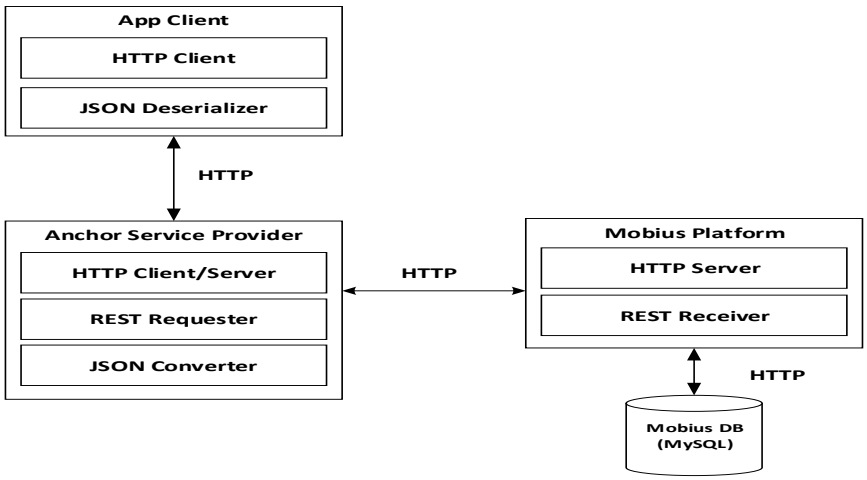

Figure 6. Detailed Interworking Architecture Based on Anchor Service Provider

Figure 7 shows anchor service provider with REST API detailed architecture. Service interface provides access interface to outside service. Sensor middleware is responsible for the device management such as device registration and control. Sensing web sensing service is used to access the sensor state management and sensing data receiver that collecting the sensing data from physical devices. In order to provide communication functionality with the Mobius platform. The Mobius interworking service is implemented in the Service control module. We also developed a user interface to control the interworking service in the service interface module. The communication between the app client and the service provider is using HTTP protocol and the format for data exchange is in JSON format.

Figure 8 illustrates the sequence of the proposed interworking process. In this case, IoT device establishes the connection via a secure association to perform the registration in IoT gateway. After the connection is successfully established, the IoT device starts to send the information for the registration CREATE procedure. IoT gateway receives the request info and on successful validation of the CREATE request, the IoT gateway creates the requested resource. IoT gateway responds with a response message to the IoT device. IoT gateway sends a CREATE request for the created resource to Mobius platform. When the validation and verification is completed successfully, Mobius platform creates the requested resource. Mobius platform sends a successful response to the IoT gateway. The created resource is stored in Mobius DB. Then the user starts the app client and request the service provider URI to perform resource querying service. The service handler in the service provider handles the request and then request the resource URI in Mobius platform. Mobius platform retrieves the resource info from related DB and respond the resource info in Xml format to the service provider. The service provider converts the response info from Xml to JSON format and respond to app client. And then the app client displays the response information.

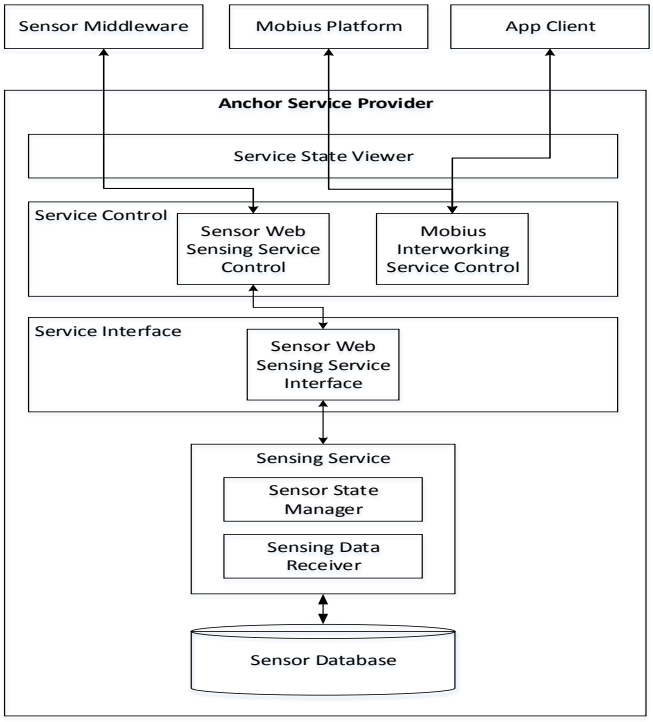

Figure 7. Detailed Anchor Service Provider Configuration

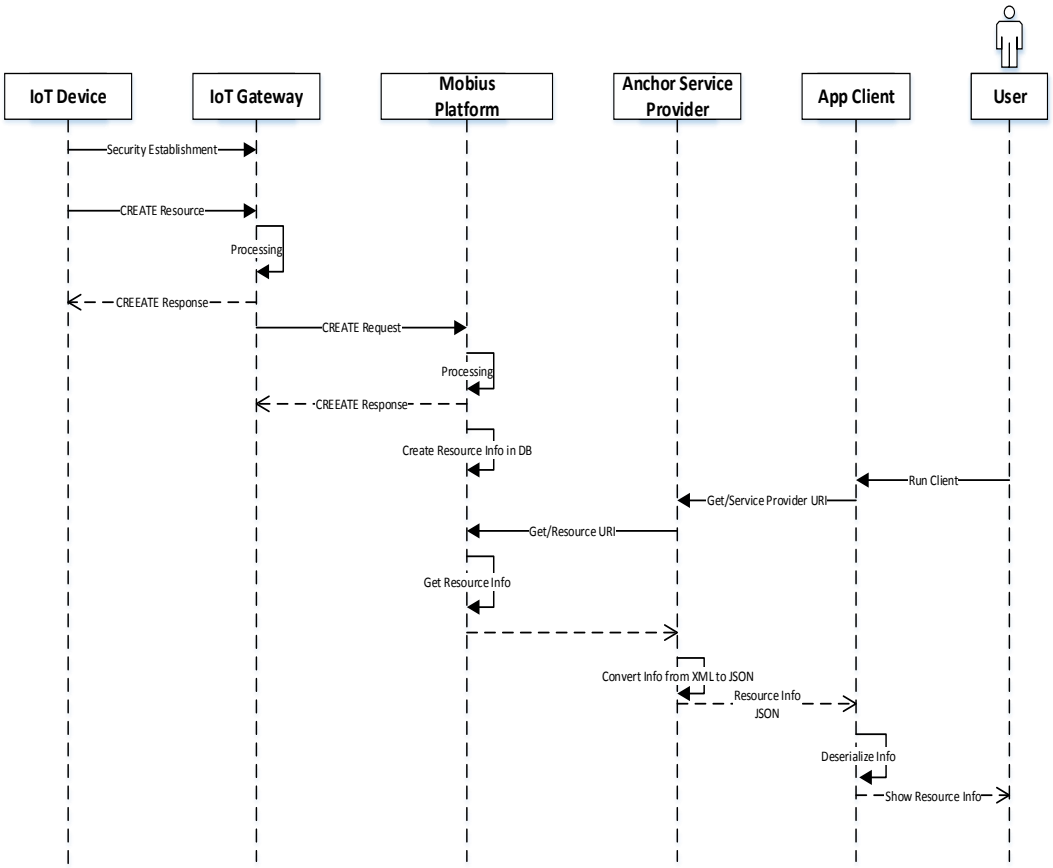

Figure 8. Proposed Interworking Sequence Diagram

5. Implementation and Performance Evaluation

The app client implementation environment is described in Table1 for the proposed interworking configuration. This application is developed with Visual Studio2015 using C#. We apply Newtonsoft.Json, .Net.Http and Windows Forms libraries in this application. The application is developed in Windows 7 Ultimate 64 bit.

Table 1. Implementation Environment of App Client

App Client	
Development Tool	Visual Studio Community 2015
Library	Newtonsoft.Json, .Net.Http, Windows.Forms
Language	C#
OS	Windows 7 Ultimate 64 bit

The anchor service provider implementation environment is described in Table 2 for the proposed interworking configuration. This application is developed with Visual Studio2015 using C#. We apply XML, Newtonsoft.Json, .Net.Http and Windows Forms libraries in this application. The application is also developed in Windows 7 Ultimate 64 bit.

There is a user interface which provides a graphics for human-machine interaction. User can start or stop the service and also check the service status intuitively from the user interface. Service Host module provides a host for services. This module defines the type and endpoint of service and its base address. Service module defines the main function of the service. The request and response operations are performed in the module. Value Object module is used to represent the resource entity. To support all the functionalities, we apply four libraries (Service Model, .Net HTTP, XML and Windows Forms).

Table 2. Implementation Environment of Anchor Service Provider

Anchor Service Provider	
Development Tool	Visual Studio Community 2015
Library	XML, .Net.Http, Windows.Forms, ServiceModel
Language	C#
OS	Windows 7 Ultimate 64 bit

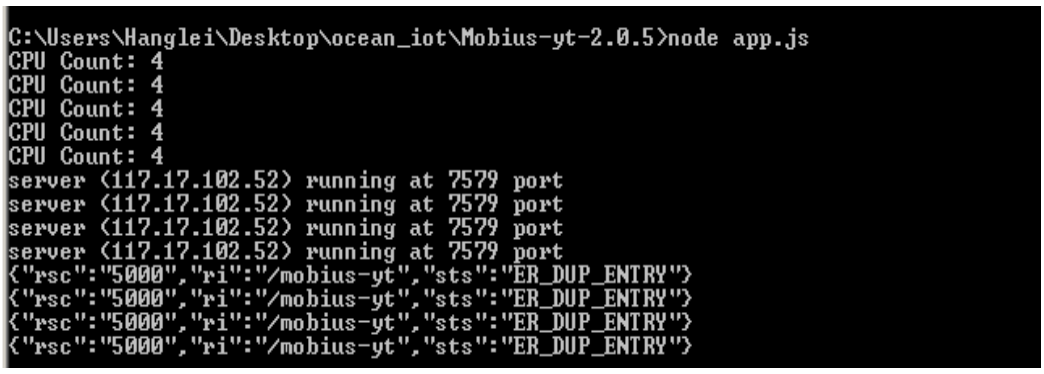

Figure 9. Mobius Server Initialization in Command Prompt

Figure 9 shows the operating result of starting the Mobius platform. To start the Mobius platform, we need to open a new console window and navigate to the root directory of the Mobius platform. After typing the following command, the Mobius

platform is normally started. The console window provides information (ip address and port) of the server.

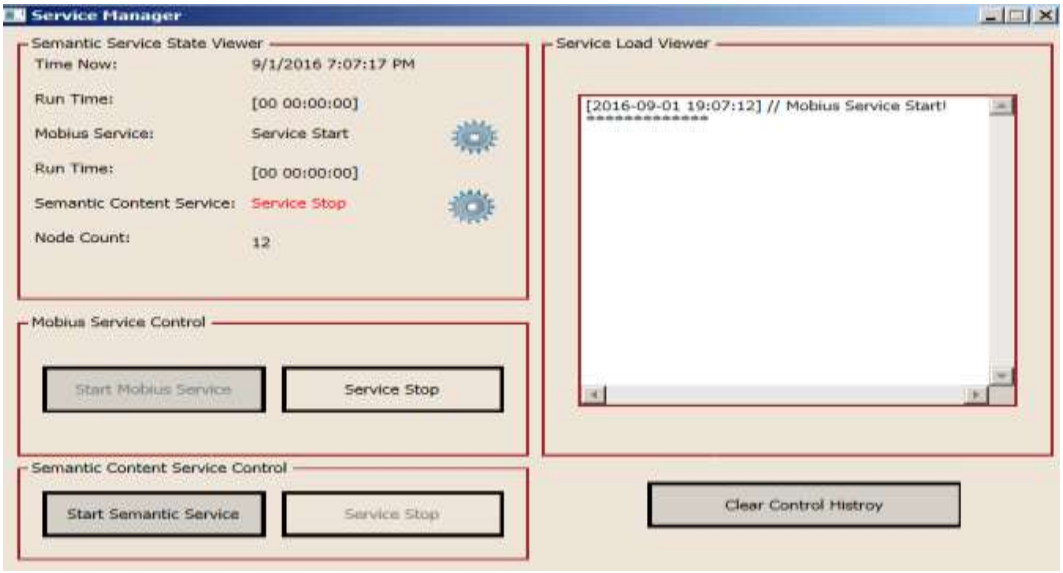

Figure 10. Display of Anchor Service Provider

Anchor service provider has a user interface as shown in Figure 10. This interface is responsible of monitoring and controlling the interworking API to connect Mobius platform. The interworking API is started after user click the start button and the service provider will immediately notifies the result to user. The notification information is displayed on the user interface as shown in the following figure.

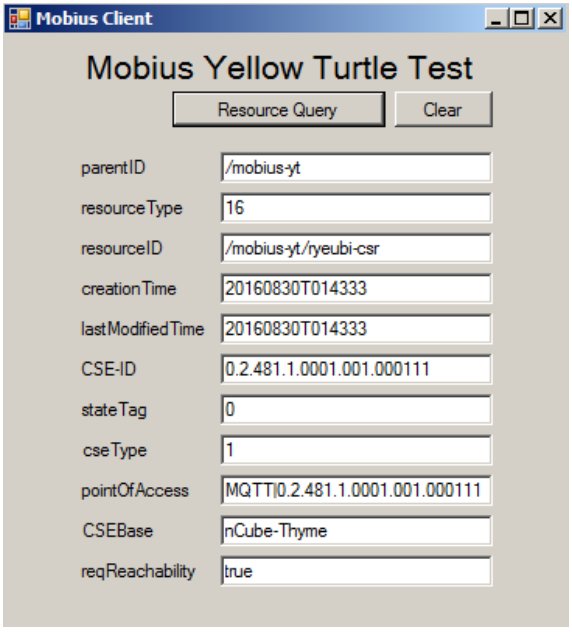

Figure 11. Display of App Client for Mobius Platform

```
C:\Users\Hanglei\Desktop\ocean_iot\Mobius-yt-2.0.5>node app.js
CPU Count: 4
CPU Count: 4
CPU Count: 4
CPU Count: 4
CPU Count: 4
server (117.17.102.52) running at 7579 port
server (117.17.102.52) running at 7579 port
server (117.17.102.52) running at 7579 port
server (117.17.102.52) running at 7579 port
{"rsc":"5000","ri":"/mobius-yt","sts":"ER_DUP_ENTRY"}
{"rsc":"5000","ri":"/mobius-yt","sts":"ER_DUP_ENTRY"}
{"rsc":"5000","ri":"/mobius-yt","sts":"ER_DUP_ENTRY"}
{"rsc":"5000","ri":"/mobius-yt","sts":"ER_DUP_ENTRY"}
resource_retrieve: 7ms
{"rsc":2000,"ri":"GET-/mobius-yt/ryeubi-csr","sts":""}
```

Figure 12. Operating Results in Mobius Platform

In order to test the feasibility of cross domain interworking, we implement a simple HTTP client based simulator shown in Figure 11. After clicking the query button, the client requests the interworking API in the anchor service provider for retrieving resource information and then the anchor service provider access the API receiver in Mobius platform. Figure 12 shows the operation result of the request. The process result of Mobius platform is displayed in the console prompt. Obviously, the request is successfully received and performed by the Mobius platform according to the text message from the console window. The resource retrieve time as well as the Uri path is shown in yellow. The retrieved resource information is then sent back to the service provider and the service provider responds to the client. The client receives the response and then parse the resource information from the response body. Finally, resource information is displayed in the list as shown in Figure 11.

Figure 13. Interworking Round Trip Time Evaluation

We measured the whole interworking system round trip time by the following equation:

$$R_t = \frac{1}{n}\sum_{i=0}^{n} t_n \tag{1}$$

Where Rt the round trip time and {t=1, …, n} is the time slots between the client and the service provider, the service provider and the Mobius Server, the Mobius Server and the service provider, the service provider and the client. The following figure illustrates the performance evaluation result of the propose system. We have recorded the response time

results in 20 seconds at one second intervals. The average costs of the proposed interworking is 10.6 milliseconds. The latency time is such a low that it can be ignored and hardly been observed. The results show that the response time is very good.

6. Conclusion

IoT is evolving around a plethora of vertical platforms, each specifically suited to a given scenario and often adopting proprietary communications, device and resource control protocols. The need for cross-domain IoT applications that can cover multiple aspects of everyday life is becoming more apparent nowadays.

In this paper, we propose an IoT internetworking architecture based on anchor service provider for heterogeneous IoT platforms. This architecture enables the communication with anchor service provider and other IoT platforms. We design and implement the anchor service provider for internetworking to connect with the Mobius platform using a RESTful API based on HTTP protocol. We also implement an app client to test the proposed interworking architecture. And in the experiment we evaluate the performance by computing the system response time. The experiment result shows the data transmission latency time is low and the system performs very well.

In the future, we will keep research on the internetworking between different IoT platforms. A universal framework is eagerly needed to satisfy the requirement of interoperability across IoT platforms for a unified and secure sharing of and access to sensing/actuating resources. In addition, more protocols are also need to be investigated to satisfy compatibility of different IoT platforms.

Acknowledgments

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2017-2016-0-00311) supervised by the IITP(Institute for Information & communications Technology Promotion), and this work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2015-0-00164, Creation of PEP based on automatic protocol behavior analysis and Resource management for hyper connected for IoT Services). Corresponding author: DoHyeun Kim

References

[1] "Internet of things", https://en.wikipedia.org/wiki/Internet_of_things.
[2] Gartner, "Predicts 2015: The Internet of Things", **(2014)**.
[3] GSMA Intelligence, "From concept to delivery: the M2M market today", **(2014)**.
[4] "IoT Platforms", http://www.krnet.or.kr/board/data/dprogram/1856/A1-1.pdf.
[5] "OCEAN", http://www.iotocean.org/main/.
[6] J. Swetina, "Toward a Standardized Common M2M Service Layer Platform: Introduction to oneM2M", IEEE Wireless Commun. Mag., vol. 21, no. 3, **(2014)**, pp. 20-26.
[7] J. Kim, "M2M Service Platforms: Survey Issues and Enabling Technologies", IEEE Commun. Surveys & Tutorials, vol. 16, no. 1, **(2014)**, pp. 61-76.
[8] L. Hang and D.-H. Kim, "A Study of IoT Internetworking between service provider and Mobius platform", Advanced Science and Technology Letters, vol. 141, **(2016)**, pp.178-180.
[9] F. Tila and D. H. Kim, "service provider for Indoor Environment Control – A Sparql and SQL based hybrid model", Advanced Science and Technology Letters, vol.120, **(2015)**, pp.678-683.
[10] C. Nan, Y.J. Lee, F. Tila and D. H. Kim, "Design and Implementation of Middleware Based on ID and IP Address for Actuator Networks", International Journal of Smart Home, vol. 10, no. 1, **(2016)**, pp. 41-48.
[11] Internet of Things Architecture project, [online] Available: http://www.iot-a.eu/public/front-page.
[12] Z. Shelby, K. Hartke and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, **(2014)**.
[13] "oneM2M Protocol Analysis Technical Report", one M2M-TR-0009, vol. 0, **(2013)**.
[14] "Study on Abstraction and Semantics Enablement", one M2M-TR-0007, vol. 0.5.0, **(2013)**.

[15] oneM2M; OIC InterworkingTS-0024-V2.0.0. Available online: http://www.etsi.org/deliver/etsi_ts/118100_118199/118124/02.00.00_60/ts_118124v020000p.pdf (2016-Sept).
[16] oneM2M. TS-0021-oneM2M_and_AllJoyn_Interworking-V2_0_0. Available online: http://www.onem2m.org/images/files/deliverables/Release2/TS-0021 oneM2M_and_AllJoyn_Interworking-V2_0_0.pdf (2016-August-30).
[17] oneM2M. LWM2M Interworking TS-0014-V2.0.0. Available online: http://www.etsi.org/deliver/etsi_ts/118100_118199/118114/02.00.00_60/ts_118114v020000p.pdf (2016-Sept).
[18] oneM2M. 3GPP Release 13 Interworking TR-0024-V2.0.0. Available online: http://www.etsi.org/deliver/etsi_tr/118500_118599/118524/02.00.00_60/tr_118524v020000p.pdf (2016-Sept).

Authors

Lei Hang, he received the B.S. degree in computer engineering in 2015 and the M.S. degree from the Jejun National University, Korea, in 2017. He is currently a Ph.D student at Jeju National University, South Korea. His area of interest is sensor networks, M2M/IOT, intelligent service, and mobile computing.

Do-Hyeun Kim, he received the B.S. degree in electronics engineering from the Kyungpook National University, Korea, in 1988, and the M.S. and Ph.D. degrees in information telecommunication the Kyungpook National University, Korea, in 1990 and 2000, respectively. He joined the Agency of Defense Development (ADD), from Match 1990 to April 1995. Since 2004, he has been with the Jeju National University, Korea, where he is currently a Professor of Department of Computer Engineering. From 2008 to 2009, he has been at the Queensland University of Technology, Australia, as a visiting researcher. His research interests include sensor networks, M2M/IOT, energy optimization and prediction, intelligent service, and mobile computing.

