Industrial Internet of Things Interoperability Between OPC UA and OneM2M

Po-Wen Lai and Fuchun Joseph Lin(✉)

Department of Computer Science, College of Computer Science,
National Chiao Tung University, Hsinchu, Taiwan
{eric4025.cs05g,fjlin}@nctu.edu.tw

Abstract. With the emergence of the Industrial Internet of Things (IIoT), a huge amount of devices in the factory need be connected to the Internet. The use cases being considered include data exchange for inter-factory manufacturing, integrity of data collection and real-time monitoring, etc. It is important to allow seamless communications among IIoT devices of different protocols and standards. In this research, we investigate how to achieve IIoT interoperability via two popular global IIoT standards: oneM2M and OPC-UA by adopting the OPC-UA system in the field domain but connecting it to the oneM2M system in the infrastructure domain. We propose an optimized design of oneM2M Interworking Proxy Entity for resources mapping and procedures mapping between OPC-UA and oneM2M. The design contains both (1) interworking functions and (2) OPC-UA client APP, installed on the oneM2M Middle Node in the field domain, to handle the data exchange.

Keywords: OPC UA · oneM2M · Interoperability

1 Introduction

The Industrial Internet of Things (IIoT) is emerging in recent years with a huge amount of devices in the factory connected to the Internet [1–3]. The use cases being considered include data exchange for inter-factory manufacturing [4–6], integrity of data collection, real time monitoring [7–9], etc. It is important to allow seamless communications among IIoT devices with different protocols and standards.

There has been a large amount of existing research on interoperability between different protocols and standards for the Internet of Things [10–13]. However, many issues still remain about how to achieve interoperability between different IoT systems and protocols. In this research, we investigate how to achieve IIoT interoperability via two popular global IIoT standards: oneM2M and OPC-UA by adopting the OPC-UA system in the field domain but connecting it to the oneM2M system in the infrastructure domain.

The oneM2M standards already have published a technical report [14] to define the interoperability between OPC UA and oneM2M. However, many details are yet to be specified; thus more research is needed to clarify those details. Our key contribution in

B. Li et al. (Eds.): IoTaaS 2019, LNICST 316, pp. 439–455, 2020.
https://doi.org/10.1007/978-3-030-44751-9_37

this research includes (1) proposing a new design of IPE beyond what oneM2M specified to improve the system performance, (2) quantitatively evaluating the improvement of the new design over the oneM2M specified one by measuring the time taken in each interworking step.

The rest of this paper is organized as follows. The OPC UA and oneM2M architectures are first introduced in Sect. 2. Section 3 discusses two design alternatives of IPE: oneM2M specified one and our proposed one. Section 4 then describes the implementation and verification of each. Finally, Sect. 5 discusses our conclusion and future work.

2 Background

We first introduce OPC UA and oneM2M architectures.

2.1 OneM2M Architecture

As shown in Fig. 1, the oneM2M architecture comprises three entities:

1. *Application Entity (AE):* Application Entity is an entity that implements an M2M application service logic. Examples of AEs include remote monitoring application, machinery controlling application and power metering application.
2. *Common Service Entity (CSE):* Common Service Entity is an entity that represents a set of "common service functions" which are defined by oneM2M. Examples of common service functions include device management, group management and communication management.
3. *Network Services Entity (NSE):* Underlying Network Services Entity is an entity that provides the underlying network services to the CSEs.

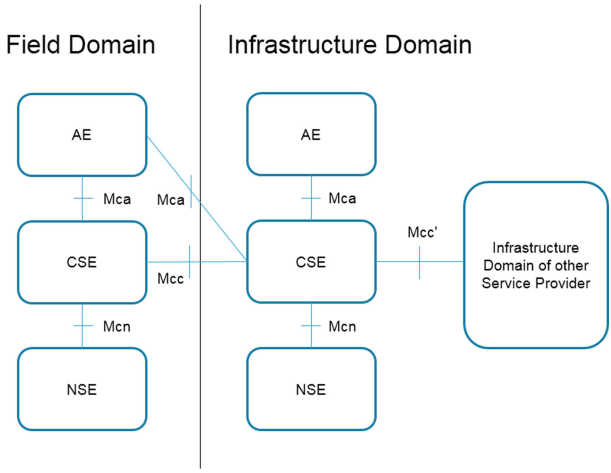

Fig. 1. oneM2M functional architecture

The communication flow between any two entities above is specified in a reference point. oneM2M reference points include the following:

1. *Mca:* specifies the communication flows between AE and CSE. These flows enable the AE to use the services support by the CSE.
2. *Mcc:* specifies the communication flows between two CSEs. These flows enable a CSE to use the services support by another CSE.
3. *Mcn:* specifies the communication flows between CSE and NSE. These flows enable the CSE to use the services support by the NSE.
4. *Mcc′:* specifies the communication flows between two CSEs in different M2M service provider domains in Infrastructure Domain. These flows enable the CSE in Infrastructure Domain to use the services supported by another CSE of a different M2M service provider.

2.2 OPC Unified Architecture

The OPC (Open Platform Communications) Unified Architecture (UA) is a platform independent service-oriented architecture developed by the OPC Foundation that integrates all the functionalities of the individual OPC Classic specifications into one extensible framework [12]. There have been a lot of research focused on the performance of OPC UA [15] and the development of new applications using OPC UA. These new applications include data monitoring [16, 17], industrial automation [18], smart grid [19], etc. [20]. Moreover, interoperability between OPC UA and other standards such as BACnet [21], KNX [22], AutomationML [23], CIM [24], etc. [25, 26] has been extensively studied. This phenomenon shows that OPC UA is a popular standard in the IIoT.

2.3 OPC UA Information Model

OPC UA provides a framework that can be used to represent complex information as Objects in an Address Space which can be accessed with standard web services. Figure 2 shows how OPC UA abstracts real objects in its Address Space. This Address Space is where real objects and their components are abstracted and represented by a set of Nodes and References (the lines between Nodes); the latter are used to relate the nodes in Address Space. A View is a subset of Address Space which is used to restrict the Nodes that the Server makes visible to the Client to simplify data acquisition and control data access in the Server [14]. In summary, Address Space is where OPC-UA Servers keep its collected industrial data.

2.4 OPC UA Resource Mode

OPC UA Clients will communicate with OPC UA Servers to acquire industrial data through Defined Service Sets. All data entities are defined as “object” and located in the OPC UA servers. The OPC UA objects are represented by a set of Nodes that are only accessible through the aforementioned Address Space. Hence whenever OPC UA Clients want to browse the industrial data, they would access the Nodes in the Address Space of an OPC UA Server [14].

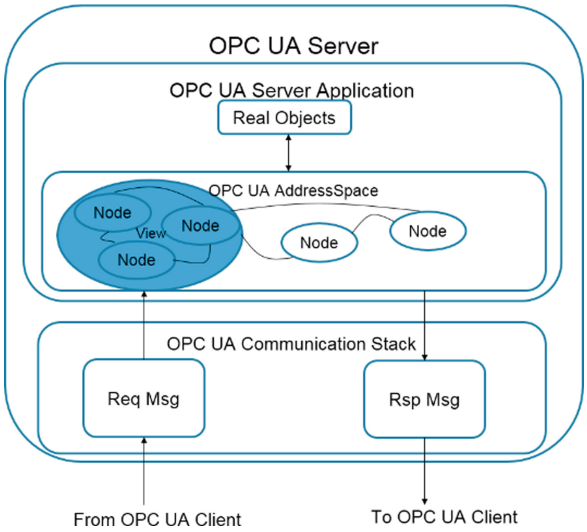

Fig. 2. OPC UA address space

3 Design of OPC UA-oneM2M Interworking IPE

In this section, we discuss the details of resource mapping and procedure mapping between OPC UA and oneM2M.

3.1 Resource Model Mapping

When the data transfer occurs between OPC UA and oneM2M, IPE should create a oneM2M <AE> resource to represent OPC UA <Root> node and map other nodes under <Root> node to the oneM2M <container> resources. An example of mapping the OPC UA resource model to the oneM2M resource tree is depicted in Fig. 3.

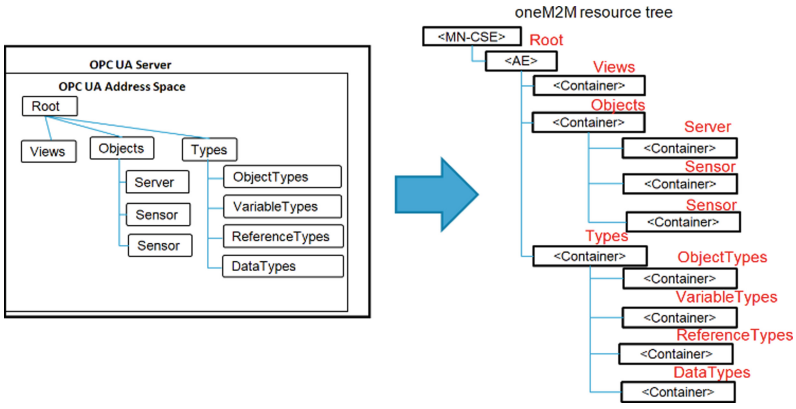

Fig. 3. OPC UA resource model mapping to oneM2M resource tree

In IIoT, there are many sensors deployed in the factory. Thus, many resources will be created in the resource model. Our goal is to design a resource structure with as minimum memory usage as possible. In [14], oneM2M recommends the use of <flexContainer> for mapping the OPC UA data to the oneM2M resource. In our approach, we use <container>/<contentInstance> instead of <flexContainer> to map the nodes under the Root of OPC UA because <container>/<contentInstance> is more suitable for our scenario. For example, if there are 10 nodes under the Root of OPC UA, according to the oneM2M specification we need to create 10 <flexContainer>'s to represent those nodes. In our approach, we use one <container> and 10 <contentInstance>'s instead of 10 <flexContainer>'s. Compared to 10 <flexContainer>'s, our approach of one <container> and 10 <contentInstance>'s uses less memory on IPE.

3.2 Procedure Mapping

The procedures of data handling in OPC UA and oneM2M are different. IPE is designed to assist OPC UA and oneM2M applications in exchanging data or requests. The concept of procedure mapping via oneM2M IPE is proposed in [14]. Nevertheless, we improve oneM2M recommendations with an optimized IPE design. To analyze the performance of both mapping methods: (1) oneM2M Specified Design of IPE and (2) Our Proposed Optimized Design of IPE, we implemented both data exchange procedures between OPC UA and oneM2M. Both methods can process data exchange successfully but exhibit different performance. In this section, we describe the details of each method. We then compare their performance in the next section.

1. *OneM2M Specified Design of IPE:* The design includes four procedures: initialization and discovery, reading, subscription and notification.

   A. *Initialization and Discovery:* To interwork with OPC UA, oneM2M needs to establish a communication session at the beginning, and represent OPC-UA device data as oneM2M resources. To achieve these objectives, the initialization and discovery procedures are proposed in oneM2M recommendations. In the initialization stage, IPE will be installed on the oneM2M MN in the field domain. In the discovery stage, IPE will connect with OPC UA devices through the OPC UA discovery mechanism and follow the mapping rules to map OPC-UA resources to oneM2M resources. As depicted in Fig. 4, the steps in the initialization and discovery procedures are:

      i. Install IPE on MN first.
      ii. After that, IPE can discover OPC UA devices utilizing OPC UA discovery mechanisms.
      iii. Map the resource on OPC UA Server to oneM2M.

   B. *Reading:* For data collection, one way to collect data from OPC UA devices is to send a <Retrieve> request from the originator AE to an OPC UA device. IPE will then translate this oneM2M <Retrieve> request to the OPC UA "Read"

message in order to get data from OPC UA devices. As shown in Fig. 5, the steps in the reading procedure are:

i. The originator AE will first send the Retrieve message to IN-CSE.
ii. IN-CSE forwards the Retrieve message to MN-CSE.
iii. MN-CSE reaches IPE to map the Retrieve message to the OPC UA "Read" message.
iv. Finally, IPE will get the required data by the "Read" message.

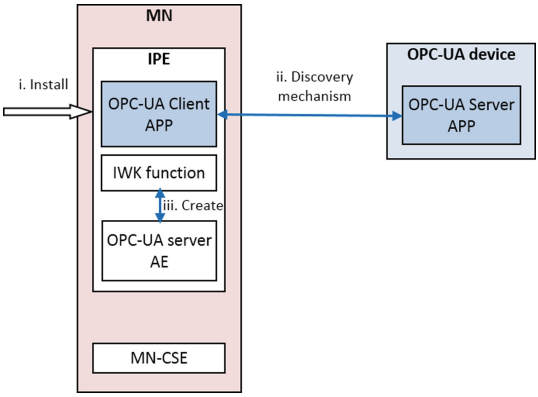

Fig. 4. Initialization and discovery procedure in non-optimized design

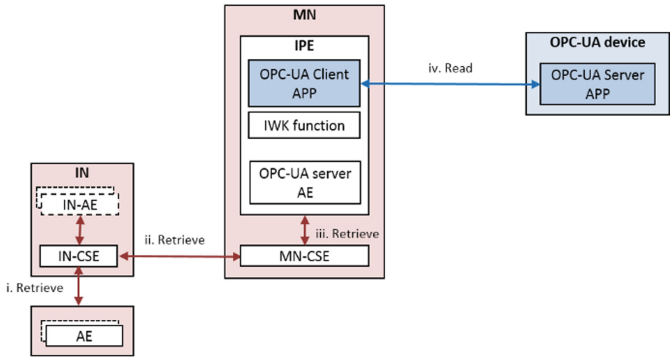

Fig. 5. Reading procedure in non-optimized design

C. *Subscription:* Another way to collect data is to use the oneM2M subscription mechanism to get notification on the change of the corresponding OPC UA resources in the address space. As shown in Fig. 6, the steps in the subscription procedure are:

i. IPE will create a subscribed-to resource for getting notifications about incoming subscription requests.

ii. IN-CSE sends a Create request of <subscription> resource targeting at the resource under an OPC UA server AE in MN-CSE.
iii. MN-CSE will reply a Create response message to the IN-CSE.
iv. MN-CSE sends a Notify request message to IPE.
v. IPE will map the Notify request message to the OPC UA Server.
vi. When OPC UA Server APP receives this message, OPC UA Server APP will send OPC UA Client a Subscription Response message.

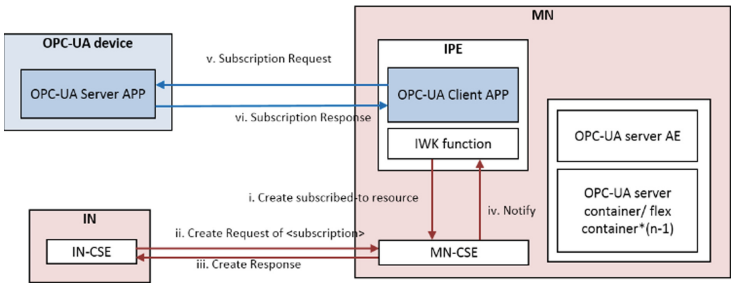

Fig. 6. Subscription procedure in non-optimized design

D. *Notification:* As shown in Fig. 7, the steps in the notification procedure are:

i. Whenever there is an update, OPC UA Server APP will send OPC UA Client APP a notification message.
ii. IPE will map the OPC UA notification message to an Update Request message to MN-CSE.
iii. After receiving the message, MN-CSE will send Notify to IN-CSE.

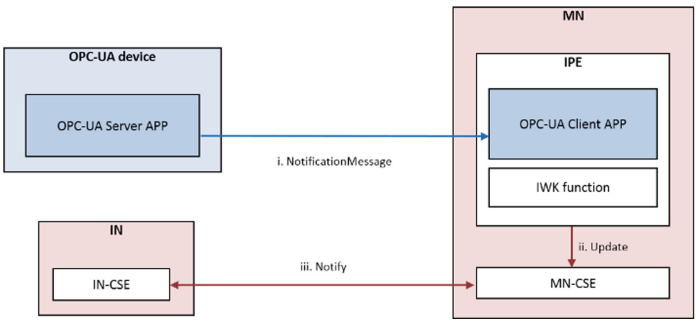

Fig. 7. Notification procedure in non-optimized design

2. *Our Proposed Optimized Design of IPE:* Compared to the oneM2M specified design of IPE, the optimized design of IPE reduces the message exchanges between IPE and the OPC UA device by caching data in the Middle Node to improve system response time. The differences between two designs are described below.

A. *Initialization and Discovery:* In the Initialization and Discovery procedure, besides the steps in the Initialization and Discovery procedure in oneM2M Non-Optimized Design of IPE, there are two more steps need to be executed as shown in Fig. 8:

i. After IPE maps the resource on OPC UA Server to oneM2M, IPE needs to subscribes to the resource on OPC UA Server APP to keep the data up to date.
ii. When the value on OPC UA Server APP is changed, IPE will get the notification and update the value on the Middle Node.

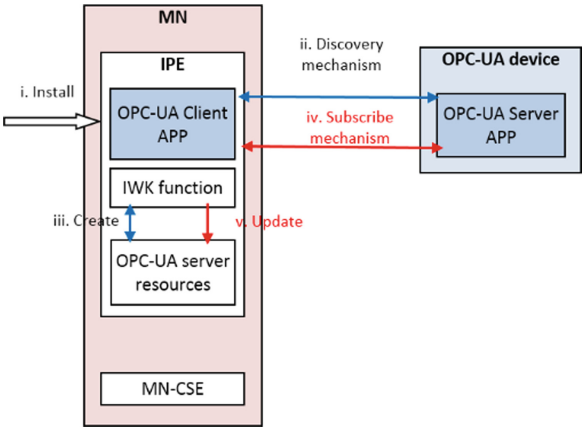

Fig. 8. Initialization and discovery procedure in optimized design

B. *Reading:* In the Reading procedure, the data already cached in the Middle Node and is up to date. IN AE can directly retrieve the data from the Middle Node, IPE does not have to map oneM2M Retrieve message which is Step iv in the Reading procedure of the non-optimized design. As shown in Fig. 9, other steps in the Reading procedure are the same as in the non-optimized design.

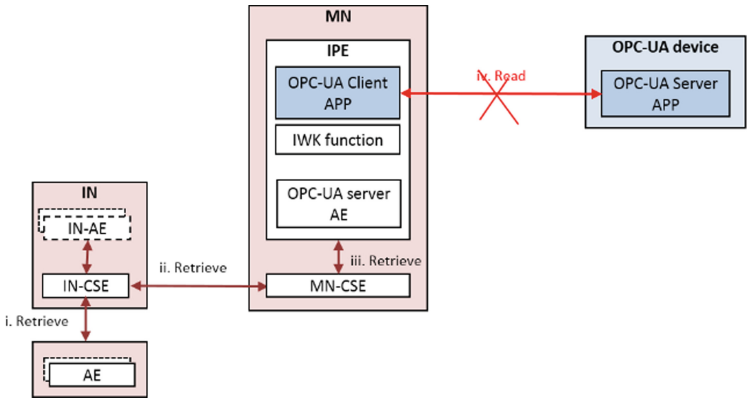

Fig. 9. Reading procedure in optimized design

C. *Subscription:* In the Subscribe procedure, we already do the subscription in the Initialization and Discovery procedure, so IPE does not need to create a subscribed-to resource and receive Notify request in order to map to the OPC-UA subscribe request which are Steps i, iv, v and vi in the Subscribe procedure of the non-optimized design. As shown in Fig. 10, other steps in the Subscribe procedure are the same as in the non-optimized design.

D. *Notification:* In the Notification procedure, we already do the update in the Initialization and Discovery procedure, so IPE does not need to map Notification message to an Update message which are Steps i and ii in the Notification procedure of the non-optimized design. As shown in Fig. 11, other steps in the Notify procedure are the same as in the non-optimized design.

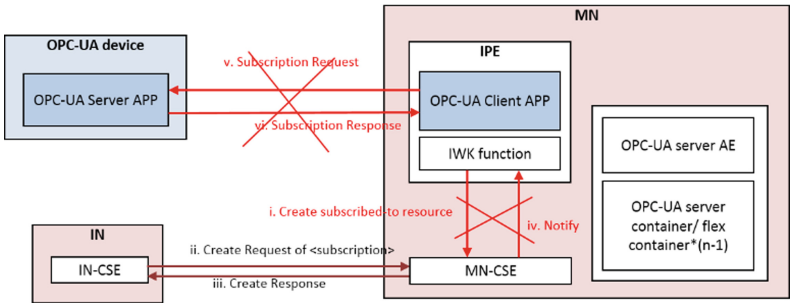

Fig. 10. Subscription procedure in optimized design

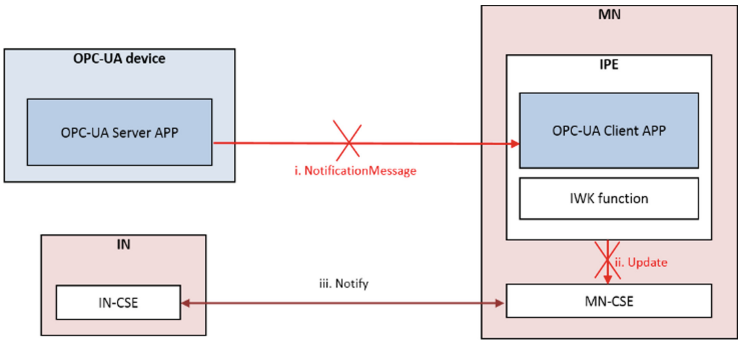

Fig. 11. Notification procedure in optimized design

4 Detailed Design and Implementation

In this section, the system implementation is explained first. Then, we describe the testing environment. Next, we show our experimental results. Finally, the testing result is analyzed.

4.1 System Implementation

Figure 12 depicts the system architecture for handling data exchange based on the scenario addressed by this research. In this system, sensors will collect the data from the environment and send the data to the OPC UA Server residing in a Raspberry Pi. OPC UA Server then sends data to OPC UA Client in the IPE on a laptop computer through the OPC UA interface. The IWK (Interworking) functions in IPE will map OPC UA data into the oneM2M resource tree and send the resources to IN-AE user applications through Mca and Mcc reference points.

In our implementation, OPC UA server and client are based on open62541 [27] which is written in C; on the other hand, oneM2M MN and IN are based on OpenMTC [28] which is written in Python. A Python program is also implemented to get the data

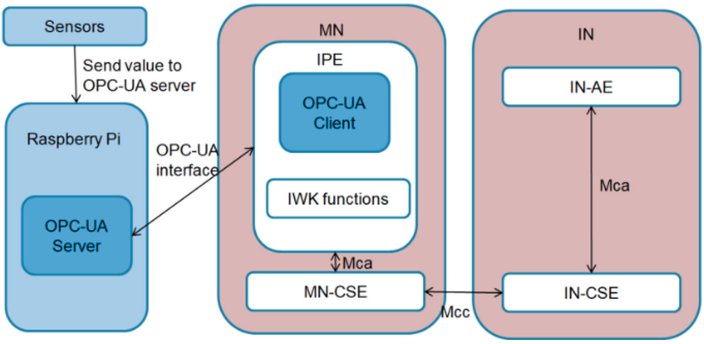

Fig. 12. System architecture design

out of the sensors. Thus we need to deal with data transmission from Python to C and from C to Python: When the sensors get the data, a Python program will store those data into a file, OPC UA Server can get the data by reading the file and manage the resource. After OPC UA Client gets the data from OPC UA Server through OPC UA interface, it will store the data into another file. IWK functions which are written in python will then get the data from this file to do the mapping procedures. After that, the data can be transmitted through Mca and Mcc reference points to the oneM2M application using RESTful communications.

In OPC UA Server, we use the functions which are provided by open62541 to create, update and transmit the data to OPC UA client. In the OPC UA client, we also use the functions provided by open62541 to subscribe the resource and send the request back to the OPC UA Server. In the IWK function, we use RESTful APIs to handle the resource in MN to fulfill the demand of mapping.

4.2 Testing Environment

Figure 13 depicts the testing environment for this research. We used a Raspberry Pi installed with OPC-UA Server APP to represent an OPC UA device that can collect the temperature and humidity data in the factory. The collected data then is sent to OPC-UA Client APP installed with the IPE. With the interworking functions in IPE, OPC-UA data can be converted to oneM2M data, then sent to oneM2M APP by OpenMTC for remote monitoring. For example, if the temperature is detected too high, oneM2M APP can send a request to turn on the fan in the factory.

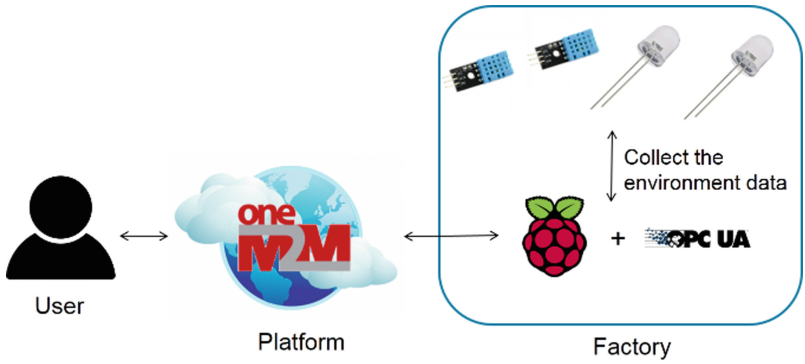

Fig. 13. Use case of system

4.3 Testing Results

We measure the response time of each step described in Sect. 3 to get the overall response time of a procedure under two designs. Each response time is calculated from the average of the measurements collected from 10 executions of the procedure. Besides the response time, we also count the lines of source code to estimate the cost of each system.

1. *Response time:* Table 1 shows the response times of two designs in the Initialization and Discovery procedure. If the time required for a particular step is not included in the overall response time, it will be marked with a diagonal. Table 2 shows the response times of two designs in the Reading procedure. Steps i, ii and iii compose a complete retrieve request from IN to MN. Table 3 shows the response times of two designs in the Subscribe procedure. Steps ii and iii are the request and the response of Subscribe between IN and MN. Steps v and vi are the request and the response of Subscribe between OPC UA server and client. Table 4 shows the response times of two designs in the Notify procedure. Table 5 shows the overall response times of two designs without transmission overhead. The response time of each procedure is the sum of the response times of all its steps. Table 6 shows the overall response times of two designs with transmission overhead. The response time of each procedure is the time consumed from the start to the end of each procedure including the delay of data transmission.

Table 1. Response time of initialization and discovery

	Non-Optimized Design	***Optimized Design***
Step i		
Step ii	622(ms)	Same as left field
Step iii	30(ms)	Same as left field
Step iv		55(ms)
Step v		

Table 2. Response time of reading

	Non-Optimized Design	***Optimized Design***
Step i&ii&iii	2(ms)	Same as left field
Step iv	29(ms)	

Table 3. Response time of subscribe

	Non-Optimized Design	***Optimized Design***
Step i	3(ms)	
Step ii&iii	7(ms)	Same as left field
Step iv	2(ms)	
Step v&vi	55(ms)	

Table 4. Response time of notify

	Non-Optimized Design	***Optimized Design***
Step i	54(ms)	
Step ii	8(ms)	
Step iii	2(ms)	Same as left field

2. *Source Line of Code:* We count the lines of source code in IPE which is composed of OPC UA Client and IWK functions to estimate the cost of our system. Table 7 shows the lines of source code in OPC UA Client and IWK functions in the non-optimized and the optimized systems, respectively, to indicate the complexity of each design.

Table 5. Overall response time (without transmission overhead)

Procedure	Non-optimized design	Optimized design
Initialization and discovery	652 (ms)	707 (ms)
Reading	31 (ms)	2 (ms)
Subscribe	67 (ms)	7 (ms)
Notify	64 (ms)	2 (ms)

Table 6. Overall response time (without transmission overhead)

Procedure	Non-optimized design	Optimized design
Initialization and discovery	674 (ms)	738 (ms)
Reading	33 (ms)	2 (ms)
Subscribe	73 (ms)	9 (ms)
Notify	68 (ms)	3 (ms)

Table 7. Source lines of code

	Non-optimized design	Optimized design
OPC UA client	240 (lines)	215 (lines)
IWK functions	121 (lines)	104 (lines)

4.4 Testing Result Analysis

The results show that all procedures of our proposed method have better response times except the Initialization and Discovery procedure because two new steps need to be added. Comparing the different steps between the optimized design and the non-optimized one, we find that most of message exchanges between IPE and the OPC UA device are removed under the optimized design without adding any new steps. Consequently, the optimized design is able to provide shorter response time.

The results of the overall response time without transmission overhead and the one with transmission overhead are about the same. This leads us to conclude that the transmission overhead does not significantly affect the overall response time. The difference of response times between two designs is primarily caused by the different numbers of message exchanges between IPE and the OPC UA device.

This research focuses on industrial IoT applications such as remote monitoring, controlling application and power metering application. As these applications require real-time processing for all incoming data, it is important to quickly respond. Consequently, our evaluation focuses mostly in the response time with an objective of shortening the response time of the system as much as possible. Therefore, we don't consider throughput, the usage of memory and CPU, or the power consumption of the system in our analysis.

We also count the lines of source code in two designs in order to estimate their complexity of software implementation. The results show that the lines of source code for the optimized design is also less than that of the non-optimized design. The reduction of complexity in the optimized design comes from the removal of some steps from the non-optimized design. Consequently, the optimized design is able to provide better software complexity than the non-optimized design.

In summary, by reducing the number of message exchanges between IPE and the OPC UA device, not only the response time of our proposed design is much faster than that of a non-optimized design, its software complexity is also improved over the non-optimized design.

5 Conclusion and Future Work

We propose an optimized IPE design to improve IIoT interoperability between oneM2M and OPC UA for the scenario where an OPC-UA system is in the field domain and oneM2M is in the infrastructure domain. For resource mapping, a new mapping method for oneM2M and OPC UA is proposed. For procedure mapping, we also propose new procedures by removing some procedural steps and thus reducing the overall response times.

To verify our optimized design, we tested our system based on a use case that simulates the scenario of data collection in the factory. In this environment, there will be many devices with different standards. Consequently, a generic standard such as oneM2M could be very useful. To achieve interoperability, we just need to make every devices communicate with oneM2M.

Our test system employs a Raspberry Pi installed with OPC-UA Server APP to represent a non-oneM2M device in the factory, designed for collecting the temperature

and humidity in the room. This data is sent to OPC-UA Client APP installed in the IPE of a oneM2M Middle Node. The interworking function in the IPE then converts this OPC-UA data to oneM2M data. Finally, this data is sent to oneM2M APP by OpenMTC for remote monitoring.

As shown in the result analysis, comparing to the original oneM2M design, our optimized design improves the performance when using oneM2M in the infrastructure domain and an OPC-UA system in the field domain to collect data.

In conclusion, we propose two mapping methods which are different from the recommendation of oneM2M for resource mapping and procedure mapping, respectively. According to the result analysis, our approach improves the performance of the system where oneM2M is used in the infrastructure domain while OPC-UA is deployed in the field domain.

In our design, because the data need to be kept up to date in MN, this implies that the data need to be constantly retrieved from OPC UA devices; such operations will be costly. Also, the situation of race condition may occur which would lead to out of date data. According to the issues above, our proposed approach is more suitable for the scenario where the cost of constant data update is affordable and the race condition is acceptable.

Though the optimized design is suitable for our scenario, it may not have better performance than the original oneM2M design for other scenarios. Different resource and procedure mapping methods may be required for those other scenarios. Also, we only test our test environment with few devices, the testing scenario with a large amount of OPC UA devices should be considered for future work.

Moreover, more efficient mapping methods for other scenarios can be developed by building a test environment for each scenario. Also, the solutions of cost reduction and race condition avoidance as discussed above can be explored.

Acknowledgement. This study is conducted under the "Open Service Platform of Hybrid Networks and Intelligent Low-carbon Application Technology Project(4/4)" of the Institute for Information Industry which is subsidized by the Ministry of Economic Affairs of the Republic of China.

References

1. Derhamy, H., Eliasson, J., Delsing, J., Priller, P.: A survey of commercial frameworks for the Internet of Things. In: IEEE International Conference on Emerging Technologies and Factory Automation (EFTA) (2015)
2. Gazis, V., et al.: A survey of technologies for the Internet of Things. In: Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC), August 2015, pp. 1090–1095 (2015)
3. Kim, J., Lee, J., Kim, J., Yun, J.: M2M service platforms: survey, issues, and enabling technologies. IEEE Commun. Surv. Tutor. **16**(1), 61–76 (2014)
4. Jung, J., et al.: Design of smart factory web services based on the industrial Internet of Things. In: Proceedings of the 50th Hawaii International Conference on System Sciences (2017)
5. De Brito, M.S., Hoque, S., Steinke, R., Willner, A.: Towards programmable fog nodes in smart factories. In: IEEE International Workshops on Foundations and Applications of Self* Systems, pp. 236–241. IEEE, September 2016

6. Lee, A., Lastra, J.: Data aggregation at field device level for industrial ambient monitoring using web services. In: 9th IEEE International Conference on Industrial Informatics (INDIN), July 2011, pp. 491–496 (2011)
7. Granzer, W., Kastner, W.: Information modeling in heterogeneous building automation systems. In: Proceedings of the 9th IEEE International Workshop on Factory Communication Systems (WFCS), Lemgo, 21–24 May 2012, pp. 291–300 (2012)
8. Schachinger, D., Kastner, W.: Model-driven integration of building automation systems into Web service gateways. In: 2015 IEEE World Conference on Factory Communication Systems (WFCS). IEEE (2015)
9. Komoda, N.: Service oriented architecture (SOA) in industrial systems. In: Proceedings of the IEEE International Conference on Industrial Informatics, August 2006, pp. 1–5 (2006)
10. Derhamy, H., Eliasson, J., Delsing, J.: IoT interoperability – on demand and low latency transparent multi-protocol translator. Trans. Serv. Comput. (2016)
11. Castellani, A.P., Loreto, S., Bui, N., Zorzi, M.: Quickly interoperable Internet of Things using simple transparent gateways. In: Interconnecting Smart Objects with the Internet Workshop, 25 March 2011 (2011)
12. Delsing, J., Rosenqvist, F., Carlsson, O., Colombo, A., Bangemann, T.: Migration of industrial process control systems into service oriented architecture. In: 38th Annual Conference on IEEE Industrial Electronics Society, IECON 2012, Montreal, Canada, 25–28 October 2012, pp. 5786–5792 (2012). https://doi.org/10.1109/iecon.2012.6389039
13. Wu, C.-W., Lin, F.J., Wang, C.-H., Chang, N.: OneM2M-based IoT protocol integration. In: IEEE Conference on Standards for Communications & Networking, Helsinki, Finland, 18–20 September 2017 (2017)
14. oneM2M, TR-0018 V2.5.0: Industrial Domain Enablement (2017)
15. Cavalieri, S., Cutuli, G.: Performance evaluation of OPC UA. In: Emerging Technologies and Factory Automation (ETFA), p. 18 (2010)
16. Verma, N.K., Sharma, T., Maurya, S., Singh, D.J., Salour, A.: Real-time monitoring of machines using open platform communication. In: 2017 IEEE International Conference on Prognostics and Health Management (ICPHM), pp. 124–129. IEEE, June 2017
17. Hästbacka, D., Barna, L., Karaila, M., Liang, Y., Tuominen, P., Kuikka, S.: Device status information service architecture for condition monitoring using OPC UA. In: 2014 IEEE Emerging Technology and Factory Automation (ETFA), pp. 1–7. IEEE, September 2014
18. Palm, F., Grüner, S., Pfrommer, J., Graube, M., Urbas, L.: Open source as enabler for OPC UA in industrial automation. In: 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), Luxembourg, pp. 1–6 (2015)
19. Claassen, A., Rohjans, S., Lehnhoff, S.: Application of the OPC UA for the smart grid. In: 2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe), pp. 1–8. IEEE (2011)
20. Melik-Merkumians, M., Baier, T., Steinegger, M., Lepuschitz, W., Hegny, I., Zoitl, A.: Towards OPC UA as portable SOA middleware between control software and external added value applications. In: 17th IEEE Conference on Emerging Technologies Factory Automation (ETFA), September 2012, pp. 1–8 (2012)
21. Fernbach, A., Granzer, W., Kastner, W.: Interoperability at the management level of building automation systems: a case study for BACnet and OPC UA. In: 2011 IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA), September 2011, pp. 1–8 (2011)
22. Cavalieri, S., Chiacchio, F., Di Savia Puglisi, A.: Integrating KNX and OPC UA information model. J. Comput. **9**(7), 1536–1541 (2014)
23. Henßen, R., Schleipen, M.: Interoperability between OPC UA and AutomationML. Proc. CIRP **25**, 297–304 (2014)
24. Kim, J.S., Park, H.J., Choi, S.H.: CIM and OPC-UA based integrated platform development for ensuring interoperability. KEPCO J. Electr. Power Energy **2**(2), 233–244 (2016)

25. Cavalieri, S., Di Stefano, D., Salafia, M.G., Scroppo, M.S.: Integration of OPC UA into a web-based platform to enhance interoperability. In: 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE), pp. 1206–1211. IEEE, June 2017
26. Gruner, S., Pfrommer, J., Palm, F.: RESTful industrial communication with OPC UA. IEEE Trans. Ind. Informat. **12**(5), 1832–1841 (2016)
27. open62541. https://open62541.org/
28. OpenMTC. https://www.openmtc.org/