*Transdisciplinary Engineering for Complex Socio-technical Systems*
*K. Hiekata et al. (Eds.)*

*doi:10.3233/ATDE190142*

# Ontology-Based Interworking Framework Among Heterogeneous IoT Protocols

Jingyu SUN[1], Shko KATAYAMA, Susumu TAKEUCHI, Seiji OHMORI, Hiroyuki MAEOMICHI, Masato KAMIYA and Ikuo YAMASAKI
*Network Innovation Laboratories, NTT*

**Abstract.** In nowadays' IoT, data have to be exchanged among heterogeneous protocols in many practical scenarios. The differences of the data representations and data structures among these protocols dramatically hinder the interworking, the accumulation as well as the analysis and utilization of IoT data. A famous IoT horizontal platform standardization called oneM2M was established. Besides the conventional specific interworking, oneM2M proposed option interworking method called Ontology Based Interworking for the data exchange between oneM2M and other protocols. This paper analyzes the problems existing in Ontology Based Interworking of oneM2M and proposes the framework to solve it. The heterogeneous IoT data are first abstracted using the Base Ontology introduced by oneM2M. By extending the Base Ontology with new meta-fields, the I/O API interfaces of other IoT protocols are represented in the composed ontology, which facilitates the common design of the data exchange processes among different IoT protocols. For evaluation of the proposed framework, interworking applications are made for some representative IoT protocols/PFs, such as LoRaWAN and Device Web API. The effectiveness of the common data exchange processes proposed in this paper is evaluated. The development cost of these interworking proxy entities is dramatically reduced, and the flexibility of the framework when new devices and protocols try to connect to the oneM2M platform turns to be obviously improved after applying the common interworking flows.

**Keywords.** IoT Protocols, oneM2M, Base Ontology, Ontology Based Interworking

## Introduction

It is said that there will be at least 70 billion IoT (Internet of Things) devices connected to Internet by 2025[1]. These devices complying with over 200 different IoT protocols exist in various domains. To effectively utilize the data from these extraordinary amount of devices, a standardized common IoT Platform which can store these data properly is not the only thing needed, how to easily transform the data from different IoT protocols to the common platform is also important as well as the effective comprehensive, analysis and utilization of the accumulated data. In other words, if we take the transformation cost between different IoT protocols as the Investment and the eventually effective utilization of the data as the Return, then how to maximize the ROI (Return on Investment) turns out to be the main problem of the next generation IoT.

However, considering the variety of the existing IoT protocols and the heterogeneous interfaces that they provide for data exchange, not to mention the different

[1] Corresponding Author, Email: jingyu.sun.pu@hco.ntt.co.jp.

data models in which they manage their data, a considerable software (accessing and converting) development cost would happen when we try to accumulate and utilize the data of them commonly. On the other hand, the applications who eventually utilize the data accumulated by the common platform would prefer a common data structure and common data accessing method from the platform.

An international standardization called oneM2M[2] was established in 2012 for the horizontal integrated IoT platform. oneM2M provides several kinds of interworking method between oneM2M and other IoT protocols, including the Specific Interworking and Ontology Based Interworking. The Ontology Based Interworking utilizes Base Ontology for the device and data representation of other protocols.

This research analyzes the problems existing in oneM2M's existing interworking method and constructs an extended ontology model based on Base Ontology for data representation of various IoT protocols. Data exchange flows for the high-level abstracted data in the ontology are designed in this paper. Library supporting the data exchange between the common platform and other IoT protocols is developed. Based on the developed library, interworking applications between oneM2M and two IoT protocols including DWAPI (Device Web API)[3] and LoRaWAN [4] are developed and evaluated in this paper.

## 1. Interworking in oneM2M

### *1.1. Outline of oneM2M*

oneM2M aims at establishing a standardization of the horizontal IoT platform for accommodating various different IoT protocols by providing over 14 common IoT functions including the data management, access control and so on. As shown in Figure 1(a), oneM2M consists of CSE (Common Services Entity) providing common IoT functions and AE (Application Entity) which communicates with CSE for data communications. A special AE so called IPE (Interworking Proxy Application Entity) which is on the other hand responsible of the data model mapping and data transformation between oneM2M and other IoT protocols.

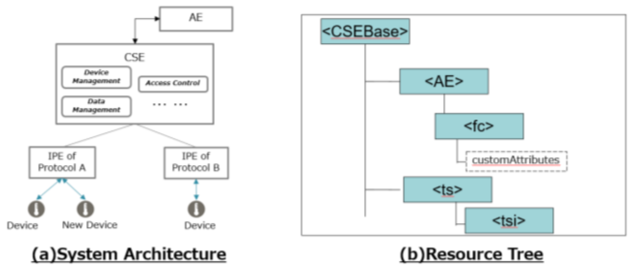

**Figure 1.** Outline of oneM2M.

oneM2M manages the devices and data it accumulated as resources which are stored in a resource tree as shown in Figure 1(b). There are over 60 kinds of different resources defined by oneM2M for various utilizations. For instance, <CSEBase> represents a CSE and is regarded as the root of the resource tree under which different kind of resources can be linked, such as <AE> resource representing an AE and <fc> whose attributes are usually used for storing data obtained from devices. <ts> (time series resource) keeps multiple <tsi>s (time series instance resource) for storing the time series data. Moreover, oneM2M performs subscription and notification processing towards resources (e.g. <fc>) by adding <subscription> resource under them.

In addition, oneM2M provides multiple ways for designing and developing IPE including Specific Interworking and Ontology Based Interworking, which will be illustrated in the following sections.

## *1.2. Specific Interworking in oneM2M*

When data are exchanged between oneM2M and other IoT protocols, without any abstraction of the data representations, a data exchange method provided by oneM2M so called Specific Interworking method designs and develops the exchange flow for every single data element in the data model respectively.

For example, in case of HAIM[5], HAIM defined a concept framework including the data model within the template called SDT (Smart Device Template). oneM2M's Specific Interworking utilizes HAIM (Home Appliance Information Model) for interworking with some of the consumer electronics. oneM2M developed specific interworking standards for every single entity in HAIM's definition. IPE development and test should follow these standards to make sure it is oneM2M compliant. There are also some standards designed for other protocols such as OPC UA and so on. Due to the specific design of the resource mapping with oneM2M, various patterns of data representation and resource mapping in oneM2M platform are expected.

## *1.3. Ontology Based Interworking in oneM2M*

Besides Specific Interworking, oneM2M proposed an option interworking method so called OBI (Ontology Based Interworking), in which Base Ontology[6] is used for the abstraction of the device/data representation in oneM2M. The abstracted information is stored in ontology.

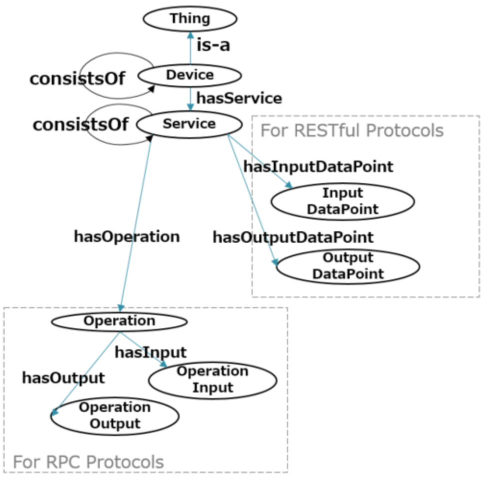

**Figure 2.** Abstraction of device representation using Base Ontology

Figure 2 illustrates Base Ontology. According to oneM2M's concept, the data sending to and receiving from IoT devices are though services/operations. If the protocol is RESTful, then the DataPoints will be put just under the Services. For the RPC interfaced protocols, the Operation will be used to represent the RPC function of the protocol. To adjust to the complicated devices with multi-functions or consisting of multiple sub devices, we can expand the ontology by using the metadata filed "consistsOf" defined by Base Ontology.

Figure 3 gives a simple sample of how to mapping the devices' functions to Base Ontology. The device, "light" is mapped to the Device, under which the "SwitchOnService" is provided as a Service. The switch's status can be changed by updating the InputDataPoint "BinaryInput" under the "SwitchOnService". On the other hand, OutputDataPoint "status" is used to obtain the status of the light.

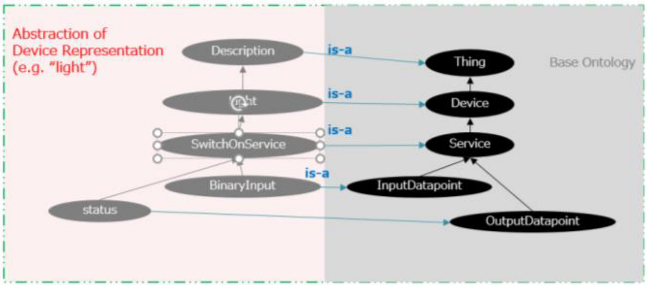

**Figure 3.** Device mapping example

After the data of other protocols are abstracted into Base Ontology, oneM2M uses the ontology to do the common data mapping to oneM2M resources: manage Device in <AE> (application resources), Service in <fc> (Flex Container) resources, DataPoint as the <fc>'s custom attribute.

## *1.4. Issues of Interworking in oneM2M*

oneM2M accumulates and utilizes the data from other protocols through IPEs. Under different interworking situations especially for the large-scale implementation, the development cost of the IPEs when try to connect with new protocols and new devices needs to be reduced into an acceptable level. Moreover, oneM2M has no common data exchange flows designed for supporting multiple data exchange patterns such as on-demand data accumulation or time series accumulation, which means interworking flows need to be developed respectively. In addition, a common data structure and data accessing method need to be provided to the AEs for the effective data utilization.

Unlike Specific Interworking, naive OBI can promise a common data representation in oneM2M's resource tree and a common way for AEs to access the data. However, issues still exist in large-scale development even applying naive OBI.

(a) Since oneM2M has no common processing flows about how to communicate with other protocols through their specific APIs, different Services of other protocols still need to be triggered respectively, and values of different DataPoints existing in the response still need to be obtained respectively. The IPE's development cost for the data accumulation and service/operation triggering stays high.

(b) For different typical data exchange patterns (e.g. on-demand data retrieving and newest value retrieving), interworking flows cannot be reused among different protocols also causes the extra development cost.

The remaining of this paper proposed a framework called "CommonOBI" solving the issues mentioned above.

## 2. CommonOBI

By extending oneM2M's Base Ontology, CommonOBI framework proposed in this paper abstracts the device representation including other protocols' I/O API information into high-level abstraction entities such like "devices", "services", etc. All the data exchange flows are designed for the abstracted entities instead of every single concrete service or data element. This framework aims at the reduction of the IPE development cost and adapting for various typical data exchange patterns in IoT.

### *2.1. Metadata Field Design for Interworking*

In order to reduce the IPE development cost when connecting to new devices or new protocols, the details of data exchange API on Service/DataPoint levels are composed into the ontology too. As shown in Figure 4, metadata field "Protocol:hasAPI" is used to record the APIs which is used to trigger the service/function of other protocols, at the same time identify the OutputDataPoints from other protocol's responses and the InputDataPoints as the parameters in the requests to other protocols. Besides, to help the platform and applications understand the accumulated data, metadata field "Protocol:hasUnit" is designed to record the data's unit.

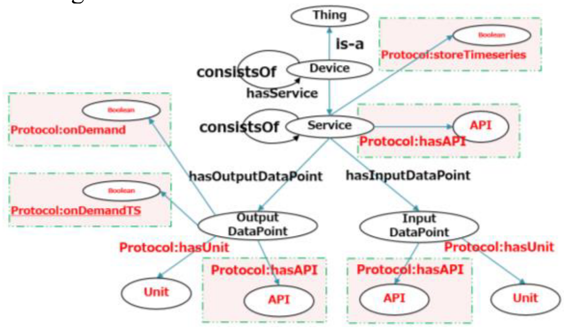

**Figure 4.** Meta data field design for interworking

On the other hand, typical data exchange patterns of IoT platform can be categorized into the following:

(1): obtains the newest values and stores them
(2): obtains the newest value and stores the history time series
(3): only get the value on the applications' demand
(4): get the time series on the applications' demand

Metadata field "Protocol:storeTimeseries" with the Boolean value is designed towards Services to indicate if the history time series data of that service need to be stored in the platform or not. Metadata field "Protocol:onDemand" and "Protocol:onDemandTS" with Boolean value is designed towards OutputDataPoint to indicate if the value of that OutputDataPoint need to be obtained on demand of AE instead of polling the newest value all the time.

### *2.2. Utilization of Extended Ontology for Interworking*

Figure 5 illustrates the architecture of the proposed CommonOBI. IPE reads the details of Device/Service/DataPoint representation from ontology, and performs common interworking processing commonly. Protocol dependent unit is designed to deal with the

API's processing for each protocol and can be used commonly by different devices complying with that protocol.

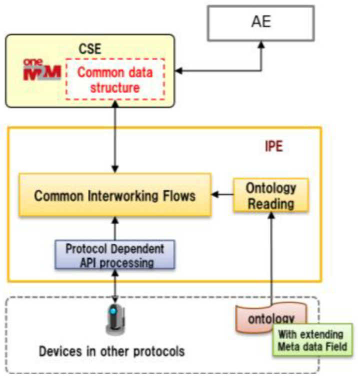
AE
CSE
oneM2M
Common data structure
IPE
Common Interworking Flows
Ontology Reading
Protocol Dependent API processing
ontology
With extending Meta data Field
Devices in other protocols

**Figure 5.** System architecture used by CommonOBI

Firstly, "Protocol Dependent API Processing" unit is developed for the service/function triggering of other protocols and the interfaces can be broken down into:

i) Data obtaining from other protocols (in this case, DWAPI and LoRaWAN)
  - ✓ Input: protocol API identify strings and OutputDataPoint strings
  - ✓ Output: the values of the specified DataPoints

ii) Data/command sending to other protocols
  - ✓ Input: protocol API identify strings and InputDataPoint strings
  - ✓ Output: response of the command execution from other protocols

API identify strings and DataPoint strings mentioned above are read out from the ontology in real time by the "Ontology Reading" unit.

Then we illustrate the common interworking flow with the example as shown in Figure 6 in which on the left is the oneM2M's resource tree, on the right is the ontology used by IPE. The sub devices and sub services are extracted from the ontology and recursively processed following the same processing workflow. Except for the steps we specially point out, "Common Interworking Flows" unit in Figure 5 performs the interworking flows below.

*(A) Common Processing For Devices*

i) "Protocol Dependent API Processing Unit" obtains the device list using the device discovery API provided by other protocols (e.g. DWAPI). For the protocols, which own no device discovery API, such as LoraWAN, extract the device information the data firstly obtained from the protocol.

ii) Judge if each device in the device list is supported by the ontology or not using the following SPARQL statement.

*{?Device RDFS:subClassof BO:Device}*

iii) Create <AE> (Application Entity) resource representing the device in oneM2M for the devices supported by the extended ontology.

*(B) Common Processing For Services*

i) Extract services (e.g. "getData" and "ManageDev") of Device from the ontology according to the following SPARQL statement.

*{?Device BO:hasService ?Service}*

ii) Create <FC> (FlexContainer) resource representing every service.

iii) Extract APIs, such as "DWAPI:Sensor/Service1", for triggering the function/service of DWAPI with the "DWAPI:hasAPI" metadata field.

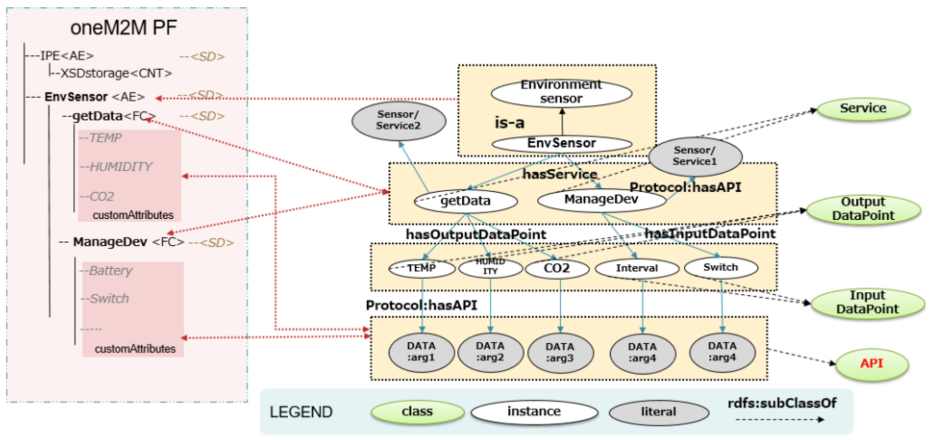

**Figure 6.** Data mapping between oneM2M and extended ontology

(C) *Common Processing For OutputDataPoints*

i) Extract OutputDataPoints ("humidity", etc.) of each service from the ontology according to the following SPARQL statement.

*{?Service BO:hasOutputDataPoint ?OutputDataPoint}*

ii) Create thread for the service to observe the status of the DataPoints if there are any OutputDataPoints under a service.

iii) Extract APIs of the DataPoint, such as "DWAPI:DATA:arg1" by looking up the "DWAPI:hasAPI" metadata filed in the ontology.

iv) Identify the DataPoint's value from the response using APIs every time "Protocol Dependent API Processing Unit" receiving the response from device.

v) Update oneM2M resource representing the service with these identified values of the OutputDataPoints.

vi) Create <tsi> for current value if the OutputDataPoint's "storeTimeseries" field is true.

vii) Waits until the data obtaining request "demanding" from oneM2M platform to obtain the value or the time series data for that OutputDataPoint initiatively if the OutputDataPoint's "onDemand" or "onDemandTS" field is true.

(D) *Common Processing For InputDataPoints*

i) Extract InputDataPoints ("switch", etc.) of each service from the ontology according to the following SPARQL statement.

{?Service BO:hasInputDataPoint ?InputDataPoint}

ii) Create oneM2M <Subscription> resource under the service's resource to make sure that the IPE will be announced when the value of each InputDataPoint is updated if there are any InputDataPoints under a service.

iii) Trigger the Service of other protocols using the service's APIs extracted from ontology when receiving the InputDataPoint's updating notification from CSE.

## 3. Case Study

For the evaluation of CommonOBI, IPEs connecting oneM2M and two of the major IoT protocols (DWAPI and LoRaWAN) were made. In case of Common OBI, the Common Interworking Flows (protocol independent) mentioned in Figure 5 were developed as a java library. The protocol dependent parts of IPEs were developed for the interworking. Moreover, the IPEs based on Specific Interworking were also made for comparison. In this case study, all the IPEs were realized for accommodating the devices illustrated in Table 1.

**Table 1.** Case study settings

| | DWAPI | LoRaWAN |
|---|---|---|
| Kind of Devices | 5 | 5 |
| Services/Device | 3 | 2 |
| DataPoints/Service | 3 | 2 |

The case study evaluates the proposed framework from the following aspects:

(1) Simplification of processes for interworking
(2) Reduction of IPE development cost
(3) Cost of ontology composition

## *3.1. Simplification of Processes for Interworking*

How CommonOBI simplified the processes for interworking between oneM2M and other protocols are analyzed according to the IPE development in this case study.
Figure 7(a) illustrates the processes for connecting devices through the proposed framework, CommonOBI. Only for new protocols, the protocol dependent part (common API processing between IPE and devices) needs to be developed. For the already supported protocols, we only need to add the device representations to the ontology for connecting new types of devices.

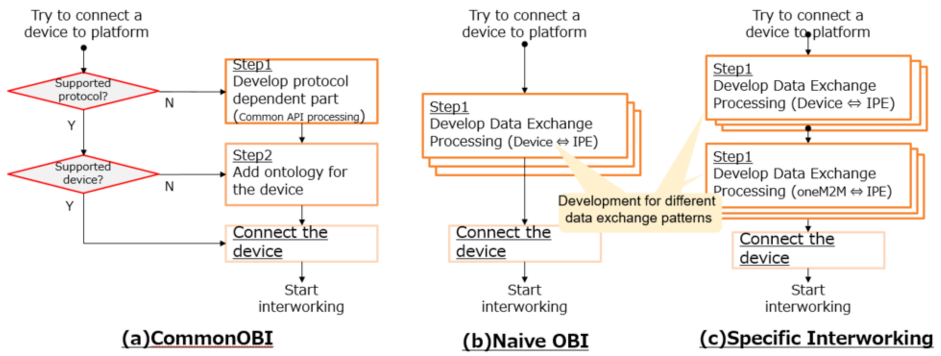
Try to connect a device to platform
Supported protocol?
N
Y
Step1
Develop protocol dependent part
(Common API processing)
Supported device?
N
Y
Step2
Add ontology for the device
Connect the device
Start interworking
(a)CommonOBI
Try to connect a device to platform
Step1
Develop Data Exchange Processing (Device ⇔ IPE)
Connect the device
Start interworking
(b)Naive OBI
Development for different data exchange patterns
Try to connect a device to platform
Step1
Develop Data Exchange Processing (Device ⇔ IPE)
Step1
Develop Data Exchange Processing (oneM2M ⇔ IPE)
Connect the device
Start interworking
(c)Specific Interworking

**Figure 7.** Procedures for connecting devices to oneM2M platform

In contrast, as shown in Figure 7(b) and (c) with the naive OBI or Specific Interworking proposed by oneM2M, development was necessary for the data exchange between IPE and device of other protocols when trying to connect to a new type of device. Furthermore, for Specific Interworking, design and development for the data transformation and exchange between IPE and oneM2M platform were conducted as shown in Figure 7(c). In addition, these processing flows need to be designed and developed for every data exchange patterns respectively.

Therefore, the processes of interworking between oneM2M and other protocols were simplified by applying CommonOBI encouraging the reduction of the development cost of IPEs.

## *3.2. Reduction of IPE Development Cost*

As introduced in Section 2.2, protocol dependent API processing in CommonOBI are mainly responsible for the service/function triggering of other protocols. Table 2 compares the protocol dependent source codes' lines count of CommonOBI and the Specific Interworking for five devices.

**Table 2.** Protocol dependent source codes (lines count) for connecting five different devices

| | DWAPI | LoRaWAN |
|---|---|---|
| CommonOBI | 98 | 56 |
| Specific Interworking | 700 | 200 |

The differences are illustrated as shown in Figure 8: 1) For Specific Interworking, the source codes of protocol dependent part increases with the connected devices. However, CommonOBI keeps a low development cost for the protocol dependent part for dealing with different amount of devices. 2) Even for connecting same amount of devices, since CommonOBI utilizes the common processing to deal with different services and DataPoints, it has much less source code development comparing with Specific Interworking.

Naive OBI proposed by oneM2M standardizes the data exchange between IPE and devices, development of it does not increase as sharply as Specific Interworking, however still increases linearly along with the count of connecting devices. Moreover, since naïve OBI does not have common processing flows for data exchange between device and IPE, development cost of it stays same as Specific Interworking when connecting only one kind of device.

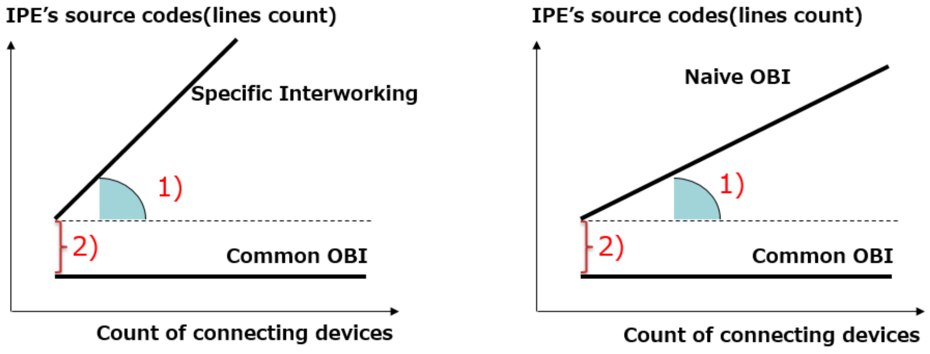

**Figure 8.** Comparison of specific interworking, naive OBI and CommonOBI

## *3.3. Evaluation of Ontology Composition Cost*

Different from the Specific Interworking, ontologies need to be composed manually for the interworking when trying to connect with new protocols or new types of devices. However, the result of this case study shows that for who is familiar with the ontology composition language and the knowledge of the protocol to be connected, it only cost about 1minute/1device to compose ontology for typical IoT device (with no more than five services).

## 4. Discussion

By introducing CommonOBI, the interworking process was proved to be dramatically simplified. However, from the view point of network transmission cost, an issue occurs when implement the CSE and IPE into different hosts. As shown in Figure 9(a), <Protocol:storeTimeseries> extention forces IPE to send multiple requests with same values obtained from devices to CSE for (1) updating <fc>'s customAttribute and (2) create new <tsi>. Solutions for this issue are illustrated in Figure 9(b) as: (2) Let CSE be responsible for duplicating the data from <fc> to <tsi>; or (2') make an AE for this work. Therefore, only one time of data transmission between IPE and CSE will be needed.

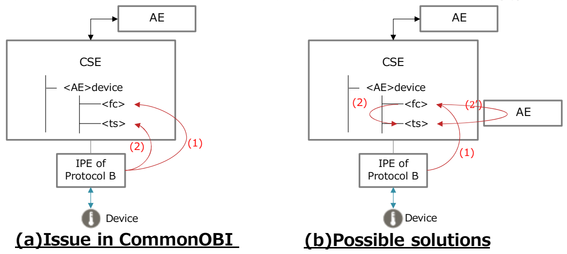

**Figure 9.** Issue in CommonOBI and solution

Meanwhile, for the devices with complicated or special structures, IoT-Lite[7] or IoT-O[8], which also provide an ontology with high-level abstracted representation of different data models aiming at the common data representation among different data models, are considered as options of Base Ontology in this research. The common interworking flows can be redesigned adapting for these ontologies too.

## 5. Conclusion

This research proposed the efficient data exchange framework by utilizing ontology. The API's information and data exchange patterns are composed into the ontology by adding the corresponding meta data fields to the base ontology provided by oneM2M. The device representation of other protocols can therefore be structured into the high-level abstracted ontology, and the common data exchange workflow is designed.

By applying the proposed framework to the interworking between oneM2M and two other protocols, the dramatic reduction of the IPE development is confirmed. In future work, explanation and utilization of the accumulated data will also be explored.

## References

[1] S. Lucero, IoT platforms: enabling the Internet of Things, https://cdn.ihs.com/www/pdf/enabling-IOT.pdf
[2] oneM2M, http://www.onem2m.org/
[3] DWAPI, https://device-webapi.org/link.html
[4] LoRaWAN, https://lora-alliance.org/resource-hub/lorawantm-specification-v102
[5] HAIM, http://www.ttc.or.jp/jp/document_list/pdf/j/TS/TS-M2M-0023v2.0.2.pdf
[6] Base Ontology, http://www.onem2m.org/technical/onem2m-ontologies
[7] M. Bermudez-Edo et al., IoT-Lite: A Lightweight Semantic Model for the Internet of Things, 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, 2016.
[8] N. Seydoux et al., IoT-O, a Core-Domain IoT Ontology to Represent Connected Devices Networks, EKAW 2016: Knowledge Engineering and Knowledge Management pp 561-576.