Multilayer Internet-of-Things Middleware Based on Knowledge Graph

Cheng Xie, *Member, IEEE*, Beibei Yu, Zuoying Zeng, Yun Yang, *Member, IEEE*, and Qing Liu

***Abstract*—Internet of Things (IoT) provides ubiquitous intelligence and pervasive interconnections to diverse physical objects. A key technology to seamlessly integrate different IoT devices into an IoT system is IoT middleware, a software system layer designed to be the intermediary between IoT devices and applications. However, two issues, *communication gap* and *heterogeneous access*, prevent the existing IoT middleware from effective application in an IoT system with heterogeneous standards and interfaces. To address this problem, inspired by a graph-based knowledge system for eliminating heterogeneity in business systems, we, in this article, propose a knowledge graph-based multilayer IoT middleware. The proposed multilayer IoT middleware introduces a new layer to bridge the gap between IoT devices with different communication protocols. It is able to uniformly manage all IoT devices by using an IoT knowledge graph. We evaluate the applicability of the proposed approach by a real-life IoT project, a remote monitoring project of rural sewage treatment stations located in Yunnan Province, China. We find that the proposed approach effectively resolves the communication gap and heterogeneous access problems that occurred in the system.**

***Index Terms*—Internet of Things (IoT), IoT application, IoT middleware, knowledge graph, ontology, Web service.**

I. Introduction

THE Internet of Things (IoT) is emerging as an information revolution in the new Internet era [1], [2]. Numerous devices connected to the Internet substantially influence various aspects of our lives and work [3]. Sophisticated IoT systems implement fruitful functionalities by combining and integrating IoT devices into a large-scale system.

However, IoT devices that differ in terms of their functions are typically manufactured by different companies; thus, they may use completely different access standards, thereby demonstrating prominent heterogeneity in terms of the access method. Therefore, determining how IoT devices that differ in terms of access methods effectively communicate with each other in a complicated system has become one of the main challenges faced by IoT applications [4], [5]. To handle this challenge, IoT middleware was proposed in the previous research [Fig. 1(a)] for bridging the gap between heterogeneous IoT devices and other systems or applications [4], [6]–[9].

Although the IoT middleware methodology is feasible in theory, several problems are encountered in practical applications.

1) *Communication Gap:* Some IoT devices are nonsmart and have limited computing and communication capabilities (e.g., temperature sensors and humidity sensors). They only support field communication protocols, such as RS485 and Modbus, and are unable to support IoT communication protocols, such as CoAP, MQTT, and ZigBee. Thus, these IoT devices cannot directly access IoT middleware.
2) *Heterogeneous Access:* Machine-to-machine (M2M) communication plays a key role in the IoT middleware framework. However, there are many M2M standards (e.g., oneM2M [10], LWM2M [11], FIWARE/NGSI-LD [12], etc.), and they are not all compatible with one another. Moreover, IoT devices may differ in terms of the M2M standards to which they conform, thereby leading to the heterogeneous access problem. In some cases, a subset of the devices in the system may even use a customized standard instead of an existing communication standard, which further exacerbates the heterogeneous access problem.

Therefore, to successfully apply the IoT middleware approach in an IoT system with complex communication networks, we must find an effective solution to the communication gap problem and the heterogeneous access problem.

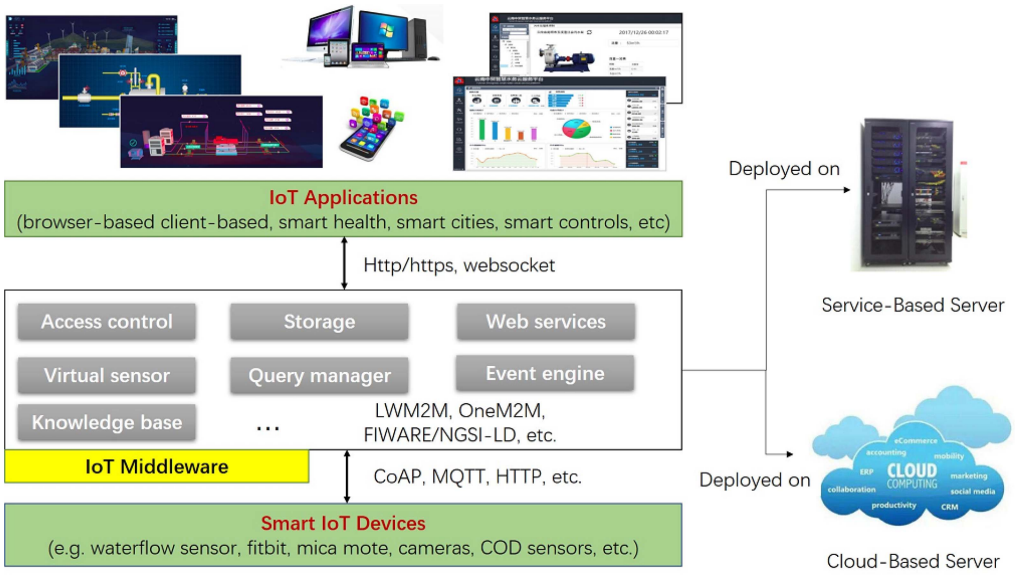
In recent years, many companies have begun to use knowledge graph-based business systems to replace their traditional APIs-based systems. A prominent example is GraphQL,[1] which has been used by GitHub-v4,[2] Facebook,[3] YELP,[4] and AWS.[5] Compared with APIs-based systems, knowledge graph-based systems can provide more flexible access methods and effectively reduce the heterogeneity of the system by building a uniform graph. Inspired by GraphQL, this article envisages whether the IoT devices accessed by APIs can be transformed into the ones accessed by a knowledge graph so that the above-mentioned problems in the IoT systems can be resolved.

Manuscript received June 15, 2020; revised July 26, 2020; accepted August 19, 2020. Date of publication August 26, 2020; date of current version February 4, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61876166 and Grant 61663046; in part by the Youth Talent Promotion Project of China Association for Science and Technology under Grant W8193209; and in part by the Department of Science and Technology of Yunnan Province under Grant 202002AB080001-5 and Grant K264202002220. *(Corresponding author: Yun Yang.)*

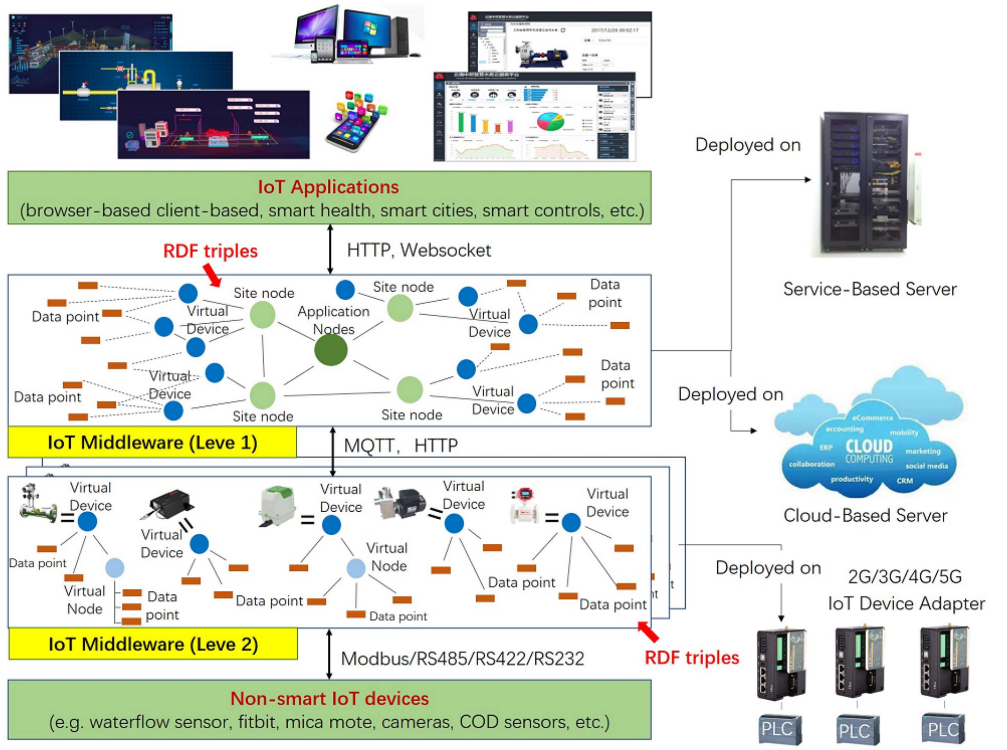
The authors are with the School of Software, Yunnan University, Kunming 650504, China (e-mail: yangyan19@hotmail.com).

Digital Object Identifier 10.1109/JIOT.2020.3019707

[1]https://www.apollographql.com
[2]https://developer.github.com/v4/
[3]https://developers.facebook.com/docs/graph-api
[4]https://www.yelp.com/developers/graphql/guides/intro
[5]https://docs.aws.amazon.com/appsync/index.html#lang/en_us



(a)



(b)

Fig. 1. (a) Classic IoT middleware for industry application with smart IoT devices. (b) Knowledge graph-based IoT middleware for industry application with nonsmart IoT devices.

In this article, we propose a novel multilayer IoT middleware method based on a knowledge graph for IoT device management in a complex IoT system. The new method is illustrated in Fig. 1(b). The major contributions of this new method are twofold.

1) Similar to GraphQL, we propose a novel method based on knowledge graph theory for uniform management of the communications of all devices. The proposed method maps an IoT device to a subgraph in a knowledge graph. As a result, the operation of the device no longer calls the APIs of the IoT devices, except to query or modify the corresponding edges and nodes in the mapped graph. Then, communication and interoperation between IoT devices can be realized via attribute manipulation in a knowledge graph. Therefore, the heterogeneous access problem discussed above is effectively resolved.
2) We introduce a new layer, namely, level-2 (L2 for short), of IoT middleware. The new layer serves as a bridge

between the IoT devices that only support field communication protocols (e.g., RS485 and ModBus), and the existing IoT middleware that deploys IoT protocols (e.g., MQTT, HTTP, and other IoT protocols). L2 IoT middleware effectively bridges the communication gap.

We evaluate the applicability of the proposed multilayer IoT middleware by considering a real-life IoT project: a remote monitoring project of rural sewage treatment stations in Yunnan Province, China; additional details are provided in Section IV. We found that the proposed approach effectively resolves the communication gap and heterogeneous access problems in the system. We have successfully implemented the proposed approach in 15 sewage treatment stations around the province. This success paves the way for the future application of our approach in other IoT systems with the same problems.

The remainder of this article is organized as follows. Section II discusses the related work. Section III describes the models and methods proposed in this work in detail. Then, Section IV discusses the application of the method to the real project. Finally, Section V presents the conclusions and a discussion of future work.

II. Related Works

A. IoT Middleware

Although IoT has high potential and offers numerous exciting opportunities, effectively managing "things" to realize a seamless integration of the physical world and the cyber world remains challenging [13]–[15]. One of the difficulties in the IoT industry is related to machine-to-machine (M2M) communication [9]. An enabling technology for integrating IoT devices in a system is middleware, which is a software system layer designed to serve as an intermediary between IoT devices and applications [4]. An IoT middleware is a software that provides interoperability among incompatible devices and applications [16], [17]. Nowadays, there are many IoT middlewares, such as FIWARE-IoT-Agent [18], FIWARE-ORION [19] Linksmart [20], OpenIoT [21], and Devicehive [22]. These IoT middlewares require MQTT, REST, HTTP, or other high-level communication protocols. They are suitable for smart IoT devices, which have adequate computation and communication capacity. But they would not work on nonsmart IoT devices, which do not support the high-level communication protocols. Some other IoT middleware, such as GSN [23] and Xively,[6] are all based on Web service (or are API based). They use service composition and discovery to handle the heterogeneity of IoT devices, which requires substantial manual and programming efforts. Moreover, these middlewares use WAN communication protocols (e.g., HTTP, MQTT, ZigBee, and CoAP) that cannot be used on many nonsmart IoT devices that only support field communication protocols (e.g., RS 232, RS423, RS485, and ModBus); this is the communication gap issue discussed in Section I. In this article, in comparison with the existing IoT middleware frameworks, we propose a multilayer IoT middleware based on a knowledge graph for bridging the communication gap and resolving heterogeneous access issues.

[6]http://xively.com

B. Knowledge Graph

Knowledge graphs are widely used in question answering [24], [25], natural language processing [26], hospital information system [27], association prediction [28], visual computing [29], cloud manufacturing [30], and other artificial intelligence applications and have yielded outstanding results. In recent years, many companies have begun to use knowledge graph-based systems for their business-related systems as well.

1) Structure of Knowledge Graph: Popular knowledge graphs, such as DBpedia [31], YAGO [32], NELL [33], and KnowledgeVault [34], are composed of a base ontology and multiple domain-specific ontologies and instances. The base ontology of a knowledge graph is used to make uniform the "terms" and "relations" in the "top level." The domain-specific ontology of a knowledge graph is used to specify the "terms," "relations," and restrictions in a specified domain. The instances are the data of the knowledge graph.

2) Representation of Knowledge Graph: Knowledge graphs are directed graphs composed of entities as nodes and relations as edges. They store information in the form of a triplet (head, relation, and tail), where head and tail are entities and relation represents the relationship from the head to the tail [35], [36]. RDF N-triples is one of the most popular implementations of knowledge graph triplets. RDF N-triples is a line-based, plaintext format for encoding an RDF graph. It uses "subject" and "object" to represent the head and tail nodes, which are linked by "predicate." An RDF triple in a knowledge graph is expressed as <Subject> <Predicate> <Object>. Today's largest knowledge graph, namely, Google KG,[7] contains 18 billion RDF triples [37]. Widely used knowledge graphs DBpedia and Wikidata each contain more than 200 million RDF triples [37]. In this article, the proposed IoT knowledge graph uses RDF N-triples as its base format.

3) Graph-Based Middleware: Graph-based data are applied to many domains, such as social networks, street maps, etc. Therefore, graph-based middleware has received considerable attention in recent years. For example, GraPS [38] can handle graph-based publications and subscriptions by several middleware servers. It demonstrates that middleware is effective in processing graph-based knowledge through three case studies: gaming, traffic control, and social networks. Inspired by the graph-based system, we uniformly manage the communications of all devices based on the knowledge graph. The proposed method effectively resolves the heterogeneous access problem instead of IoT devices accessed by APIs.

III. Proposal

A. IoT Knowledge Graph

According to the basic definition of a knowledge graph, an IoT knowledge graph consists of three parts: 1) an IoT base ontology; 2) an IoT domain ontology; and 3) an IoT instance.

[7]https://www.google.com/intl/bn/insidesearch/features/search/knowledge.html

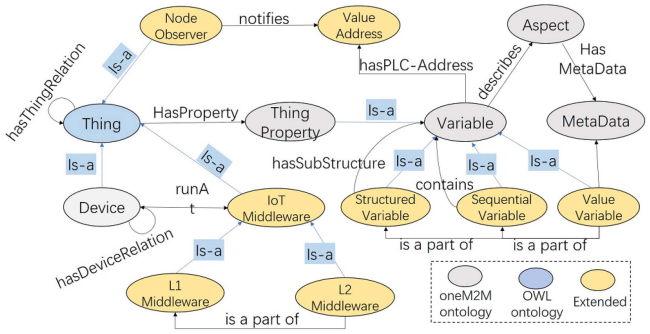


Fig. 2. Base ontology of the top-level knowledge graph of the framework.

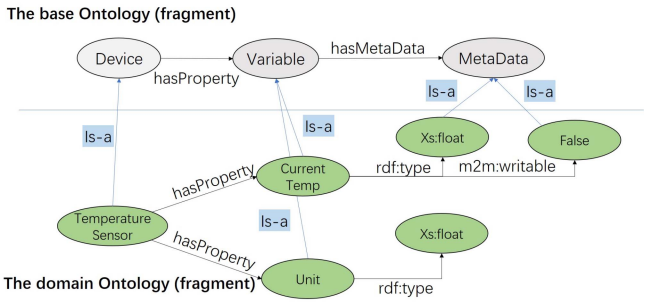


Fig. 3. Example of an oneM2M domain ontology for describing a thermometer.

1) IoT Base Ontology: The IoT base ontology can be understood as a specification or standard for unifying the naming of relationships and attributes of IoT devices. Based on the oneM2M base ontology (Y.4500.12(18)_F01),[8] we define the basic structure of an IoT device in the knowledge graph from a high level, as shown in Fig. 2.

In the IoT base ontology, a device (class: Device) is a thing (a subclass of class Thing) designed to accomplish a particular task via the functions the device performs. A variable (class: Variable) constitutes a superclass to the following classes: ValueVariable, StructuredVariable, and SequentialVariable. Its members are entities that have data (integers, text, or structured data) that can change over time. These data of the variable typically describe real-world aspects (e.g., temperature) and can include metadata (e.g., units and precision). A device can run on IoT middleware, which can be further divided into L1 and L2 IoT middleware. For each variable of a device, there is a ValueAddress that specifies the corresponding physical address of the variable. A NodeObserver is a trigger for change. When the value of the registered variable changes, NodeObserver will notify the ValueAddress to synchronously update.

2) IoT-Domain Ontology: An IoT base ontology defines the basic structure and relationships of IoT devices in an abstract domain. In contrast, an IoT domain ontology is designed to unify attributes, names, data points, and interfaces of IoT devices in a specific domain. In this work, the oneM2M domain ontology[9] is applied with the IoT base ontology. The oneM2M domain ontology intends to provide a unification means in the oneM2M system by defining an information model for the specific domain devices, such as thermometers, flowmeters, water sensors, smoke sensors, refrigerators, and air conditioners. Fig. 3 presents an example of a oneM2M domain ontology for describing a thermometer.

In Fig. 3, the classes colored in gray (e.g., device and variable) are the abstract concepts from the IoT base ontology. The classes that are colored in green (e.g., thermometer and currentTemp) are the concepts from the IoT-domain ontology. The IoT-domain ontology not only specifies the attributes of a device but also specifies the metadata of these attributes.

[8]http://www.onem2m.org/technical/published-drafts
[9]http://www.onem2m.org/component/rsfiles

B. IoT Device to Knowledge Graph Mapping

Thus, far, we have constructed the IoT base ontology and the IoT-domain ontology. This section describes how to map an IoT device to an IoT instance graph such that a complete IoT knowledge graph is constructed.

1) Device URI Generation: IoT device mapping to an IoT instance graph requires the assignment of a globally unique identifier to the device. The common practice is to use the device's URI as the unique identifier. The standard definition of the URI is as follows:

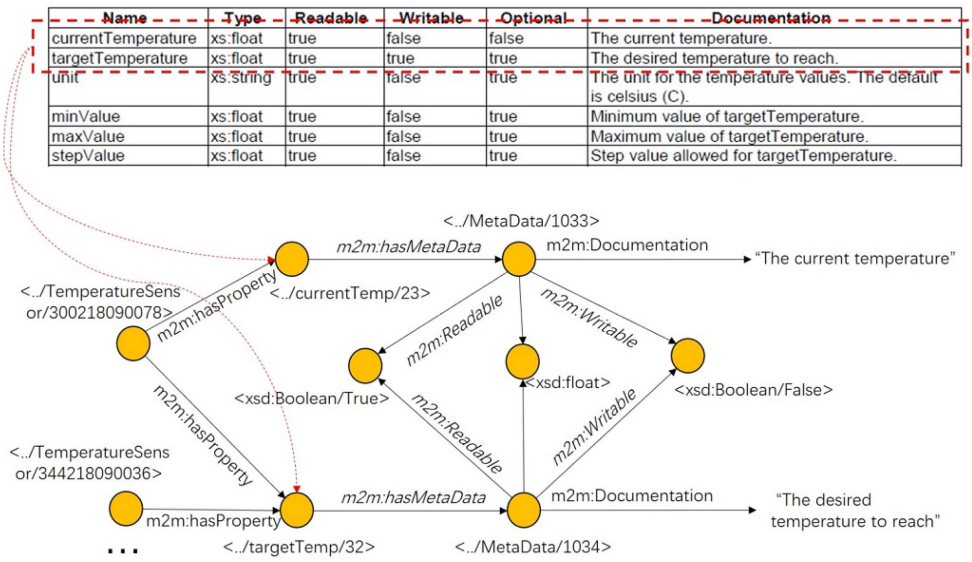
$$\text{URI} = [\text{namesapce}]/[\text{localpath}]/[\text{localname}].$$

In this work, we use the network-domain name of the deployed application as the namespace (e.g., https://www.zm-iotplatform. com), the class name of the device in the IoT-domain ontology as the localpath (e.g., thermometer), and the serial number of the device as the localname (e.g., 300218090078). Thus, the IoT URI is defined as follows:

$$\text{URI} = [\text{ApplicationDomainName}]/[\text{BaseOntology}] /[\text{DomainOntology}]/[\text{DeviceID}].$$

First, a device is mapped to a device class in the IoT base ontology. Then, the name of the device is mapped to the standard name in the IoT domain ontology, which, along with the device class of the base ontology, serves as the localpath of the URI. The serial number of the device is placed at the end of the URI as the localname. We can directly access the URI to obtain the root node of the device in the IoT knowledge graph. For instance, by querying <https://www.zm-iot-platform.com/Device/thermometer/300218090078>, we can obtain the root node of the thermometer in the IoT knowledge graph.

Once the root node of the device has been obtained, the data contained in the device must be further manipulated. This requires the data and the attributes in the device to be mapped to the IoT knowledge graph. To map the attributes and data of a device to an IoT knowledge graph, the core strategy is to regard each attribute or value not as a locally existing value but as a globally accessible reference (a node in the IoT knowledge graph). For instance, the temperature collected by a thermometer is considered a node in the IoT knowledge graph, not a value. Multiple thermometers could share the same temperature node if the values of their temperatures are the same. Thus, each attribute of a device should



Name	Type	Readable	Writable	Optional	Documentation
currentTemperature	xs:float	true	false	false	The current temperature.
targetTemperature	xs:float	true	true	true	The desired temperature to reach.
unit	xs:string	true	false	true	The unit for the temperature values. The default is celsius (C).
minValue	xs:float	true	false	true	Minimum value of targetTemperature.
maxValue	xs:float	true	false	true	Maximum value of targetTemperature.
stepValue	xs:float	true	false	true	Step value allowed for targetTemperature.

Fig. 4. Example of value variable mapping for a temperature value.

also be associated with a URI. According to the attribute structure of the device, attribute mapping can be divided into three categories: 1) value variable mapping; 3) structured variable mapping; and 3) sequential variable mapping.

2) Value Variable: A value variable refers to the primitive data types defined in W3C XSD 1.1,[10] such as integers, floating-point numbers, time, and characters. The mapping rules for this type are as follows:

$$\text{URI} = [\text{ApplicationDomainName}]/\text{ValueVariable} /[\text{DomainOntology}]/[\text{Value}].$$

According to the mapping rules, the attributes of the device are mapped to a ValueVariable in BaseOntology. Subsequently, the name of each attribute is mapped to the standard attribute name in DomainOntology, which, together with ValueVariable, acts as the localpath of the URI. The value of the attribute is placed at the end of the URI as the localname. For example, a value 23° collected by a thermometer is mapped to a node of the IoT knowledge graph as <https://www.zmiot-platform.com/ValueVariable/CurrentTemp/23°>. Here, "/ValueVariable" specifies that this node is a value node. "/CurrentTemp" is a standard attribute name in the IoT-domain ontology. The data type and the constraints of the value node are represented by the "MetaData" node in the IoT base ontology. To constrain the value node, we only need to create an RDF triple that links the value node to the "MetaData" node: <../CurrentTemp/23°> <hasMetaData> <../MetaData>. Fig. 4 presents an example of mapping a temperature value to an IoT knowledge graph.

In the above example, the two temperature sensors (300218090078 and 344218090036) share"targetTem" on the same value node, namely, "<./targetTemp/32°>." In the knowledge graph, the attribute values of IoT devices are represented as nodes, and only one node exists for the same attribute value, regardless of how many devices use the value. When the value of a device changes, instead of modifying the value of the node, we modify the relationship between the device and the node to connect it with another value node (if the value node already exists in the knowledge graph) or create a new value node (if the value node does not exist in the knowledge graph). In addition, a value node that loses connection with other nodes will be reclaimed by the system after a period of time.

3) Structured Variable: A structured variable is a combination of one or more sets of data that cannot be mapped directly to a primitive data type. A structured variable is a tree structure with nonleaf nodes that represent a substructure and leaf nodes that store data. In the IoT base ontology, a structured variable is defined as a subclass of the variable. A structured variable uses "hasSubStructure" to link with substructures. In the knowledge graph, it is expressed as follows: <StructuredVariable> <hasSubStructure> <Variable>. For leaf nodes, we can directly map using ValueVariable mapping rules. For nonleaf nodes, we define the following URI mapping rules:

$$\text{URI} = [\text{ApplicationDomainName}]/\text{StructuredVariable} /[\text{DomainOntology}]/[\text{AutoID}].$$

According to the above rules, the structured attribute of a device is first mapped to the StructuredVariable type in the IoT base ontology. The nodes of a structured variable have custom IDs that do not store data. A structured variable is associated with substructure nodes through "hasSubstructure" relationships. Substructure nodes can be either a structured variable or a value variable. Structured variables are common in the oneM2M IoT standard. For example, a variable that represents the battery state of a temperature sensor is defined as a structure with three attributes, and its mapping is illustrated in Fig. 5.

By definition, a structured variable can contain both substructures and value nodes, and the depth of the substructures is not limited. However, based on practical experience, the

[10]https://www.w3.org/TR/xmlschema11-1/

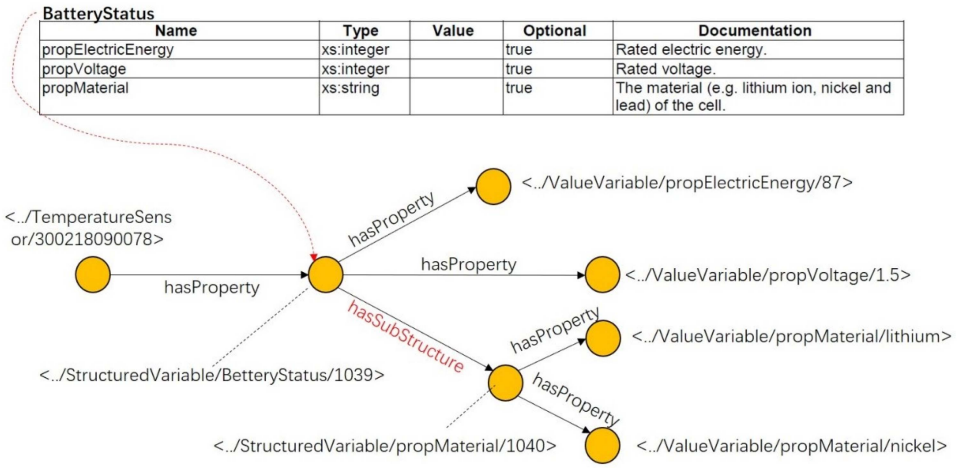


Fig. 5. Example of structured variable mapping for BatteryStatus of a temperature sensor.

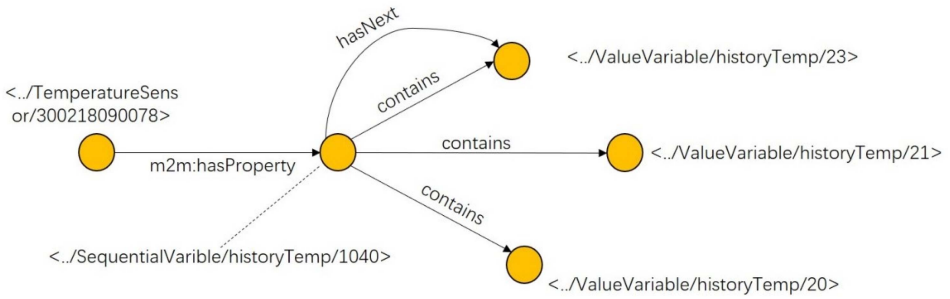


Fig. 6. Example of sequential variable mapping for historyTemp of a temperature sensor.

depth of complex structured variables typically does not exceed three. In addition, "MetaData" in the IoT base ontology does not restrict structured variables. MetaData only restrict value variables.

4) Sequential Variable: A sequential variable is a set of variables. It can be ordered or disordered. A sequential variable cannot be directly mapped to a value variable. In the IoT base ontology, a sequential variable is defined as a subclass of the variable that contains multiple variables. These variables can be value variables, structured variables, or sequential variables. Sequential variable uses a "contains" relationship to link with the variable it contains and orderly obtains the next variable through a "hasNext" relationship. The URI mapping rules between a sequential variable and the knowledge graph are as follows:

$$\text{URI} = [\text{ApplicationDomainName}]/\text{SequentialVariable} /[\text{DomainOntology}]/[\text{AutoID}].$$

According to the above rules, the sequential attribute of a device is mapped to the sequential variable in the IoT base ontology. A node of the sequential variable has a custom ID that does not store data. The sequential variable associates with its inner nodes by using a "contain" relationship and traverses them by using a "hasNext" relationship. Fig. 6 presents an example of mapping the historical temperature data collected by a temperature sensor to the sequential variable node in the IoT knowledge graph.

Each set of relationships in the above knowledge graph is expressed as an RDF triple. When the RDF triple is generated, an additional tuple, namely, <Date>, is created for recording the creation time of the relationship. In the example of historyTemp, for each historical temperature data item collected, the collection date (the date when the relationship was created) is recorded by the fourth tuple: <SequantialVariable/historyTemp/1040> <contains> <ValueVariable/historyTemp/20° > <02-01-2019 20:32:18>. The fourth tuple <02-01-2019 20:32:18> specifies that historyTemp 20° was collected on 02-01-2019.

C. IoT Knowledge Graph Service

Through the mapping rules, a device with its attributes can be transformed into an IoT knowledge graph, and the operation of the device is also transformed into the operation of the IoT knowledge graph. For RDF-triple-based knowledge graph services, there are many supported frameworks. Apache Jena SPARQL is a relatively mature knowledge graph service framework. It has been widely applied in many knowledge graph- based applications. In this work, Apache Jena SPARQL is deployed in L1 IoT middleware for the IoT knowledge graph service. Fig. 7 illustrates the structures of the L1 and L2 IoT middlewares.

In Fig. 7, the SPARQL endpoint is a universal service access terminal, which provides standard SPARQL services. Through the SPARQL endpoint, the nodes and edges in the

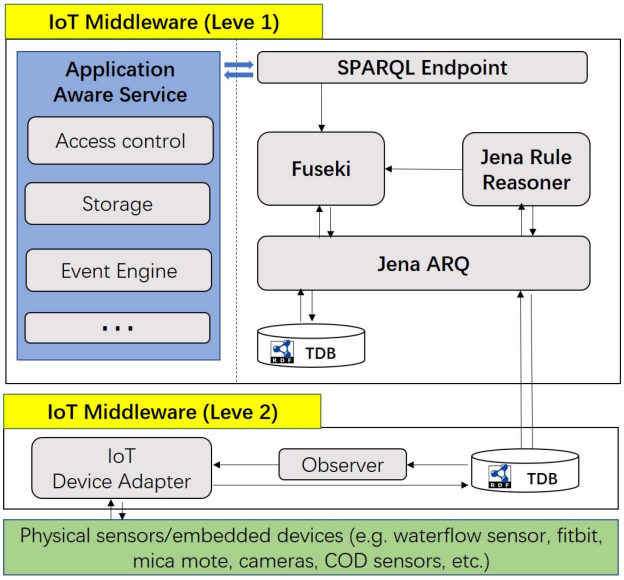


Fig. 7. Framework of IoT knowledge graph service.

IoT knowledge graph can be queried, created, modified, and deleted. Fuseki is the execution engine of SPARQL services, which provides the parsing and execution of SPARQL statements. Jena Rule Reasoner is an ontology rule reasoner engine, which can infer the SPARQL execution according to the rules defined in the IoT base ontology and in the domain ontology. For example, when querying the variables of a temperature sensor, the sequential variable, structured variable, and value variable are all queried. This is because in the definition of the IoT base ontology, the sequential variable, structured variable, and value variable are all subclasses of the variable, and the Jena Rule Reasoner will automatically identify these potential relationships. TDB is the database in which the knowledge graph is stored. It optimizes the storage structure of the RDF triple and improves the efficiency of the SPARQL service in the knowledge graph query. The L1 IoT middleware only stores the root node and the static attributes of a device while the dynamic data of the device are stored in the L2 IoT middleware. The knowledge graph in the L2 IoT middleware can be registered in an observer, and any modification of the graph will trigger the observer. Then, the observer will pass the changed data to the IoT device adapter to update the physical device accordingly. At the same time, the IoT device adapter will generate RDF triples according to the mapping rules and will store them in TDB in the L2 IoT middleware. In Fig. 7, application-aware service can apply application frameworks according to business requirements. An application-aware service is used to generate business-related control logic, roles, UI, and other business functions. The application-aware service does not directly access the IoT devices or knowledge graph. It operates the knowledge graph through the SPARQL endpoint and indirectly operates the IoT devices.

IV. Case Study

A. Project Description

We evaluate the applicability of the proposed multilayer IoT middleware by implementing it in a real-life IoT project.

TABLE I
SENSOR AS KNOWLEDGE GRAPH

	currentCOD	Name	COD sensor
	historyCOD	Type	HNT-B
	status	Protocol	ModBus/RTU
	...	Brand	Chemins

IoT BaseOntology (RDF triple)

```
<CODSensor> <rdf:type> <m2m:Device>
<CODSensor> <zm:runAt> <middleware/l2/1001>
<CODSensor> <zm:runAt> <middleware/l1/01 >
...
```

IoT DomainOntology (RDF triple)

```
<CODSensor> <CurrentCOD> <owl:cardinality/1>
<CODSensor> <HistoryCOD> <owl:cardinality/1>
<CODSensor> <Status> <owl:cardinality/1>
<currentCOD> <rdf:type> <ValueVariable>
<historyCOD> <rdf:type> <SequentialVariable>
...
```

IoT Instance Graph (RDF triple)

```
<CODSensor/[id]> <hasProperty> <CurrentCOD/[value]>
<CODSensor/[id]]> <hasProperty> <SequantialVariable/historyCOD/[id]>
<SequantialVariable/historyCOD/[id]> <contains> <historyCOD/[value]>
<SequantialVariable/historyCOD/[id]> <contains> <historyCOD/[value]>
<SequantialVariable/historyCOD/[id]> <contains> <historyCOD/[value]>
...
<CODSensor> <hasProperty> <status/[value]>
```

The main objective of this project is to provide an automatic sewage treatment system for rural areas in Yunnan province, China. All sensors and controllers must connect to an IoT environment such that an administration center can monitor the quality of the water to make timely decisions in an emergent situation and remotely control all actuators (devices).

In this sewage treatment system, the *communication gap* and *heterogeneous access* issues discussed in Section I are encountered. The sewage treatment system consists of two groups of blowers, a group of hoist pumps, a group of self-priming pumps, a group of backwashing pumps, two groups of backflow pumps, two solenoid valves, two COD sensors, a flow sensor, a pH sensor, a temperature sensor, and a turbidity sensor. Fig. 8 depicts the system architecture of the sewage treatment station considered in this case study. However, these pumps and sensors do not have sufficient computing and communication capabilities and cannot be directly connected with the existing IoT middleware. In addition, some devices have incompatible network access methods. Therefore, we cannot directly use the existing IoT middleware technique to uniformly manage all devices in this system.

In contrast to the existing IoT middleware technique, the proposed method can effectively resolve these two issues. We implement the multilayer IoT middleware in this sewage treatment system to provide uniform management for all devices in this system. We implement the L2 IoT middleware, which uses an IoT adapter to communicate with small IoT devices through the RS232, RS422, RS485, and other low-level protocols. The L2 IoT middleware communicates with the L1 IoT middleware in a WAN through the MQTT and HTTP protocols, which provides a feasible access path for small IoT devices. Then, we use the knowledge graph mapping rule proposed in Section III to map devices to nodes in our IoT knowledge graph, and as a result, all communication between the devices and the system can be conducted via the manipulation of nodes in the knowledge graph. Next, we describe in detail the proposed multilayer IoT middleware in this sewage treatment project.

In this case, the devices that must be mapped into a knowledge graph can be divided into three categories: 1) sensors; 2) pumps; and 3) solenoid valves. Tables I–III, respectively,

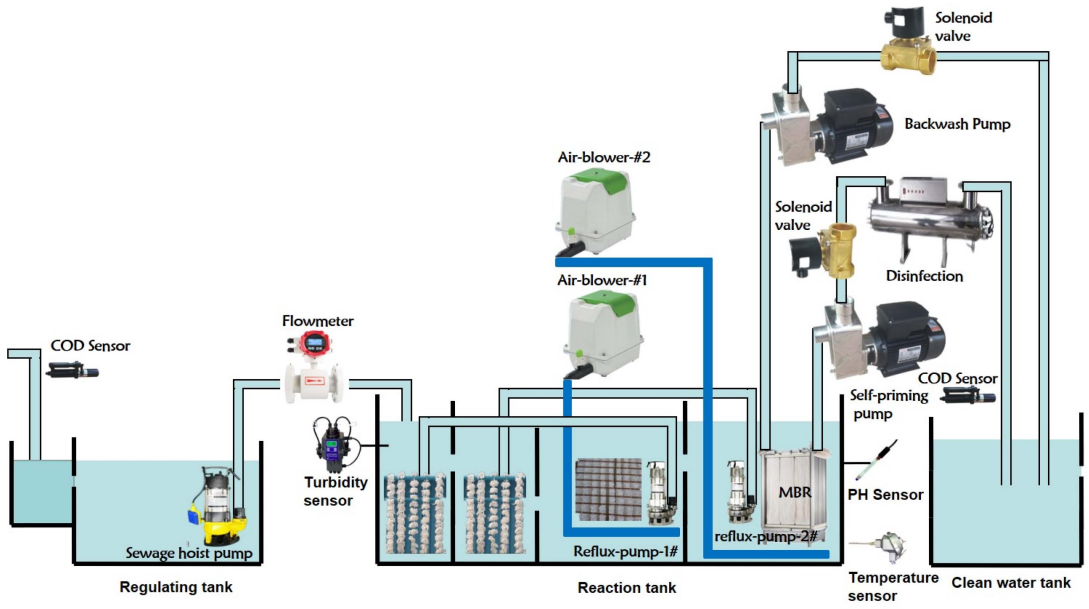


Fig. 8. Nonsmart IoT devices configuration of a sewage treatment station of the project.

TABLE II
PUMP AS KNOWLEDGE GRAPH

	runMode	Name	Air Pump
	alarmStatus	Type	JDK 80
	faultStatus	Protocol	ModBus/RTU
	...	Brand	SECOH
IoT BaseOntology (RDF triple)			
<AirPump> <rdf:type> <m2m:Device> <AirPump> <zm:runAt> <middleware/l2/1001> <AirPump> <zm:runAt> <middleware/l1/01 > ...			
IoT DomainOntology (RDF triple)			
<AirPump> <runMode> <owl:cardinality/1> <AirPump> <alarmStatus> <owl:cardinality/1> <AirPump> <faultStatus> <owl:cardinality/1> <runMode> <rdf:type> <ValueVariable> <alarmStatus> <rdf:type> <ValueVariable> ...			
IoT Instance Graph (RDF triple)			
<AirPump/[id]> <hasProperty> <runMode/[value]> <AirPump/[id]]> <hasProperty> <alarmStatus/[value]> <AirPump/[id]> <hasProperty> <faultStatus/[value]> ...			

TABLE III
VALVE AS KNOWLEDGE GRAPH

	valveStatus	Name	Solenoid valve
	valveState	Type	ZCF52-15ML
	duration	Protocol	ModBus
	alarm	Brand	**Dunming**
IoT BaseOntology (RDF triple)			
<Valve> <rdf:type> <m2m:Device> <Valve> <zm:runAt> <middleware/l2/1001> <Valve> <zm:runAt> <middleware/l1/01 > ...			
IoT DomainOntology (RDF triple)			
<Valve> <valveStatus> <owl:cardinality/1> <valveStatus> <rdf:type> <StructuredVariable> ...			
IoT Instance Graph (RDF triple)			
<Valve/[id]> <hasProperty> <StructureVariable/valveStatus/[id]> <StructureVariable/valveStatus/[id]> <hasProperty> <valveState/[value]> <StructureVariable/valveStatus/[id]> <hasProperty> <duration/[value]> <StructureVariable/valveStatus/[id]> <hasProperty> <alarm/[value]> ...			

describe the processes of mapping these devices into a knowledge graph.

B. Experimental Sewage Disposal Equipment

A self-developed experimental sewage disposal equipment is tested before application deployment. This equipment is a scaled-down version of the actual sewage disposal site. It consists of an L2 IoT middleware, an air pump, a lifting pump, a reflux pump, a reaction pool, a COD sensor, an oxygen sensor, a turbidimeter, etc., as shown in Fig. 9.

By applying the proposed method, all the devices of the equipment are mapped to the nodes of an IoT knowledge graph, as shown in Fig. 10.

The small green nodes in Fig. 11 are the nodes of the IoT knowledge graph that is connected to L2 IoT middleware. The value node can be device URI, value variable, structured variable, or sequential variable according to the

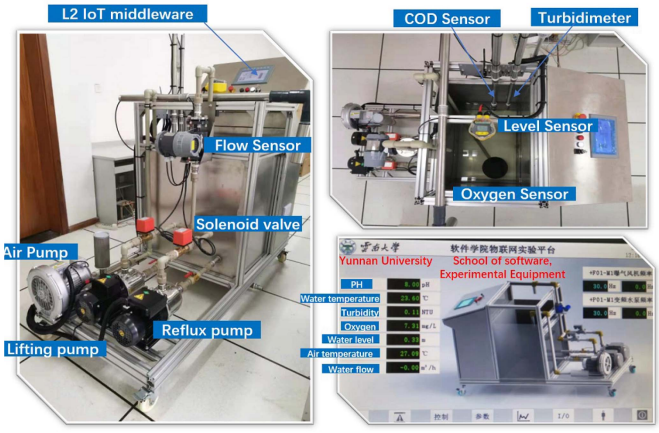


Fig. 9. Experimental sewage disposal equipment with nonsmart IoT devices.

proposed method. By using the knowledge graph mapping, the complex equipment is decomposed into atomic value nodes of the graph, which are easy to manipulate through L2 IoT

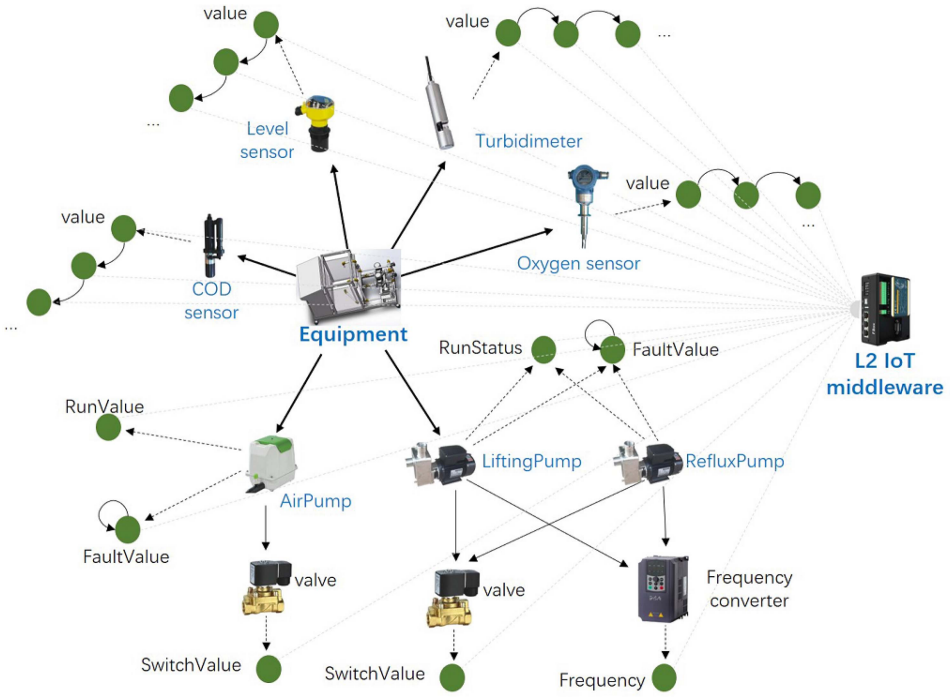


Fig. 10. Knowledge graph of the corresponding experimental equipment.

middleware. It can solve the communication problems for small IoT devices. Since all terms of nodes are unified by IoT base and domain ontology (mentioned in Section III-A), the heterogeneous problems among IoT devices can be solved.

C. Application

The generated IoT knowledge graph is simultaneously deployed in the L1 and L2 IoT middlewares. The L1 IoT middleware stores only the root node of the device and the profile of the device, such as the brand, model, and name. The L2 IoT middleware stores the dynamic data of the device, such as the values collected by the sensor and the running state of the device. The service for accessing the knowledge graph is provided through the SPARQL endpoint, as described in Section III-C. In the query or modification operation, the L1 IoT middleware can further access the L2 IoT middleware according to the actual query and operation. In this case, we use ElementUI,[11] Electron,[12] and Spring Boot[13] to build a client and an application server. The L1 IoT middleware is also deployed on the application server. The client and the application server access the IoT knowledge graph via the SPARQL endpoint. After the knowledge graph mapping, the IoT devices of each station became the device nodes and value nodes of the IoT knowledge graph. The device nodes and value nodes are then stored in both L1 and L2 IoT middleware relatively. The system can easily query or modify these nodes from the client. By querying the root nodes in the IoT knowledge graph, the list of stations and devices can be obtained. By querying the corresponding value nodes of a device in the IoT knowledge graph, the data collected by the sensors of the device can be obtained. By modifying the corresponding value nodes and their edges in the IoT knowledge graph, the status of the device can be changed.

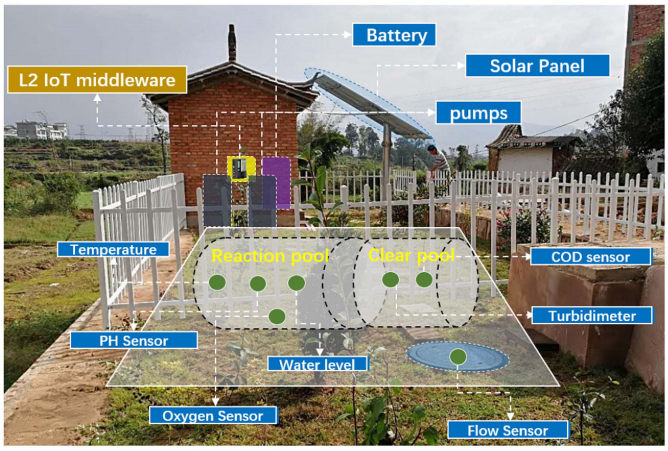


Fig. 11. Representative sewage treatment station applied the proposed multilayer IoT middleware.

Fig. 11 presents a representative sewage treatment station that applies the proposed method. Similar to the experimental sewage treatment equipment, the IoT devices of the station are decomposed into atomic value nodes of the graph, which are easy to manipulate through L2 IoT middleware. It solves the communication problems for small IoT devices. Since all terms of value nodes are unified by IoT base ontology and IoT domain ontology, the heterogeneous problems among IoT devices can be also solved.

In Fig. 11, the green nodes are the value nodes (value variable, structured variable, or sequential variable) of the IoT

[11] https://element.eleme.io/
[12] https://electronjs.org/
[13] http://spring.io/projects/spring-boot/

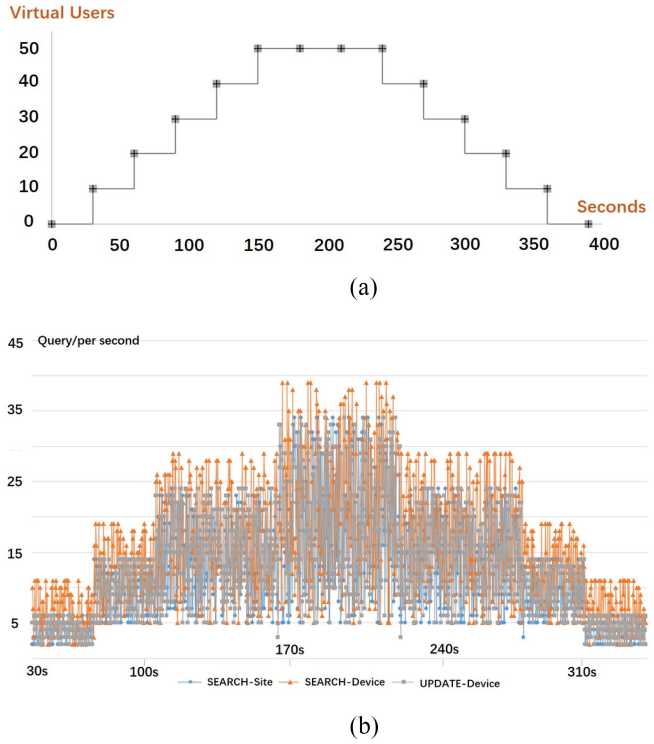


Fig. 12. (a) Simulative stress of gradually increase in the number of concurrent users. *x*-axis represents the time. *y*-axis represents the number of concurrent users. (b) Concurrent queries of knowledge graph service invoked by the virtual users.

knowledge graph. All the IoT devices are mapped to device URI in the knowledge graph that is linked with its value nodes. Furthermore, value nodes are connected with L2 IoT middleware that can be manipulated from L1 IoT middleware.

D. Evaluation

In this case, the core function of the sewage treatment system is an IoT knowledge graph service. IoT knowledge graph service is a type of Web service, and its performance is typically measured by the quality of service. In this section, we will use the main indicators of QoS, namely, the response time, reliability, availability, and scalability, to quantitatively evaluate the performance of the proposed method in this case study.

1) Response Time: In this work, we use the dedicated performance testing tool performance runner[14] to test the deployed IoT system. First, we set up 50 virtual users to simulate concurrency in the performance runner. The initial number of virtual users is 10, increasing by 10 every 30 s. Hold for 90 s after adding 50 virtual users. After that, the number of virtual users began to decrease gradually, by 10 per 30 s, until the number of virtual users stopped at 0. Fig. 12(a) plots the changes in the number of virtual users. Fig. 12(b) plots the corresponding number of concurrent queries invoked by the virtual users.

Based on virtual concurrent users, we choose a typical service query process for this case. First, a sewage treatment station root node (SEARCH-Site) in the IoT knowledge graph is queried. Then, according to the site node, the statuses of all devices and sensors in the sewage treatment station are queried (SEARCH-Device). Finally, the sewage reflux pump is selected, and the IoT knowledge graph modification operation is performed to modify the states of the reflux pump (UPDATE-Device). Fig. 13 plots the response time of the IoT knowledge graph service of the sewage disposal system.

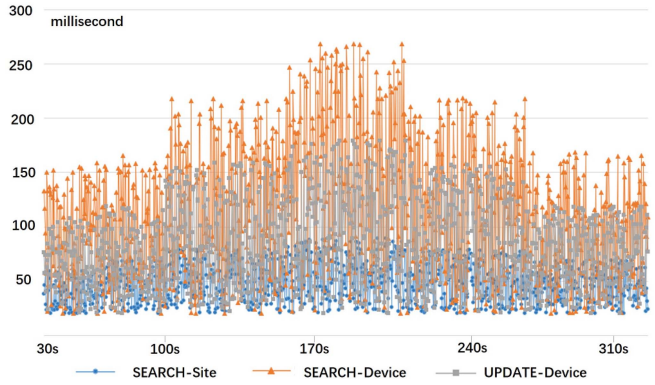
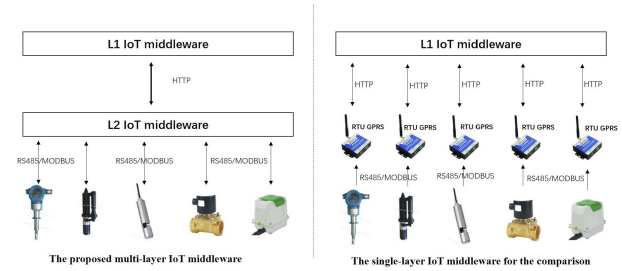


Fig. 13. Response time of the IoT knowledge graph service of the sewage disposal system.

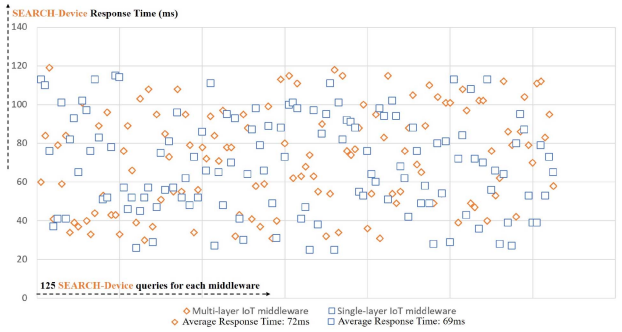
According to Figs. 12(b) and 13, with fewer than 50 concurrent virtual users, the response times of the three types of IoT knowledge graph service can be kept below 300 ms. The overall average response time is 127 ms, which satisfies the response time requirements of general service-based application systems (< 200 milliseconds). For a single node query of a site (SEARCH-Site), the average response time is 43 ms. For multinode query service of devices (SEARCH-Device), the average response time is 151 ms. The time consumed by SEARCH-Device is three times that by SEARCH-Site, which is due to two main reasons: 1) SEARCH-Site only needs to locate the root node of the site and does not need to go deep into the knowledge graph; thus, the response time is short and 2) SEARCH-Device must call the L2 IoT middleware where the device is located (SEARCH-Site only needs to access the L1 IoT middleware), which requires more graph node queries, so the response time increases accordingly.

As the number of concurrent users is gradually increased, the response times of services are affected; however, the effect differs among services. When the number of virtual concurrent users increases to 50, the response time of SEARCH-Site increases by approximately 20%, the response time of SEARCH-Device increases by approximately 80%, and the response time of UPDATE-Device increases by approximately 50%. The response time of SEARCH-Site shows that the latency of the knowledge graph service is insensitive to concurrency increases. This is because SEARCH-Site can be directly executed in L1 IoT middleware. SEARCH-Device and UPDATE-Device are evidently affected by the increasing of concurrent queries due to the limited computing and communication ability of L2 IoT middleware. But, the increasing ratio of latency is much lower than the growth rate of concurrent users. In practical applications, the L2 IoT middleware is distributed in the sewage treatment station. It will be called only

[14]http://www.spasvo.com/Products/PerformanceRunner.asp



(a)



(b)

Fig. 14. (a) Setup of the middleware comparison in the term of SEARCH-Device response time. (b) Response time of SEARCH-Device in the proposed multilayer IoT middleware and the single-layer IoT middleware.

when the device operation is needed and it will not produce high concurrency.

We also compared the response time of the proposed multilayer IoT middleware with a single-layer IoT middleware (L2 middleware is unloaded). L1 IoT middleware requires devices to access to the IoT network first. Then, the devices use CoAP, MQTT, or HTTP requests to communicate with L1 IoT middleware. However, without L2 IoT middleware, most of the small devices, such as a valve, turbidimeter, oxygen sensor, etc., cannot access to the IoT network independently. Therefore, we add RTUs (remote terminal unit, GPRS version) in a selected sewage treatment station for each small device to access to the L1 IoT middleware. In this setup, the L1 IoT middleware can only receive the data from small devices through the added RTUs. It cannot control the small devices since the added RTUs cannot hold the knowledge graph for these devices. Fig. 14(a) shows the set up of the middleware comparison. Fig. 14(b) shows the comparison result of the proposed multilayer IoT middleware with a single-layer IoT middleware.

As shown in Fig. 14(b), the average response time of the proposed multilayer IoT middleware and the single-layer IoT middleware is close. L2 IoT middleware does not bring new overheads into the architecture. Besides, L2 IoT middleware with a knowledge graph solves the problem of the communication gap and heterogeneity of the small IoT devices.

2) Reliability and Usability: Reliability is an important index for measuring the running state of an IoT application. Reliability refers to the ability to satisfy the criteria of normal and healthy operation of its functions at special intervals. Reliability can be measured by the mean time between failures (MTBF). For this case, reliability refers to how long the IoT knowledge graph service can run normally before a failure occurs. Faults of a device or a sensor are not included in the reliability evaluation of the IoT knowledge graph service. The system was launched on September 20, 2018, with 15 sewage treatment stations accessing the system. As of December 20, 2018, the IoT knowledge graph service faults that have been recorded by the system are listed in Table IV.

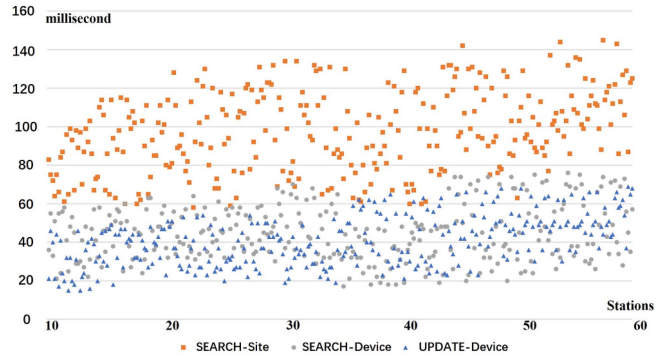
TABLE IV
MATURE TECHNOLOGIES AND PLATFORMS USED IN THIS STUDY

Site ID	Failure Source	Failure Date	Recovery Date
300218040314	Network failure	2018-09-27 19	2018-09-27 21
300218100136	Network failure	2018-09-29 12	2018-09-29 15
300217090161	Network failure	2018-10-14 09	2018-10-14 11
300217090165	Power failure	2018-10-22 14	2018-10-22 15
300218090078	out of credit	2018-11-08 01	2018-11-08 10
300218100136	Network failure	2018-11-19 04	2018-11-19 11
300218090068	Power failure	2018-12-01 09	2018-12-01 10
300218070562	Network failure	2018-12-05 16	2018-12-05 18

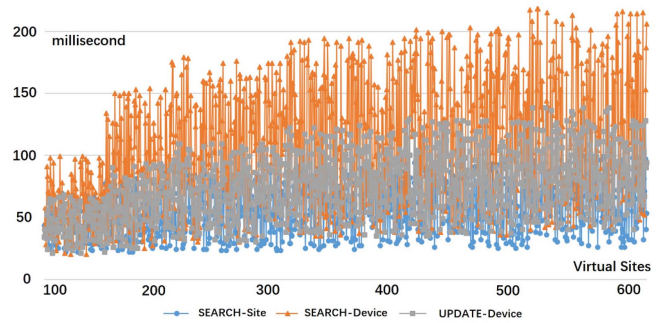
According to the fault records in Table IV, there are 91 total operating days of the system, which is equivalent to 2184 hr. In these three months, the IoT knowledge graph service in a site was unavailable five times due to network failure, two times due to power failure, and one time due to "out of credit" failure, for a total of nine times. These faults occur at sewage treatment stations in mountainous areas and correspond to the L2 IoT middleware in the system. The L1 IoT middleware was in operation for three months without failure. According to the MTBF calculation formula, the failure rate (λ) is 0.0041 times per hour, and MTBF $== 1/\lambda ==$ 242 hr. According to an in-depth investigation, two power failures were due to the local long overcast days, on which the solar battery was exhausted, and the L2 IoT middleware was offline due to power failure. The one "out of credit" failure was caused by maintenance personnel forgetting to recharge the IoT SIM card. This type of malfunction was caused by the irregular operation of personnel, which can be gradually eliminated via standardized training.

The other five failures were caused by network failure. The causes of these faults are complex and it is difficult to determine the factors responsible for them. By calculating the mean time to restoration (MTTR) and MTBF, we determine the availability of the system as $=$ MTBF /(MTBF $+$ MTTR) $=$ 98.8%. As can be calculated from Table IV, the MTTR is 3 hr. In actual maintenance, the MTTR is mainly spent in the process of maintenance personnel rushing to the site, not in the process of equipment maintenance. Therefore, the availability of the system can be further improved via a reasonable allocation of maintenance personnel to manage the area.

3) Scalability: Our cooperative company plans to build 1500 rural sewage treatment stations by 2022, all of which require access to the current IoT system. Therefore, scalability is an important index for measuring whether the proposed IoT system can meet the practical application needs. There are 103 sewage treatment stations, and 63 of them have applied the proposed multilayer IoT middleware. The scale of the sewage treatment stations is still inadequate for scalability testing. Therefore, we test both real and virtual sewage treatment stations. For the virtual station, we set up three cloud



(a)



(b)

Fig. 15. (a) Response time of the IoT knowledge graph service with the growth of the real stations. (b) Response time of the IoT knowledge graph service with the growth of the virtual stations.

servers distributed in various areas. Up to 200 L2 IoT middlewares are virtualized on each cloud server for simulating up to 200 sewage treatment stations. The L2 IoT middleware, which is virtual in cloud servers, has a stable network and low network latency, which affects the response time in the simulation. However, in this case, the scalability is used to evaluate whether the system can effectively handle service requests under a continuous increase of load (sewage treatment station), which does not focus on the absolute value of the response time. Therefore, the virtual sewage treatment station can also restore the load of the real sewage treatment station well.

In the test, we mainly examine the change in the response time of the IoT knowledge graph service under increasing load. Here, we also use performance runner, set up ten virtual users, and concurrently call the services (SEARCH-Site, SEARCH-Device, and UPDATE-Device) of the IoT knowledge graph. Fig. 15 gives the changes in the response time of the IoT knowledge graph service with the growth of virtual sites.

According to Fig. 15(a), as the number of stations increases from 10 to 60, the response time of IoT knowledge graph service has no significant change. According to Fig. 15(b), as the number of virtual stations increases from 100 to 600, the response times of the three services increases slightly. The response times of SEARCH-Site, SEARCH-Device, and UPDATE-Device increase by approximately 30%, 90%, and 50%, respectively. The response times of the three types of services rapidly increase from 100 to 200 sites. When the number of sites exceeds 400, the response times of the three types of services are relatively stable, with only a small increase. Since SEARCH-Site only queries the root node of the knowledge graph stored in the L1 IoT middleware, which has adequate computing and communication capacities, the increase in the number of sites has little effect on the performance of the service. For SEARCH-Device and UPDATE-Device, the number of L2 IoT middlewares increases as the number of sites increases. The execution of these services is distributed in the L2 IoT middlewares, which can better disperse computing requirements. Although the number of virtual sites continues to increase, the response times of the services have not doubled.

In this simulation, the average response times of SEARCH-Device and UPDATE-Device are lower than the previously measured response times. This might be due to the deployment of the L2 IoT middleware in cloud servers rather than in IoT adapters with limited computing and network capabilities in real sewage treatment stations. This might cause the computing power and the network conditions of the virtual site to be superior to those of the actual environment, which might decrease the average response times of SEARCH-Device and UPDATE-Device.

V. Conclusion

In this article, a multilayer IoT middleware for providing flexible IoT device access has been proposed. The proposed solution is based on an IoT knowledge graph, which aims at reducing the heterogeneity among IoT devices. The multilayer middleware supports both WAN (high level) and field (low level) protocols that enable small IoT devices also to be accessed. The feasibility of the proposed solution, along with its limited overhead, has been demonstrated through a real-life IoT project, namely, a remote monitoring project of rural sewage treatment stations in Yunnan Province, China. The performance and scalability evaluation demonstrate the effectiveness of the proposed solution. In the near future, the proposed solution will combine with artificial intelligence technology to realize adaptive remote monitoring. It will be a significant increase in the efficiency of rural sewage treatment in Yunnan Province, China. By cooperating with environmental companies, it is estimated that the proposed solution is planned to apply to more than 1500 rural sewage treatment stations. Furthermore, the proposed solution could also be applied to other IoT-related industries, such as environmental monitoring, protection area monitoring, mountain forestry, rural agriculture, etc.

References

[1] O. B. Sezer, E. Dogdu, and A. M. Ozbayoglu, "Context-aware computing, learning, and big data in Internet of Things: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 1–27, Feb. 2018.

[2] G. Tanganelli, C. Vallati, and E. Mingozzi, "Edge-centric distributed discovery and access in the Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 425–438, Feb. 2018.

[3] B. Cheng, S. Zhao, J. Qian, Z. Zhai, and J. Chen, "Lightweight service mashup middleware with rest style architecture for IoT applications," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 3, pp. 1063–1075, Sep. 2018.

[4] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT middleware: A survey on issues and enabling technologies," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 1–20, Feb. 2017.

[5] W. R. A. R. Giancarlo Fortino, and C. Savaglio, "On the classification of cyberphysical smart objects in the Internet of Things," in *Proc. CEUR Workshop*, 2014, pp. 86–94.

[6] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 1, pp. 70–95, Feb. 2016.

[7] M. A. A. da Cruz, J. J. P. C. Rodrigues, J. Al-Muhtadi, V. V. Korotaev, and V. H. C. de Albuquerque, "A reference model for Internet of Things middleware," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 871–883, Apr. 2018.

[8] R. T. Tiburski, L. A. Amaral, E. D. Matos, and F. Hessel, "The importance of a standard securit y archit ecture for SOA-based IoT middleware," *IEEE Commun. Mag.*, vol. 53, no. 12, pp. 20–26, Dec. 2015.

[9] M. A. da Cruz, J. J. Rodrigues, A. K. Sangaiah, J. Al-Muhtadi, and V. Korotaev, "Performance evaluation of IoT middleware," *J. Netw. Comput. Appl.*, vol. 109, pp. 53–65, May 2018.

[10] *Standards for M2M and the Internet of Things*, oneM2M.Org, USA, 2012. [Online]. Available: https://www.onem2m.org/

[11] *Lightweight M2M*, OMA-SpecWorks, San Diego, CA, USA, 2016. [Online]. Available: http://www.openmobilealliance.org/wp/Overviews/lightweightm2m_overview.html

[12] *Fiware NGSI V2*, ETSI-ISG-CIM-Initiative, Sophia Antipolis, France, 2018. [Online]. Available: https://fiware-datamodels.readthedocs.io/en/latest/ngsi-ld_howto/index.html

[13] D. Raggett, "The Web of things: Challenges and opportunities," *Computer*, vol. 48, no. 5, pp. 26–32, May 2015.

[14] L. Yao, Q. Z. Sheng, and S. Dustdar, "Web-based management of the Internet of Things," *IEEE Internet Comput.*, vol. 19, no. 4, pp. 60–67, Jul./Aug. 2015.

[15] Y. Qin, Q. Z. Sheng, N. J. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos, "When things matter: A survey on data-centric Internet of Things," *J. Netw. Comput. Appl.*, vol. 64, pp. 137–153, Apr. 2016.

[16] A. Farahzadi, P. Shams, J. Rezazadeh, and R. Farahbakhsh, "Middleware technologies for cloud of things: A survey," *Digit. Commun. Netw.*, vol. 4, no. 3, pp. 176–188, 2018.

[17] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[18] *Fiware-IoT-Agent*, FIWARE-Agent, Berlin, Germany, 2019. [Online]. Available: https://fiware-academy.readthedocs.io/en/latest/iot-agents/idas/index.html

[19] *Orion Context Broker*, FIWARE-ORION, Berlin, Germany, 2020. [Online]. Available: https://fiware-orion.readthedocs.io/en/master/

[20] M. Eisenhauer, P. Rosengren, and P. Antolin, "A development platform for integrating wireless devices and sensors into ambient intelligence systems," in *Proc. 6th IEEE Annu. Commun. Soc. Conf. Sens. Mesh Ad Hoc Commun. Netw. Workshops*, Rome, Italy, Jun. 2009, pp. 1–3.

[21] J. Soldatos *et al.*, "OpenIoT: Open source Internet-of-Things in the cloud," in *Interoperability and Open-Source Solutions for the Internet of Things*, I. P. Žarko, K. Pripužić, and M. Serrano, Eds. Cham, Switzerland: Springer, 2015, pp. 13–25.

[22] *Open Source IoT Data Platform With the Wide Range of Integration Options*, DataArt, New York, NY, USA, 2019. [Online]. Available: https://devicehive.com/

[23] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 1199–1202. [Online]. Available: http://lsir.epfl.ch/research/current/gsn/

[24] Y. Qu, L. Jie, L. Kang, Q. Shi, and Y. Dan, "Question answering over freebase via attentive RNN with similarity matrix based CNN," 2018. [Online]. Available: arXiv:1804.03317.

[25] Y. Zhang, H. Dai, Z. Kozareva, A. J. Smola, and L. Song, "Variational reasoning for question answering with knowledge graph," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 6069–6076.

[26] K. M. Annervaz, S. B. R. Chowdhury, and A. Dukkipati, "Learning beyond datasets: Knowledge graph augmented neural networks for natural language processing," in *Proc. 16th Annu. Conf. North Amer. Ch. Assoc. Comput. Linguist. Hum. Lang. Technol.*, 2018, pp. 313–322.

[27] C. Xie, P. Yang, and Y. Yang, "Open knowledge accessing method in IoT-based hospital information system for medical record enrichment," *IEEE Access*, vol. 6, pp. 15202–15211, 2018.

[28] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2D knowledge graph embeddings," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1811–1818.

[29] D. Zhang *et al.*, "Knowledge graph-based image classification refinement," *IEEE Access*, vol. 7, pp. 57678–57690, 2019.

[30] C. Xie, H. Cai, L. Xu, L. Jiang, and F. Bu, "Linked semantic model for information resource service toward cloud manufacturing," *IEEE Trans. Ind. Informat.*, vol. 13, no. 6, pp. 3338–3349, Dec. 2017.

[31] J. Lehmann *et al.*, "Dbpedia—A large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, Jan. 2015.

[32] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 697–706.

[33] A. Carlson, J. Betteridge, R. C. Wang, E. R. Hruschka, Jr., and T. M. Mitchell, "Coupled semi-supervised learning for information extraction," in *Proc. 3rd ACM Int. Conf. Web Search Data Min.*, 2010, pp. 101–110.

[34] X. Dong *et al.*, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discover. Data Min.*, 2014, pp. 601–610.

[35] G. Ji, K. Liu, S. He, and J. Zhao, "Knowledge graph completion with adaptive sparse transfer matrix," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 985–991.

[36] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proc. 28th AAAI Conf. Artif. Intell.*, 2014, pp. 1112–1119.

[37] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic Web*, vol. 8, no. 3, pp. 489–508, 2016.

[38] C. Cañas, E. Pacheco, B. Kemme, J. Kienzle, and H. Arno, "GraPS: A graph publish/subscribe middleware," in *Proc. 16th Annu. Middleware Conf.*, Vancouver, BC, Canada, 2015, pp. 1–12. [Online]. Available: https://doi.org/10.1145/2814576.2814812

Cheng Xie (Member, IEEE) received the B.S. degree in software engineering from the Minzu University of China, Beijing, China, in 2009, and the M.S. and Ph.D. degrees in software engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012 and 2017, respectively.

From 2015 to 2016, he was a Visiting Scholar with the Data and Web Science Group, University of Mannheim, Mannheim, Germany. He is currently an Associate Professor with the School of Software, Yunnan University, Kunming, China. His research interests include knowledge graph and industrial informatics.

Dr. Xie received the Shanghai Science and Technology Progress Award (Second Prize) in 2014, the Shanghai Outstanding Graduates in 2017, the Youth Support Project of China Association for Science and Technology in 2018, and the Thousand Talents Program of Yunnan in 2018.

Beibei Yu received the B.S. degree in information security from Yunnan University, Kunming, China, where she is currently pursuing the M.S. degree in data science and engineering with the School of Software.

Her current research interests include knowledge graph and industrial informatics.

Zuoying Zeng received the B.Sc. degree in software engineering from Sichuan Normal University, Chengdu, China, in 2018. He is currently pursuing the M.Sc. degree in artificial intelligence and machine learning with Yunnan University, Kunming, China.

His current research interests include knowledge graph, machine learning, and Internet of Things.

Yun Yang (Member, IEEE) received the B.Sc. (Hons.) degree in information technology and telecommunication from Lancaster University, Lancaster, U.K., in 2004, the M.Sc. degree in advanced computing from Bristol University, Bristol, U.K., in 2005, and the M.Phil. degree in informatics and the Ph.D. degree in computer science from the University of Manchester, Manchester, U.K., in 2006 and 2011, respectively.

He was a Research Fellow with the University of Surrey, Surrey, U.K., from 2012 to 2013. He is currently with the National Pilot School of Software, Yunnan University, Kunming, China, as a Full Professor of machine learning. His current research interests include machine learning, data mining, pattern recognition, and temporal data process and analysis.

Prof. Yang serves as an Associate Editor for *Neural Networks*.

Qing Liu received the bachelor's degree from Yunnan University, Kunming, China, in 1985, and was selected to be a teacher with the Department of Computer Science, Yunnan University, and the master's degree in computer science from the University of Electronic Science and Technology of China, Chengdu, China, in 1993.

He was a visiting Scholar with the University of Illinois, Champaign, IL, USA, from 1999 to 2000, and an Advanced Researcher with the De Montfort University, Leicester, U.K., in 2006. He is currently a Professor with the National Pilot School of Software, Yunnan University, also as the Deputy Director of the Software Engineering Key-Lab of Yunnan Province. His current research interests include software engineering, data science, and intelligent computing.