IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 24, 2018, accepted September 28, 2018, date of publication October 15, 2018,
date of current version November 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2876197

A OneM2M-Compliant Stacked Middleware Promoting IoT Research and Development

RONGZHEN ZHAO[1], LITIAN WANG[1], XINGZHE ZHANG[1], YU ZHANG[1], LIZHI WANG[2], AND HONGZHAO PENG[1]

[1]College of Electronic Information and Optical Engineering, Nankai University, Tianjin 300350, China
[2]College of Computer and Control Engineering, Nankai University, Tianjin 300350, China

Corresponding author: Litian Wang (wanglitianrf@sina.com)

This work was supported by Nankai University.

ABSTRACT Internet of Things' (IoT) applications keep emerging in all living and production scenarios. However, development tasks of support, communication, and computing are usually troublesome. An IoT middleware that completes such common basic tasks and assists advanced development is desired. Numerous general-purpose IoT instances have been tried in both academia and industry, while each has its own advantages and disadvantages. We abstract their characteristics into an ideal concept then build a reliable, modular, and oneM2M-compliant middleware accordingly. Our contributions include: proposing a stack of support-communication-computing to integrate excellent open-source projects; devising techniques variable and McSugar to enable flexible uniform human-thing interactions; and building implementation foundations for cutting-edge technologies such as fog computing and semantic reasoning. This middleware has been verified and applied: the case of field-cloud computing shows its efficacy in facilitating IoT research; the case of smart floriculture proves its capability in boosting IoT development. In short, with this middleware, developers and researchers can focus on top-level requirements of IoT development or experiment, instead of being trapped in the underlying technologies.

INDEX TERMS Development & research, human-thing interaction, Internet of Things (IoT), middleware, oneM2M.

I. INTRODUCTION

The Internet of Things (IoT) has been changing the way in which humans interact with the surrounding world greatly, using technologies like intelligent perception, identification, pervasive computing and so on. The IoT is therefore recognized as the third revolution in the information industry, after invention of the computer and Internet. Large quantities of IoT applications are running, such as smart homes, smart transportation, precision agriculture, remote healthcare and industrial automation, allowing people to enjoy the fruits of human intelligence [1]–[5].

The oneM2M is a globally adopted IoT standard. It divides IoT environments into two domains, namely, the infrastructure and field domain; and defines nodes into five types, namely, the infrastructure node (IN), middle node (MN), application service node (ASN), application dedicated node (ADN) and non-oneM2M device node (NoDN), which are shown in Figure 1 [6], [7]. Mapped to a common IoT application system in Figure 2, NoDNs correspond to sensors and actuators in a WSAN (Wireless Sensors and Actuators Network), where protocol conversion via gateway is required for them to join an IoT network; ADNs can be smart objects, which have constrained resources but support IoT protocols; ASNs correspond to smart objects or client devices, which have sufficient resources and support IoT protocols; MNs can be gateways in the field domain, which also have sufficient resources and support both field protocols and IoT protocols; INs correspond to cloud platforms or servers. With an IoT application system, users can monitor and control field conditions via the link of client device - cloud platform - field devices. As the number of unconstrained nodes (ASNs, MNs) grows, we can infer that the enriching computing resources in field domain will enable fog/cloud computing or their fusion, where tasks are computed near data sources to improve instantaneity [8], [9].

Besides, oneM2M abstracts nodes into three layers, namely, the application layer, common service layer and network service layer; but not every node has all layers,

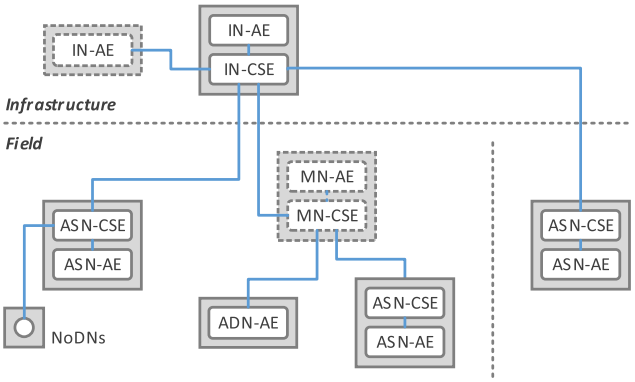

FIGURE 1. A possible configuration of oneM2M architecture.

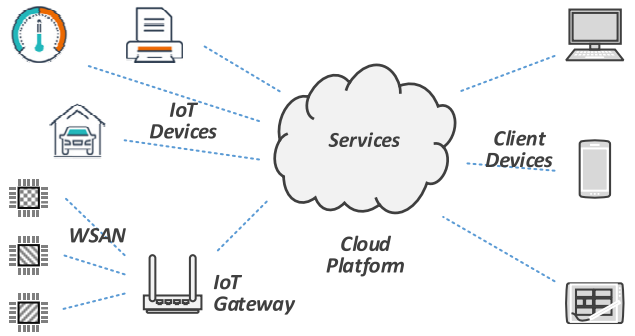

FIGURE 2. Composition of a common IoT application instance.

as shown in Figure 1. Each layer has its own entities: application entities (AEs) stand for the applications in devices, gateways or servers; common service entity (CSE) is a set of common service functions for IoT services; network service entities (NSEs) (omitted in Figure 1) are the underlying network services that are available for CSE. The oneM2M adopts the ROA (Resource Oriented Architecture) model, where all devices can be handled as resources using a hierarchical structure. So, based on the URI (uniform resource identifier) technique, which also has a hierarchical structure, and the CRUD (create, retrieve, update, delete) methods, which are highly uniform, comprehensive human-thing interactions can be implemented using simple interfaces like Web APIs; but how to describe those heterogeneous devices into uniform resources is a challenge. oneM2M supports various IoT protocols like MQTT (Message Queuing Telemetry Transport) and CoAP (Constrained Application Protocol), and thus can connect human and things sufficiently; yet the messaging mechanisms of those protocols are too different to develop uniform APIs over them. In addition, semantic reasoning is introduced by oneM2M, which is a potential capability that oneM2M-compliant IoT instances have [10], [11]. In a word, conforming oneM2M when building IoT systems is worthwhile.

According to the latest highly-cited IoT surveys [12]–[15], tasks of building an IoT application system can be divided into common basic and advanced parts, as shown in Figure 3. The common basic tasks of three aspects

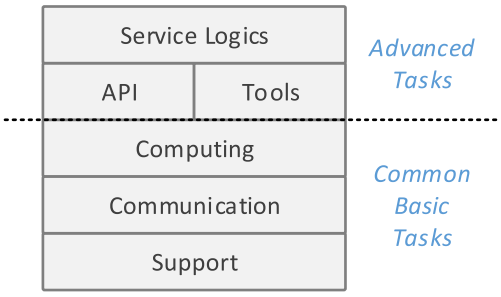

FIGURE 3. Tasks in IoT development.

must be addressed firstly: (1) Support, including real-time operating system, hardware abstraction, app engine/runtime and so on.; (2) Communication, including field protocols (Z-Wave, CAN Bus, Bluetooth, etc.), IoT protocols (MQTT, CoAP, etc.), various types of networks (Ethernet, Cellular, Wi-Fi, etc.) and so on; (3) Computing, including read/write of end devices, management of heterogeneous devices/data, service logic decomposition/mapping/execution, security and so on. Then comes the advanced tasks: building service logics over APIs and tools to fulfill specific requirements, which can also be seen as computing. Fulfilling these tasks is no doubt a great challenge for developers. Clearly, a middleware where those common basic tasks are fulfilled and framework or toolkit is provided for the advanced tasks is desired.

Although many IoT instances for general purposes have been tried out in both academia and industry, each has its own strengths and weaknesses. Commercial instances, IoT service platforms in particular, tend to be overly generalized, with no sufficient support for specific requirements; the academic ones are either too ahead of the time to be put into practice, or too limited to certain scenarios to leave room for cutting-edge technologies. Many of them didn't take oneM2M into consideration. Meanwhile, excellent open-source projects for IoT are just being used in isolation, and no IoT instance has ever tried to utilize them as a whole so far.

Based on the discussions above, we build a oneM2M-compliant stacked IoT middleware, where mature IoT open-source projects are integrated and the great obstacle that developers fall into common basic tasks instead of focusing on advanced development is overcome. Functioning throughout the middleware, techniques Variable and McSugar are devised to utilize heterogeneous device resources and realize uniform human-thing interactions. These techniques also provide the foundation for cutting-edge IoT researches like semantic reasoning and fog computing. With this middleware, developers can build IoT applications for specific requirements in many scenarios quickly, and researchers can setup testbed easily to conduct their experiments.

The remainder of this paper is organized as follows. The second section analyzes the advantages and disadvantages of both commercial and academic IoT instances which are intended to promote IoT prototyping or instantiating. In section three, the overall goals for an excellent IoT middleware are summarized according to these cases.

The fourth section elaborates such goals into technical characteristics to guide the implementation, and introduces two enabling techniques for the building of this middleware. In section five, steps to take this middleware into effect are presented, and two case studies are described to demonstrate the powerful role that this middleware plays in IoT research and development. Section six provides the conclusions.

II. RELATED WORK

There are now a large number of commercial platforms providing common services to simplify IoT development in the fast-growing IoT market. EvryThing is a cloud platform for smart objects [16], where sensing data are utilized in encapsulated virtual bodies according to users' rules to realize personalized applications. Web UI (User Interface) is adopted for development and management, and it can interwork with other platforms. IFTTT (If This Then That) provides Web interoperation services among different platforms using a set of normalized triggering conditions [17]. It provides rule-tree based APIs for programming, but with no ability to handle execution conflicts caused by complex rules. SensorCloud allows users to access sensing data via RESTful APIs [18]. Advanced mathematical processing tools are provided for data analysis and visualization, and it also supports services based on triggering conditions. Device resource aggregation requires users to appoint the URIs. ThingSpeak allows sensors to be linked using HTTP, and users can access their multidimensional sensing data via RESTful API [19]. Complex data processing can be achieved based on MATLAB programming; data management, analysis, visualization and event triggering are also provided. ThingWorx provides virtual entity services based on Axeda cloud platform [20]. It is a typical cloud-based IoT system, just like other commercial platforms. It is also equipped with GUI (Graphics User Interface) for application editing, device/data management and service logic execution.

Such platforms are mostly Web-based, so their adoption of HTTP (HyperText Transfer Protocol) makes themselves unsuitable for the human-thing interactions in IoT scenarios. Their high-level abstraction does contribute to applicability but also raises the issue that developers still need to do a lot to fulfill specific requirements. The cloud-based architecture requires to upload all data and thus isn't suitable for the applications whose data are confidential and require frequent and real-time exchange. Their profitable nature drives them to develop differentiated services to compete with their opponents, which in turn limits the interoperability among different platforms.

Academia goes even further to lower the threshold of IoT development. Reference [21] proposes a modular architecture for IoT development, and its application instances in healthcare, structural health detection, agriculture and indoor tour guide are presented. But it is just a technology selection guidance, and much work still remains if developers want to build their IoT applications. Reference [22] proposes a middleware, MinT, which features interoperability and sharing in distributed IoT. It realizes active interworking between local and external devices based on the abstract-system-interaction architecture and APIs. MinT focuses on the field part, but common basic tasks in the cloud and user parts aren't addressed sufficiently. Reference [23] proposes an IoT development platform for precision agriculture to promote quick prototyping. Its architecture is derived using concurrent views, where requirements of the developer, user and researcher are considered together. It can be implemented on various cloud platforms. It doesn't address issues like device resource acquisition and management. Reference [24] proposes a message orientated middleware, MoM, to integrate resources in smart building management systems (SBMS). It can adapt to various SBMSs thanks to its distributed architecture and improved interoperability, which is achieved by abstracting the devices and management software. But this proposal needs further generality. Reference [25] proposes a middleware named SITE for smart home scenarios, enabling developers and even users to drive smart objects. This is realized by providing uniform operation rules and correlated UI for data visualization. Yet this middleware shows no effective solution to issues like remote access, device management and mash-up.

These cases are distinctively innovative but not sufficiently practical. Some conduct frontier investigations with assumptions ahead of the time. Some have prominent features, but lack reliability, standard compatibility, and upgrade potentials to new technologies. Some are too scenario-specific to be applied widely. Most of them employ components developed by themselves even though there are many excellent open-source projects addressing IoT issues of all levels, which means they can be utilized in an organic integrity. A similar example is LAMP (Linux, Apache, MySQL and PHP/Perl), which gains great popularity and success in Web development by stacking several open-source technologies together [26]. The stacked design is also well worth learning in IoT development.

III. OVERALL GOALS

According to the existing middleware or platforms for IoT development, a qualified IoT middleware should meet the following requirements:

(1) Complete common basic tasks of IoT development for developers, namely, tasks of support, communication, and computing.

(2) Mask underlying technical details, and provide auxiliary toolkits for developers, such as simple-text or GUI APIs.

(3) Comply with the global IoT standard oneM2M, and provide a foundation for data sharing and integration among different systems, as well as for the implementation of cutting-edge technologies.

(4) Be sufficiently reliable, not only for the security of user information but also for the maintainability of the system.

(5) Be compatible with multiple software and hardware platforms, including different PaaS (Platform-as-a-Service) for cloud platforms, embedded hardware and legacy WSANs.

TABLE 1. Support-communication-computing stack for this middleware.

	Device	Gateway	Server	Client
Advanced	- APIs, debug tools - remote management		- APIs, tools, visualization interfaces - human-thing interaction framework - remote management	- UI for human-thing interaction
Common Basic		- data management - device management	- data management - device management	- visualization - authenticity
	- field / IoT protocols	- local / global communication - field / IoT protocols	- global communication - IoT protocols	- global communication - IoT Protocols
	- hardware abstraction - low-power RTOS	- app engine or runtime - high-efficiency OS	- common PaaS platforms	- common mobile platforms

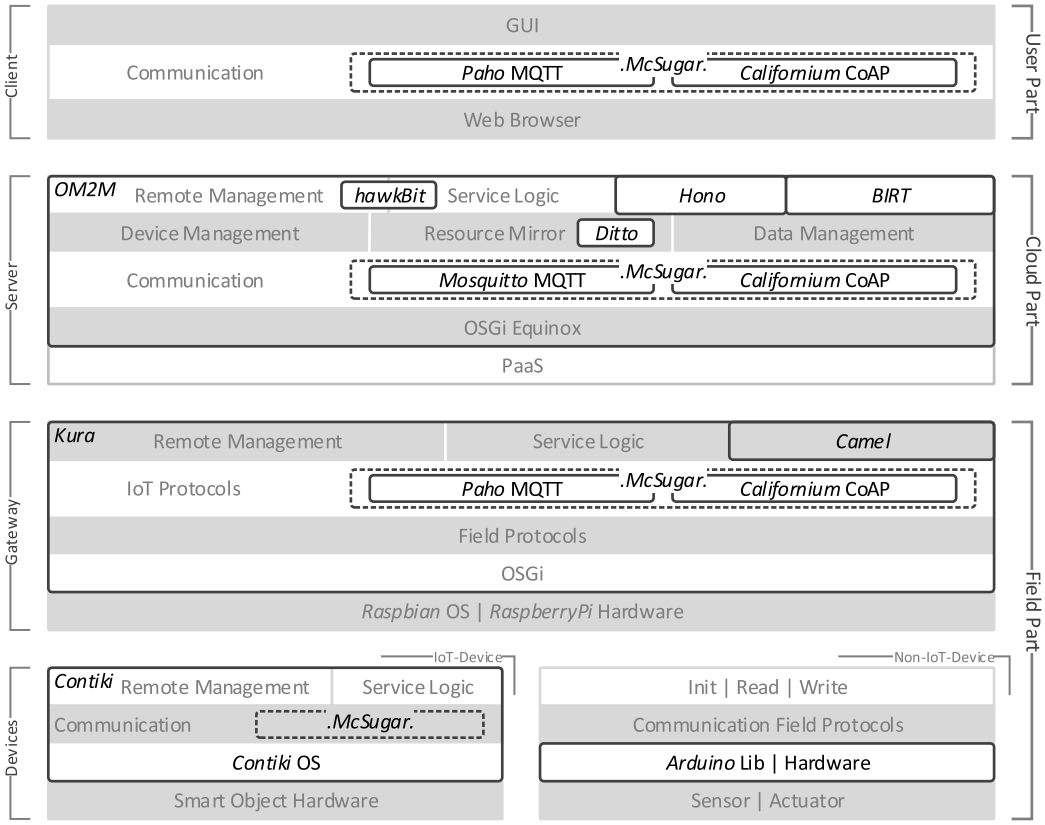

FIGURE 4. Implementation of the middleware stack.

(6) Be modularized and loosely coupled among its components so that IoT systems can be built concurrently and maintained or upgraded convenient.

(7) Ensure high efficiency, low latency and flexibility in human-thing interactions. Flexibility means complex interactions can be realized by composing simple ones simply and briefly.

This middleware, briefly speaking, will enable quick prototyping for IoT application so that developers can focus on advanced development. It will also work as a reliable test bed for novel IoT techniques so that researchers don't need to worry about interfering factors.

IV. MIDDLEWARE DETAILS

Guided by some leading surveys [27]–[30] on IoT middleware, we designed a support-communication- computing stack for the ideal IoT middleware to meet the overall goals, as shown in Table 1. It has four components for the device, gateway, server and client. Open-source projects and hardware platforms are innovatively incorporated together to construct the middleware according to this stack. In addition, enabling techniques Variable and McSugar are proposed to intersect all parts of the middleware to enable flexible and uniform human-thing interactions.

A. IMPLEMENTATION

IoT communities have created a lot of excellent open-source projects that can solve problems of all levels in IoT applications. Many are not originally designed to work together or even for IoT, yet they can be integrated in a comprehensive manner according to our support-communication-computing stack, as shown in Figure 4.

1) FIELD PART

The middleware's component for non-IoT devices (bottom right of Figure 4, Mw-Dev in short) can easily drive various kinds of MCUs (Micro-Controller Units) with the help of Arduino, which is widely used due to its extensive support to hardware platforms [31]. We have extended those widely used field protocols including ZigBee, BLE, CANBus and ZWave into the Arduino library. In the Arduino IDE (Integrated Development Environment), developers can use simple text to program to read and write those devices.

The middleware's component for IoT devices (bottom left of Figure 4, Mw-Dev in short) utilizes Contiki and Paho [32], [33]. As a lightweight real-time operating system, Contiki abstracts hardware details like ARM and their IO. The CoAP of Contiki and the MQTT of Paho are integrated into McSugar for developers to choose for specific scenarios. With McSugar, firmware remote management and resource monitoring are supported. Sub-functionalities of the middleware component can be disabled or enabled according to device performance. Non-IoT devices here are the aforementioned the NoDNs of oneM2M, which have simple functionality and are large in quantity; IoT devices are the ADNs of oneM2M, which are generally complex in functionality. To become a part of an IoT instance, the former needs the driver of MCUs and the organization of WSANs via field protocols; the latter uses IoT protocols directly or with slight transformations.

The component for gateways of the middleware (the last but one layer of Figure 4, Mw-Gtw in short) is built on Kura, Californium and Camel [34]–[36]. Kura realizes compatibility with widely used hardware platforms such as Raspberry Pi; provides OSGi (Open Service Gateway Initiative) runtime for the M2M service logic and APIs for IO such as Serial, RS-232 and BLE; supports multiple field protocols like ModBus, CANBus, ZigBee, BLE and ZWave; supports networking based on different network facilities including Cellular, WiFi and Ethernet. The CoAP of Californium and the MQTT of Kura are integrated into McSugar so that gateways can communicate transparently with the cloud platform, allowing remote customization and maintenance. The Camel engine simplifies the chaotic routing of messages.

2) CLOUD PART

The component for this middleware's cloud part (the second layer of Figure 4, Mw-Svr in short) is built on OM2M, which conforms to oneM2M [37]. Based on Mosquitto and Californium, MQTT and CoAP are bound and integrated into McSugar [38]. Human-thing interaction is decoupled from protocols using Hono [39]. Available device resources can be discovered, abstracted and organized in the style of technique Variable via Ditto [40]. HawkBit enables remote maintenance and service definition [41]. BIRT provides adaptive statistical graphs for data analysis and visualization [42]. The OSGi Equinox runtime enables modularity and extensibility. Based on all above, RESTful APIs allowing simple text service logic defining are provided. Through such APIs, service logic working by being aggregated, decomposed and matched to corresponding device resources.

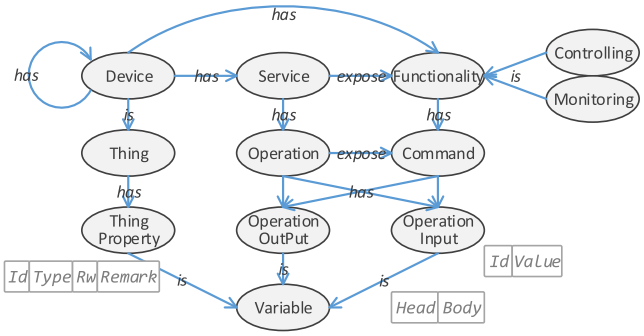

FIGURE 5. Base ontology of oneM2M (partial).

3) USER PART

The component for the user part of the middleware (the top layer of Figure 4, Mw-Clt in short) realizes a webapp client with Material Design (a widely-used beautiful UI design language by Google) styles, so it theoretically can run on all platforms. The client focuses on human-thing interactions and data visualization. McSugar, the integration of Californium and Paho, together with the uniform APIs supported by the cloud platform are the basis of human-thing interactions. Graphical monitor/control interfaces and visualization tools based on BIRT provide users with intuitive visualized analysis and regulation of field conditions.

B. VARIABLES

Technique Variable refers to the Base Ontology of oneM2M, which captures a shared understanding of IoT and provides a formal and machine-manipulable model for IoT, thus inherits its potentials in semantic reasoning [43]. It acts as loosely coupled interfaces between components, making human-thing interactions simplified. Its lightweight characteristic also enables fog computing in the field, which is examined in section case study.

According to oneM2M Base Ontology, as shown in Figure 5, device description and the implementation of Service and its Functionality all ultimately rely on Variable, which is defined as {Head + Body}.

According to Figure 5, Device is the "Things" of the "Internet of Things". A Thing may have several ThingProperties, which can be abstracted into Variables as {Id + Type + Rw + Remark}: Id is identifier, which adopts the lightweight OID solution of oneM2M and is thus of high-efficiency [49]; Rw indicates whether the ThingProperty is readable/sensing or writable/executing by 0 and 1; Remark keeps texts for auxiliary understanding. Type has the following three cases: (1) enum, a device supports a limited number of values; (2) multi, a device supports data within a range and with a precision; (3) chars, a device support character strings.

TABLE 2. McSugar methods based on MQTT and CoAP for interactions.

Monitor	One-Off	CoAP GET, CON	McSugar GET One-Off
	Continual	CoAP GET, Obs, CON/NON; MQTT Subscribe, QoS0	McSugar GET Continual
	Historical	HTTP GET; CoAP GET, CON	McSugar GET Historical
Control	One-Off	CoAP POST, CON	McSugar POST One-Off
	Continual	MQTT Publish, QoS1/2	McSugar POST Continual

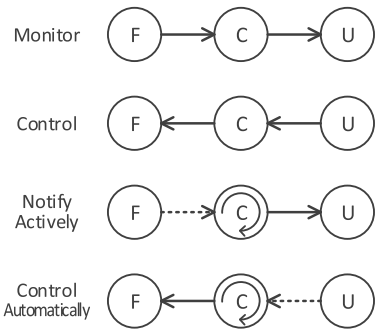

FIGURE 6. Types of human-thing interactions. F: field part; C: cloud part; U: user part; straight arrow: flow of data; arc arrow: advanced services.

Service is based on one or a set of organized devices in the field and provides human-thing interactions under certain service logics. From the perspective of interaction, all services can be categorized into the following four types, as shown in Figure 6: (1) Monitor, physical world information flows from the field via the cloud to the user part in the form of sensing data; (2) Control, user's intention of influencing the physical world flows from user via cloud to field by executing instructions; (3) Active Notification and (4) Automatic Control are when the cloud plays an intelligent role through advanced service logics in the interactions mentioned above [44], [45].

Human-thing interactions are conducted via the data that are generated or to be consumed by devices and are exchanged frequently among the three parts. So they are defined as Operation-Input/Output in {Id + Value}, where Value is the data generated or to be consumed corresponding to a certain Thing-Property. Service logics of interactions thus can be realized by Operation-Input/Outputs along with the device resource mirror, which are lightweight descriptions of devices (Thing-Properties) and are stored on cloud server or gateways.

In the field part, dynamic and heterogeneous devices are uniformly described into resources based on Thing-Property, and gathered by the gateway. In the cloud part, resource mirror can be built to support service logic, namely, service logic is collected then decomposed and matched to resources in the mirror. In various human-thing interactions, devices only need to deal with the simple Inputs/Outputs, which ensures the compatibility and scalability. This means all kinds of devices can be described and utilized, and also grouped to realize service logics either simple or complex. More specifically, for a comprehensive device or a set of devices that collaborate with one another, what is needed to describe them is no more than a set of mutually exclusive Thing-Properties; what is needed to interact with them is nothing but a set of extremely simple Operation-Inputs/Outputs.

C. McSugar

Technique McSugar in this middleware integrates MQTT and CoAP to provide human-thing interactions with a framework. We do this because both MQTT and CoAP are excellent IoT protocols, but CoAP is suitable for low-power large-scale sensing while MQTT fits real-time command transmission and execution better [46]. As shown in Figure 7, the McSugar clients of the field part and user part communicate in P2P mode via the McSugar Broker in cloud part. So users, physical world and even service logic gain the equal status, promising many potentials for human-thing interactions [47], [48].

Table 2 shows the McSugar methods for human-thing interactions based on CoAP and MQTT. User's intention to influence the field can be achieved by one-off or continual control via McSugar POST; monitoring information of the physical environment of the field can be obtained via user's McSugar GET request. Moreover, decisions made through cloud computing according to service logic can also be transmitted to the field or user through McSugar GET or POST. Note that some interactions, such as McSugar GET continual, have two options (MQTT or CoAP), so developers can choose according to the requirements. When MQTT is selected, McSugar's RESTful API is broken down into username, password, topic, and other information required for the MQTT subscribe-publish model; when CoAP is selected, the RESTful APIs directly drive the communication of CoAP request-response model.

McSugar RESTful APIs refer to the OpenAPI for oneM2M proposed by [49] which has already been applied widely. Its structure is baseUrl/resourceURI, where the structure of the resource URI is username/groupId/componentId. A resource, which can be a group of devices working together or a function of a complex device, can be identified by it. BaseUrl can be Localhost, BaseUrlOfGateway, or BaseUrlOfServer, which respectively represents directly operating smart objects, accessing devices within the LAN (Local Area Network) or accessing remotely. This technique allows authorized users and even another oneM2M system to utilize the resources of the current system. Meanwhile, McSugar supports filtering rules, which is compatible with that of MQTT [50]. In resource URI, "/" is used to separate resource levels; "+" is used to match all the resources in a single layer; "#" is used to match all the resources

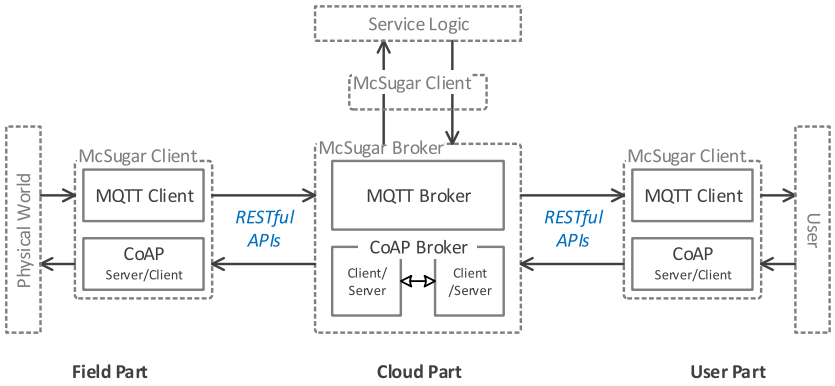

FIGURE 7. Framework for human-thing interaction.

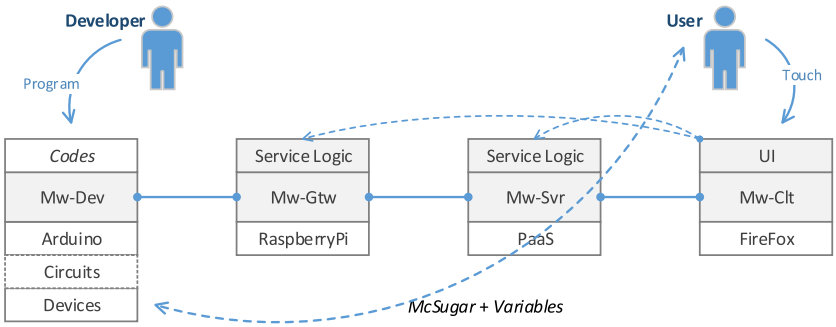

FIGURE 8. Human-thing interaction: McSugar + Variables.

in multiple layers. Combined with technique Variable, its scalability, reusability and flexibility can be fully maximized.

D. HUMAN-THING INTERACTION

Based on this middleware, human-thing interaction is fairly simple to realize. Here, we use an example to explain how the techniques Variable + McSugar addresses the 7th goal set in Section 3. As shown in Figure 8, developers just need to program on the Mw-Dev to drive devices, where devices are described into elementary resources according to Variable; and users can define the relations of the available device resources by touching the client UI, then obtain graphic interfaces for the human-thing interactions.

Assume that we want to control indoor temperature and humidity intelligently. According to Variable, if open state is marked as 1 and close state is 0, then the humidifier has only one Thing-Property and can be described as in Figure 9a; if the DHT11 sensor can measure humidity between 20 and 90% with a resolution of 1% and temperature between 0 and 50 °C with a resolution of 1 °C, then the device has two Thing-Properties, so it can be described as in Figure 9b. Once combined, the two devices can be described as shown in Figure 9c. The Operation-Input Variable of Humidifier is shown in Figure 9d; the Operation-Output Variable of DHT11 is as shown in Figure 9e. When a user wants the sensing data of DHT11, all data to be delivered are just instances of Figure 9e, which takes up just $1 + 1 + 4 + 1 + 4 = 11$ bytes only, thanks to the device resource mirror of Figure 9b. The three "1" in the equation are bytes used by the OIDs (mentioned in Section 4.2) of "dht11", "hmdt" and "tmpr"; the two "4" are bytes used by the float values of humidity and temperature sensed by DHT11. This is how human-thing interactions of high-efficiency and low-latency achieved.

Via McSugar, RESTful APIs for basic human-thing interactions can be achieved as in Table 3: ① and ② are the interaction APIs that when available device resources (Humidifier and DHT11) are discovered, and their Variables are sent to server to build resource mirror for service logics; ③ is the API when user creates the group indoorEnv for Humidifier and DHT11 to realize the automatic control mentioned in Figure 6; ④ is the API that user acquires all sensing data in group indoorEnv, and ⑤ is another approach; ⑥ is the API when user just acquires the sensing data of DHT11; and ⑦ is the API that user controls Humidifier or that server autonomously controls Humidifier when automatic control rules are set and triggered. Obviously, by combining the brief URIs and GET/POST methods, comprehensive interactions can be very simple even when it's a virtual "thing" where a group of things are aggregated. This is how the flexibility is achieved.

By the way, the human-readable Ids here are used only for understanding while the OIDs are actually employed.

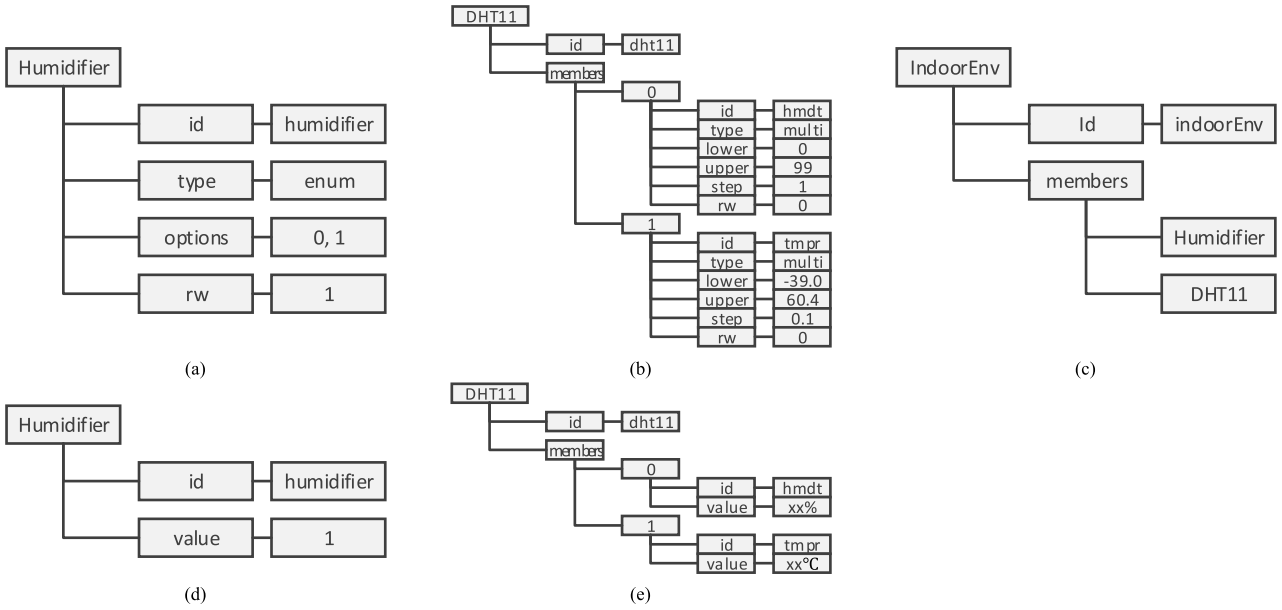

FIGURE 9. Examples of variables.

TABLE 3. Example APIs of human-thing interactions.

	McSugar Method	Base URL	Variable URI	Variable Entity
①	POST		indoorEnv/humidifier	Figure 9a
②	POST		indoorEnv/dht11	Figure 9b
③	POST		indoorEnv	Figure 9c
④	GET	baseUrlOfServer/username/	indoorEnv	null
⑤	GET		indoorEnv/+	null
⑥	GET		indoorEnv/dht11	null
⑦	POST		indoorEnv/humidifier	Figure 9d

And the structure charts of Variable instances can be described by JSON, which is widely adopted in IoT applications.

Like the McSugar, interaction methods of instances mentioned in Section 2 are also RESTful, but they do not have the same conciseness shown by the Variable URI and Entity columns of Table 3. So the unique advantages of our solution are (1) further steps in the homogenization of the heterogeneous devices for comprehensive services; (2) compatibility in multiple protocols including HTTP, CoAP and MQTT, and uniformity in APIs at the meantime; and (3) most importantly, compliance with the global standard oneM2M.

V. CASE STUDY

With this middleware, IoT development can be simplified into the following key steps:

(S1) Flash the middleware's Mw-Dev component to non-IoT devices; as for IoT devices, the operation is similar to (S2).

(S2) Install Mw-Gtw to gateway board if there is a gateway.

(S3) Deploy Mw-Svr on the PaaS platform you've bought, or on a local server.

(S4) Access the target server using your browser to obtain human-thing interactions.

(S5) Login the remote administration page, then define and enable advanced service logics.

Its take very little efforts to do (S2)-(S4) as only the registration information is needed. Here are two case studies in IoT development and research to explain these five steps and demonstrate the advantages of this middleware.

A. BOOSTING IoT DEVELOPMENT

Since temperature, humidity, light, and CO_2 concentration are key factors that influence the quality of flower cultivation, this IoT application aims at overcoming farmers' subjective judgment and enabling intelligent precise control of their greenhouse environment [51], [52].

The user part of this IoT application uses the original version of this middleware's webapp client. The cloud part is deployed on a local workstation as this is only for private use; dynamic domain name service (DDNS) is introduced to enable remote access. The field part includes a gateway and WSAN, where sensors (temperature, humidity, CO_2 concentration, illumination) are added and existing equipment (heating, sprinklers, curtains, lights, ventilation) are transformed then organized, as shown in Figure 10, Figure 11 and Table 4.

In terms of implementation, this middleware can highly simplify it. (S1) For the devices in the field, connect

TABLE 4. Details of field devices.

Devices	DHT11 × 6	GY-30 × 6	DS-CO2-20 × 6	Heating Pipes	Sprinklers	Roof Curtains	Lights	Ventilation
Output / Input	20 - 90 ± 5% 0 - 50 ± 2 °C	0 - 65535 lux	0 - 5000 ± 50 ppm	10 levels	On-Off	On-Off	On-Off	On-Off
Main Factors	Temperature, Humidity	Illumination	CO_2 Concentration	Temperature	Humidity	Illumination	Illumination	Concentration, Humidity

FIGURE 10. Equipment layout inside the flower cultivation base. Existing equipment: sprinklers, heating pipes, ventilation, top curtains and lights. Newly-add equipment: DHT11s, DS-CO2-20s and GY-30s.

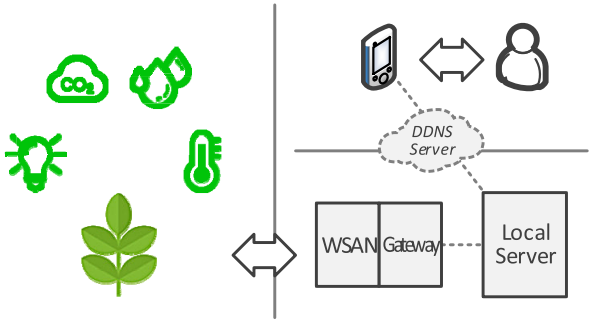

FIGURE 11. Application structure: sensors/actuators are organized by WSAN and gateway; then the field conditions can be monitored/ controlled remotely via local server with the help of a DDNS server.

legacy equipment (via proper transition circuits) and new devices (directly) to Arduino-ZigBee boards; code devices' init/read/write functions according to Variable, which is quite easy as demos are provided by their retailers. The following is the pseudo code for DHT11. When it initializes, init()describes DHT11 as two components hmdt and tmpr (see code#1 and Figure 9b); and submit it to the gateway (code#2), so as to build resource mirrors for human-thing interactions. When user requests the sensing data, all components of DHT11 can be located by McSugar API plus identifiers dht11-1/hmdt or dht11-1/tmpr; then read() packs data into "{dht11-1; {hmdt; value1 }; {tmpr; value2 } }" and upload them (see code#3 and Figure 9e). The write() is null for sensors.

```
function init() {
  str = "{
    id:'dht11-1'; type:'multi'; rw:0;
    members:{{id:'hmdt'; lower:0;
                 upper:99; step:1; },
      {id:'tmpr'; lower:-39; upper:60;
        step:0.1 }
    };
  }";                                  // c#1
  set_DHT11_resource_mirror(str);
  send_resource_mirror_via_RF_to_gateway
  (str);                               // c#2
  configure_hardware_connection_with_MCU();
}
function read() {
  data = scan_MCU_pins_for_sensing_data();
  send_sensing_data_via_RF_to_gateway(data);
                                       // c#3
}
```

Compile the coded Mw-Dev and flash it into the boards. Then forms the WSAN, which is connected to the gateway via a serial port. (S2) For the gateway, install Mw-Gtw directly to Raspberry Pi board, and do necessary configurations like username/password and target domain name to connect to the "cloud" server. (S3) For the "cloud" server, just install Mw-Svr to the local workstation; then set up the DDNS service we bought to map the local server to a public domain name to realize remote access, which is also quite simple. We do so not because the middleware doesn't support real cloud server, but for the sake of cost. (S4) For the client, visit that domain name using browser then register using the same username/password, then connections with the field are built via the cloud; after that, device resource mirrors are submitted automatically to the server by the gateway; and basic human-thing interactions powered by McSugar are thus obtained by the client, such as remote real-time monitor/control and visualized analysis of historic sensing data. (S5) For advanced services, log in the remote management page, then define service logics using simple text over the APIs. Here, based on Variable, devices are virtualized into four groups: (1) temperature, DHT11 × 6 + Heating Pipes; (2) humidity, DHT11 × 6 + Sprinklers + Ventilation; (3) illumination, GY-30 × 6 + Roof Curtains + Lights; and (4) CO_2 concentration, DS-CO2-20 × 6 + Ventilation. The rules for the system to intelligently control actuators to take actions according to sensors' sensing data are also set.

In brief, developers just need to make circuits, code functions, flash firmware/install software, input user information, and define advanced services; no need to deal with complex underlying technologies of support/communication/computing. This middleware is of course not an enclosed box, and when required, in-depth customization or optimization can be realized easily thanks to its modular and loosely coupled structure.

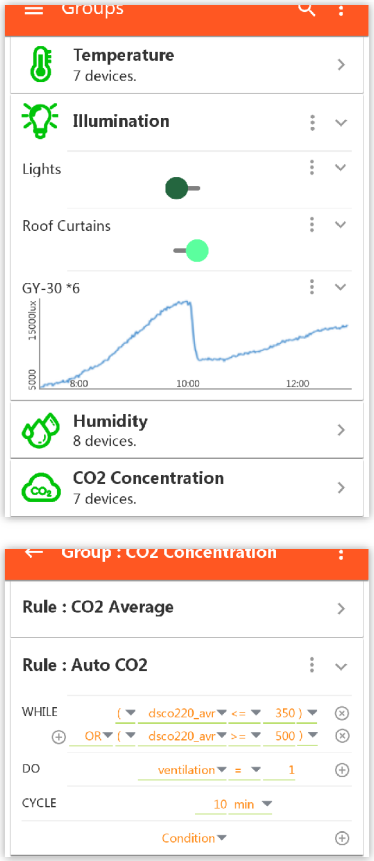

FIGURE 12. Client GUI: monitor/control (upper); adding rules (lower).

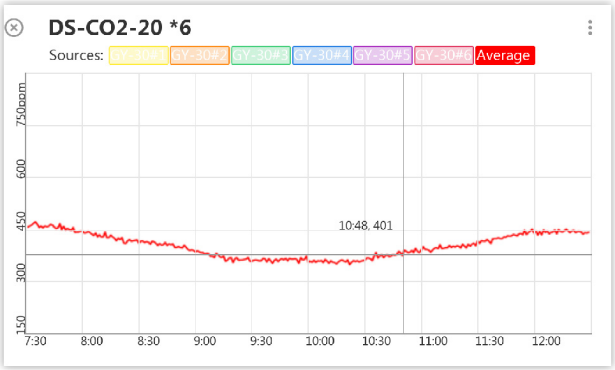

FIGURE 13. Curve of CO2 concentration with auto-control rules.

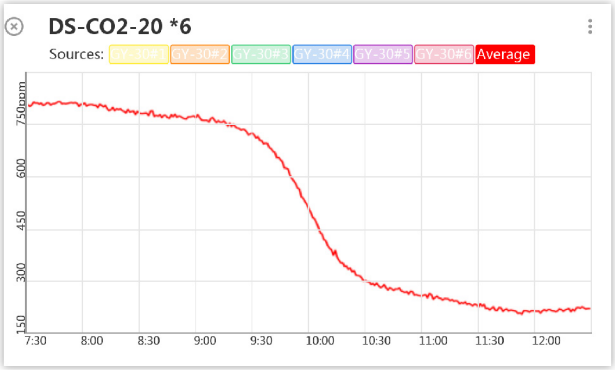

FIGURE 14. Curve of CO2 concentration without auto-control rules.

With the client, farmers can browse temperature, humidity, illumination, and CO_2 concentration of their greenhouse at anytime from anywhere. Farmers can also visually analyze the influence of these factors on the flowers. The intuitional charts can adaptively visualize the sensing data based on Thing-Properties, and allow adjustment in display styles, such as the average of several data sources and the scope of coordinate axes. In addition, users can tap the GUI views such as "switch", "progress bar" and "text box", which are matched to corresponding device resources according to their Variables, to control the greenhouse environment, as shown in Figure 12a. With user-defined rules, the system can automatically control the actuators and alert users about emergencies. For example, in rule "Auto CO2" shown in Figure 12b, when dsco220_avr (the average value of the six DS-CO2-20s, obtained via their Variables, created in rule "CO2 Average") is less than 350 or greater than 500, ventilation will be given value "1" to turn on the ventilation facilities in the field to adjust CO_2 concentration autonomously.

Maintaining a suitable CO_2 concentration is vital to the cultivation of flower quality. Figure 13 and 14 present the tendencies of CO_2 concentration over a period of two similar days, with and without auto-control. The values of CO_2 concentration are the average of the sensing data of six DS-CO2-20s. The auto-control rules are to ventilate if CO_2 concentration is not in the range of 300-475 ppm. As it shows, the concentration remained within the favorable range all the time with the auto-control rules, while the concentration varied greatly due to respiration and photosynthesis without the rules. The former clearly has an advantage in flower cultivation.

B. FACILITATING IoT RESEARCH

This middleware can be used as a testbed to put IoT research simulations into practice. The following is an example of the probe of distributed computing in elderly care.

There are many diverse devices in a nursing scenario where the service should be persistent and the emergency response should be as swift as possible [53], [54]. In a conventional cloud-centered architecture, sensing data have to be uploaded to the far cloud and congestion occurs unpredictably. It is time/resource consuming, unable to respond to emergencies timely and constantly. Distributed computing is an ideal solution: local resources are utilized, and service logics are handled nearer to the field. Assume that monitoring equipment in each room is organized by a gateway and connected to a cloud server. The gateways and server complete tasks together: the central node (a gateway) distributes computing tasks to "adjacent" nodes (other gateways or the server); each node calculates the assigned task and returns the result to the

TABLE 5. Specifications of the nodes.

Nodes	Hardware Platform	CPU	RAM	OS
Gateway (×5)	Raspberry Pi 3B+	BCM2837 Cortex-A53 ARM 1.2 GHz 64 bit	1 GB	Raspbian Desktop
Cloud Server	ThinkStation P310	Intel Core i7-6700 X86 3.4 GHz 64 bit	16 GB	Windows Server

central node; the result is very lite, so only its transmission delay is considered. If the total amount of task is A, and the computing capacity of node n c_n, communication bandwidth of the center node b_n and communication delay with the central node d_n are all known, then the task proportion done by node n is

$$p_n = \frac{a_n}{\Sigma a_i}, \quad \text{where } a_n = xc_n + yb_n + z\frac{1}{d_n} \tag{a}$$

The time node n receives its tasks, the time node n completes its tasks, and the latency node n receives/returns its task data respectively are

$$t_{bn} = \frac{Ap_n}{b_n}K_b, \quad t_{cn} = \frac{Ap_n}{c_n}K_c, \; t_{dn} = 2d_n \tag{b}$$

The aim of the collaboration is to minimize the time consumption of distributed computing:

$$\max_n t_n \rightarrow min, \quad \text{where } t_n = t_{bn} + t_{cn} + t_{dn} \tag{c}$$

The system model is shown in Figure 15, which includes the server in the cloud part and several gateways that drive some sensors and actuators in field part. Each gateway distributes sensing data among the gateways and server to complete the computing task together. For simplicity, just consider the condition of Gw0. For Gw0, the server and other gateways are all peer nodes that can be utilized. As shown in Figure 16, all these nodes logically comprise three layers: service logic, neuron and sensing layers. (1) Sensing layer collects sensing data from sensors (just consider that of Gw0). (2) Neuron layer integrates all nodes together and updates communication parameters among nodes and their free resources (c_n, b_n, d_n) and distribute tasks according to Equation (a) via McSugar. Jubatus in this layer is a powerful online machine learning tool, serving to update those parameters. (3) Service logic layer executes distributed tasks, based on resource mirrors powered by Variables.

In terms of implementation, steps (S1-S4) are similar to the former case, easy as always. For advanced tasks in (S5), Jubatus need to be introduced in. The data needed by Jubatus's online learning and dynamic updating are all described by the extremely lightweight Variables, saving a lot of communication resources. Thanks to the OSGi feature of the middleware, which makes the middleware itself highly modularized and loosely coupled, the introduce of Jubatus is just as easy as playing building blocks even though Jubatus is programmed in C++: we encapsulate it in Java and invoke it using JNI (Java Native Interface), making it a plug-and-play module running on these nodes.

In short, by conducting experiments with this middleware, researchers can set up the testbed quickly and concentrate

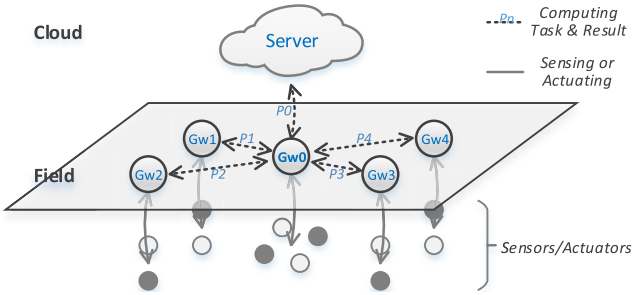

FIGURE 15. Model of the experiment. Consider Gw0 only: cloud server and other Gws (gateways) in the field are nodes can be utilized; Gw0 collects and distributes sensing data to other nodes to conduct distributed computing, then gathers computing results to finish it.

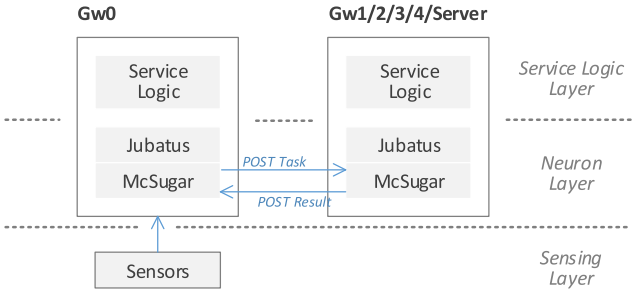

FIGURE 16. Three layers of the experiment. Sensing layer of Gw0 collects data; neuron layers of all nodes update factors needed by distributed computing; service logic layers complete tasks allocated to the nodes. Gw1-4 and server are nodes that can be utilized by Gw0.

on their research; even introducing new modules into it is no challenge.

To facilitate the comparison and analysis, the following constraints are defined for the experiment: (1) The computing task is to predict future value according to 20 DHT11s' historical sensing data, which is of linear complexity. (2) The task volume can be set by the sampling frequency, which ranges from 5 to 100 Hz. (3) The nodes are five Raspberry Pi gateways and a ThinkStation P310 server, as shown in Table 5, in the same WLAN. (4) The Raspberry Pis have had their OSs re-installed and unrelated processes killed. (5) The ThinkStation has had its OS re-installed and unrelated processes killed; latency of 20 ms is added to IP packets to simulate the communication delay of a commercial cloud platform.

Figure 17 shows the proportion of tasks undertaken by each node under different sampling frequencies when tasks are computed in field-cloud mode. Figure 18 shows the time required by modes cloud-only, field-only and field-cloud to complete a task at different sampling frequencies. According to Figure 17, the computing task is always distributed

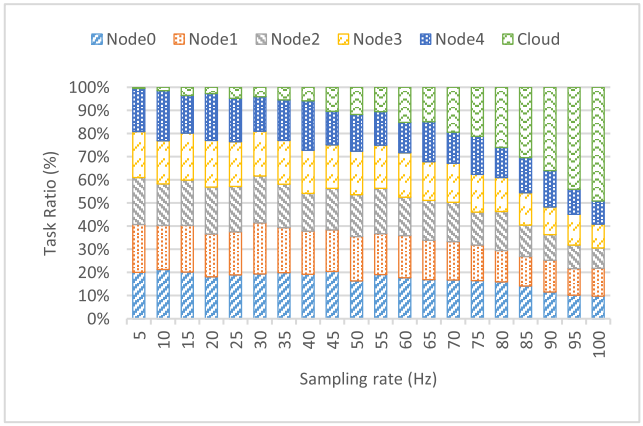

FIGURE 17. Task ratio allocated to all nodes.

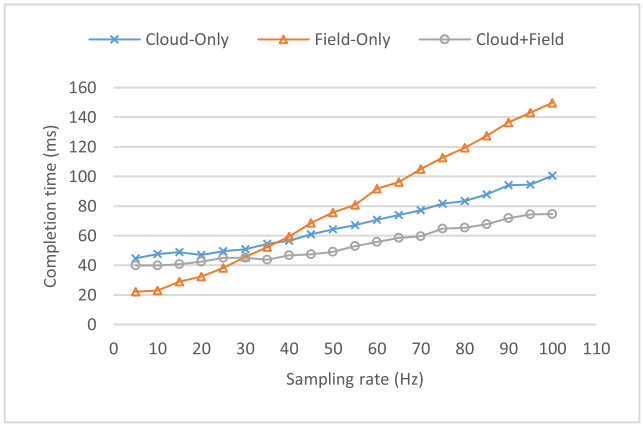

FIGURE 18. Completion time of three modes.

evenly among the field nodes. When the sampling rate is low, the completion of computing tasks depends mainly on the cooperation of field nodes. However, as task quantity increases, cloud nodes play an increasingly important role due to its strong performance. According to Figure 18, when the sampling rate is relatively small, mode field-only can complete tasks the fastest, while cloud-only has poor instantaneity performance due to its inherent delay. With the increase of sampling rate, mode field-cloud becomes the best solution. However, according to our expectations, when the sampling rate is low, the performance of field-cloud should be at least as good as field-only. Obviously, the assumption that a_n has linear relationships with c_n, b_n and d_n^{-1} is not fully reasonable.

C. DISCUSSION

As can be seen from these cases, the overall goals set in Section 3 are achieved in this middleware.

With this middleware, developers basically don't have to handle the underlying technologies in the common basic development and thus can build up their IoT application quickly. The APIs and tools for visualized analysis work well in the advanced development. The cases of both smart floriculture and field-cloud computing function smoothly in tests, which proves the middleware's maintainability and reliability. The strengths of this modular and decoupled stack are demonstrated either when Jubatus is introduced in.

The flexible, uniform and effective human-thing interactions are enabled by techniques Variable and McSugar. In the first case, the cooperation of McSugar and Variable is the key to real-time monitor/control, historic data visualization and other services; in the second case, device resource mirror based on Variable is the basis of distributed computing.

The potentials like semantic reasoning due to its compliance with oneM2M are not fully presented though, and thus further investigation is required. Although not directly illustrated by these cases, the middleware's adaptability to various software/hardware platforms is fine according to responses from IoT communities.

VI. CONCLUSIONS

In this paper, we propose a oneM2M-compliant stacked middleware for IoT development and research. Its innovative stack of support-communication-computing integrates those excellent open source software and hardware into a comprehensive entity, where those tricky common basic tasks in IoT development are fulfilled and APIs and tools are provided for advanced tasks. Particularly, its Variable and McSugar techniques enable the uniform and flexible interactions between human and things, maximizing its value in practical applications. As the tests demonstrate, with this middleware, IoT developers can focus on top-level requirements instead of being trapped in underlying technological details; researchers can build up testbeds quickly and conduct experiments for novel IoT technologies. Furthermore, this middleware complies with the global standard oneM2M and thus inherits its great potentials. Foundations for those cutting-edge IoT technologies like fog/cloud computing and semantic reasoning are established, which still require much research and development though. Our following goals are contextual resource integration and semantic reasoning guided by oneM2M ontology, so as to foster more intelligent and practical IoT applications.

ACKNOWLEDGMENT

Technological support from Fei Qin and his startup company Glrsmart Tech. Jiangsu Co., LTD is gratefully acknowledged. Fellow apprentice Hui Li helped a lot in this paper's literal and logic expression.

REFERENCES

[1] W. Ejaz, M. Naeem, A. Shahid, A. Anpalagan, and M. Jo, "Efficient energy management for the Internet of Things in smart cities," *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 84–91, Jan. 2017.

[2] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *J. Cleaner Prod.*, vol. 140, pp. 1454–1464, Jan. 2017.

[3] M. Chen, Y. Ma, Y. Li, D. Wu, Y. Zhang, and C. -H. Youn, "Wearable 2.0: Enabling human-cloud integration in next generation healthcare systems," *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 54–61, Jan. 2017.

[4] H. M. Jawad, R. Nordin, S. K. Gharghan, A. M. Jawad, and M. Ismail, "Energy-efficient wireless sensor networks for precision agriculture: A review," *Sensors*, vol. 17, no. 8, pp. 1781–1825, 2017.

[5] M. Wollschlaeger, T. Sauter, J. Jasperneite, "The future of industrial communication: Automation networks in the era of the Internet of Things and industry 4.0," *IEEE Ind. Electron. Mag.*, vol. 11, no. 1, pp. 17–27, Mar. 2017.
[6] *TS-0001-Functional_Architecture-V2_10_0.pdf*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.onem2m.org/images/files/deliverables/Release2/TS-0001-%20Functional_ArchitectureV2_10_0.pdf
[7] oneM2M. *oneM2M: Standards for M2M and the Internet of Things*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.onem2m.org
[8] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Comput. Netw.*, vol. 130, pp. 94–120, Jan. 2018.
[9] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.
[10] E. Kovacs, M. Bauer, J. Kim, J. Yun, F. Le Gall, and M. Zhao, "Standards-based worldwide semantic interoperability for IoT," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 40–46, Dec. 2016.
[11] S. Mayer, J. Hodges, D. Yu, M. Kritzler, and F. Michahelles, "An open semantic framework for the industrial Internet of Things," *IEEE Intell. Syst.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2017.
[12] 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.
[13] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.
[14] V. Gazis, "A survey of standards for machine-to-machine and the Internet of Things," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 482–511, 1st Quart., 2017.
[15] I. C. L. Ng and S. Y. L. Wakenshaw, "The Internet-of-Things: Review and research directions," *Int. J. Res. Marketing*, vol. 34, no. 1, pp. 3–21, 2017.
[16] *EVRYTHNG IoT Smart Products Platform*. Accessed: Mar. 28, 2018. [Online]. Available: http://evrythng.com
[17] *IFTTT Helps Your Apps and Devices Work Together*. Accessed: Mar. 28, 2018. [Online]. Available: https://ifttt.com
[18] *SensorCloud*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.sensorcloud.com
[19] *IoT Analytics-ThingSpeak Internet of Things*. Accessed: Mar. 28, 2018. [Online]. Available: https://thingspeak.com
[20] *ThingWorx Delivers Industrial Innovation*. Accessed: Mar. 28, 2018. [Online]. Available: https://www.thingworx.com
[21] K. Yelamarthi, S. Aman, and A. Abdelgawad, "An application-driven modular IoT architecture," *Wireless Commun. Mobile Comput.*, vol. 2017, May 2017, Art. no. 1350929. [Online]. Available: https://www.hindawi.com/journals/wcmc/2017/1350929
[22] S. Jeon and I. Jung, "MinT: Middleware for cooperative interaction of things," *Sensors*, vol. 17, no. 6, pp. 1452–1476, 2017.
[23] T. Popović, N. Latinović, A. Pešić, Ž. Zečević, B. Krstajić, S. Djukanović, "Architecting an IoT-enabled platform for precision agriculture and ecological monitoring: A case study," *Comput. Electron. Agricult.*, vol. 140, pp. 255–265, Aug. 2017.
[24] G. Lilis and M. Kayal, "A secure and distributed message oriented middleware for smart building applications," *Automat. Construct.*, vol. 86, pp. 163–175, Feb. 2018.
[25] B. Hafidh, H. Al Osman, J. S. Arteaga-Falconi, H. Dong, and A. El Saddik, "SITE: The simple Internet of Things enabler for smart homes," *IEEE Access*, vol. 5, pp. 2034–2049, 2017.
[26] U. V. Ramana and T. V. Prabhakar, "Some experiments with the performance of LAMP architecture," in *Proc. 5th Int. Conf. Comput. Inf. Technol.*, Shanghai, China, Sep. 2005, pp. 916–920.
[27] K. J. Singh and D. S. Kapoor, "Create your own Internet of Things: A survey of IoT platforms," *IEEE Consum. Electron. Mag.*, vol 6, no. 2, pp. 57–68, Apr. 2017.
[28] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, 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.
[29] 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.
[30] 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.
[31] G. Barbon, M. Margolis, F. Palumbo, F. Raimondi, and N. Weldin, "Taking Arduino to the Internet of Things: The ASIP programming model," *Comput. Commun.*, vols. 89–90, pp. 128–140, Sep. 2017.
[32] *Contiki: The Open Source OS for the Internet of Things*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.contiki-os.org
[33] *Eclipse Paho-MQTT and MQTT-SN Software*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.eclipse.org/paho
[34] *Eclipse Kura-Open Source Framework for IoT*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.eclipse.org/kura
[35] *Californium (Cf) CoAP Framework*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.eclipse.org/californium
[36] *Apache Camel*. Accessed: Mar. 28, 2018. [Online]. Available: http://camel.apache.org
[37] *Eclipse OM2M-Open Source Platform for M2M Communication*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.eclipse.org/om2m
[38] *Eclipse Mosquitto*. Accessed: Mar. 28, 2018. [Online]. Available: https://mosquitto.org
[39] *Eclipse Hono*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.eclipse.org/hono
[40] *Eclipse Ditto*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.eclipse.org/ditto
[41] *Eclipse hawkBit-IoT Software Update*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.eclipse.org/hawkbit
[42] BIRT. *BIRT Buzz*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.eclipse.org/birt
[43] *oneM2M Technical Specification TS-0012-V2.0.0 Base Ontology*. Accessed: Mar. 28, 2018. [Online]. Available: http://www.onem2m.org/images/files/deliverables/Release2/TS-0012-oneM2M-Base-Ontology-V2_0_0.zip
[44] M. Tao, J. Zuo, Z. Liu, A. Castiglione, and A. Castiglione, "Multi-layer cloud architectural model and ontology-based security service framework for IoT-based smart homes," *Future Gener. Comput. Syst.*, vol. 78, pp. 1040–1051, Jan. 2017.
[45] A. Markus, G. Kecskemeti, and A. Kertesz, "Flexible representation of IoT sensors for cloud simulators," in *Proc. 25th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, St. Petersburg, Russia, Mar. 2017, pp. 199–203.
[46] C. Pereira and A. Aguiar, "Towards efficient mobile M2M communications: Survey and open challenges," *Sensors*, vol. 14, no. 10, pp. 19582–19608, 2014.
[47] F. Khan, I. ur Rahman, M. Khan, N. Iqbal, and M. Alam, "CoAP-based request-response interaction model for the Internet of Things," in *Proc. Int. Conf. Future Intelligent Veh. Technol.*, Porto, Portugal, Sep. 2016, pp. 146–156.
[48] A. Khakimov, A. Muthanna, R. Kirichek, A. Koucheryavy, and M. S. Ali Muthanna, "Investigation of methods for remote control IoT-devices based on cloud platforms and different interaction protocols," in *Proc. IEEE Russia Section Young Researchers Elect. Electron. Eng. Conf.*, St. Petersburg, Russia, Feb. 2017, pp. 160–163.
[49] J. Kim, S.-C. Choi, I.-Y. Ahn, N.-M. Sung, and J. Yun, "From WSN towards WoT: Open API scheme based on oneM2M platforms," *Sensors*, vol. 16, no. 10, p. 1645, 2016.
[50] *MQTT Version 3.1.1*. Accessed: Mar. 28, 2018. [Online]. Available: http://docs.oasisopen.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
[51] Monteiro, J.A.; Nell, T.A.; Barrett, J.E., "Postproduction of potted miniature rose: Flower respiration and single flower longevity," *J. Amer. Soc. Horticultural Sci.*, vol. 126, no. 1, pp. 134–139, 2001.
[52] R. Şenol and K. Taşdelen, "A new approach for LED plant growth units," *Acta Polytechnica Hungarica*, vol. 11, no. 6, pp. 57–71, 2014.
[53] S. M. M. Fattah, N.-M. Sung, I.-Y. Ahn, and M. R. J. Yun, "Building IoT services for aging in place using standard-based IoT platforms and heterogeneous IoT products," *Sensors*, vol. 17, no. 10, p. 2311, 2017.
[54] J. Sun, Y. Guo, X. Wang, and Q. Zeng, "mHealth For aging China: Opportunities and challenges," *Aging Disease*, vol. 7, no. 1, pp. 53–67, 2016.

RONGZHEN ZHAO received the master's degree from Nankai University, Tianjin, China. He is currently with the Tianjin Key Laboratory of Optoelectronic Sensor and Sensing Network Technology, Nankai University.

His current research interests include Internet of Things and deep learning.

LITIAN WANG is currently pursuing the Ph.D. degree in electronic science and technology with Nankai University, Tianjin, China. His main research interests include Internet of Things, microwave passive components and systems, and HTS tunable filter design.

XINGZHE ZHANG is currently pursuing the master's degree with the College of Electronic Information and Optical Engineering, Nankai University, Tianjin, China. His current research interest is Internet of Things.

YU ZHANG is currently pursuing the master's degree with the College of Electronic Information and Optical Engineering, Nankai University, Tianjin, China. Her current research interest is Internet of Things.

LIZHI WANG is currently pursuing the bachelor's degree with the College of Computer and Control Engineering, Nankai University, Tianjin, China. His current research interest is Internet of Things.

HONGZHAO PENG is currently pursuing the master's degree with the College of Electronic Information and Optical Engineering, Nankai University, Tianjin, China. His current research interest is Internet of Things.