The International Journal of Advanced Manufacturing Technology
https://doi.org/10.1007/s00170-019-04310-z

ORIGINAL ARTICLE

Research on the overall architecture of Internet of Things middleware for intelligent industrial parks

Li Zhang[1] · Huiqun Yuan[2] · Sheng-Hung Chang[2] · Anthony Lam[3]

Received: 21 June 2019 / Accepted: 14 August 2019

Abstract
With the continuous development of the Internet of Things technology and the proposal of "Made in China 2025", the construction of IoT application models for intelligent production lines, intelligent workshops, smart factories and other manufacturing industries has attracted more and more attention. In the smart industry, there are many types of devices connected to the Internet of Things, and the data format of the devices is not uniform. Therefore, when the upper layer application collects device data and manages the devices, it is necessary to introduce middleware to solve these problems. A service-oriented IoT middleware model is designed. This middleware is built on the background of the smart campus with the above characteristics. The middleware is responsible for realizing the information interaction between the front-end sensing device and the upper-layer application, sending the commands of the upper-layer application to the device to implement real-time management of the device and providing a heterogeneous intelligent device for accessing the Internet of Things application environment, an effective solution. And through an application example, the working principle and implementation of this middleware are further illustrated.

Keywords Internet of Things · Middleware · Service oriented · Architecture

1 Introduction

On the basis of communication networks such as the Internet and mobile communication networks, the Internet of Things uses intelligent terminals with sensing, communication and computing capabilities to automatically acquire various information in the physical world for all application requirements and to enable all independently addressable physics. The objects are interconnected to realize comprehensive sensing, reliable transmission and intelligent processing and construct an intelligent information service system in which people and things and things and things are interconnected [1]. The Internet of Things is composed of various information sensing devices and systems such as simple two-dimensional bar code, radio frequency identification technology, wireless sensor network, CPS (cyber-physical systems) [2] and M2M (machine-to-machine) system [3]. An intelligent network was formed by a combination of core technologies such as ubiquitous networks and next-generation Internet. The Internet of Things includes three layers of perceptual extension layer, network layer and service and application layer [4]. The first layer is responsible for collecting information related to objects and objects; the second layer is a heterogeneous communication ubiquitous communication network; and the third layer is applications and services, providing application services for sensing information for various terminal devices such as mobile phones and PCs [5] Providing service-related services is an inherent requirement of the Internet of Things. However, the number of physical objects is large, various and heterogeneous, and virtual objects have complex characteristics of information correlation. Therefore, it is necessary to establish a universal service platform to achieve effective management, interaction and processing of objects and to ensure the provision of object-related services [6]. So, the Internet of Things middleware was created.

✉ Huiqun Yuan
yuanhuiqun@hbue.edu.cn

[1] School of Art and Design, Hubei University of Economics, Wuhan 430205, China

[2] School of Business Administration, Hubei University of Economics, Wuhan 430205, China

[3] Faculty of Economics and Business, KU Leuven, 3000 Leuven, Belgium

Published online: 14 September 2019

2 Introduction to relevant theory

2.1 Internet of Things

The Internet of Things (IoT) is an intelligent network formed by combining various information sensing devices such as RFID technology, sensor technology, wireless sensor network, CPS and M2M system with the first-class key technologies of ubiquitous networks and next-generation Internet. The overall structure consists of three layers, namely a perceptual extension layer composed of objects, a network layer composed of various heterogeneous converged communication networks and an application service layer deployed on different devices to provide different application services. In 1999, MIT Auto ID Centre first of all, the concept of "Internet of Things" was put forward: on the basis of computer interconnection network, using RFID, wireless data communication and other technologies to establish a network that integrates all objects in the world to realize automatic identification and information of all items in the network. Internet sharing is one of the four core technologies for realizing the Internet of Things: RFID, nanotechnology, sensor technology and intelligent embedded technology. In 2009, IBM proposed the "Smart Earth". The concept has since aroused the Internet of Things boom in the world; many countries will develop the Internet of Things. Network technology is included in the major information development strategy.

2.2 Middleware concept and classification

Middleware is not a specific piece of software, but a series of software that provides basic services between the operating system and the application system. The original intention of middleware is to shield various complex technical details, simplify technical problems and reduce the complexity of programming. Application developers do not need to consider the migration of software programs in different systems, reducing the workload and shortening. The development cycle and the work of maintaining and managing the operating system are reduced. Based on the role of middleware in the system and the technology used, we divide the middleware into the following categories:

1. Based on event middleware

   Event-based middleware, whose components, applications and all other participants, communicates through events. Different events have different types and parameters, and changes in parameter values reflect changes in event generators. The driver for event-based middleware must consist of two parts: the event generator and the event consumer. The event generator generates an event and the event consumer subscribes to the event. In an event middleware system, there may be many application components or entities that generate and consume events. The publish/subscribe mode can provide one-to-many and many-to-many asynchronous communication modes at the same time, which can completely solve the problem in time and space, and well meets the requirements of large-scale distributed applications. This design approach addresses non-functional requirements such as reliability, availability, real-time performance, scalability and security.
2. Service-oriented middleware

   Service-oriented middleware (SOM) is designed based on service oriented architecture (SOA). SOM is characterized by technology neutrality, loose service coordination, high reusability and discoverability of services, all of which are in line with the actual needs of IoT applications. In the SOA architecture, the underlying details are transparent to the upper-level services, and the interfaces between the services are an important part of the software design flow. When the upper service requires the implementation of some complex services, the basic services can be split and combined without redevelopment. Service-oriented middleware can serve as a foundational platform for providing service (PaaS) models for many business-specific middleware, applications or computing platforms.
3. Based on agent middleware

   In agent-based middleware, applications are divided into modular programs to facilitate the injection and utilization of mobile agents' network distribution. While migrating from one node to another, the agent maintains their execution state. This helps the dispersion to tolerate partial system design failures. In addition, the agent middleware can talk to other software, and the agent actively collects data and updates some of the applications. At the same time, resource-constrained issues should be considered in the development of agent-based middleware.
4. Message-oriented middleware

   The primary feature of message-oriented middleware is looseness, a feature that provides reliable, cross-platform data transfer possibilities between processes. "Message" refers to any content customized by the user, such as text, numbers and video streams. The message-oriented middleware provides synchronous communication mode and asynchronous generalization. The asynchronous communication method is based on the principle of store and forward. The processes do not directly exchange information with each other, but communicate indirectly through the intermediary. This communication mode also plays a role in data buffering, the role of the asynchronous transmission of the message.
5. Database middleware

   Database middleware is produced in the early days of the middleware, and it has the longest development and the most mature. The database middleware is between the application and the database, providing a unified data interface to solve access problems between the application and the database.

6. Remote procedure call middleware

   The remote procedure call middleware communication mode adopts the synchronous mode. When one application A needs to exchange information remotely with another application B, a request is generated locally, and the information is received through the communication link B, and after B completes the processing, the result is returned to A. The flexibility of remote procedure call middleware makes it more versatile and can be used in more complex computing environments. However, this synchronous communication method is not suitable for large-type applications, needs to deal with network or system failures during operation, handles a series of complicated problems such as concurrent operations and process synchronization and is suitable for small applications.

2.3 Service-oriented architecture

1. Service-oriented architecture concept

Service-oriented architecture (SOA) is a component model that follows specific standards and design methodology to properly deploy, assemble and apply application components with loose, coarse-grained features. SOA shields the difference in information between the technology layer and the business layer, allowing the application layer to get rid of the negative problems caused by the different information. Communication between SOA services requires a simple, well-defined interface, regardless of the underlying programming interface and operating system.

2. Characteristics of service-oriented architecture

Service-oriented architecture is a coarse-grained, loosely coupled service architecture. Communication between services requires a simple, precisely defined interface, regardless of the underlying programming interface and operating system. This architectural model usually has the following characteristics:

Firstly, the service is loosely coupled Fig 1. The role of loose coupling is to enable the service requester to communicate with the service provider at the interface level. As for how the internal implementation of the service is independent of the service requester, the service interface and the service implementation are isolated, and the service requester does not need to know the service provision. The technical details of the implementation, such as the language and development environment, are used in SOA-based software design; if the service provider does not change the interface, only internal changes will not affect the service user.

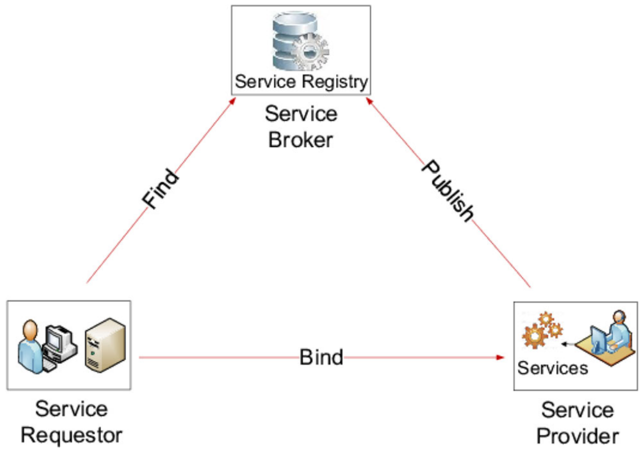

Fig. 1 The role of each part of SOA

Secondly, the coarse-grained service. In general, service granularity refers to the scope of the functions exposed by the service, which are divided into fine-grained and coarse-grained services. A coarse-grained service can provide a rich logical availability service to an application. Conversely, a fine-grained service can only provide a small or specific logical availability service. The coarse-grained service is flexible and reusable and can be formed by fine-grained services.

Thirdly, the encapsulation of the service. Service encapsulation, as its name suggests, encapsulates services to form business components that can be reused for different business processes. It can mask the technical details used to implement the service. If the service provider does not change the interface, only internal changes will not affect the service user.

Fourthly, the reusability of services. The reusability of the service means that all services are independent of each other and do not affect each other. Each service independently implements one or a function, and the service user does not change the service itself when using it, so the service can be reused, and it has nothing to do with the underlying technology and user needs and can play a role in reducing development costs.

Fifthly, interoperability between services. Services can communicate with each other, and you need to choose synchronous or asynchronous communication according to the defined protocol.

Sixthly, the location of the service is transparent. As a service consumer, there is no need to know the specific location of the service provider, and it is not necessary to know which service responded to its request, and you only need to care whether the service provider provides the required service.

3 Smart Park IoT architecture

At present, the Internet of Things is in the development stage, and the projects for the construction of smart parks are also in the process of continuous exploration and practice. Therefore, cities and researchers have different interpretations and implementations of the concept of smart city smart park construction. It is widely believed that smart parks refer to the succession of new-generation information technologies such as the Internet, mobile Internet, Internet of Things and cloud computing, providing a safe, comfortable and convenient modern, intelligent living environment for the residents of the park, and then forming information based on A new management form of the park, intelligent social management and service [7].

3.1 Demand analysis

The scenario built in this paper is a smart campus. Since different areas in the campus need to access different terminal devices, the upper-layer applications that need to be developed are also different. Therefore, the middleware needs to support the system expansion function and improve the flexibility and scalability of system development. This will facilitate the deployment of the middleware platform in the smart campus. The main function of the middleware is to acquire the data collected by the terminal device under the control of the upper-layer application, transmit the data back to the upper layer application, and then control the operation of the corresponding device according to the collected data. If the data collected by the humidity sensor exceeds a certain value, the ventilation device needs to be turned on. The information perceived by the smart campus has the characteristics of multi-source, format diversity and real-time dynamic changes. Therefore, when designing the IoT middleware, we must consider the problems of handling multiple access methods, real-time data processing and the adoption of technology platforms. So, this paper designs an Internet of Things middleware based on SOA architecture. The coarse-grained, loosely coupled, highly reusable and protocol-independent features of the SOA architecture meet the needs of IoT middleware in this scenario.

According to the analysis in the previous section, we divide the IoT middleware for the smart campus into three layers: the device driver layer, the service scheduling layer and the application business layer. The following is a detailed analysis of the functions that need to be implemented at each layer. The device driver layer needs to be analysed and researched: the transmission technology of the smart campus is different, and the terminal devices are different [8]. Therefore, the transmission protocol used and the format of the collected device data are different. Therefore, different protocols need to be parsed, the protocol adaptation layer is designed, and a unified data interface is adopted. To handle various common communication protocols, In order to simplify the operation steps, we designed a plug-and-play method for devices based on TCP/IP protocol. The service scheduling layer needs analysis and research: device-driven scheduling and management, application service scheduling and management, log system, and management of the entire middleware platform. The application business layer needs analysis and research: it is mainly designed for the specific IoT application system, providing various data interaction methods with the application system and designing a unified interface in the application business layer, and different application services must inherit this interface Fig 2.

3.2 Internet of Things middleware

Based on the need's analysis of the IoT middleware in the opposite smart campus, we have implemented the functional layers of this middleware platform and introduced the implementation methods of key modules. Figure 3 is a detailed structural diagram of the Internet of Things middleware platform.

The communication between the various functional layers of the middleware is driven by events, and the data format is the object data type class [9]. First, the data collected by the device driver layer and the command data obtained by the application service layer are classified; that is different events are called according to different command codes of the data and transmitted to the service scheduling layer, and the service scheduling layer further transmits the messages. The data format passed between the middleware and the upper application and the database is an xml file. Messages from the middleware can be provided directly to the application in the form of xml or can be stored in the database via the database interface.

4 System design

4.1 Implementation of the device driver layer

The device driver layer is located at the bottom of the architecture and interacts directly with the terminal device. Because the device accesses the middleware in different ways, the data format uploaded by the device is different. In order to shield the underlying device from the upper layer, the communication interface needs to be driven at the device driver layer, the transmission protocol is parsed and the data format is unified. The terminal device uses a unique identifier in the database; i.e. the ID is unique. The way to get a unique ID is to call the .NET Framework's Get Guid () function to get a global variable when reading a device data for the first time. Driving the communication interface module and using the visual view, the configuration parameters are placed in the XML file, and the data parsing module also uses the XML file to facilitate subsequent rewriting. In order to realize unified management of devices by the service scheduling layer, all device drivers need to inherit the unified driver standard interface IDevice Driver. The filtering of the data collected by the middleware to the terminal

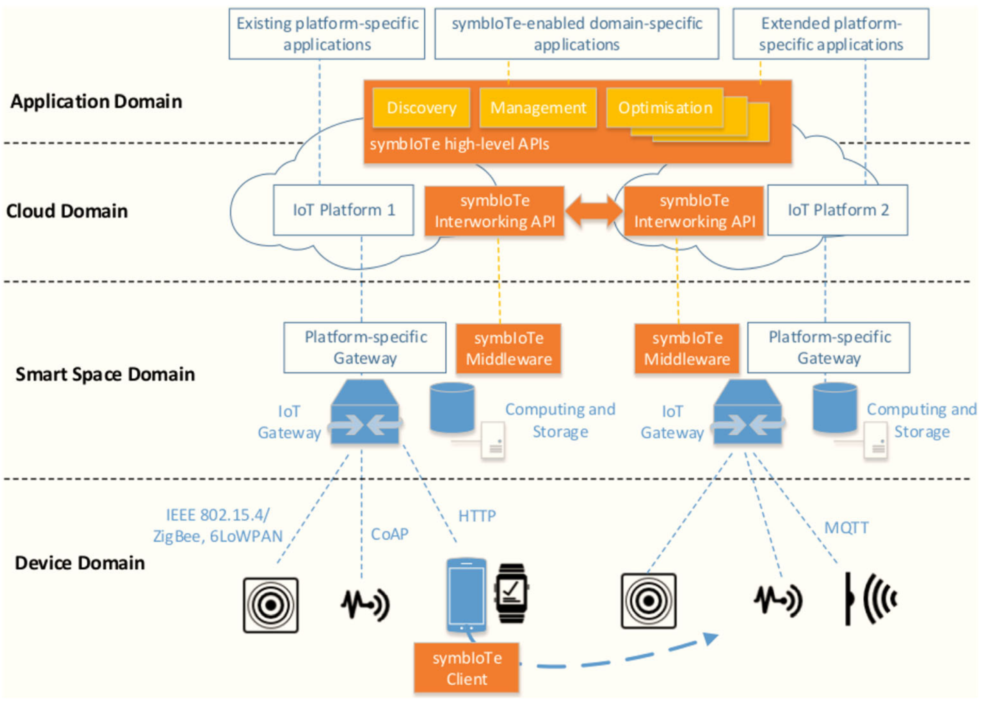

Fig. 2 Middleware platform overall structure

device is also completed internally by the driver and is transmitted to the middleware through a unified interface method.

The workflow of the device driver layer receiving the terminal device data is as follows: (1) the device driver is initialized, waiting for the port to read the data. (2) The device driver receives the data packet read by the port and performs and verifies the data packet according to the protocol format already specified. If the verification error occurs, the data packet is discarded and the received data is cleared. If the verification is correct, go to the next step. (3) Parse the packet and retrieve the command code CMD. (4) If the command code is 41, the device returns a query status result, executes a query status result event and notifies the service scheduling layer. (5) Plus command code is 42, the device returns the read data result, performs the data read result event and notifies the service scheduling layer. (6) Plus fruit command code is 43, then write the device data operation result, execute the device write result event and notify the service scheduling layer. (7) Plus fruit command code is 44; in order to set the device parameter operation results, perform device parameter setting result event and notify the service scheduling layer.

1. Device driver layer unified interface

The device driver layer provides a common set of interfaces that shield the differences between different RFID devices and sensors. The service scheduling layer uses the same interface to invoke the device without knowing the specific implementation of the device driver. Another advantage of using a unified interface is that the device driver layer can be developed only according to the interface specification, which greatly shortens the development cycle. A driver unified interface is a set of specifications for a set of operating devices that specify the uniform interface specifications that need to be followed for different device drivers.

2. Data filtering module

This module is only responsible for the basic filtering of device data, that is filtering the duplicate data within the defined time. It is optional. For example when the terminal device is RFID, this module can be called to filter the tags repeatedly read within a certain time interval. The middleware allocates a read thread and a data buffer to each device while collecting device data. Different devices need to call different data parsing methods to complete data security authentication and verification, and then filter the data at different levels through the filtering mode of the configuration file [10].

In the filtering process, the read data is put into a hash table, and the device ID is set as the keyword of the hash table. In this module, you need to define a time interval repeat time. When the device driver reads one data, check the hash table according to the device ID in the data. If the same data exists in the same device ID, verify both. Whether the time difference is smaller than repeat time and if it is less than, the data is considered to be read repeatedly and should be filtered out, and the reading time of the data of the device ID in the hash table is updated. If the read time difference between the two is greater than the repeat time, the data is considered to be the newly read device data, the data is retained, and the latest read time of the device data in the hash table is updated. The algorithm flow of this module is shown in Fig. 4.

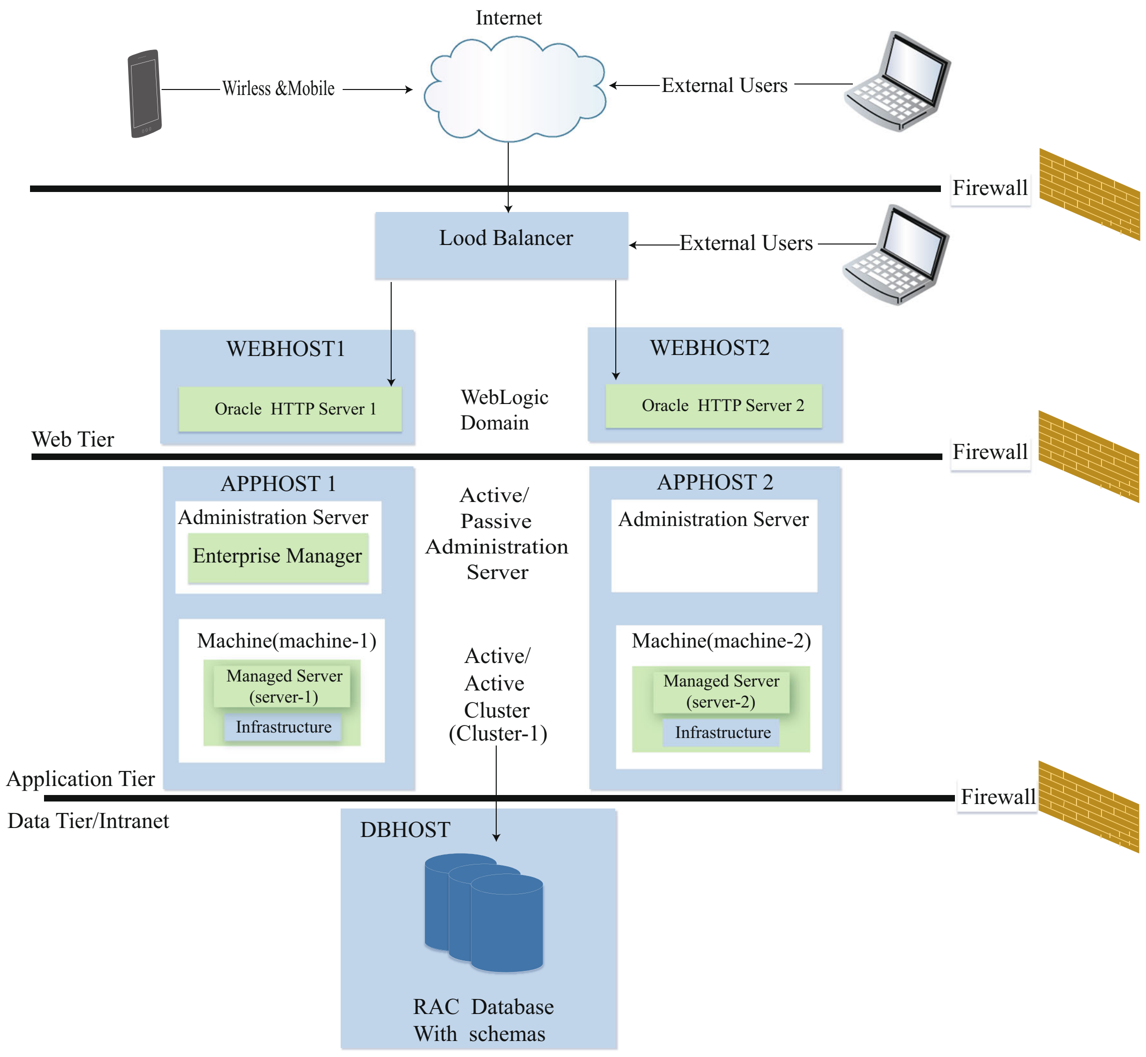

Fig. 3 Middleware platform detailed structure diagram

4.2 Implementation of the service scheduling layer

1. Event delivery module

The middle-time service scheduling layer's time delivery module is divided into two parts: one is the receiving device driver layer event module. The other is to receive the application business layer event module. The key code of the device driver layer event module is an object event listener, corresponding to a device connection event of the driver layer, a device disconnection event, a device event state, a device data event, a device data event and a corresponding device data event. The event handler is listening. The key code is also an object listener, corresponding to the service application layer to obtain the instruction to be issued, the read status command event, the read data command event and the write back data command event, respectively, and their corresponding event handlers are monitored.

2. System log

In a complete software system, the logging system is a very important part of the function. It records all the behaviours generated by the system and expresses them according to certain specifications. We can use the information recorded by the logging system to troubleshoot the system, display the running status of the program or adjust the behaviour of the system based on this information. In the middleware platform, we implemented (1) terminal device status monitoring: the terminal device status is obtained in real time through the terminal status function driven by the round-robin device and displayed in the device list. (2) Data monitoring: real-time display of all data communication information of terminal equipment and middleware, including status, data transmission and reception, can view all equipment and can only view one piece of equipment information. (3) Performance monitoring: display and

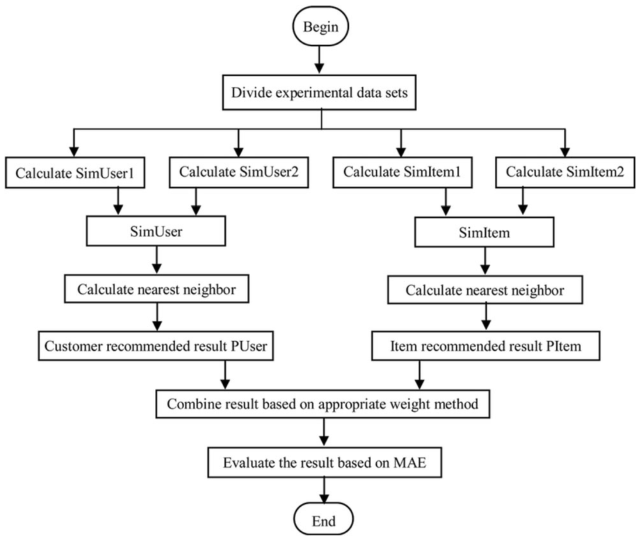
Begin
Divide experimental data sets
Calculate SimUser1
Calculate SimUser2
Calculate SimItem1
Calculate SimItem2
SimUser
SimItem
Calculate nearest neighbor
Calculate nearest neighbor
Customer recommended result PUser
Item recommended result PItem
Combine result based on appropriate weight method
Evaluate the result based on MAE
End

Fig. 4 Filtering algorithm flow charts

record the number of round robins per device, the number of readings, the number of business application processing completions, the number of business application processing failures and the processing time.

4.3 Application business layer implementation

Application business layer: The application business layer is responsible for directly interacting with the upper application or database. The middleware service object scheduling layer needs to implement indiscriminate calls to all services, so different services must inherit the unified interface method and define it as the IBusines interface class method. Since the upper layer may have many different applications, in order for each application to call the middleware data, a standard external publishing API function must be defined. In addition, some data passing through the middleware needs to be directly called by the application, and other data may need to be saved into the database, and an XML data conversion module is designed in the application business layer, and the data transmitted from the service scheduling layer is converted into various XML data.

1. Application service layer unified interface

The application business layer unified interface defines a set of basic operations. The main functions are implemented as follows: Parameter setting Setup: When the business processing program needs to use the middleware call to perform parameter configuration, it can be implemented in the Setup() method, and the middleware driver management interface Select the driver and click "Parameter Settings" to call the Setup method. Start the application service Start (): This method is called when the middleware starts the application business, mainly completes the business class initialization, loads the business layer parameter information and starts the business processing thread, such as the device state round-robin thread and the device data collection thread. Stop the application business Stop (): This method is called when the middleware stops the application business, mainly stops the business processing thread and releases the business layer memory [11, 12].

2. Data uniform format

Before providing the data to the upper application terminal, it is necessary to unify the different data formats sent by different terminal devices. This is an important role of the Internet of Things middleware. The unified data of the middleware platform of this paper is expressed in XML format, and the specific number is as follows:

```
<?xml version=" 1.0" encoding="utf-8"?>
<Unit>
<Device-ID=" 1 "></Device-ID>
<Device-Info>
<Device-SN> 10000001 </Device-SN>
<Device-Type>ZIGBEE</Device-Type>
<Device-Port>0</Device-Port>
<Device-Area> 1 </Device-Area>
</Device-Info>
<Data-Info>
<Data. -Type> 1 </Data. -Type>
</Unit>
```

In this XML data, the start identifier is defined as <Unit>; the device unique identifier symbol is defined as <Device-ID>, which is the unique identifier of the device in the database to distinguish different devices; the device information identifier symbol is defined as <Device- Info>, which contains several parameters; the data information identification symbol is defined as <Data-Info>, which also contains several parameters; and the end identification symbol is defined as </Unit>. In the device information, the related parameters are as follows: <Device-SN> is the device serial number, or physical address such as MAC address; <Device-Type> is the device type; <Device-Port> is the device port, if no port is used, 0″ means; and <Device-Area> is the area where the device is located. For example "1" is the plant no. 1. In the data information, the relevant parameters are as follows: <Data-Type> is the data type; for example "1" is the temperature information; <Time> records the current data collection time: and <Data> is the currently collected data. The information that can be obtained from the above XML file is that the device marked as "1" is of the ZIGBEE type, and the plant of the area "1" has no port information. It was measured at 10:42:56 on November 23, 2015. The temperature is 12.10 degrees.

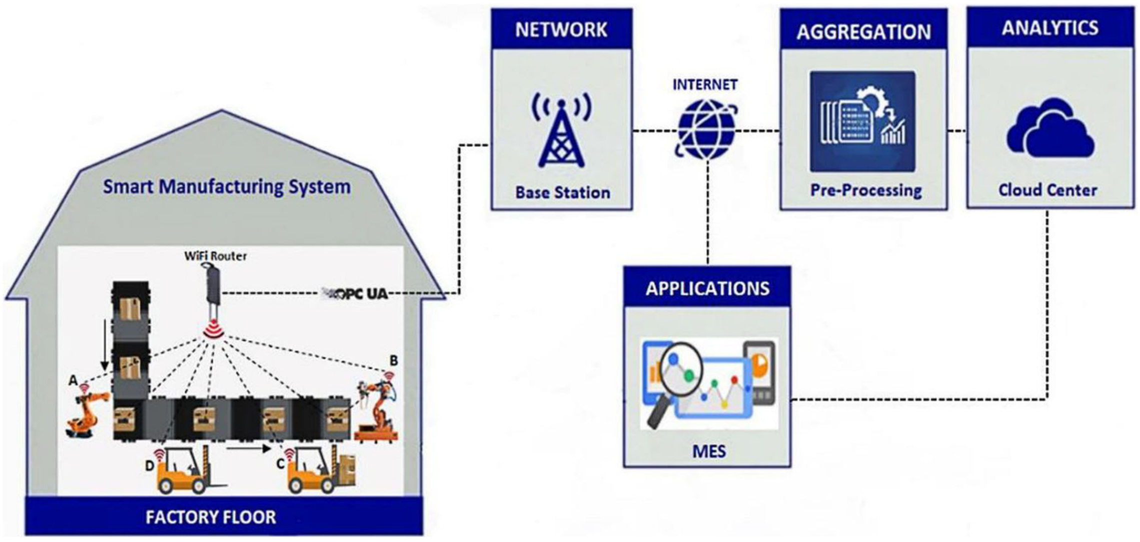

Fig. 5 Middleware-based factory floor environmental monitoring system

5 System testing

This case implements a middleware-based factory floor environment monitoring system. The system is divided into a host computer part and a lower computer part. The upper computer part includes environmental monitoring page, middleware and database. The lower computer part includes a gateway, two ZIGBEE devices, a sensor acquisition module and a sensor control module.

The greenhouse environment information is collected through the ZIGBEE terminal node, and the networking is wirelessly sent to the ZIGBEE central node, and the central node sends the data to the gateway, and the gateway sends the data to the middleware through the serial port. The middleware is responsible for parsing the gateway data and converting the temperature, humidity and smoke data into XML data of the same format, which is directly used by the monitoring page application. The monitoring page performs parameter configuration and sends commands to the terminal device by calling the API function of the middleware. After receiving the command data, the middleware parses the command and sends the data to the gateway at the driver layer according to the pre-defined protocol format, and the gateway sends the data to the ZIGBEE central node, and the central node sets the information through the wireless networking, sent to the ZIGBEE terminal device. The terminal device operates the control module according to the received command: The relay controls the heating device and the ventilation device. The middleware-based factory floor environment monitoring system is shown in Fig. 5.

6 Conclusion

The middleware designed in this paper is between the sensing device and the upper layer service system. It adopts the SOA architecture design and has the characteristics of coarse granularity and loose coupling, which satisfies the requirements of multiple access devices of the Internet of Things and supports multiple applications. To implement access management for multiple types of sensing devices, each device can be flexibly designated to provide information services for specific application systems and to get rid of the limitations of development, maintenance and expansion brought about by non-standardized protocols for IoT terminal devices. Finally, the above functions are verified by specific cases.

References

1. Ghosh A, Sarkar S (2016) Strategic interaction among different entities in internet of things. J Pharm Sci 65(12):1827–1831
2. Fang S, Li DX, Zhu Y, Ahati J, Pei H, Yan J et al (2014) An integrated system for regional environmental monitoring and management based on internet of things. IEEE Trans Ind Inf 10(2): 1596–1605
3. Cruz HR, Rt DSJ, de Holanda MT, De OAR, García Villalba LJ, Kim TH (2017) Distributed data service for data management in internet of things middleware. Sensors 17(5):977
4. Nordahl M, Magnusson B (2016) A lightweight data interchange format for internet of things in the Palcom middleware framework. J Ambient Intell Humaniz Comput 7(4):523–532
5. Ferreira HGC, Junior RTDS (2017) Security analysis of a proposed internet of things middleware. Clust Comput 20(1):1–10
6. Gomes B, Muniz L, Da FSES, Dos DS, Lopes RF, Coutinho LR et al (2017) A middleware with comprehensive quality of context support for the internet of things applications. Sensors 17(12):2853
7. Shi, Y. Q., & Zhu, Y. L. (2013). The application of wireless sensor network and middleware in the internet of things. Applied Mechanics & Materials,427-429(5):611-615
8. Chelloug SA, Elzawawy MA (2017) Middleware for internet of things: survey and challenges. Intell Automation Soft Comput 24(2):309–317

9. Blair G, Schmidt D, Taconet C (2016) Middleware for internet distribution in the context of cloud computing and the internet of things. Ann Telecommun 71(3–4):87–92
10. Jesús RM, José-Fernán M, Pedro C, Lourdes L (2013) Combining wireless sensor networks and semantic middleware for an internet of things-based sportsman/woman monitoring application. Sensors 13(2):1787–1835
11. Rausch T, Dustdar S, Ranjan R (2018) Osmotic message-oriented middleware for the internet of things. IEEE Cloud Computing 5(2): 17–25
12. Leng, K., Bi,Y., Jing, L., Fu H., Van Nieuwenhuse, I. (2018). Research on agricultural supply chain system with double chain architecture based on blockchain technology. Futur Gener Comput Syst, 86: 641–649