MiWoT: An Application-Oriented Programming Supporting Framework for Web of Things

Lin Yan, Tao Feng, Gang Chen, Yao Guo, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education)
Institute of Software, School of EECS, Peking University, Beijing, China
{yanlin10, fengtao09, chengang10, yaoguo, cherry}@sei.pku.edu.cn

ABSTRACT

With the development of Internet of Things (IoT) in recent years, Web protocols have been adopted as a key communication method, thus IoT evolves into Web of Things (WoT). Different web service implementations have been proposed to provide a standard framework to connect heterogenous devices in WoT. However, most of the existing efforts focus only on providing device-level web services, making it difficult for programmers to develop complicated applications. This paper proposes an application-oriented programming supporting framework for WoT called MiWoT, which provides a uniform framework to hide the heterogeneity of smart devices. In order to enhance programming support of WoT applications, MiWoT also provides a high-level application-oriented device abstraction interface, which allows developers great flexibility and simplicity to develop applications. We build a prototype of MiWoT in a SmartLab project and show its feasibility through a case study.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: [Network Architecture and Design]

General Terms

Design, Management

Keywords

Internet of things, web service, application programming interface

1. INTRODUCTION

Seamlessly integrating physical world with digital world is a main objective in pervasive computing research. To achieve this goal, the Internet of Things(IoT) provides a vision that physical objects such as sensors, actuators and even home appliances can be integrated into the Internet as the information source or the action performer.

Due to the fast advances in microelectronics and the falling price of processors and communication modules, physical objects are becoming smart objects with additional computing and communication capabilities, which can be easily interconnected in a uniform addressing scheme like IP Address and a communication protocol like the Internet Protocol(IP). Ideally, under the concept of IoT, all the interconnected smart objects would eventually speak the same language.

However, hundreds of IP based application protocols, for example X10, AMQP, ModBus and Megaco, have been created by different academic groups, industrial organizations and enterprises on behalf of their own benefits. These proprietary protocols divides IoT into hundreds of isolated networks thus creating a great challenge for IoT application development since very few application developers know the whole spectrum of technologies across organizational boundaries.

The notion of the Web of Things(WoT)[10], which enhances the interoperability of IoT, aims at eliminating the tight coupling within each aforementioned isolated network. WoT enables application developers to integrate those smart things into the Web as resources leveraging widely adopted Web standards and principles(e.g. HTTP, URI, HTML, XML, REST).

In order to realize such a scenario, Web Service has been leveraged as an key enabler to shield heterogeneity and provide interoperability. Web servers have been implemented directly on embedded devices thus providing web services[13][5]. For those severely resource-constrained devices, the web server is implemented on the gateway which connects the devices with the Internet[9].

However, most of the existing efforts focus only on providing device-level web services, making it difficult for programmers to develop complicated applications. To the best of our knowledge, the APIs offered in existing WoT frameworks are device-oriented, which means that they just forward the raw sensing data from the specified device to applications. While in fact, APIs that an application developer really needs are more meaningful than sensing data from each device. For example, the temperature of a specific location (such as a conference room) provides more meaningful results compared to a collection of temperatures from all sensors in the location. One of the key challenges here is how to provide application developers with application-oriented APIs.

To solve the above problems, this paper proposes an application-oriented programming support framework for

WoT called MiWoT, which provides a uniform framework to hide the heterogeneity of smart devices. In order to enhance programming support of WoT applications, MiWoT also provides a high-level application-oriented device abstraction interface, which allows developers great flexibility and simplicity to write applications. We build a prototype of MiWoT in a SmartLab project and show its feasibility through a case study.

The main contributions of MiWoT include:

- MiWoT implements a uniform framework supporting heterogenous physical devices that could have different implementations of web services.

  For smart objects with relatively abundant resources, MiWoT provides a light-weight web service implementation, such that these objects could be accessed directly through web services. We implement these web services on a wireless sensor network mote (IRIS) running Tiny OS.

  Because many objects do not have enough resources to run web services (such as RFIDs), MiWoT also provides a central proxy to connect these objects through web services.

- Besides device-oriented APIs accessing each sensor directly, a key feature of MiWoT is that it also provides application-oriented APIs, which allow application developers to specify their requirements for application-oriented data that is more meaningful. The data requirements can be specified easily with a "Filter and Combiner" mechanism.

- We evaluate MiWoT thoroughly through a case study in a SmartLab project, in which several different sensors and objects are deployed in a real environment. Programming examples demonstrate that MiWoT provides developers great flexibility and simplicity when developing WoT applications.

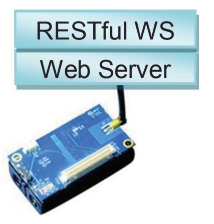
This paper is organized as follows: in Section 2 we discuss the background and related work. Section 3 presents the details of the MiWoT architecture. We present a case study in SmartLab project in Section 4. Section 5 concludes this paper.

2. BACKGROUND AND RELATED WORK

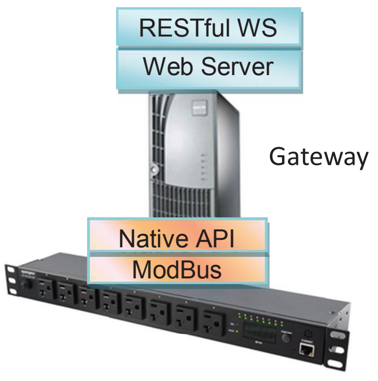
2.1 Web of Things

With the diminishing price of embedded devices, predictions have been made that the number of networked sensors and actuators will eventually exceed the number of networked computers by several orders of magnitude[3]. The Web of Things(WoT) vision leverages the existing ubiquitous Web protocols and standards to interconnect these devices. Compared to those proprietary protocols, the Web protocols and standards like HTTP, URI, HTML, etc. are much more open, flexible, scalable and widely accepted. So integrating the smart objects and devices into the Web really means integrating them into our daily lives.

The WoT implementations introduce web service as an enabler of the interoperability for a loose coupling among distributed smart objects and devices. These web services are often RESTful web services[8] which are simpler and more lightweight than WS-* Web Services.



(a) Direct Implementation



(b) Intermediate Implementation

Figure 1: Two ways of implementing Web Service

2.2 Web Service Implementation

There are two ways to implement Web Services as depicted in Figure 1 . For devices with enough resources to support a web server, such as WSN motes, Web Services are directly implemented on them like Tiny Web Services[13] and sMAP[5]. For other devices with insufficient resources like smart sockets, Web Services are intermediately implemented on a gateway, which originally works as a proxy connecting devices with the Internet. In MiWoT, the Device Service Proxy plays the role of gateway in the latter way.

End user applications are developed in both works by calling these web services implemented. While this kind of implementation tackles a stunning technical challenge by giving resource-constrained devices a web server, there are still a lot of concerns to make use of these web services directly on device. Resource-constrained devices are not powerful enough to support concurrent applications and complicated access control mechanism.

2.3 Application-Supporting Frameworks for WoT

WoT frameworks have been proposed to address the above issues. One category of such frameworks is like Sensor.Network[11] and Pachube[2]. They facilitate sensor and actuator to connect together and provide APIs to access the data on them and to make use of their functions. However, they just provide basic device-level APIs which just forwards the raw sensing data to applications.

They other category of frameworks is like SOCRADES[6]. SOCRADES is an architecture focusing on coupling web

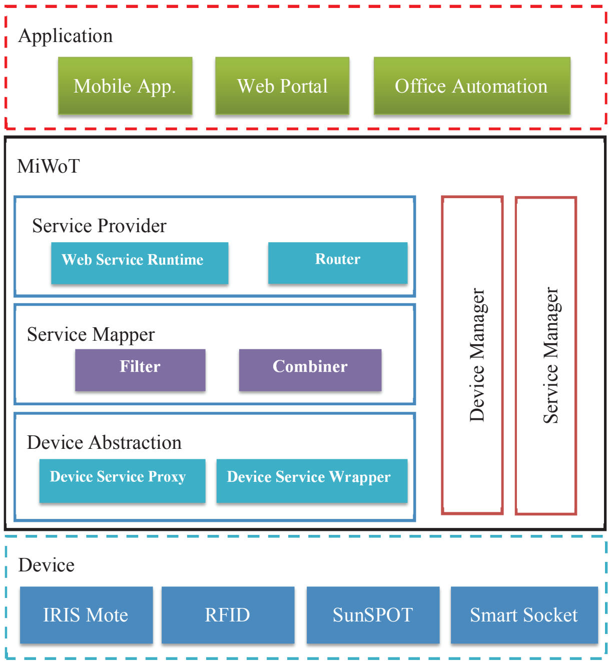


Figure 2: Architecture of MiWoT

service enabled devices with enterprise applications such as ERP Systems. However, it doesn't explicitly show its support for application development.

3. ARCHITECTURE OF MiWoT

3.1 Overview

In this section we present an overview of MiWoT's whole architecture. As shown in Figure 2, based on smart devices, MiWoT offers application-oriented programming support for programmers.

There are mainly three layers of components in MiWoT: **Device Abstraction**, **Service Mapper** and **Service Provider**. These three layers can handle invocations from application and map them to invocations of services on device and calculate the final results or execute actions as defined by application developers. Besides these layers of components, MiWoT offers two GUIs called Device Manager and Service Manager for application developers to accomplish the work of manage devices and services. In the following, each part of MiWoT and its basis is described in a bottom-up sequence.

Device: Heterogenous devices are connected to MiWoT, like nodes from Wireless Sensor Networks measuring temperature, illumination and humidity, smart sockets controlling power supply to home appliances as well as RFID readers and tags to detect the presence of objects being traced. Device types are not limited to this list and in general, devices can be classified into two categories: sensors and actuators.

Device Abstraction: For some devices, they offer web services themselves like motes in Wireless Sensor Networks(WSN). For other dumb devices, such as smart sockets and RFID readers and tags, web services are not directly offered. In MiWoT, both kinds of devices can be integrated with the

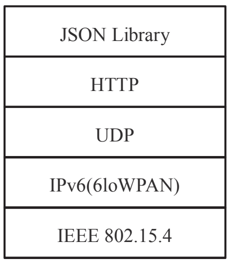


Figure 3: Layers in the protocol stack on IRIS mote

help of Device Abstraction. For devices with discoverable web services on themselves, the Device Service Proxy just forward their services to upper layers with almost no modification. For dumb devices without the notion of web services, the Device Service Wrapper communicates with them in an ad-hoc manner and wrap their functionalities into web services.

Service Mapper: Application Services offered by MiWoT need to be mapped into services on devices. The Filter component filters device services for desired ones and the Combiner calculates the final results or execute actions as defined by application developers.

Service Provider: The MiWoT provides both device-oriented APIs and application-oriented APIs. All these interfaces or services reside in the Application Service Runtime and invocations from applications are responded by Router.

Device Manager and Service Manager: The application developers can add new devices, update or delete existing devices with Device Manager. This GUI also allows developers to know the running status of each device. The Service Manger offers the developers with the ability to define new application-oriented services by giving the Filter and Combiner Specifications. Besides, developers can edit or search services using this module.

3.2 Device-level Web Service Implementation

As mentioned before, web service has been chosen as an key enabler of the WoT scenario. In our architecture, Device Abstraction is responsible for wrapping up devices' functionalities into device-level Web Services.

In the following we discuss how Web Services are directly implemented on WSN motes. In our approach we implement Web Services on three different hardware platforms: IRIS, SunSPOT and Arduino Duemilanove. Here we focus on implementation of IRIS motes since implementation on Java Virtual Machine enabled SunSPOT and Processing based Arduino Duemilanove are relatively less challenging.

IRIS mote runs a TinyOS operating system on batteries with only 8Kbytes of RAM. Targeting at such an resource-constrained hardware platform, we implement a lightweight protocol stack offering Web Services in a REpresentational State Transfer(REST) manner which reuses the GET, POST, PUT, DELETE methods of HTTP[8]. RESTful Web Service with JavaScript Object Notation(JSON) as data representation[4] is a lightweight replacement of Simple Object Access Protocol(SOAP) and XML based Web Service since it reduces the parsing and generating overhead.

As depicted in Figure 3, IRIS motes communicate in an IEEE 802.15.4 based network. We make them IP addressable by adopting *blip*, an IPv6 stack included in TinyOS 2.2 conforming to the 6loWPAN standard. The 6loWPAN standard[12] introduces an adaptation layer to compress IPv6 packets in order to reduce the communication overhead. Upon IPv6 layer, we implemented a compressed HTTP layer in conjunction with the JSON library just serving basic requirement of RESTful Web Service provision.

3.3 Data Abstraction

One critical issue of MiWoT is the sensor data abstraction. At the core of the MiWoT architecture is the concept of a *datastream.* A datastream is a time-series of one or more sensor values sampled by the same entity. The entity could be actual sensors like aforementioned IRIS motes. The datastream of an IRIS mote mentioned in Section 3.2 includes two values at a specific time point: temperature and illumination. However, in MiWoT, the entity could also be an environment, like a conference room or a car, defined by application developers. The values of an environment entity are fused from readings of actual sensors located in the same environment by the "Filter and Combiner"mechanism.

For each datastream, the following attributes must be specified when the datastream is first introduced to MiWoT. They are Name, Type, Environment Tags, Sampling period and Data values. We will discuss each attribute in detail.

Name Each datastream should have a Name indicating what the datastream is and the Name must be unique, e.g. "IRIS#001"

Type According to the sampling entity mentioned before, a datastream should be referred as **Sensed** type or **Environment** type.

Environment Tags A datastream must be associated with one or more Environment Tags to describe those environment in which it is fit in, like "John's Car", "Lily's Office" or "ConferenceRoom 101".

Sampling Period MiWoT will generated data values corresponding to this attribute. Besides, MiWoT could detect devices' status using this period. Sampling period should be specified in milliseconds.

Data Values A datastream could have one or more data values. Each data value stands for a type of data that the datastream can offer. For data values of a **Sensed** datastream, each of them has a sensor type, a data type and a unit. For example, the temperature value of the IRIS mote mentioned in Section 3.2 has a *temperature* sensor type, a *float* data type and a unit of *Celsius.* For data values of user-defined **Environment** datastream, each of them should be associated with a **Filter** function and a **Combiner** function and we will discuss these two functions in Section 3.4.

3.4 The "Filter and Combiner" Mechanism

The "Filter and Combiner" mechanism enables MiWoT to support application-oriented programming. As stated before, when application developers want to define an Environment datastream, he should specify the Filter function and Combiner function for each data value inside the datastream.

Developers can use the Filter to select a collection of datastreams from raw sensing data. Then Combiner will generate the newly defined datastream from the collection.

The **Filter** function and **Combiner** function are specifications for the **Filter** component and the **Combiner** component in the MiWoT architecture. Developers can specify Filter function and Combiner function similar to a Map-Reduce style[7].

Consider a motivating example in which the application needs to get the temperature of a conference room whose number is 101. Suppose that there are ten smart objects providing temperature reading of different locations in conference room 101. Without MiWoT, the programmer should manage to get each temperature reading separately on those ten objects and then calculate them in the application logic to get the value wanted.

However, with the support of MiWoT, the developer can get the temperature of conference room 101 directly in a "Filter and Combiner" way. The developer should define an Environment datastream named "ConferenceRoom101" and add a data value of "temperature" sensor type to this datastream. Then the developer should specify the Filter and Combiner telling MiWoT how to generate this data value.

The Filter function takes an object of DataStream type as input and its return value is a DataValue object. The Filter component of MiWoT will traverse all existing DataStreams executing this function and finally get a DataValueCollection object. Obviously, in function logic, the developer should express the conditions of attributes that a datastream must satisfy and return the desired data value of the datastream. As for the motivating example, the Filter function is like the following codes.

```
DataValue Filter(DataStream dataStream){
 if(dataStream.type == "Sensed"){
  foreach tag in dataStream.Environment_tags
   if(tag == "conference␣room␣101")
    foreach datavalue in dataStream.Datavalues
     if(datavalue.Sensor_type == temperature)
      return datavalue;
 }
 return null;
}
```

The Combiner function takes a ValueCollection object as input and returns a DataValue object. In function logic, a developer should specify how to map the data in the ValueCollection into a single data value. The Combiner function will be executed by the Combiner component in MiWoT. The Combiner function of the motivating example is like the codes below to calculate the average value of all temperature readings in conference room 101.

```
DataValue Combiner(ValueCollection vCollection){
  double sum=0;
  foreach datavalue in vCollection
  {
    sum+=(double)datavalue.Value;
  }
  return sum/vCollection.Count;
}
```

3.5 Multi-level API

As stated before, our framework implements RESTful APIs on smart objects to provide access to sensors and actuators through the web. We can provide not only basic device-level APIs to access data from actual sensors, but also application-oriented APIs to access data pre-processed by MiWoT based on application requirements.

To access sensor data, we provide a uniform format of API for both datastream from actual sensors and datas-

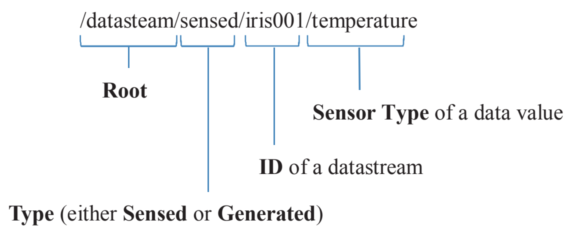

/datasteam/sensed/iris001/temperature
Root
Sensor Type of a data value
ID of a datastream
Type (either Sensed or Generated)

Figure 4: The URL structure for a temperature reading on a sensed datastream named IRIS001

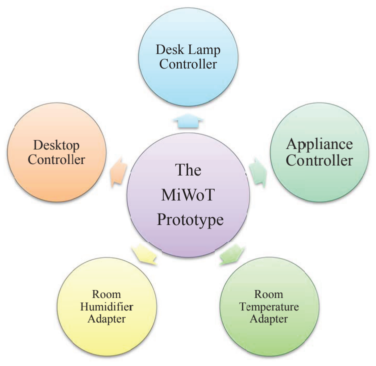


Figure 5: Five applications based on the MiWoT prototype

tream defined by application developers. Figure 4 shows the whole URI structure of the API by an illustrating example to read a datastream(URI of the server such as "http://www.miwot.com" has been left out here for neatness).

As REST implies[14], the datastream accessing API is a Resource Oriented Architecture. Table 1 illustrates the resource hierarchy of the datastream accessing API. All APIs on the resource hierarchy should be accessed using the GET method of HTTP.

To perform actions with actuators, we also provide a uniform format of API. Different from datastream accessing APIs, action performing APIs make use of both GET method and POST method of HTTP. The resource hierarchy of action performing APIs is illustrated in Table 2, in which a smart socket is taken for example.

4. MiWoT IN SMARTLAB PROJECT: A CASE STUDY

4.1 Project Scenario

Being on the market for more than two decades, home and office automation has already been a commonplace both for high-end and low-end user around the world[15]. SmartLab Project envisions a office automation scenario of a computer laboratory. The main goal of SmartLab project is to provide convenience for lab members' daily life and make the lab energy-efficient . To accomplish this goal, we use MiWoT to wrap up all the physical devices with RESTful APIs thus providing better programmability to make the lab smarter.

4.2 The Prototype

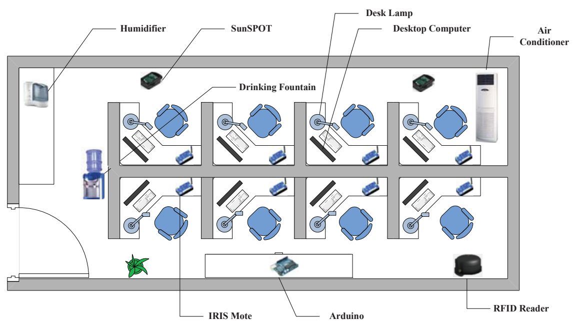


Figure 6: The plan view of the lab as SmartLab prototype

Table 3: Application Description of the prototype

Application	Description
Desk Lamp Controller	Turning the desk lamps on and off depending on the presence of their owners and illumination
Appliance Controller	Cutting off the power supply when nobody in the lab and recovering when somebody shows up
Room Temperature Adapter	Maintain the room's temperature
Room Humidity Adapter	Maintain the room's humidity
Desktop Controller	Booting the desktop computer when the owner shows up; Hibernating it when the owner leaves

4.2.1 *Overview*

We build a prototype of MiWoT in a SmartLab project based on our own lab environment. A plan view of our lab is shown in Figure 6. The physical objects deployed in the lab include several IRIS motes and SunSPOTs, one Arduino, one RFID reader and several tags, appliances such as an air conditioner, a humidifier, a drinking fountain, desk lamps and desktop computers. These objects are all managed by MiWoT. As depicted in Figure 5, the prototype consists of five separate applications developed by the use of RESTful APIs offered by MiWoT.

4.2.2 *Smart objects*

Smart objects are enhanced physical objects with additional computing and communication capabilities. As depicted in Figure 6, in our prototype, some objects are already smart, such as IRIS, SunSPOT, Arduino, RFID reader and tags and desktop computers. Other objects can be combined with smart ones into smart objects like smart desk lamp, smart humidifier, smart air conditioner and smart drinking fountain.

In Table 4, we describe these smart objects in detail such as sensors/actuators on them and in which way they can be integrated into MiWoT. If web service has been directly implemented on the smart object, then it is *directly*

Table 1: Example of resource hierarchy of datastream accessing API provided by MiWoT

/datastream	root collection of datastreams
/datastream/sensed	collection of sensed datastreams
/datastream/generated	collection of generated datastreams
/datastream/sensed/1001	collection of sensor types
/datastream/sensed/1001/temperature	collection of temperature readings on sensor datastream 1001
/datastream/sensed/1001/illumination	collection of temperature readings on sensor datastream 1001

Table 2: Example of resource hierarchy of action performing API provided by MiWoT

HTTP Method	URI	Purpose
GET	/actuator	To list all the actuators available
GET	/actuator/smartsocket	To list all the outlets on the smartsocket
GET	/actuator/smartsocket/outlet1	To get the status of outlet 1
POST	/actuator/smartsocket/outlet1/1	To change the status of outlet 1 into 1 which stands for 'ON'

Table 4: Details of smart objects in the prototype

Name	Sensor	Actuator	Integration Method
IRIS	temperature, light	N/A	Directly Integrated
SunSPOT	temperature, light, 3-axis accelerometer, humidity	N/A	Directly integrated
RFID reader and tags	RSSI of each Tag	N/A	Proxy style
Desktop computer(Wake On Lan(WOL)[1] Enabled)	N/A	Booting or Hibernating the desktop computer	Proxy style
Smart air conditioner(Arduino+ Air conditioner)	temperature	Heating or Cooling hardware	Directly integrated
Smart desk lamp(smart socket+ desk lamp)	N/A	socket	Proxy style
Smart humidifier(smart socket+ humidifier)	N/A	socket	Proxy style
Smart drinking fountain(smart socket+ drinking fountain)	N/A	socket	Proxy style

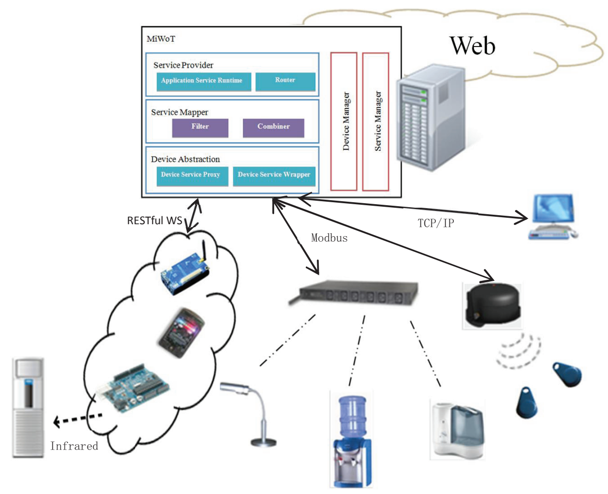


Figure 7: Deployment of hardware and software components

integrated into MiWoT. Otherwise, if the smart object takes MiWoT as a gateway to offer web service, it is integrated in a *proxy* style.

To sum up, the deployment of hardware and software components is illustrated in the Figure 7.

4.3 Desk Lamp Controller: An Example Implementation

To show the flexibility and simplicity in developing applications on MiWoT, we presented the implementation of Desk Lamp Controller which is one of the five applications in our prototype.

The desk lamp controller application controls desk lamps on each working desk. A desk lamp is turned on and off based on the presence of its owner and the light condition. A desk lamp will always stay off when its owner is not in the lab. When the owner shows up in the lab, the application will keep the illumination around him in a pre-defined range.

To fulfil the design functionalities, the programmer should know how to access the RFID sensing data to determine whether the desk owner is in the lab, the illumination around the desk and average illumination of the whole lab. Besides, the programmer should have a method to turn the lamp on and off.

Before programming, an Upper Limit Illumination Threshold(ULIT) and an Lower Limit Illumination Threshold(LLIT) must be given to determine the aforementioned range. When the owner is at the desk, the application logic to maintain the illumination in the range is described in Table.

Without the notion of application-oriented programming supported by MiWoT, the application get device-oriented APIs. The key algorithm of the application logic in pseudocode is like Algorithm 1.

With MiWoT, we can simplify this algorithm by leveraging the application-oriented programming support. The room illumination belongs to a Environment datastream which is created by application developer using "Filter and Combiner" mechanism.

In our approach, Filter and Combiner functions are agnostic of where the readings come from. The Filter and Combiner functions are defined as follows:

Algorithm 1 Algorithm to maintain illumination around the desk

```
1: $illumination[5]$; /*initialization. There are five illumination sensors in the lab*/
2: $illumination[0]$ = Get illumination reading from the first sensor with its API;/*This is the reading around the desk*/
3: $illumination[1]$ = Get illumination reading from the second sensor with its API;
4: $illumination[2]$ = Get illumination reading from the third sensor with its API;
5: $illumination[3]$ = Get illumination reading from the fourth sensor with its API;
6: $illumination[4]$ = Get illumination reading from the fifth sensor with its API;
7: $sum = 0$;
8: $average = 0$;
9: for (int $i = 0$; $i \leq 4$; $i++$) do
10:   $sum+ = illumination[i]$;
11: end for
12: $average = sum/5$;
13: if $average$>ULIT&&$illumination[0]$>ULIT then
14:   Turn the desk lamp off through API;
15: else
16:   if $average$>LLIT&&$illumination[0]$>ULIT &&$illumination[0]$<ULIT then
17:     Turn the desk lamp off through API;
18:   end if
19: else
20:   if $average$>LLIT&&$average$<ULIT &&$illumination[0]$>ULIT&&$illumination[0]$<ULIT then
21:     Turn the desk lamp off through API;
22:   end if
23: else
24:   if $average$<ULIT&&$illumination[0]$<ULIT then
25:     Turn the desk lamp on through API;
26:   end if
27: end if
```

```
DataValue Filter(DataStream dataStream){
 if(dataStream.type == "Sensed"){
  foreach tag in dataStream.Environment_tags
   if(tag == "lab")
    foreach datavalue in dataStream.Datavalues
     if(datavalue.Sensor_type == illumination)
      return datavalue;
 }
 return null;
}
```

```
DataValue Combiner(ValueCollection vCollection){
  double sum=0;
  foreach datavalue in vCollection
  {
    sum+=(double)datavalue.Value;
  }
  return sum/vCollection.Count;
}
```

Since MiWoT provides RESTful APIs, the application development is not limited to any specific programming

Table 5: Illumination Maintaining Logic."ON" stands for turning the lamp on; "OFF" stands for turning the lamp off; "N/A" stands for there is no such a circumstance; "Remain" stands for remaining the status of the lamp;

	Desk illumination > ULIT	**LLIT < Desk illumination < ULIT**	**Desk illumination < LLIT**
Lab illumination > ULIT	ON	N/A	N/A
LLIT < Lab illumination < ULIT	OFF	OFF	N/A
Lab illumination < LLIT	Remain	Remain	ON

Algorithm 2 MiWoT-based Algorithm to maintain illumination around the desk

```
    average = Get the reading from /datastream/generated/lab/illumination
 2: illumination0 = Get the reading from /datastream/generated/iris000/illumination
    if average>ULIT&&illumination0>ULIT then
 4:     Turn the desk lamp off through API;
    else
 6:     if average>LLIT&&illumination0>ULIT &&illumination0<ULIT then
            Turn the desk lamp off through API;
 8:     end if
    else
10:     if average>LLIT&&average<ULIT &&illumination0>ULIT&&illumination0<ULIT then
            Turn the desk lamp off through API;
12:     end if
    else
14:     if average<ULIT&&illumination0<ULIT then
            Turn the desk lamp on through API;
16:     end if
    end if
```

language or platform. The only requirement is that the code to access a HTTP server can be written in the chosen programming language. Continuing the above example of the Desk Lamp Controller, the application could access the room illumination using the URI:

/datastream/generated/lab/illumination

with GET method of HTTP. And the application can access the illumination of the desk from IRIS mote whose number is 000 using the URI:

/datastream/generated/iris000/illumination

To control desk lamp, the application could use the URI:
/actuator/desklamp/0/

with GET method to check the status of it and use the URI:

/actuator/desklamp/0/on

with POST method to turn on the lamp or with "off" at the end to turn off the lamp.

Based on these APIs described, key algorithm of the application logic of Desk Lamp Controller can be written as Algorithm 2.

5. CONCLUSIONS

In this paper, we presented the design and implementation of MiWoT, a Web Service based framework to interconnect heterogenous devices and provide both the device-oriented and high-level application-oriented APIs for application developers. With MiWoT, we build a prototype in a SmartLab project and show MiWoT's feasibility through a case study.

For the near future, we plan to add new features to MiWoT such as access control mechanism,and an eventing module to support event-driven programming.

6. ACKNOWLEDGMENTS

This work is supported by the National Basic Research Program of China (973) under Grant No. 2009CB320703; the National Natural Science Foundation of China under Grant No. 60821003; the National Science and Technology Major Project 2011ZX01043-001-002.

7. REFERENCES

[1] AMD White Paper on WOL. http://support.amd.com/us/Embedded_TechDocs/20213.pdf.

[2] Pachube. http://www.pachube.org.

[3] D. D. Clark, C. Partridge, R. T. Braden, B. Davie, S. Floyd, V. Jacobson, D. Katabi, G. Minshall, K. K. Ramakrishnan, T. Roscoe, I. Stoica, J. Wroclawski, and L. Zhang. Making the world (of communications) a different place. *SIGCOMM Comput. Commun. Rev.*, 35:91–96, July 2005.

[4] D. Crockford. RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON). Technical report.

[5] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. smap: a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 197–210, New York, NY, USA, 2010. ACM.

[6] L. M. S. De Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. Socrades: a web service based shop floor integration infrastructure. In *Proceedings of the 1st international conference on The internet of things*, IOT'08, pages 50–67, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[8] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. In *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, pages 407–416, New York, NY, USA, 2000. ACM.

[9] D. Guinard. Towards the web of things: Web mashups for embedded devices. In *In MEM 2009 in Proceedings of WWW 2009. ACM*, 2009.

[10] D. Guinard, V. Trifa, and E. Wilde. Architecting a mashable open world wide web of things. Technical Report 663, Department of Computer Science, ETH Zurich, Feb. 2010.

[11] V. Gupta, A. Poursohi, and P. Udupi. Sensor.network: An open data exchange for the web of things. In *PerCom Workshops*, pages 753–755, 2010.

[12] J. W. Hui and D. E. Culler. Extending ip to low-power, wireless personal area networks. *Internet Computing, IEEE*, 12(4):37–45, July-Aug. 2008.

[13] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 253–266, New York, NY, USA, 2008. ACM.

[14] L. Richardson and S. Ruby. *RESTful web services.* O'Reilly Series. O'Reilly, 2007.

[15] J.-P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP - The Next Internet.* Morgan Kaufmann, 2010.