Cloud-based Programmable Sensor Data Provision

Lin Yan, Yao Guo and Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education)
School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
{yanlin10, yaoguo, cherry}@sei.pku.edu.cn

***Abstract*—As sensor data grow towards an explosion due to the popularity of Internet of Things and mobile computing, many sensor data sharing platforms are developed to support various sensor-based applications. Although these platforms are able to provide capabilities such as collecting data from sensors and sensor data provision for applications, their capabilities are normally confined in direct retrieval of sensor data with little composition such as SQL aggregation or even no composition at all. This kind of raw sensor data provision not only increases the network traffic between platforms and applications, but also put most computation burden on the client side, which poses big challenges for applications running on resource-constrained devices such as mobile phones.**

In this paper, we propose cloud-based programmable sensor data provision, which moves the sensor data processing logic from client applications to cloud-based services. The key technique behind this is *FilterCombine*, a two-step sensor programming support framework that enables developers to specify sensor processing logic in the cloud service. By moving sensor data processing logic to the cloud, we not only reduce network traffic due to data transfer and computation on the client side, we also improve code reusability in the cloud side, as many sensor data processing logic can be shared among multiple applications. We build a prototype platform of cloud-based programming sensor data provision called MiWoT, which implements the proposed *FilterCombine* mechanism on the cloud side. We demonstrate the feasibility of the proposed techniques through case studies.

I. Introduction

With the advancement of Internet of Things and mobile computing, sensors are becoming pervasive in our daily life. For example, there are billions of dedicated sensors planted in our surrounding environment and billions of mobile sensors embedded in our smartphones. As an example, dedicated sensor networks have also been planted as a public infrastructure by governments globally to improve city management[1].

As these sensors are generating an overwhelmingly huge amount of data everyday[2], sensor data management has become a big challenge. At first, sensor data are managed by the corresponding sensor owners in an ad-hoc manner. Sensor owners could develop various applications based on these sensor data to fulfill their various data usage intentions. However, it soon became evident that such scenarios cannot exploit the full power of sensor data. On one hand, one sensor owner alone has limited insights, resources and technical skills. On the other hand, some sensor data based trends or conclusions can only be drawn when taking a large scale of geographically distributed sensors into consideration, instead of some small set of sensors which belong to a single owner.

In order to unleash the full potential of sensor data, many people are advocating open data initiatives for numerous sensors planted worldwide[3], [4], [5]. *Open Data* encourages the sharing of formatted government or private entity data regardless of copyright restrictions, in order to support the creation of innovative services and applications. It is believed that the open data initiatives could boost open innovation of citizens in a smart city. For example, Chicago is already planning to open source its sensor data to the public[6].

As examples of open data platforms, a number of sensor data sharing platforms have been built to provide data collection as well as data provision services. Representative platforms in the research community include Microsoft SenseWeb[7], Global Sensor Networks (GSN)[8], SensorBase[9], IrisNet[10], Sensor Data as a Service[11], etc. There are also some commercial platforms such as Xively[12] (a.k.a. COSM or Pachube), Wikisensing[13], SensorCloud[14]. For data producers (i.e. sensors), they provide web service based APIs for data upload. For data consumers (i.e. applications), they provide web service based APIs for data retrieval. However, these APIs are mostly data-centric APIs in the sense that they focus on the creation, update, query and deletion of specific sensor data streams in a simple “store-and-retrieve” manner. Applications typically make direct use of the raw sensor data with little processing provided by the platforms. The situation is referred as “*raw sensor data provision*” in this paper, which is illustrated in the top part of Figure 1.

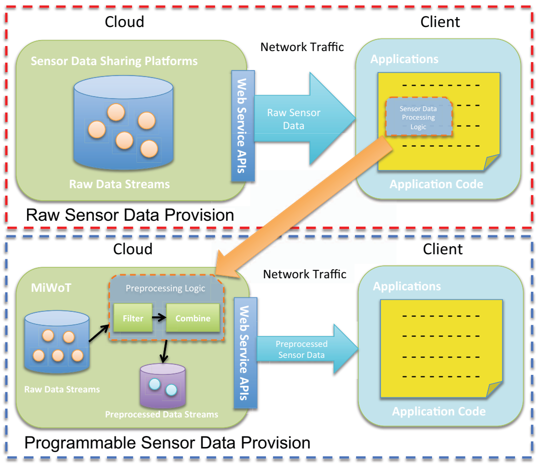


Fig. 1: Paradigm shift of sensor data provision.

Developing applications with raw sensor data provision services brings some serious challenges. First of all, many applications might require polling a big number of sensors regularly, which results in high network traffic. For instance, if a certain application needs to do some computation with all temperature sensor readings on a city level, it has to acquire all related (tens of thousands, or even more) raw data streams and perform the computation itself. This is especially challenging for applications running on resource-constrained mobile devices as network communication consumes a significant amount of energy and the computation power is limited.

One possible solution to this issue is moving sensor data processing from the client side (applications) to the cloud side (service provider), as shown in the bottom part of Figure 1. In this case, developers can specify a sensor data processing logic that runs in the cloud, which automatically collects the required sensor data and computes the required results. The client application can get the final results directly (for example, through web services), instead of doing all the tasks on its own.

However, in the new data provision paradigm, sensor data providers need to provide sensor data programming support, such that application developers can specify their data processing logic. Some platforms, such as GSN[8], allow data consumers to compose new data streams based on existing ones with simple range selectors or SQL selection and aggregators. Although this can provide some degree of programming support in defining the data processing logic, its capability is very limited due to its reliance on SQL queries.

In this paper, we propose MiWoT, a cloud-based framework that provides programmable sensor data provision. MiWoT is designed as a typically sensor data sharing framework based on cloud storage, in that it is able to provide a set of RESTful APIs for sensors to upload readings and for applications to consume sensor data. At the center of MiWoT, we propose a new sensor data programming mechanism called *FilterCombine*.

As shown in the bottom part of Figure 1, *FilterCombine* introduces a two-step programming support approach for developers to specify processing logic on sensor data. The two steps are called Filter and Combine, respectively. Just as its name indicates, *FilterCombine* can filter the sensor data and then combine the selected data into final query results as requested by application developers. In order to achieve this, *FilterCombine* will ask application developers to provide code snippets including a `Filter` and a `Combiner`[1]. The `Filter` tells the criteria to screen out qualified data streams and the `Combiner` will guide the platform to transform these qualified data streams into the target data stream. Both the `Filter` and the `Combiner` are all written in a full-blown scripting language.

For example, if an application wants to calculate the average temperature of a whole city with *FilterCombine*, it only needs to define a selection criteria (location tags in this case) called `Filter` and a set of composition rules (calculating average value logic in this case) called `Combiner`. Then it can retrieve the average temperature of the whole city directly from the cloud service provider.

The contributions of this paper are three-fold.

- We propose MiWoT as a cloud-based programmable sensor data provision platform, which is able to move data processing logic from the client application to the cloud-based service provider. MiWoT is also capable of managing heterogeneous sensor data.
- We propose *FilterCombine* as an open sensor data programming mechanism, which facilitates software reuse as `Filters` and `Combiners` can be shared and combined freely among developers.
- MiWoT could also reduce network traffic between applications and data sharing platforms, which is particularly meaningful for mobile applications.
- We have implemented a prototype of MiWoT, and demonstrated the feasibility of cloud-based programmable sensor data provision through case studies.

[1]FilterCombine can be compared to the popular Map/Reduce paradigm[15], however focusing on a different area.

The rest of this paper is organized as follows. We first discuss the background and related work in Section II. Section III presents the details of the MiWoT architecture. We present detailed design and implementation of the *FilterCombine* mechanism in Section IV and V. Section VI concludes this paper.

II. Related Work

The Internet of Things (IoT) [16], [17], [18] is a vision of a world where all kinds of heterogeneous smart objects and devices are uniformly and inherently networked together through the Internet Protocol(IP). Hopefully, in the realm of IoT, all the IP-enabled heterogeneous smart things will eventually speak the same language, while hundreds of communication protocols above IP such as AMQP[19], ModBus[20], Megaco[21], etc., have been created by different academic groups, industrial organizations and enterprises on behalf of their own benefit. As a result, IoT was initially a set of isolated networks of smart things as few developer know the whole spectrum of technologies related to these ad-hoc systems.

One major perspective to solve the heterogeneity issue is pushing standards. At the hardware level, IEEE 1451.4 provides a Transducer Electronic Data Sheet (TEDS) standard for transducers[22]. TEDS compatible sensors can self-identify and provide metadata about itself, such as manufacturer, version, serial number, type and parameters for translating and interpreting its raw readings of physical quantities such as temperature, air pressure etc.

The Open Geospatial Consortium (OGC) proposes another standard called Sensor Model Language, a.k.a SensorML[23]. The primary goal of SensorML is to reduce heterogeneity of different sensor data structure thus enabling interoperability. SensorML allows sensor providers to describe their data formats with a XML-based specification so that sensor data can be translated into understandable formats.

Although these standards are effective in eliminating heterogeneity of sensor data, they care more on the data collection side. In other words, they provide a tool mainly for data producers. As for data consumers, the difference made by these standards is reducing the set of protocols they have to master. Yet what data consumers really need is an efficient way to develop new applications or services based on a large set of sensor data.

There are a number of efforts, which have been made to focus more on data consumer side. As a pioneer, TinyDB[24] helps data consumers access sensor data by using SQL-like queries and push the computation of these queries inside a sensor network. Though TinyDB is mainly focusing on private homogeneous sensor networks, it shows the feasibility of the concept of an on-demand provisioning of sensor data.

Open data sharing platform is another mainstream initiative for on-demand sensor data provisioning. The earliest platform is SensorBase[9] built by the Center For Embedded Networked Sensing (CENS) of UCLA. It is a centralized sensor data storage and management platform, which allows users publish and share their sensor data after defining data streams and access control policies. For data consumers, SensorBase allows users to query data streams using SQL-centric APIs. Users can add filters to data selection in SQL language on location, type, time range, or other relevant fields in the relational database table.

Similar sensor data sharing platforms for research purposes include Sensorpedia[25] by Oak Ridge National Laboratory (ORNL), Microsoft's SenseWeb[26], Nokia's SensorPlanet[27], Sun's Sensor.Network[28] and SAP's notion of Sensor Data as a Service (SDaaS)[11]. All these platforms provide web-based APIs for data producers to upload and manage sensor data and for data consumers to retrieve data. In particular, both SenseWeb and SDaaS mainly adopt a SOA centric interface in order to support data retrieval and SOA based service composition. Sensor.Network implements a light-weight architecture to offer RESTful web service APIs.

There are also commercial sensor data sharing platforms; representative examples are TempoDB[29], Xively[12], Wikisensing[30] and SensorCloud[14]. These platforms share some common features such as focusing on storing time-indexed sensor data and providing REST APIs to facilitate storing, retrieving, and querying time series data.

Besides, some of those commercial platforms like SensorCloud provide the ability to visualize sensor data. Users can also run data analysis applications on the platform.

In summary, the above mentioned related work all focus on providing sensor data provision support, which means that client applications can only acquire raw sensor data with little processing provided by the platforms. Thus they typically lack of support to facilitate client end application development.

Yet another work Global Sensor Networks (GSN)[8] enables users to define new sensor data streams called virtual sensor data streams which are combinations of existing data streams collected from physical sensors. However, the aggregation method provided by GSN is based on SQL language whose expressiveness is limited by the expressibility of SQL.

III. The MiWoT Platform

We first presents MiWoT, a cloud-based programmable sensor data provision platform. At the center of MiWoT is the proposed programming support mechanism *FilterCombine*, which will be described in the next section. MiWoT stands for "middleware for web of things", which is designed as a centralized sensor sharing platform. With MiWoT, heterogeneous IoT smart devices could be connected to fulfill the open data initiative. MiWoT is a sensor data sharing platform with both GUI support for user configuration and RESTful web service APIs. By leveraging the RESTful APIs offered by MiWoT, users could just provide the format of their sensor data streams and then wire sensors in. Meanwhile, users could also develop innovative services and application upon these shared sensor data.

A. Architectural Overview

Figure 2 shows an overview of the MiWoT architecture. Based on smart devices, MiWoT offers both an sensor data sharing interface and an application-oriented programming support for developers.

There are mainly three layers in the MiWoT architecture: `Device Abstraction`, `Service Mapper` and `Service Provider`. These three layers can handle invocations from application and map them to invocations of services on devices and calculate the final results or perform actions as defined by application developers. Besides these layers of components, MiWoT offers two management interfaces called `Device Manager` and `Service Manager` for application developers to accomplish the work of managing devices and services. Next, we describe each part of MiWoT in a bottom-up sequence.

Device Heterogeneous devices are connected to MiWoT, such as sensor nodes 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, or even sensors in our smartphones. Device types are not limited to this list. 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 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 wraps 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.

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

Device Manager and Service Manager Application developers can add new devices, update or delete existing devices with **Device Manager**. This GUI component also allows developers to monitor the list of online sensors and the running status of each device. The **Service Manger** offers developers

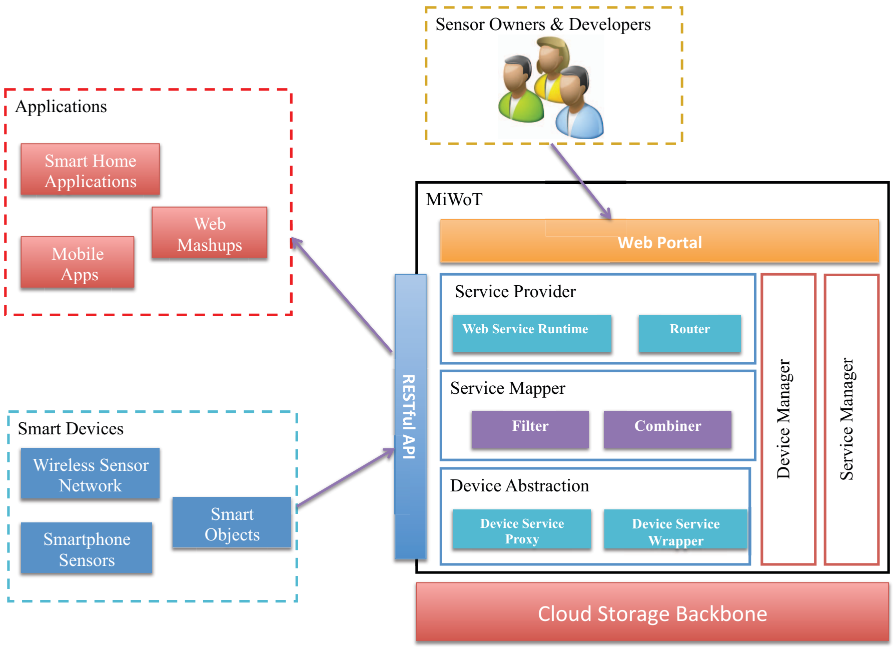


Fig. 2: An overview of the MiWoT architecture

the ability to define new application-oriented services by providing the `Filter` and `Combiner` code snippets. Besides, developers can edit or search services using this module.

B. Key Data Abstraction

The key data abstraction in MiWoT is transforming physical sensor readings into data streams. In order to upload and store sensor data on the platform, data producers, i.e. sensor owners have to register the corresponding data streams first. A data stream is a time series of one or more data readings sampled by the same sensor, for example the temperature readings. Meanwhile, a data stream also represents the concept seen by data consumers. Data consumers can search, query and retrieve these data streams in order to build applications to meet their various usage scenarios.

The following metadata must be specified when a data stream is introduced to MiWoT for the first time.

- **Identifier** Each data stream should provide this attribute to uniquely identify itself, such as IRIS001.
- **Description** A brief description of the data stream specified in natural language. In fact, this attribute is reserved for human recognition, not for any machine-related calculation.
- **Location** The exact location of the corresponding sensor of the data stream must be specified in a `<latitude, longitude>` pair.
- **Environment Tags** A data stream should be associated with one or more Environment Tags to describe those environment in which it fits in, for example, Alice's Car, Bob's Office, ConferenceRoom 101 and so on. This attribute is specified in natural language as well. The tags could be picked from existing ones.
- **Sampling Period** MiWoT will generated data readings corresponding to this attribute. Besides, MiWoT could detect devices' status using this attribute. It should be specified in the unit of milliseconds.
- **Data Value** A data stream must have a valid data value. The data value stands for a type of data the data stream can offer. More specifically, the data value contains a sensor type, a data type and a unit. For example, the temperature value of an IRIS mote has a temperature sensor type, a float data type and a unit of Celsius. All these fields could be selected in a supported set provided by MiWoT as drop-down lists.
- **Permissions** Permissions of a data stream must be specified in order to let MiWoT enforce its role-based access control on sensor data.

C. Data Stream Manipulation APIs

Once a data stream is established, MiWoT will assign an URI for it. In the meantime, a Representational State Transfer (REST) style[31] web API set is also activated for this URI. Authorized users could upload or retrieve sensor data using this API set.

REST style APIs, or RESTful APIs, is a light-weight implementation of web services that reuses the GET, POST, PUT, DELETE methods of standard HTTP protocol. RESTful APIs with JavaScript Object Notation (JSON)[32] as data representation is a lightweight replacement of traditional Simple Object Access Protocol (SOAP)[33] and XML-based web services since it reduces the parsing and encapsulating overhead. The

TABLE I: Basic Data Stream Manipulation APIs

URI \ HTTP Method	GET	POST
`http://miwot-platform/datastream/<id>`	Response is an JSON object which includes the complete metadata of the corresponding sensor data stream specified by id. As shown in Example 1.	Invalid
`http://miwot-platform/datastream/<id>/data`	Response is a JSON array includes all data readings of a specific sensor data stream. As shown in Example 2.	Upload this method to upload new sensor readings of a specific sensor data stream in JSON format.

combination of RESTful APIs and JSON data format has become a de facto standard for open APIs on the web.

Table I shows the detail of the basic RESTful data stream manipulation APIs, which mainly focus on sensor data upload and retrieval. However, the `FilterCombine` mechanism will introduce a new API format. We will cover it in the next section.

Example 1: An example of metadata querying response

```
{
    "id": "Office101Sensor1",
    "description": "First temperature sensor in
        Office101."
    "latitude" : "116.32298703399",
    "longitude" : "39.983424051248",
    "tags" : "office 101|temperature|science
        building 1|Peking University",
    "samplingPeriod": "1000",
    "dataValue": {
        "sensorType": "temperature",
        "dataType": "float",
        "unit": "celsius"
    }
}
```

Example 2: An example of data upload and retrieval message

```
[
        {
          "id": "Office101Sensor1",
           "timeStamp": "1415935877.21"
           "reading" : "26.2",
        },
        {
          "id": "Office101Sensor1",
           "timeStamp": "1415935878.21"
           "reading" : "25.7",
        }
]
```

IV. THE *FilterCombine* MECHANISM

Figure 3 illustrates the key idea of *FilterCombine*, which represents our core data preprocessing mechanism in the programmable sensor provision platform MiWoT. As depicted in Figure 3, sensor streams have many features binding with them. By letting application developers define a `Filter` as well as a `Combiner` in code snippets, *FilterCombine* will screen out a subset of sensor data streams into an intermediate data collection and combine these data streams into one desired data stream.

FilterCombine enables a different perspective on sensor data for sensor data consumers, i.e. application developers. Originally, as other sensor data sharing platforms do, MiWoT only provides a set of sensor data manipulation APIs. These APIs mainly support sensor data insertion initiated from smart devices and sensor retrieval performed by application developers. This scheme is more like a simple accessing wrapper on raw sensor readings, which enables data-oriented provisioning.

However, the *FilterCombine* mechanism provides more flexibility to data consumers, which enables application-oriented on-demand sensor data provisioning. This is more suitable to application developers in real scenarios as getting the full set of raw data streams is both network-intensive and bringing extra handling work for application developers. *FilterCombine* also promotes code reuse in the sense that those previously defined `Filters` and `Combiners` could be shared among different developers.

A. A Motivating Example

Consider an example in which the application needs to get the average temperature of a whole building to indicate its heating status in winter. Suppose that there are hundreds of sensors providing temperature reading of different locations in the building. Without MiWoT, the programmer have to manage to get each temperature reading separately on all those sensors and then average them in the application logic. However, with the support of MiWoT, the developer can create a data stream of the building's average temperature in a *FilterCombine* way.

Application developers first need to define the `Filter` as well as the `Combiner` code snippets on a GUI interface provided by MiWoT. Both the `Filter` and the `Combiner` code snippets have their own programming rules.

B. The `Filter`

There are two parts in defining a `Filter`: the unique identifier and the `Filter` code snippet.

The unique identifier is a string to indicate the `Filter`'s intention. For the motivating example, the identifier can be defined as "scienceBlding1TempSensors".

The `Filter` code snippet takes an object of `DataStream` class as parameter and its return value is a `DSCollection` array, which stands for data stream collection. The Filter component of MiWoT will traverse all existing data streams executing this function and finally get a `DSCollection` object, which contains all data streams wanted. Obviously, in function logic, the developer should express the conditions of attributes that a data stream must

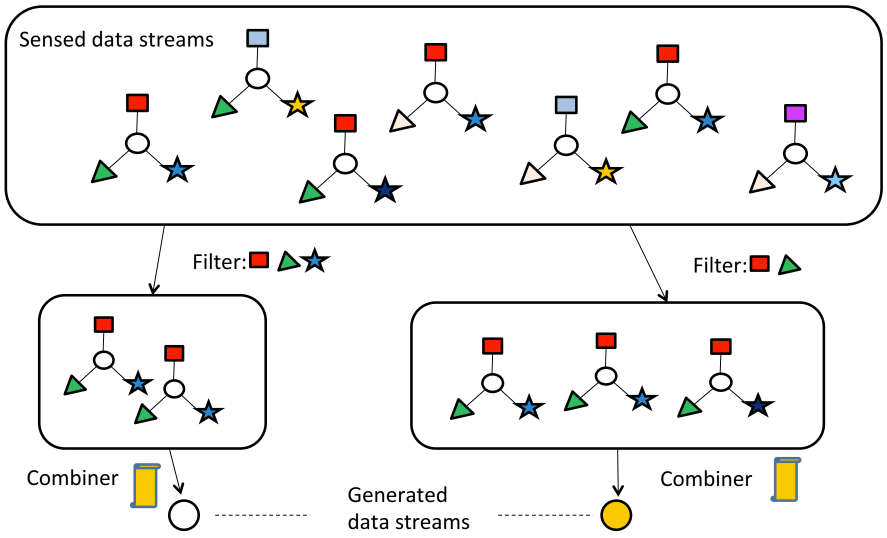


Fig. 3: The *FilterCombine* Mechanism

satisfy and return qualified data streams. As for the motivating example, The `Filter` code snippet is shown in Example 3.

Example 3: An example of the `Filter` code snippet in PHP

```
public function Filter($dataStream){
  array $DSCollection=new array{};
  if($dataStream->sesnorType  == DSSenorTypes::
      Temperature) {
   foreach $tag in $dataStream->tags
     if($tag == "science building 1")
       $DSCollection.push($dataStream);
   }
   return $DSCollection;
}
```

C. The `Combiner`

Definition of a `Combiner` also contains two parts: the unique identifier and the `Combiner` code snippet.

Just like in defining the `Filter`, the unique identifier of a `Combiner` is a string to indicate the `Combiner`'s intention, for the motivating example, the identifier can be "average".

Based on the result of a `Filter`, the `Combiner` code snippet takes a `DSCollection` object as input and returns a new data stream value. In the function logic, a developer specifies how to map various data streams in the `DSCollection` into a single data stream. the `Combiner` code snippet will be executed by the Combiner component in MiWoT. The `Combiner` code snippet of the motivating example is shown in Example 4 , which calculates the average value of all temperature readings in a whole building.

Example 4: An example of the `Combiner` code snippet in PHP

```
public function Combiner($DSCollection){
  $sum = 0;
  $count = 0;

  foreach dataStream in $DSCollection
  {
    $sum += dataStream.reading;
    $count++;
  }

  if($count == 0)
    return null;
  else
    return ($sum/$count);
}
```

We can observe from this example that a `Combiner` does not necessarily need be bound to a specific `Filter`. For example, the "average" `Combiner` can be applied to any data stream collections which need to be averaged. Thus the *FilterCombiner* can significantly boost code reuse from two aspects. 1) Both `Filters` and `Combiners` can be reused by future applications; and 2) `Filters` and `Combiners` can be freely combined as to support different usage scenarios.

One may argue that the *FilterCombiner* mechanism can also be accomplished with the existing SQL-based composition mechanisms. For example, there is a simple pre-defined `avg` aggregator in SQL language to calculate the average of a group of data streams. However, with the *FilterCombiner* mechanism, application developers can express almost unlimited processing capabilities on a group of data streams as the `Filter` and the `Combiner` code snippets are written in a full-blown scripting language. The semantic of the preprocessing is not confined to the expressibility of the SQL language.

Another issue that needs clarification here is that although data streams in a `DSCollection` are filtered according to the same `filter`, the sampling periods of them may still differ significantly. The question is how to merge these data streams with different sampling periods. In the current design of MiWoT, we choose to use a "minimum sampling period" among a specific `DSCollection` as the sampling period of the generate data stream. We can of course use other time window based algorithms to deal with this. We leave this issue to future work.

D. The FilterCombiner API

It must be noted that the configuration interface for data consumers to specify both the `Filter` and the `Combiner` is through a GUI component on the web portal of the MiWoT platform. Once configured, a dedicated RESTful API is automatically activated by MiWoT. The *FilterCombiner* API can be identified through the combination of the identifiers of the `Filter` and the `Combiner`, the URI format is:

```
http://miwot-platform/datastream/<filter id>/<
    combiner id>
```

By using the GET HTTP method with this URI, MiWoT will return the data stream values generated by the combination of the specified `Filter` and `Combiner`. The format of the response message is just like normal data retrieval results as shown in Example 2.

All other HTTP methods performed on this URI will return error message as a *FilterCombiner*-generated data stream does not support data insertion.

As an example, the URI of the *FilterCombiner* API of the motivating example in Section IV-A is shown below.

```
http://miwot-platform/datastream/
    scienceBlding1TempSensors/average
```

V. Implementation

A. Implementation Environment

We build a small IoT testbed in our lab with several types of sensors as well as actuators, which belongs to part of our "Smart Lab" project. In this paper, we wired in all the sensors in our testbed to the MiWoT prototype system. Here is a brief introduction to these sensors.

IRIS Motes. As shown in Figure 4a, IRIS is a WSN mote manufactured by MEMSIC. In our experiments, ten IRIS motes are distributed in our office to acquire environment context data, including indoor temperature and humidity continuously. Each of the motes runs TinyOS 2.1 operating system and has network connections wire a base station.

RFID Readers and Tags. As shown in Figure 4b, these radio-frequency based tags are carried by our lab members. When a tag, which stands for the man who carries it, enters the laboratory, the RFID reader will detect the event in real time. The RFID reader is connected to a PC as a gateway via the ModBus protocol.

B. Enabling Technologies

We adopted a Model-View-Controller architecture when implementing the MiWoT platform and the *FilterCombiner* mechanism. Key enabling technology and programming frameworks involved are: Relational Database, the MVC programming framework, the Responsive Front-end framework.

Relational database. Although NoSQL database prevails nowadays, in this project we select table-oriented traditional relational database as our sensor data repository. The reason we choose the old-school database is because key-value based database is efficient in handling heterogeneous data on one hand, but it is slow when handling random data selection with criteria on some specific fields. Meanwhile, NoSQL databases emphasize more on the size of the storage it can handle rather than the availability and fault tolerance on the data. As we need to implement the the *FilterCombiner* mechanism which contains a lot random selections on data repository, we choose MySQL as our database.

The MVC programming framework. There are a lot of MVC frameworks available in various programming languages such as Django on Python, Ruby on Rails, CodeIgniter on PHP and so on. In this project, we choose CodeIgniter[34] as it supports rapid development with rich pre-defined classes and libraries and PHP is among the oldest optimized scripting languages specialized in web development. Of course, the *FilterCombiner* is also realized in PHP. Besides, CodeIgniter has also integrated the ActiveRecord which provides an abstract layer of SQL language to perform sanity checks on queries and transaction management.

Responsive front-end framework. We choose Twitter's Bootstrap framework[35] for our web portal design and implementation. Bootstrap provides a rich HTML, JavaScript and CSS-based design templates for web pages, typography, forms, buttons, navigation and other dynamic interface components. It is an easy-to-use toolset to rapidly develop professional web interfaces for programmers who are not good at art design.

In Figure 5, we present some representative screenshots in MiWoT. Figure 5a shows the homepage of MiWoT in which we list a table of all available sensors and its metadata. Figure 5b and 5c shows the graphical interface to define a `Filter` and a `Combiner`, respectively.

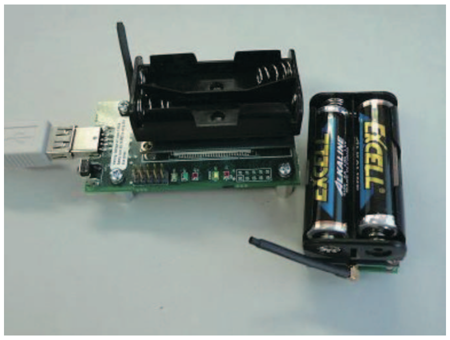
VI. Conclusion

In this paper, we introduce the idea of cloud-based programmable sensor data provision, in which we propose a new sensor data programming support framework called *FilterCombine*. *FilterCombine* allows developers to define sensor data processing logic in two easy steps: filters and combiners. The `Filter` indicates which subset of sensor data the application desires while the `Combiner` expresses how these selected data will be combined into a meaningful result to the application. As the filters and combiners are generic enough, they can be reused among different developers to eliminate duplicated coding efforts. We have implemented the *FilterCombine* mechanism in a sensor data sharing platform prototype MiWoT, which demonstrates our idea of cloud-based programmable sensor data provision.

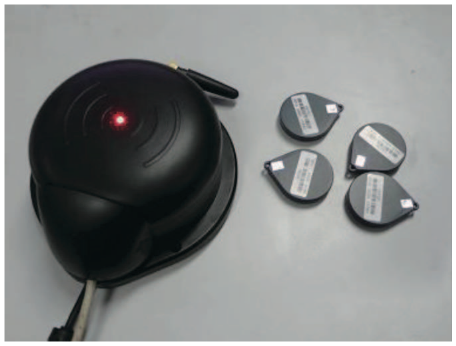
Our future work includes improving the functionalities of the MiWoT data sharing platform, developing API standards for filters and combiners, and developing new sensor applications.

Acknowledgment

This work is supported in part by the High-Tech Research and Development Program of China under Grant No.2013AA01A605, the National Natural Science Foundation of China under Grant No.61421091 and 61103026.

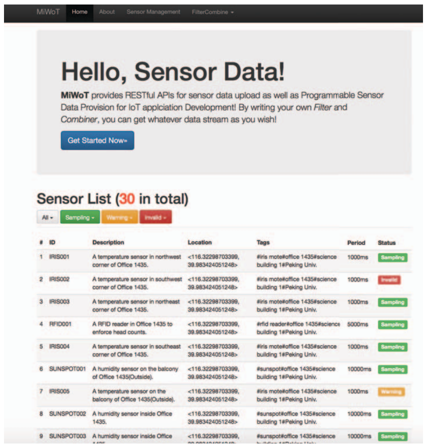


(a) IRIS Motes



(b) RFID Reader and Tags

Fig. 4: Sensors for MiWoT Prototype Implementation



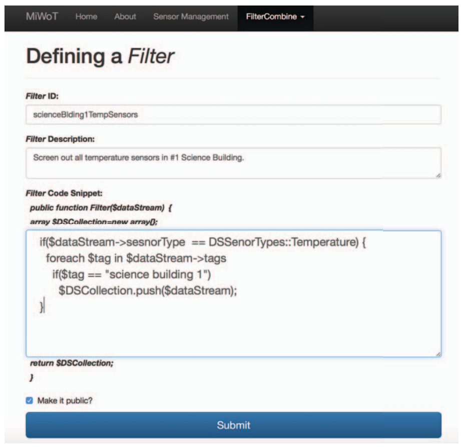
Hello, Sensor Data!

MiWoT provides RESTful APIs for sensor data upload as well as Programmable Sensor Data Provision for IoT applciation Development! By writing your own *Filter* and *Combiner*, you can get whatever data stream as you wish!

Sensor List (30 in total)

#	ID	Description	Location	Tags	Period	Status
1	IRIS001	A temperature sensor in northwest corner of Office 1435.	<116.32298703399, 39.983424051248>	#iris mote#office 1435#science building 1#Peking Univ.	1000ms	Sampling
2	IRIS002	A temperature sensor in southwest corner of Office 1435.	<116.32298703399, 39.983424051248>	#iris mote#office 1435#science building 1#Peking Univ.	1000ms	Invalid
3	IRIS003	A temperature sensor in northeast corner of Office 1435.	<116.32298703399, 39.983424051248>	#iris mote#office 1435#science building 1#Peking Univ.	1000ms	Sampling
4	RFID001	A RFID reader in Office 1435 to enforce head counts.	<116.32298703399, 39.983424051248>	#rfid reader#office 1435#science building 1#Peking Univ.	5000ms	Sampling
5	IRIS004	A temperature sensor in southeast corner of Office 1435.	<116.32298703399, 39.983424051248>	#iris mote#office 1435#science building 1#Peking Univ.	1000ms	Sampling
6	SUNSPOT001	A humidity sensor on the balcony of Office 1435(Outside).	<116.32298703399, 39.983424051248>	#sunspot#office 1435#science building 1#Peking Univ.	10000ms	Sampling
7	IRIS005	A temperature sensor on the balcony of Office 1435(Outside).	<116.32298703399, 39.983424051248>	#iris mote#office 1435#science building 1#Peking Univ.	1000ms	Warning
8	SUNSPOT002	A humidity sensor inside Office 1435.	<116.32298703399, 39.983424051248>	#sunspot#office 1435#science building 1#Peking Univ.	10000ms	Sampling
9	SUNSPOT003	A humidity sensor inside Office	<116.32298703399,	#sunspot#office 1435#science	10000ms	Sampling

(a) Sensor List



Defining a *Filter*

***Filter* ID:** scienceBlding1TempSensors

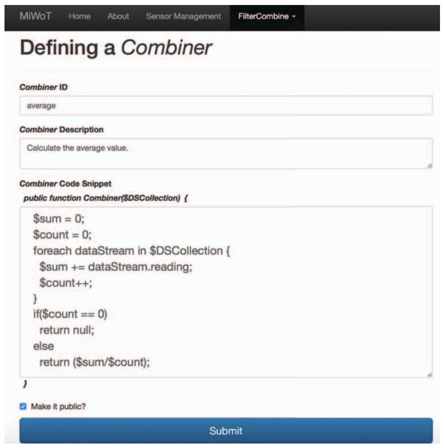
***Filter* Description:** Screen out all temperature sensors in #1 Science Building.

***Filter* Code Snippet:**

```
public function Filter($dataStream) {
array $DSCollection=new array();
  if($dataStream->sesnorType  == DSSenorTypes::Temperature) {
   foreach $tag in $dataStream->tags
     if($tag == "science building 1")
       $DSCollection.push($dataStream);
  }
return $DSCollection;
}
```

☑ Make it public?

(b) Defining a `Filter`



Defining a *Combiner*

***Combiner* ID** average

***Combiner* Description** Calculate the average value.

***Combiner* Code Snippet**

```
public function Combiner($DSCollection) {
  $sum = 0;
  $count = 0;
  foreach dataStream in $DSCollection {
    $sum += dataStream.reading;
    $count++;
  }
  if($count == 0)
    return null;
  else
    return ($sum/$count);
}
```

☑ Make it public?

(c) Defining a `Combiner`

Fig. 5: Screenshots of MiWoT

REFERENCES

[1] (2014, May) Santander: The smartest smart city. [Online]. Available: http://www.governing.com/topics/urban/gov-santander-spain-smart-city.html

[2] Gartner. (2014, March) Gartner says the internet of things will transform the data center. [Online]. Available: http://www.gartner.com/newsroom/id/2684616

[3] A. for Computing Machinery U.S. Public Policy Committee, "Recommendations on open government," 2012. [Online]. Available: http://www.acm.org/public- policy/open-government

[4] O. D. risp Community, "Open data decalogue," 2012. [Online]. Available: http://red.gnoss.com/en/community/OpenData/

[5] A. Domingo, B. Bellalta, M. Palacin, M. Oliver, and E. Almirall, "Public open sensor data: Revolutionizing smart cities," *IEEE Technology and Society Magazine*, vol. 32, no. 4, pp. 50–56, winter 2013.

[6] (2014, August) From sensors to big data: Chicago is becoming a smart city. [Online]. Available: http://smartdatacollective.com/bigdatastartups/229281/sensors-big-data-chicago-becoming-smart-city

[7] W. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao, "Senseweb: An infrastructure for shared sensing," *IEEE MultiMedia*, vol. 14, no. 4, pp. 8–13, Oct 2007.

[8] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," in *2007 International Conference on Mobile Data Management*, May 2007, pp. 198–205.

[9] G. Chen, N. Yau, M. Hansen, and D. Estrin, "Sharing sensor network data," Center for Embedded Networked Sensing, UCLA, Tech. Rep., March 2007.

[10] P. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "Irisnet: an architecture for a worldwide sensor web," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 22–33, Oct 2003.

[11] J. Zhang, B. Iannucci, M. Hennessy, K. Gopal, S. Xiao, S. Kumar, D. Pfeffer, B. Aljedia, Y. Ren, M. Griss, S. Rosenberg, J. Cao, and A. Rowe, "Sensor data as a service – a federated platform for mobile data-centric service development and sharing," in *2013 IEEE International Conference on Services Computing (SCC)*, June 2013, pp. 446–453.

[12] Xively. [Online]. Available: http://xively.com

[13] Wikisensing. [Online]. Available: http://wikisensing.org/

[14] Sensor cloud. [Online]. Available: http://www.sensorcloud.com/

[15] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[16] H.-D. Ma, "Internet of things: Objectives and scientific challenges," *J. Comput. Sci. Technol.*, vol. 26, no. 6, pp. 919–924, Nov. 2011.

[Online]. Available: http://dx.doi.org/10.1007/s11390-011-1189-5

[17] P. Pereira, J. Eliasson, R. Kyusakov, J. Delsing, A. Raayatinezhad, and M. Johansson, “Enabling cloud connectivity for mobile internet of things applications,” in *2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, March 2013, pp. 518–526.

[18] G. Li, Q. Wei, L. Li, Z. Jin, Y. Xu, and L. Zheng, “Environment based modeling approach for services in the internet of things.” *SCIENCE CHINA Information Sciences*, vol. 43, no. 10, p. 1198, 2013.

[19] Advanced messaging queuing protocol. [Online]. Available: http://www.amqp.org/

[20] The modbus protocol. [Online]. Available: http://www.modbus.org/

[21] Gateway control protocol. [Online]. Available: http://en.wikipedia.org/wiki/H.248

[22] An overview of ieee 1451.4 transducer electronic data sheets. [Online]. Available: http://standards.ieee.org/develop/regauth/tut/teds.pdf

[23] The sensor modeling language. [Online]. Available: http://www.opengeospatial.org/standards/sensorml

[24] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “Tinydb: An acquisitional query processing system for sensor networks,” *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005.

[25] B. Gorman, D. Resseguie, and C. Tomkins-Tinch, “Sensorpedia: Information sharing across incompatible sensor systems,” in *International Symposium on Collaborative Technologies and Systems, 2009. CTS ’09.*, May 2009, pp. 448–454.

[26] S. Nath, J. Liu, J. Miller, F. Zhao, and A. Santanche, “Sensormap: A web site for sensors world-wide,” in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys ’06. New York, NY, USA: ACM, 2006, pp. 373–374. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182861

[27] N. Corporation. Sensor planet. [Online]. Available: http://www.sensorplanet.org

[28] V. Gupta, P. Udupi, and A. Poursohi, “Early lessons from building sensor.network: an open data exchange for the web of things,” in *The 8th IEEE International Conference on Pervasive Computing and Communications Workshops*, March 2010, pp. 738–744.

[29] Tempodb. [Online]. Available: http://tempo-db.com/features

[30] D. Silva, M. Ghanem, and Y. Guo, “Wikisensing: An online collaborative approach for sensor data management,” *Sensors*, vol. 12, no. 10, pp. 13 295–13 332, 2012. [Online]. Available: http://www.mdpi.com/1424-8220/12/10/13295

[31] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, Irvine, 2000.

[32] Javascript object notation. [Online]. Available: http://www.json.org/

[33] Simple object access protocol. [Online]. Available: http://www.w3.org/TR/soap/

[34] Codeigniter. [Online]. Available: https://ellislab.com/codeigniter

[35] Bootstrap. [Online]. Available: http://getbootstrap.com/