DESIGN AND IMPLEMENT AN INTEROPERABLE INTERNET OF THINGS APPLICATION BASED ON AN EXTENDED OGC SENSORTHINGS API STANDARD

C. Y. Huang [a, *], C. H. Wu [a]

[a] Center for Space and Remote Sensing Research, National Central University, Taiwan – cyhuang@csrsr.ncu.edu.tw, 103322089@cc.ncu.edu.tw

Commission IV, WG IV/6

KEY WORDS: Internet of Things, Interoperable, OGC SensorThings API

ABSTRACT:

The Internet of Things (IoT) is an infrastructure that interconnects uniquely-identifiable devices using the Internet. By interconnecting everyday appliances, various monitoring and physical mashup applications can be constructed to improve people's daily life. However, IoT devices created by different manufacturers follow different proprietary protocols and cannot communicate with each other. This heterogeneity issue causes different products to be locked in multiple closed ecosystems that we call *IoT silos*. In order to address this issue, a common industrial solution is the hub approach, which implements connectors to communicate with IoT devices following different protocols. However, with the growing number of proprietary protocols proposed by device manufacturers, IoT hubs need to support and maintain a lot of customized connectors. Hence, we believe the ultimate solution to address the heterogeneity issue is to follow open and interoperable standard. Among the existing IoT standards, the Open Geospatial Consortium (OGC) SensorThings API standard supports comprehensive conceptual model and query functionalities. The first version of SensorThings API mainly focuses on connecting to IoT devices and sharing sensor observations online, which is the *sensing capability*. Besides the sensing capability, IoT devices could also be controlled via the Internet, which is the *tasking capability*. While the tasking capability was not included in the first version of the SensorThings API standard, this research aims on defining the tasking capability profile and integrates with the SensorThings API standard, which we call the *extended-SensorThings API* in this paper. In general, this research proposes a lightweight JSON-based web service description, the "Tasking Capability Description", allowing device owners and manufacturers to describe different IoT device protocols. Through the extended-SensorThings API, users and applications can follow a coherent protocol to control IoT devices that use different communication protocols, which could consequently achieve the interoperable Internet of Things infrastructure.

1. INTRODUCTION

1.1 Background

The Internet of Things (IoT) has been attracting attentions from various fields in recent years. While the Internet provides the global and pervasive connectivity infrastructure, the IoT concept was proposed to connect everyday devices to the Internet. The Internet of Things (IoT) is an infrastructure that interconnects uniquely-identifiable devices using the Internet. At its early stage, the IoT concept mainly focused on the identification and tracking of physical things. Technologies like the bar code and Radio Frequency Identification (RFID). Many RFID-based applications were proposed, such as warehouse management and logistic applications.

However, in recent years, with the advance of communication and sensor technologies, the Internet of Things is no longer confined to the applications of object identification. Everyday appliances (e.g., TV, oven, heater, lamp, door lock) can be connected to the Internet via different local communicating technologies (e.g., Wifi, Bluetooth, and Zigbee). By connecting devices to the Internet, two main IoT capabilities can be realized, which are the *sensing* and *tasking* capabilities.

The sensing capability of IoT devices allows users to monitor the device status as well as the environmental properties of their surroundings, such as air temperature, humidity, and air quality. On the other hand, the tasking capability of IoT devices provides services for users to control the devices and execute feasible tasks. Most importantly, since IoT devices are connected with the Internet, both the sensing and tasking capabilities can be achieve in a remote and real-time manner. Because of these two main capabilities of the IoT, many applications has been proposed or envisioned, such as smart home, smart city, smart agriculture, industry and logistics (Atzori, 2010).

1.2 The Internet of Things definition and architecture

Before introducing the target problem of this research, we define the IoT definition and architecture first to help clearly explain the scope of this research.

In terms of the definition of IoT, the International Telecommunication Union (ITU) defined the IoT as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" (ITU 2005). We believe this definition is clear and general enough to cover different potential architecture of the IoT. Hence, this paper follows this IoT definition.

In terms of the IoT architecture, we generalize it into four layers, including the device, gateway, web service, and application layers. First of all, the device layer contains the IoT devices that can provide sensing and/or tasking capabilities. While some of the devices have enough computation capability to connect to web services by themselves, most of IoT devices

are resource-constraint (in terms of computation and/or power supply) devices and cannot communicate via the Internet directly. For the resource-constraint devices, gateways in the gateway layer can help these devices to communicate with the web service layers (Bormann et al., 2014). The web services in the web service layer act as the data services and the intermediary between the devices and applications. While web services can host IoT sensor data for applications to retrieve (i.e., the sensing capability), web services can also forward tasking commands from applications to the IoT devices (i.e., the tasking capability). Finally, the application layer contains the applications that connect with the web service layer to utilize the sensor data and/or controllable capabilities from IoT devices.

1.3 Problem and objective

While the IoT is attracting attention from various field and many manufacturers have produced different Internet-connected devices. IoT devices created by different manufacturers follow different proprietary protocols and cannot communicate with each other. This heterogeneity issue causes different products to be locked in multiple closed ecosystems, which include proprietary device, gateway, web service, and applications. We define these closed ecosystems as the *IoT silos*.

In order to address this IoT silo issue, a common industrial solution is the *hub* approach, which implements multiple connectors to communicate with IoT devices following different protocols. Alphabet (or Google) Nest, Apple HomeKit, and Sentri are some examples of this IoT hub solution.

While the hub approach can effectively address the IoT silo issue, we argue that this solution still faces serious heterogeneity problem. With the growing number of proprietary protocols proposed by different device manufacturers, IoT hubs need to support and maintain a lot of customized connectors. Maintaining up-to-date connectors to communicate with every IoT devices may not be a realistic solution.

Another approach to solve the IoT silo issue is to define and follow open standards. Among the existing IoT standards, the Open Geospatial Consortium (OGC) SensorThings API standard supports comprehensive conceptual model and query functionalities. The first version of SensorThings API mainly focuses on connecting to IoT devices and sharing sensor observations online, which is what we call the *sensing capability*. By following open standards, the communication between layers can be unified to achieve interoperability. To be specific, with the SensorThings API, IoT devices and gateways can upload sensor observations to web services by following the same protocol. Applications can also retrieve observations generated by different devices from web services only by following the SensorThings API. In this case, no customized connectors are required.

However, while the first version of the SensorThings API only defines the data model and communication protocol for the sensing capability. This research tries to extend the SensorThings API by proposing the data model and protocol of IoT tasking capability. To be specific, this research tries to propose a solution that can be integrated with the existing SensorThings API and provide a uniform web service interface for users to control different IoT devices.

While the nature of the sensing and tasking capabilities are different, the required functionalities of these two capabilities are different as well. As the sensing capability can simply serve as a data service, the tasking capability needs to understand the communication protocols of different IoT devices in order to forward the tasking commands to them. One of the key design decisions this research made is to allow manufacturers to design device protocols as long as the protocols can be described by a uniform service protocol description standard. By defining the service protocol description standard, we can extend the SensorThings API service to automatically translate users' tasking commands into device protocol commands.

In general, the proposed solution can achieve two major contributions: (1) manufacturers can design different IoT device protocols, and (2) users/applications can control different IoT devices with a uniform service interface even if the devices follow different device protocols.

2. METHODOLOGY

2.1 Tasking capability description

As we mentioned earlier, the tasking capability of IoT devices allow users/applications to remotely control the devices. In this case, the IoT devices act similar to a web service allowing clients to communication with the devices via the Internet. While this research allows manufacturers to design their own proprietary device protocols, we need a standard description document that can describe any possible device protocols that manufacturers may create. In this case, one of the main objectives of this research is to define a web service protocol description format, which we call the *tasking capability description*.

While most of existing web service descriptions are based on XML format (Chinnici et al, 2007; Kopecky et al, 2008), they may not be suitable for resource-constraint IoT devices. Hence, this research tries to implement a JSON-based tasking capability document. First, we design the necessary elements to describe possible device protocols. Table 1 shows the main elements and their descriptions.

Table 1. The elements of tasking capability description

Element	Description
TaskingCapability	A Primary key for identifying the TaskingCapability.
Thing	A Primary key for identifying the Thing that provides the TaskingCapability, which can be integrated with the SensorThings API's Thing entity.
Description	A human-readable description for the TaskingCapability.
Parameters	A list of settable parameters for this TaskingCapability.
Protocols	A list of available desvice protocols for this TaskingCapability.
Actuator	A Primary key for identifying the Actuator that provides the TaskingCapability.

In the Table 1, "TaskingCapability" is to uniquely identify different tasking capabilities in a web service. While each tasking capability links to one "Thing", one "Thing" could have more than one tasking capabilities. "Description" is a human-readable description to describe the tasking capability. "Parameters" are used to describe the accepted parameters for this tasking capability. The "Protocols" describe the communication protocols that the IoT device supports for this

tasking capability. In the current stage, this research only focus on the HTTP-based protocols. Finally, the "Actuator" is used to identify and describe the actuator used to support this tasking capability.

The "Parameter" of tasking capability contains four main properties for describing every allowed and settable parameter for this tasking capability. The details are shown in the Table 2. The "ParameterID" represents the unique ID of an allowed parameter, which will be needed to identify the input value in users' tasking commands. The "Description" can be used to describe the meaning of the parameter so that user could understand the use of it. The "Use" property is used to describe whether the parameter is "Optional" or "Mandatory". If the "Use" of parameter is "Mandatory", when a user submits a task, the user must include this parameter in the task. Finally, the "Definition" defines the allowed values of the parameter while explaining the data type and the unit of measurement.

Table 2. The properties of "Parameter"

Property	Description
ParameterID	A unique identifier for an allowed parameter.
Description	A human-readable description for the parameter.
Use	The necessity of the parameter, i.e., optional or mandatory.
Definition	The detail definition of the parameter.

Table 3 shows the properties of protocols. In this current study, we focus on the HTTP-based protocol. The possible properties of "Protocols" include HTTP method, resource path, header, message body, query string, and fragment. These properties allow manufacturers to completely describe their proprietary device protocols. By integrating the information specified in the "Protocol", we can compose an HTTP request following the IoT device protocol.

Table 3. The properties of "Protocols"

Property	Description
HTTPMethod	HTTP method.
AbsoluteResourcePath	The path of communication protocol.
MessageBody	Message body.
QueryString	QueryString.
Headers	Headers.
Fragment	Fragment.

In addition, for users/applications to submit a controlling command, we also define the data model for the task. As shown in Table 4, a task needs to specify the tasking capability ID to determinate the corresponding IoT device that the user wants to control. Then, the "Inputs" property allows users to set acceptable as well as necessary parameters and input values of the task. Furthermore, the "Time" property allows users to specify the time that a user wants to execute the task.

Table 4. The properties of "Task"

Property	Description
TaskingCapability	The Primary key for identifying the TaskingCapability
Inputs	The parameters and input values
Time	The executing time

2.2 Tasking capability workflow

The tasking capability description allows manufacturers to describe different device protocols in a uniform format. This description document needs to be understood by a web service so that the web service knows how to communicate with the devices. In this research, we try to extend the existing SensorThings API web service by implementing the proposed tasking capability profile. In this case, the extended SensorThings API is able to support both sensing and tasking capabilities in a uniform manner.

In general, Figure 4 shows the sequence diagram of the tasking capability procedure. First, users or devices can register IoT devices to the extend SensorThings API service by using the proposed tasking capability description. The information specified in the tasking capability description can help the web service automatically understand the device protocols. Users/Applications can then find the registered tasking capabilities from the web service by following standard SensorThings API protocol. Therefore, users can know the details of any available tasking capabilities, including the TaskingCapabilityID, Description, Parameters, etc.

Users can then create and send a task to the extended SenosrThings API service by setting up the acceptable input values. When the web service receives the task, the web service will parse the task according to the defined tasking capability description. After retrieving the executing time, input parameters and corresponding values, the web service will fill the input values of each parameter into corresponding locations and compose an HTTP request following the device protocol. Finally, the web service will send the device request at the user-specified time.

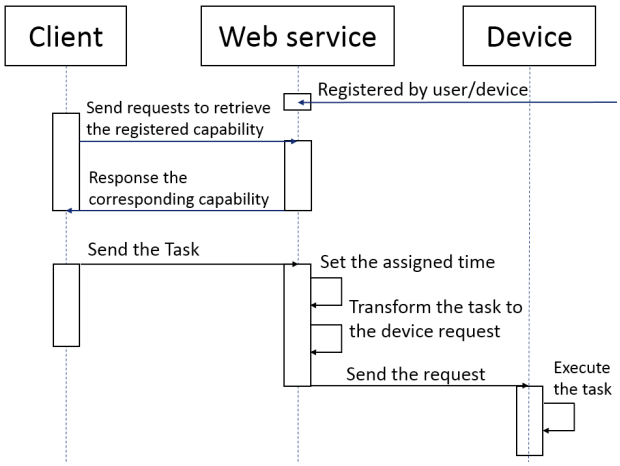

Figure 4. The sequence diagram of proposed web service

3. RESULT

In order to demonstrate the contribution of the proposed solution of this research, we implement an application utilizing the sensing and tasking capabilities provided by the extended SensorThings API. The demonstration application is an automatic dehumidifier that can automatically turn on and off according to the humidity. However, as there have not been any sensor products that can directly upload sensor observations to a SensorThings API service, we use the Arduino UNO as the controller and connect sensors and communication modules to monitor the humidity and upload the observations to the extended SensorThings API service.

On the other hand, regarding the controlling of the dehumidifier, we use a traditional mechanical dehumidifier and connect it to the WeMo smart plug. The WeMo smart plug is an IoT product that can control the provision of electricity for appliances. Therefore, by controlling the WeMo plug, we can turn on and off the dehumidifier remotely.

To register the WeMo smart plug to the extended SensorThings API service, we first identify the device protocol of the WeMo plug and then create a tasking capability description by following the proposed solution. After linking the humidity sensor and WeMo plug to the extended SensorThings API, an automatic dehumidifier application can be implemented. First, the application periodically retrieves the humidity observations from the service, e.g., every 10 minutes. When a new humidity observation is larger than a predefined higher-bound threshold, e.g., 80%, the application will automatically create a task following the properties defined in Table 4 and send the task to the extended SensorThings API to turn on the dehumidifier. Then when a new humidity observation is smaller than a predefined lower-bound threshold, the application will automatically create and send a task to turn off the dehumidifier.

While the automatic dehumidifier application is only one example, the same concept and workflow can be applied to many IoT applications. The extended SensorThings API not only can act as a sensor observation data service for sensors to upload real-time observations, but also can serve as an intermediary for users/applications to control IoT devices.

4. CONCLUSIONS AND FUTURE WORK

In this research, we propose a solution that can effectively address the heterogeneous IoT tasking capability issue. To allow manufacturers to have the flexibility of defining their own proprietary device protocols, we propose the tasking capability description standard that can describe any possible protocols in a uniform format. By combining the proposed solution with the OGC SensorThings API, the extended SensorThings API service is able to automatically translate users' tasks into requests following device protocols. As a result, end users and applications do not need to handle the heterogeneous device protocols and can only follow a single web service interface to communicate with every IoT device.

In addition, by integrating the proposed solution with the OGC SensorThings API standard, the extended SensorThings API can support both IoT sensing and tasking capabilities and provide a comprehensive solution for the IoT web service infrastructure. Overall, we believe this extended SensorThings API has the potential be become the efficient and interoperable IoT infrastructure and realize the IoT vision.

For our future work, we will promote the proposed solution to the OGC SensorThings API standard working group. By collecting opinions from multiple parties, the proposed solution could be revised and standardized.

REFERENCES

Atzori, L., Iera, A., Morabito, G., 2010. The internet of things: A survey. *Computer Networks*, 54(15), pp. 2787–2805

Bormann, C., Ersue, M., Keranen, A., 2014. Terminology for Constrained-Node Networks. *Internet Engineering Task Force, RFC*. 7228.

Chinnici, R., Moreau, J. J., Ryman, A., & Weerawarana, S., 2007. Web services description language (wsdl) version 2.0 part 1: Core language. *W3C recommendation*, 26, 19.

ITU-T, 2012. *Overview of Internet of Things*, Recommendation ITU-TY.2060.

Kopecky, Jonathon, Karthik, G., and Tomas, V., 2008. hrests: An html microformat for describing restful web services. In: *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 1.