Journal of Ambient Intelligence and Humanized Computing
https://doi.org/10.1007/s12652-019-01464-2

ORIGINAL RESEARCH

A generic Internet of things (IoT) platform supporting plug-and-play device management based on the semantic web

Woongsup Kim[1] · Haram Ko[1] · Hyeonjin Yun[1] · Jiae Sung[1] · Seeun Kim[1] · Jiseung Nam[1]

Received: 28 September 2018 / Accepted: 1 August 2019

Abstract
There is a large variety of Internet of things (IoT) devices and their peripherals available in consumer markets, and IoT deployers should work on customizing device drivers that are compatible with their peripherals. Implementing compatible device drivers, however, often requires a burden of work. This paper proposes a generic platform that enables plug-and-play (PnP) integration for sensors and actuators to allow the addition and removal of IoT device peripherals without re-customizing all the device drivers. To this end, we employ IoT ontologies and semantics to represent IoT device characteristics and to infer IoT device behaviors. IoT device behavior is then passed to the generic device driver to cover device-specific operation. Since the generic device driver selectively operates most of the available function calls required in IoT devices, most of the programming work that is normally required for device customization is removed, and management overhead for software installation and maintenance can be minimized. To this end, we employ IoT ontologies and semantics as well as generic programming techniques in the generic platform in order to configure and control IoT devices. In the proposed platform, IoT device characteristics, including I/O functions and configuration rules, are defined using custom-built IoT ontologies, and operational behaviors are inferred through SPARQL queries. The generic platform then passes function-call name and configuration rules corresponding to the newly added peripheral device's specification. The experimental results show that our generic platform covers most of the popular sensors available in the market. Our solution therefore enables a true PnP experience of sensors and actuator peripherals in IoT devices.

Keywords Ontology · Semantic web · IoT · Plug-and-play · SPARQL

1 Introduction

The recent arrival and proliferation of the Internet of things (IoT) has led to the development of infrastructure for capturing and storing data, ranging from small smart devices to smart cities (Mois et al. 2017). In an IoT infrastructure, data collected from a large set of sensors is processed, analyzed, and often fed back to actuators or smart devices to control nearby environments (Chindenga et al. 2016; Suryani et al. 2017; Vanus et al. 2018). The deployment of IoT devices with a richer variety of sensors and actuators can provide fine-grained monitoring and control larger regions with various usage contexts.

IoT is composed of embedded devices equipped with low power sensors and actuators that are connected in wireless networking environments (Yang et al. 2016; Uddin et al. 2018). A large variety of IoT devices and their peripherals are available in consumer markets, and compatible device drivers should be developed for each peripheral (Toosi et al.

✉ Woongsup Kim
woongsup@dongguk.edu

Haram Ko
jjycoco@dongguk.edu

Hyeonjin Yun
dbs5560@dongguk.edu

Jiae Sung
wldo8615@dongguk.edu

Seeun Kim
s95834@dongguk.edu

Jiseung Nam
njs51@dongguk.edu

[1] Department of Information and Communication Engineering, Dongguk University, Seoul 100715, South Korea

Published online: 10 September 2019

2018). Such device drivers, however, should be customized for sensing and actuating nearby environments, and overhead is often involved due to human intervention for customizing and maintaining device software (Tucci et al. 2017). For example, devices deployed in a home may initially be configured to monitor light intensity. After some time, a temperature sensor is needed in the home and a light sensor is no longer required. In such a case, one cannot remove the light sensor and directly attach a temperature sensor. Instead, a new device driver is required to operate the temperature sensor and to integrate the temperature sensor and its peripherals. This implies that IoT application deployers must work on customizing appropriate device drivers whenever new service demands arrive or service demands change, and the complexity of IoT development and deployment will increase given the variety of IoT devices and their peripherals that should be customized. Whenever it is necessary to reconfigure a variety of IoT devices and their peripherals, a massive amount of programming and maintenance work is required to prepare and connect the IoT devices peripherals.

This paper proposes a generic platform that enables PnP integration for various IoT sensors and actuators. The main advantage of PnP architecture is that it allows for the addition and removal of IoT peripherals independent of sensors and actuator device types without re-customizing or re-installing all of the local device drivers. In particular, all local device peripherals are classified into digital/analog types and sensor/actuator types, and they require their own device configuration rules and input/output (I/O) behaviors corresponding to their device specifications. For example, digital sensors and analog sensors gather and process data in different ways and generate data in different formats. Replacing device peripherals, therefore, requires new device configurations and different I/O function calls according to their specifications.

The goal of this study is to provide PnP IoT devices with over-the-air deployable, scalable, and robust drivers. To this end, we employ IoT ontologies and web semantics as well as generic programming techniques in order to configure and control IoT devices requiring minimum human involvement. Ontology is a formal and explicit specification of knowledge in the domain of interest (Guarino 1998). Ontology defines shared knowledge in terms of classes, relationships, and axioms (Islam and Islam 2016). Ontology addresses the existence of entities and the relationships among entities, which allows for the drawing of inferences about how they interact with other distinct entities in the real world (Vijayarajan et al. 2016). In the proposed platform, IoT device characteristics, including I/O functions and configuration rules, are defined using our custom-built IoT ontologies and written in resource description framework (RDF) format (Hjelm 2001) such that appropriate device function calls and behaviors are inferred through SPARQL (Prud'hommeaux and Seaborne 2008) queries. Our generic platform then passes function-call name and configuration rules corresponding to the newly added peripheral device's specification through a wireless network/serial port.

We implement device drivers using generic programming techniques accepting most of the function calls available in the Arduino (Severance 2014) platform, which we focused on for the IoT target application. Using generic implementation and semantic information provided by a semantic inference engine, IoT devices can select the appropriate device function calls for correct device behavior without reinstalling device drivers or changing IoT modules. The raw data gathered from sensors is transmitted to a local server and converted using a pre-defined formula to create a user-understandable sensing value that is later visualized on the dashboard in the central server. In this platform, therefore, most of the programming work that is required for normal device configuration is unnecessary, and as a result, this platform enables PnP device experiences responding to dynamic IoT service demands.

The remainder of this paper is structured as follows: Sect. 2 reviews related works regarding semantic supports for device communication and the corresponding operations. Section 3 provides an overview of our generic IoT PnP platform. Section 4 presents our IoT ontologies and their semantic representation for IoT sensors and actuators, focusing on generic device operations. Section 5 describes how ontologies and semantics are integrated into the generic architecture and how inferring device characteristics enables appropriate function calls to run a newly added device correctly without reinstalling drivers or changing the whole IoT module. Additionally, this section explores how the generic platform is implemented from the IoT device side. Finally, Sect. 6 concludes our contribution from the paper and discusses future work.

2 Related work

Semantics and ontologies have been part of IoT from the beginning and enable the automation of IoT devices (Kovacs and Csizmas 2018; Fattah and Chong 2018). The most prominent examples of IoT ontologies are the semantic sensor network ontology (Haller et al. 2017), the web of things (WOT) by Kaebisch et al. (2019), IoT-lite (Bermudez-Edo et al. 2015), smart appliances reference (SAREF) ontology (Roes et al. 2015), IoT ontology (Kotis and Katasonov 2012), and IoT-O (Seydoux et al. 2016). Internet of things vocabularies and ontologies defined from these examples can be imported from each other and are therefore able to cover most of the concepts used in IoT environments. Regarding the current approach of building a generic PnP device platform applicable to low-powered embedded devices, it is necessary to

extend existing successful IoT vocabularies and therefore to facilitate low-level control of the embedded devices using I/O function calls and hardware-level configurations.

Machine-to-machine (M2M) technology has been widely studied and used in many diverse realms to enhance interoperability among large-scale heterogeneous devices and services through device abstraction and semantic support (Cackovic and Popovic 2013; Wu et al. 2011). Current IoT devices and applications have been designed and deployed independently; hence, all interacting applications must agree beforehand on a specific terminology before establishing any communications. The semantic web is often considered a key to integrating global M2M standards and various industry applications (Ayalla et al. 2017; Chellouche et al. 2013). Object vocabularies discovered and inferred from web semantics enhance interconnection between device communications. A semantic-based reasoning mechanism enables the discovery of IoT device-specific information and the determination of required actions for the device peripherals (Park et al. 2014; Charpenay et al. 2015). As a result, the management overhead on a growing number of interconnected IoT devices can be minimized as the capability of semantic inference can reduce the human intervention cost needed for IoT device management. However, M2M technology using semantics mostly focuses on device interoperability and data integration generated from heterogeneous IoT data sources. M2M technology does not address issues over generic application architecture handling heterogeneous device peripherals, which require customized software architecture responding to dynamic device specific semantics.

There are open standards to be proposed for M2M communications, such as 3GPP (Roessler 2017), OMA LWM2M (Mcllroy 2019), ETSI SmartM2M (Buonaccorsi et al. 2012), and oneM2M (Swetina 2018). Those M2M standard protocols are designed to ensure interoperable and cost-effective operations for the sensor networks and M2M environment, managing lightweight and low-power devices on a variety of networks, and hence delivering services for the combination of IoT with the description of things and the context in which they are used. They often require remote management of IoT devices and define an extensible resource and data model. Some of those standards additionally propose IoT data management mechanisms for secure data exchanges.

Firmata (Langbridge 2015) is a generic protocol for communicating with microcontrollers from software on a host computer. Firmata allows the control of IoT devices from software on the host computer using client libraries implemented on a generic communication protocol to IoT devices. Through Firmata, a software that controls devices can be written with various programming language in which a programmable interface is provided. Even though Firmata offers easy implementation of a software that controls devices, it does not support dynamic discovery and selection of available software modules that respond to dynamic device operational semantics.

PnP enables device compatibility and on-the-fly scalability and reconfigurability (Ghazanfari et al. 2018). PnP design has been widely studied, and many standards exist to achieve universal interoperability between systems (Tan et al. 2015; Androcec and Vrcek 2016). These PnP standards mostly focus on interoperable protocols and exchanging data representation. Regarding IoT peripherals that do not have enough computing capacity, existing PnP standards cannot be applied because PnP standards require enough computing capability to process the suggested protocols and manage data representation. Works from (Liu et al. 2016; Bordel et al. 2016) implemented PnP frameworks over the sensor networks called cyber-physical systems (Ribeiro and Bjorkman 2018; Serpanos 2018). Their PnP connectivity works between sensor devices and a local gateway. For each sensor device, device drivers should be implemented individually; however, those approaches are not based on the generic architecture. (Farzaneh and Knoll 2016) applied PnP ontologies to sensor networking, but they used ontologies for device operational configurations based on pre-defined device characteristics. This approach still requires device driver customizations before integrating devices into their sensor network.

As IoTs has become the part of our daily life due to their communication and computing capabilities, IoTs requires seamless interactions among different types of many cheap sensors such as medical sensors, monitoring cameras, home appliance and so on. One of notable example is IoT-based healthcare system (Gope and Hwang 2015), where sensors in healthcare systems monitors and collect the human health functions and surrounding environment. As sensors in the healthcare system handle sensitive information, strict security mechanisms are required to prevent malicious interactions with the system. To address security issues on IoT healthcase system, Gope and Hwang (2015, 2016) proposes a lightweight IoT based security platform to be used in the healthcare system. In their proposal, three components, named Body Sensor Network, Local Process Unit and BSN-care server, work together for user-authentication and secure data transmission, which has similar architectural shape with our approach where we maintain three independent corresponding components (IoT device, local server, central server). We believe Gope and Hwang's proposal (2015, 2016) can be easily implemented into our proposed platform by separately incorporating their protocols for Body Sensor Network, Local Process Unit and BSN-care server into our IoT devices, local server and central server respectively. Therefore our PnP IoT platform can become trustworthy in hostile IoT environments.

Generic programming allows programmers to instantiate a single logic for a variety of data types and algorithms

(Altenkirch and McBride 2003). There are many approaches to make solutions generic when there are many individual logics that require infrastructural knowledge in order to generate the corresponding components (Aoudia et al. 2016; Atanasov et al. 2016; Cresson and Hautreux 2016; Le Nir and Scheers 2017). However, their implementation focuses on middleware, not on device-level firmware architecture.

3 A PnP architecture for generic IoT devices

This study presents a PnP generic platform for IoT devices using ontology and the semantic web. We use the Arduino platform for IoT devices and their peripherals and Apache Jena API and SPARQL for semantic processing, which are located in a local server.

Figure 1 illustrates our semantic-based approach to build PnP architecture for generic IoT device management. The proposed architecture has three main components: IoT devices, a local server, and a central server. The descriptions of these components are as follows:

- *Internet of things* devices are either IoT sensors or actuators intended to monitor/control their nearby environments. IoT devices are composed of sensing/actuating peripherals and a network interface for communicating with a local server. Generic software drivers handling device dynamics are installed here. IoT devices accept device customization information from remote servers and configure themselves to correctly run the attached peripherals. They also send sensing data/actuating feedback to the local server in raw format.
- *The local server* sends device customization information to IoT devices and converts raw data obtained from IoT devices to a user-understandable format. To generate device customization information, a local server receives device knowledge from the central server, and an inference engine inside the local server infers the appropriate configuration. The inference engine in this study uses Jena API. A *Post Evaluator* inside the local server translates raw data from IoT devices to a user-understandable format using an equation formula provided in the central server.
- *The central server* has a dashboard, knowledge base, and device database. The dashboard displays data collected from IoT devices and triggers, creating new device information. The knowledge base stores IoT device ontologies to be used to infer device configuration information. Device DB contains the current configuration information of the devices along with their sensed dataset.

The service flow of the proposed generic platform is described as follows. When a peripheral is removed and new peripheral is attached to an IoT device, the IoT device driver needs to be re-configured for the new peripheral's characteristics. Therefore, the device triggers a reconfiguration request and waits for device configuration information from the local server. The device reconfiguration request is then delivered to the central server through a local server connected to the IoT device. Once the trigger event handover from the local server to the central server, the local server locates the device identification (ID) from a list of directly connected IoT devices and adds it to the triggering message.

At the central server side, the local deploying manager checks a re-configuration request with the device's ID obtained from the IoT device. Given the device ID, the

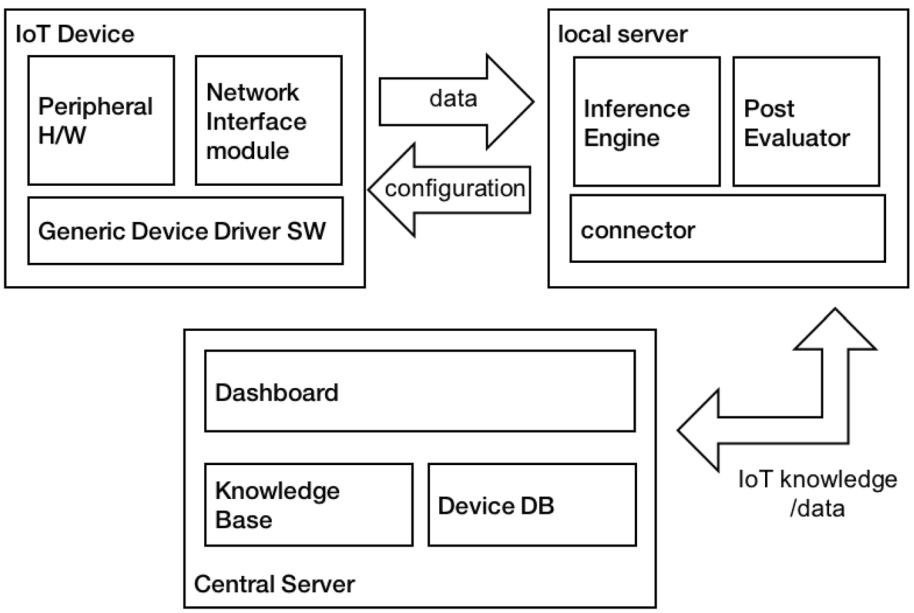

Fig. 1 A PnP architecture for generic IoT device management

central server locates the device type and all the configurable information that can be applied directly in IoT devices. The device configuration information includes the hardware connection configuration from an IoT deployer and the name of the appropriate I/O function calls with necessary additional function calls. The device configuration information is inferred based on our knowledge base stored in the central server and displayed on the dashboard. The dashboard in the central server provides available options regarding the device configuration information through an inference process such that an IoT deployer manually selects the appropriate configuration through the dashboard with respect to the designated behaviors of newly added IoT devices. Parameters regarding delay time or variable initialization needed for operating native device functions can be selected through a dashboard provided in the central server.

Once the IoT deployer finishes the selection process required for device configuration, the configuration information is sent to the local server, which is connected to the corresponding IoT device. The device configuration information is written in JavaScript Object Notation (JSON) format and composed of message ID, the device type and device parameters which is selected from our dashboard UI forms. Our dashboard maintains dynamic UI form to accept device specific parameters in runtime. So, for each new device peripherals attached, a device-specific data selection form is dynamically displayed in the dashboard.

The local server translates the information to the device dependent specific format used for customization of the generic software modules. The IoT device is implemented generic and has no device-specific information. Therefore, the local server should maintain a list of available device types which can be attachable as peripherals. As each peripheral runs on its own properties, additional device parameters which are used to define behavior mechanisms should be created at the local server and delivered to the device along with parameters selected at the dashboard. Parameters regarding device behavior are created using inference engine and knowledge based. Inference engine infers necessary function calls of generic software driver using knowledge base defined based on our IoT ontology. Inference engine in the local server creates all the necessary function calls to run newly attached peripheral devices and attach the function calls to the messages from the central server.

The local server then sends the configuration information to the corresponding IoT device. The configuration information is then composed of message ID, parameters which IoT deployer selected at the dashboard, and parameters of a list of function calls needed for operation of newly attached device peripherals. After IoT devices accept configuration information, they begin to collect data from IoT peripherals. Because sensed data in analog sensors is represented by the degree of voltage, the data should be translated to a user-understandable format employing an evaluation function provided by sensor specification. For example, if the sensed value obtained from an analog temperature sensor was 3.73, this value should be translated into the user-understandable value of 21.3 °C. The local server maintains the sensor device's post-evaluation functions from the knowledge base and converts raw data obtained from IoT devices to a user-understandable ranged value. The local server then sends the data to the central server to be displayed on the dashboard and to be stored in the device DB.

4 Ontologies and semantic web representation for PnP generic device management

This section describes our ontologies that enable PnP device operations over our generic platform (Fig. 2). Based on the present ontology, IoT devices are identified as *sensor/actuator* and *analog/digital*. The device also has several characteristics regarding device behavior and architecture. In the present ontology, devices have four properties, namely *hasBehavior, handleEvent, hasCommunication* and *handleException. hasBehavior* represents the basic behavior of IoT devices. *handleEvent* describes how a device would manage event notification. *hasCommunication* describes whether a device uses FIFO buffers or memory for storing stream data. The data fetched from sensors is in raw format and must be transformed in an understandable way. Through *ExceptionHandling,* an IoT deployer can designate the way of handling exceptions. We provide three exception-handling methods: suspend, fault, and watchdog. In case a device software is constructed with multiple patterns, an IoT deployer can select a specific pattern to efficiently conduct tasks.

From our experiment, some device software can be constructed with object-oriented language, such as C++, and designing architecture with patterns are more efficient in supporting various device operations. *hasBehavior* defines common device behaviors. We have identified 17 common device *behaviors,* which are integrated into device driver software. Device behaviors include several native steps, such as data preparation and I/O system calls. Figure 3 describes an ontology of behavior *Debounce*. Behavior *Debounce* is composed of parameters of initializing database, database input/output, and other necessary parameters required for correct execution.

Figure 4 outlines our semantic representation of *BlinkWithoutDelay* behavior in RDF format, which needs several parameters for variable initialization and loop inputs. Figure 4 describes the parameters required for the device drivers. The behavior is stored in the knowledge base located in the central server. The central server infers the needed parameters from the knowledge base and displays them on

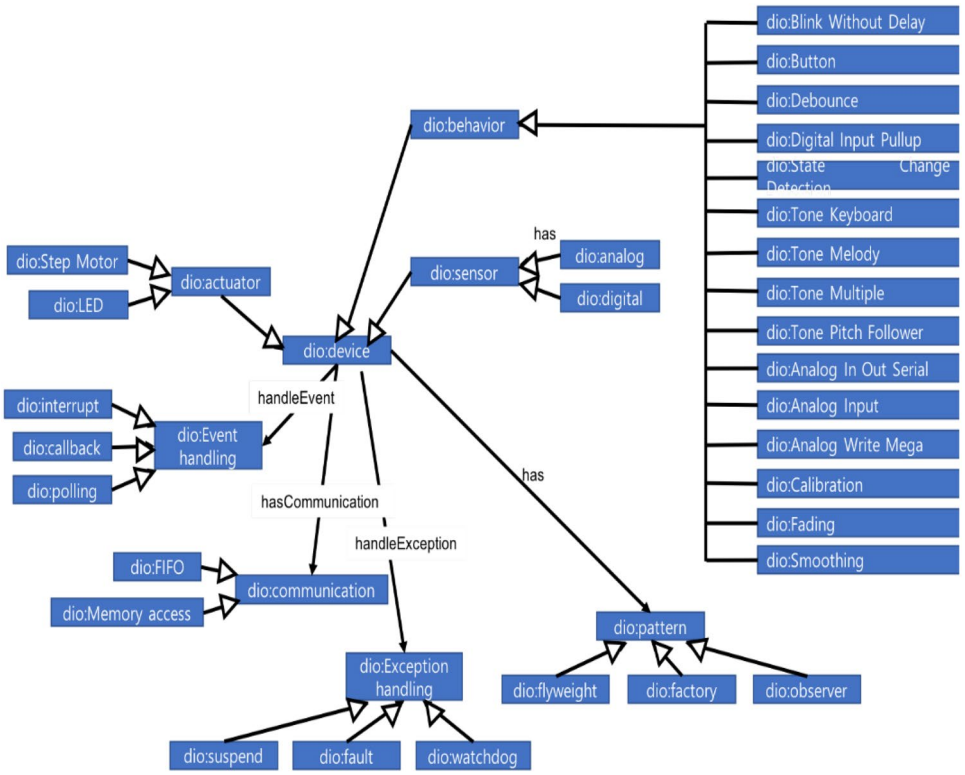

Fig. 2 Ontologies used for implementing our generic PnP platform

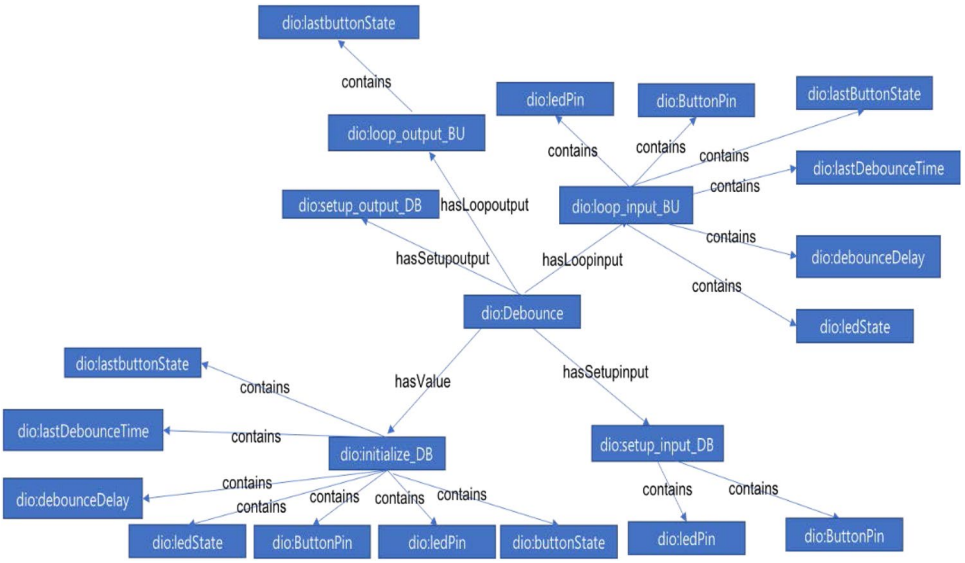

Fig. 3 Ontologies for *Debouncing* behavior

the dashboard such that IoT deployers select the appropriate values for the device parameters.

5 Implementation

We have implemented our generic platform on a customized Linux server with Jena API for the central server, Raspberry Pi for the local server, and the Arduino platform as an embedded IoT device. We connect the Arduino IoT platform to a Rasberry Pi local server directly, and peripherals of sensors and actuators are attached to the Arduino IoT platform.

Figure 5 illustrates a dashboard hosted in the central server. Through a dashboard, an IoT deploying engineer catches newly replaced sensor information and available configuration options. The central server infers all the available configuration options through Jena API with SPARQL queries. To update new peripheral device information, a

Fig. 4 Representing *Blink-WithoutDelay* behavior in RDF Format. This file is stored in the knowledge base of the central server and used to infer necessary parameters for a newly replaced sensor peripheral

```
<!DOCTYPE rdf:RDF [
        <!ENTITY rdfs  "http://www.w3.org/2000/01/rdf-schema#">
        <!ENTITY owl   "http://www.w3.org/2002/07/owl#" >
        <!ENTITY dio "http://www.dongguk.edu/ice/dio#">
        <!ENTITY m3 "http://sensormeasurement.appspot.com/m3#">
        <!ENTITY foaf "http://xmlns.com/foaf/0.1/">
        <!ENTITY xsd   "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="&rdfs;"
        xmlns="&dio-dataset;"
        xml:base="&dio-dataset;"
        xmlns:dio-dataset="&dio-dataset;"
        xmlns:dio="&dio;"
        xmlns:foaf="&foaf;"
        xmlns:m3="&m3;"
        xmlns:owl="&owl;"
        xmlns:xsd="&xsd;"
        xmlns:vann="http://purl.org/vocab/vann/"
        xmlns:vs="http://www.w3.org/2003/06/sw-vocab-status/ns#">

    <dio:Behavior rdf:about="&dio;BlinkWithoutDelay">
        <dio:Variable_Initialize>#ledPin</dio:Variable_Initialize>
        <dio:Variable_Initialize>#ledState</dio:Variable_Initialize>
        <dio:Variable_Initialize>#previousMillis</dio:Variable_Initialize>
        <dio:Variable_Initialize>#interval</dio:Variable_Initialize>
        <dio:Setup_Input>#ledPin</dio:Setup_Input>
        <dio:Loop_Input>#Interval</dio:Loop_Input>
        <dio:Loop_Input>#previousMillis</dio:Loop_Input>
        <dio:Loop_Input>#ledState</dio:Loop_Input>
    </dio:Behavior>
```

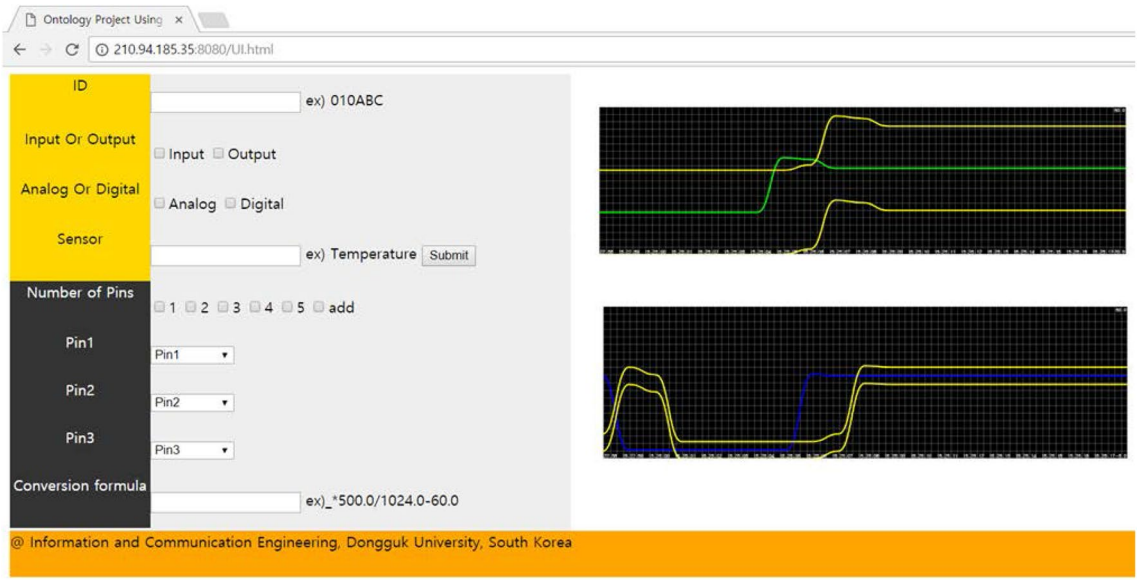

Fig. 5 Dashboard in central server

user should enter the device ID pre-set before installation, the operation type (either sensor or actuator), the sensing type (either analog or digital), and the sensor's name. The information received from the central server is then passed from the central server to a local server. Once the deploying engineer completes the device configuration process, he will see the data stream collected from the IoT devices.

The central server uses SPARQL to infer the device configuration and operational information from device ontologies and device information written in RDF. Figure 6 is a

```
PREFIX dio: <http://www.dongguk.edu/ICE/don>
SELECT  ?ID
        ?operationType
        ?SensingType
WHERE
{
    ?ID            dio:02a1ae01      ?ID .
    ?operationType dio:operationType ?operationType .
    ?SensingType   dio:SensingType   ?SensingType .
}
PREFIX dio: <http://www.dongguk.edu./ICE/don>
SELECT  ?sensor
        ?formula
        ?delay_value
        ?pin1
        ?pin2
        ...
WHERE
{
    ?sensor      dio:Temperature ?Sensor .
    ?formula     dio:formula     ?formula .
    ?delay_value dio:delay_value ?delay_value
    ?pin1        dio:pin1        ?pin1 .
    ?pin2        dio:pin2        ?pin2 .
    ...
}
```

Fig. 6 A SPARQL query sample to infer device configuration options

```
{
    "ID": "02a1ae01",
    "operationType": "sensor",
    "SensingType": "analog",
    "Sensor": "temperature",
    "Formula": "_*500.0/1024.0-60.0",
    "delay_value": "3000",
    "Pin": [
    {
        "number": "pin1",
        "type": "Enable",
        "value": "A0"
    },
    {
        "number": "pin2",
        "type": "GND",
        "value": "GND"
    },
    ...
}
```

Fig. 7 An example of a device configuration message from a central server to a local server

SPARQL query example corresponding to the IoT deployer's input. The IoT deployer obtains device information, such as the *operationType* "sensor" and the *sensingType* "Analog" for the device ID "02a1ae01." Using the information provided from SPARQL queries, the local server infers the desired behaviors, the sensor conversion equation, the desired delay value, and the pin configuration from knowledge written in RDF. An IoT deployer then provides device configuration values that will complete the device replacement process.

Figure 7 describes a message sent to the local server. The outputs shown in Fig. 7 are written in JSON format and fed into a local server through a wireless network or serial port. The values of the JSON attribute *operatingType*, *SensingType,* and *Sensors* as information from device configuration information on this generic platform. These values are converted into the appropriate integer values and sent to IoT devices, especially for the Arduino device.

In addition to message transfer and conversion, a local server conducts post-processing tasks, which are required for most types of sensors. Post-processing involves making the raw data obtained from the IoT device into user-understandable ranged values through the conversion formula. As each sensor requires its own the conversion formula, the dashboard enables the user to personally enter the conversion formula or retrieve the formula from the knowledge base. The conversion formula entered by the user is stored separately in the RDF format and sent to the local server.

To be generic over various types of device peripherals, we employ a generic software architecture in which device-native functions are operated selectively based on configuration information. To create generic software, we first identify characteristics of IoT device operation which most IoT sensors/actuators hold. From our research, most IoT sensors runs periodic operations with some external interrupts. So the outmost layer of generic programming structure is a loop structure in which device specific operating routines, timers and interrupt handlers are located. We adopt Façade patterns to make device operate in selective way based on external parameters delivered from the local server. We also adopt state machine pattern to compose generic programming structure. State machine holds the runtime device' internal state and enables decisive operational behavior based on the device's state. Our PnP platform adoptS state machine pattern and façade patterns to enable dynamic operations of generic software modules.

Toward generic implementation, we used function pointer arrays to enable the optional function selection. Attributes of Fig. 7 match the function pointer array name, and the values denote the index of a function pointer array. For example, when the inferred output includes *sensor* in *operationType*, the *operationType* function array is activated and function *isSensor* is called. When the device is detected as an analog sensor, the corresponding sensor function will be activated based on the sensor name. As Arduino's serial communication method does not support complex communication and we found that JSON data processing consumes a considerable amount of Arduino CPU power, a local server takes on the role of JSON processing and sending value-only messages to IoT devices. The message includes the function call ID and parameter values with delimiters only. The order of values in a message is predetermined and shared between the local server and IoT devices.

We provided most of the primary sensing operations presented in the Arduino platform. Because the Arduino platform is so simple and the number of function calls is limited, enumerating Arduino system calls is not a huge task and does not take much memory space. The following are parts of the generically implemented device source code:

```
void (*operationType[])(int i) = {isSensor, isActua-
tor};
void (*SensingType[])(int i) = {isAnalog, isDig-
ital};
void (*AnalogSensorParameters[])(int k) = {iValue,
fValue,  acc, In-
fraDis};

void (*setupMenu[])(byte pin2, byte pin3) = {setupEx-
tra,setupFourDis, setupSlope, setupGyro, setupOutput};
...
void iValue( ){
  setupMenu(pin2, pin3);
  int value = ANALOGREAD(pin1);
  delay(delay_value);
}
```

As a result, our generic approach covers various types of sensors/actuators supporting diverse device behavior. Table 1 shows the coverage of our generic IoT platform. Our generic driver covers most of analog sensors available in the market and supports most of the required behaviors. As for the actuator, our generic driver works on LED, step motor, and speakers as they do not need additional complex operating logic, which is required for robot arm. Our approach could not cover some digital sensors, for example, infrared rays and digital cameras, because such sensors require additional hardware modules as well as a special software library provided by the vendor.

For analog sensors operation, parameters related to software function calls fed into IoT devices are mostly identical. Most operation except timers are identical for all the analog sensor listed in Table 1. For digital sensor, we use façade pattern inside main loop structure, in order to generate sensing value cause most digital sensors require vendor specific libraries.

For actuators, we apply digital actuators only as market does not provide analog actuators currently. The operations of actuators are more complex as the actuating operation requires its own operational logic that is not covered by periodic digital write operation. We use state machine pattern that supports decision process requires to cover non-periodical operation.

Table 1 Sensor coverage over a single generic driver

Device type	Device type	Behavior support
Sensor/analog	Temperature	Analog in/out control, input calibration, value smoothing
Sensor/analog	Humidity	Analog in/out control, input calibration, value smoothing
Sensor/analog	Gas	Analog in/out control, input calibration, value smoothing
Sensor/analog	Lumity	Analog in/out control, input calibration, value smoothing
Sensor/analog	Distance	Analog in/out control, input calibration, value smoothing
Sensor/analog	Force	Analog in/out control, input calibration, value smoothing
Sensor/digital	Temperature	Digital value read, digital input pullup
Sensor/digital	Distance	Digital value read, digital input pullup
Sensor/digital	Vibration	Digital value read, digital input pullup
Actuator/digital	LED	BlinkWithoutDelay, blink, button, debounce, fade
Actuator/digital	Step motor	DigitalWrite, play a sequence
Actuator/digital	Speaker	ToneMelogy, ToneMultiple, ToneKeyboard

6 Conclusion

This paper proposes a generic platform supporting PnP IoT device management. To this end, we employed web semantics based on IoT ontologies in order to generate a PnP device configuration that can operate IoT devices correctly without replacing IoT modules or installing new drivers. We designed our own IoT ontology to define fine-grained software operational semantics. With façade pattern, we incorporated modulized operational semantics into generic program structure. Dynamic behavior of actuating devices is supported through state machine patterns inside our generic device software architecture. The composition rules respecting device behavior are inferred using web semantics and SPARQL queries.

In this PnP approach, sensing and actuating peripherals, such as light sensors and step motors, could be added and removed on the fly, which reduces the level of complexity of implementing and maintaining an IoT environment when IoT service demands change dynamically. Considering massive and dynamic usage patterns of devices and their peripherals, we believe our platform could save tons of efforts to maintain IoT infrastructure.

In terms of future research, the plan is to improve the platform in order to cover a broader range of sensor and actuator types. Some digital sensors require a mixed set of read/write I/O functions with complex bit operations. To cover these types of sensors, we plan to expand their semantic representation to support more complex interactions between IoT devices and their peripherals. Additionally, the present approach's device coverage is currently limited to sensors that do not require third-party external libraries. Due to the limited memory size of IoT devices, a wide variety of external libraries cannot be preinstalled at all. In such cases, we will look for and implement dynamic linking solutions in IoT devices.

Acknowledgements This research was also supported by the research program of Dongguk University, 2015. And the following are partial results of a study on the "Leaders in Industry-university Cooperation +" Project, supported by the Ministry of Education and National Research Foundation of Korea.

References

Altenkirch T, McBride C (2003) Generic programming within dependently typed programming. In: Gibbons J, Jeuring J (eds) Generic programming. Springer, New York, pp 1–20

Androcec D, Vrcek N (2016) Thing as a service interoperability: review and framework proposal. In: 2016 IEEE 4th international conference on future Internet of things and cloud. IEEE, Vienna, pp 309–316. https://doi.org/10.1109/FiCloud.2016.51

Aoudia FA, Gautier M, Magno M, Berder O, Benini L (2016) A generic framework for modeling MAC protocols in wireless sensor networks. IEEE/ACM Trans Netw 25:1489–1500. https://doi.org/10.1109/TNET.2016.2631642

Atanasov I, Nikolov A, Pencheva E (2016) Study on generic functionality for quality of service control in M2M communications. In: 2016 IEEE international black sea conference on communications and networking (BlackSeaCom). IEEE, Varna, pp 1–5. https://doi.org/10.1109/blackseacom.2016.7901589

Ayalla MB, Drira K, Gharbi G (2017) Semantic-aware IoT platforms. In: Proceedings of 2017 IEEE international conference on AI and mobile services (AIMS). IEEE, Honolulu, pp 8–13. https://doi.org/10.1109/AIMS.2017.15

Bermudez-Edo M, Elsaleh T, Barnaghi P, Taylor K (2015) IoT-lite ontology, W3C member submission. https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126. Accessed 18 June 2019

Bordel B, de Rivera DS, Alcarria R (2016) Plug-and-play transducers in cyber-physical systems for device-driven applications. In: 2016 10th international conference on innovative mobile and Internet services in ubiquitous computing (IMIS). IEEE, Fukuoka, pp 316–321. https://doi.org/10.1109/IMIS.2016.68

Buonaccorsi N, Cicconetti C, Mambrini R, Podias N, Russel P (2012) ETSI M2M release 1 demonstration. In: 2012 IEEE international symposium on a world of wireless, mobile and multimedia networks. IEEE, San Francisco, pp 1–3

Cackovic V, Popovic Z (2013) Abstraction and semantics support in M2M communications. In: 2013 36th international convention on information and communication technology electronics and microelectronics (MIPRO). IEEE, Opatija, pp 404–408

Charpenay V, Kabisch S, Anicic D, Kosch H (2015) An ontology design pattern for IoT device tagging systems. In: 2015 5th international conference on the Internet of things (IoT). IEEE, Seoul, pp 138–145. https://doi.org/10.1109/IOT.2015.7356558

Chellouche SA, Chalouf MA, Lemlouma T (2013) Ontology-based pervasive M2M healthcare environment. In: 2013 first international conference on future information and communication technologies for ubiquitous healthcare (Ubi-HealthTech). IEEE, Jinhua, pp 1–5. https://doi.org/10.1109/Ubi-HealthTech.2013.6708062

Chindenga E, Gurajena C, Thinyane M (2016) Towards an adaptive ontology based model for interoperability in Internet of things (IoT). In: IST-Africa week conference. IEEE, Durban, pp 1–8. https://doi.org/10.1109/ISTAFRICA.2016.7530599

Cresson R, Hautreux G (2016) A generic framework for the development of geospatial processing pipelines on clusters. IEEE Geosci Remote Sens Lett 13:1706–1710. https://doi.org/10.1109/LGRS.2016.2605138

Farzaneh MH, Knoll A (2016) An ontology-based plug-and-play approach for in-vehicle time-sensitive networking (TSN). In: 7th annual information technology, electronics and mobile communication conference. IEEE, Vancouver, pp 1–8. https://doi.org/10.1109/IEMCON.2016.7746299

Fattah SMM, Chong I (2018) Restful web services composition using semantic ontology for elderly living assistance services. J Inf Process Syst 14:1010–1032. https://doi.org/10.3745/JIPS.04.0083

Ghazanfari A, Hamzeh M, Abdel-Rady Y, Mohamed I (2018) A resilient plug-and-play decentralized control for DC parking lots. IEEE Trans Smart Grid 9:1930–1942. https://doi.org/10.1109/TSG.2016.2602759

Gope P, Hwang T (2015) Untraceable sensor movement in distributed IoT infrastructure. IEEE Sens J 15:5340–5348. https://doi.org/10.1109/JSEN.2015.2441113

Gope P, Hwang T (2016) BSN-care: a secure IoT-based modern healthcare system using body sensor network. IEEE Sens J 16:1368–1376. https://doi.org/10.1109/JSEN.2015.2502401

Guarino, N (1998) Formal ontology in information systems. In: Proceedings of the 1st international conference on formal ontology in information systems. IOS Press, Trento, pp 3–15

Haller A, Janowicz K, Cox S, Phuoc DL, Taylor K, Lefrancois M (2017) Semantic sensor network ontology. World Wide Web Consortium. https://www.w3.org/TR/vocab-ssn. Accessed 18 June 2019

Hjelm J (2001) Creating the semantic web with RDF: professional developer's guide, 1st edn. Wiley, New York

Islam MN, Islam AKM (2016) Ontology mapping and semantics of web interface signs. Hum Centric Comput Inf Sci 6:20. https://doi.org/10.1186/s13673-016-0077-y

Kaebisch S, Kamiya T, McCool M, Charpenay V (2019) Web of things (WoT) thing description. World Wide Web Consortium. https://www.w3.org/TR/wot-thing-description. Accessed 18 June 2019

Kotis K, Katasonov A (2012) Semantic interoperability on the web of things: the semantic smart gateway framework. In: 2012 sixth international conference on complex, intelligent, and software intensive systems. IEEE, Palermo, pp 630–635. https://doi.org/10.1109/CISIS.2012.200

Kovacs L, Csizmas E (2018) Lightweight ontology in IoT architecture. In: 2018 IEEE international conference on future IoT technologies (future IoT). IEEE, Eger, pp 1–6. https://doi.org/10.1109/FIOT.2018.8325591

Langbridge JA (2015) Arduino sketches: tools and techniques for programming wizardry. Wiley, New York

Le Nir V, Scheers B (2017) Low complexity generic receiver for the NATO narrow band waveform. In: 2017 International conference on military communications and information systems (ICMCIS). IEEE, Oulu, pp 1–7. https://doi.org/10.1109/ICMCIS.2017.7956479

Liu S, Guzzo J A, Zhang L, Smith D W, Lazos J, Grossner M (2016) Plug-and-play sensor platform for legacy industrial machine monitoring. In: 2016 International symposium on flexible automation (ISFA). IEEE, Celveland, pp 432–435. https://doi.org/10.1109/ISFA.2016.7790202

Mcllroy S (2019) Lightweight machine to machine. Open Mobile Alliance. https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m. Accessed on 18 June 2019

Mois G, Folea S, Sanislav T (2017) Analysis of three IoT-based wireless sensors for environmental monitoring. IEEE Trans Instrum Meas 66:2056–2064. https://doi.org/10.1109/TIM.2017.2677619

Park DH, Bang HC, Pyo CS, Kang SJ (2014) Semantic open IoT service platform technology. In: 2014 IEEE world forum on Internet of things (WF-IoT). IEEE, Seoul, pp 85–88. https://doi.org/10.1109/WF-IoT.2014.6803125

Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF. W3C recommendation. http://www.w3.org/TR/rdf-sparql-query. Accessed 18 June 2019

Ribeiro L, Bjorkman M (2018) Transitioning from standard automation solutions to cyber-physical production systems: an assessment of critical conceptual and technical challenges. IEEE Syst J 12:3816–3827. https://doi.org/10.1109/JSYST.2017.2771139

Roes J, Hartog FD, Daniele L, Verhoosel J (2015) Smart appliances reference ontology. European Telecommunications Standard Institute. http://ontology.tno.nl/saref. Accessed 18 June 2019

Roessler (2017) Impact of spectrum sharing on 4G and 5G standards a review of how coexistence and spectrum sharing is shaping 3GPP standards. In: 2017 IEEE international symposium on electromagnetic compatibility and signal/power integrity (EMCSI). IEEE, Washington, pp 704–707. https://doi.org/10.1109/ISEMC.2017.8077958

Serpanos D (2018) The cyber-physical systems revolution. Computer 51:70–73. https://doi.org/10.1109/MC.2018.1731058

Severance C (2014) Massimo Banzi: building Arduino. Computer 47:11–12. https://doi.org/10.1109/MC.2014.19

Seydoux N, Alaya MB, Drira K, Hernandez N, Monteil T (2016) IoT-O. IRIT. https://www.irit.fr/recherches/MELODI/ontologies/IoT-O. Accessed 18 June 2019

Suryani V, Sulistyo S, Widyawan W (2017) Internet of things (IoT) framework for granting trust among objects. J Inf Process Syst 13:1613–1627. https://doi.org/10.3745/JIPS.03.0088

Swetina J (2018) BaseOntology. OneM2M, http://www.onem2m.org. Accessed on 18 June 2019

Tan S, Zhang J, Sun F, Wang S (2015) An approach to support the interoperability of intelligent grouping and resource sharing (IGRS) and universal plug and play (UPnP) in home network environment. In: 2015 IEEE international conference on computational intelligence and communication technology. IEEE, Ghaziabad, pp 682–686. https://doi.org/10.1109/CICT.2015.144

Toosi AN, Son J, Buyya R (2018) CLOUDS-Pi: a low-cost raspberry-Pi based micro data center for software-defined cloud computing. IEEE Cloud Comput 5:81–91. https://doi.org/10.1109/MCC.2018.053711669

Tucci M, Riverso S, Ferrari-Trecate G (2017) Line-independent plug-and-play controllers for voltage stabilization in DC microgrids. IEEE Trans Control Syst Technol 26:1115–1123. https://doi.org/10.1109/TCST.2017.2695167

Uddin Z, Ahmad A, Qamar A, Altaf M (2018) Recent advances of the signal processing techniques in future smart grids. Hum Centric Comput Inf Sci 8:2. https://doi.org/10.1186/s13673-018-0126-9

Vanus J, Machac J, Martinek R, Bilik P, Zidek J, Nedoma J, Fajkus M (2018) The design of an indirect method for the human presence monitoring in the intelligent building. Hum Centric Comput Inf Sci 8:28. https://doi.org/10.1186/s13673-018-0151-8

Vijayarajan V, Dinakaran M, Tejaswin P, Lohani M (2016) A generic framework for ontology-based information retrieval and image retrieveal in web data. Hum Centric Comput Inf Sci 6:18. https://doi.org/10.1186/s13673-016-0074-1

Wu G, Talwa S, Jhonsson K (2011) M2M: from mobile to embedded internet. IEEE Commun Mag 49:36–43

Yang F, Hughes D, Matthys N, Man KL (2016) The PnP web tag: a plug-and-play programming model for connecting IoT devices to the web of things. In: 2016 IEEE Asia Pacific conference on circuits and systems (APCCAS). IEEE, Jeju, pp 452–455. https://doi.org/10.1109/APCCAS.2016.7804000