DPWSim: A Devices Profile for Web Services (DPWS) Simulator

Son N. Han, *Student Member, IEEE,* Gyu Myoung Lee, *Senior Member, IEEE,* Noel Crespi, *Senior Member, IEEE*
Nguyen Van Luong, *Student Member, IEEE,* Kyoungwoo Heo, Mihaela Brut, and Patrick Gatellier

***Abstract*—The Devices Profile for Web Services (DPWS) standard enables the use of Web services for resource-constrained devices, main components of the Internet of Things (IoT). DPWS can power the next generation of IoT applications connecting millions of networked devices and services on the Web. This article presents a simulator, called DPWSim, to support the use of this technology. DPWSim featuring secure messaging, dynamic discovery, service description, service invocation, and publish-subscribe eventing can be used to prototype, develop, and test products in terms of DPWS communication protocols. It can also support the collaboration between manufacturers, developers, and designers during the new product development process.**

***Keywords*—Internet of Things, DPWS, Web Service, Simulation.**

I. Introduction

The Internet of Things (IoT) aims to connect millions of physical objects to the Internet by which to power the next generation of Web applications over these networked devices. One of the recent approaches is to use Web service to seamlessly integrate device functionalities into the Web by using OASIS standard Devices Profile for Web Services (DPWS) [1], [2]. DPWS enables secure Web service capabilities on resource-constrained devices. It has an architectural concept similar to the Web Service Architecture [3] but different in several ways to better fit in resource-constrained environments and event-driven scenarios. DPWS uses Web Service Description Language (WSDL) [4] and Simple Object Access Protocol (SOAP) [5] to describe and communicate device services; but it does not require any central service registry such as Universal Description, Discovery and Integration [6] for service discovery. Instead, it relies on SOAP-over-UDP [7] binding and UDP multicast to dynamically discover services. DPWS has a publish/subscribe eventing mechanism for clients to subscribe for device events, *e.g.*, a device switch is on/off. When an event occurs, notifications are delivered to subscribers via separate TCP connections.

Son N. Han and Noel Crespi are with the Department of Wireless Networks and Multimedia Services, Telecom SudParis, Institut Mines-Telecom, Evry, France (e-mail: {son.han, noel.crespi}@it-sudparis.eu). Gyu Myoung Lee is with the School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool, UK (e-mail: g.m.lee@ljmu.ac.uk). Nguyen Van Luong is with Percevio, Paris, France (e-mail: luongnv89@gmail.com). Kyoungwoo Heo is with Electronics and Telecommunications Research Institute, Daejeon, Korea (e-mail: hkw06@etri.re.kr). Mihaela Brut and Patrick Gatellier are with Theresis Innovation Center, Thales Services S.A., Palaiseau, France (e-mail: {mihaela.brut, patrick.gatellier}@thalesgroup.com)

DPWS is the key technology in several projects under European research and development (R&D) initiatives of Information Technology for European Advancement (ITEA) and Framework Programme (FP) such as ITEA SIRENA, ITEA SODA, FP7 SOCRATES, and FP7 IMC-AESOP. These projects have contributed to the standard specification and successfully released a number of DPWS implementations to foster the adoption of the DPWS technology. However, there are currently very few development tools supporting the technology. We therefore have developed DPWSim, a simulator for DPWS standard to help users to prototype, develop, and test their applications during the development process. DPWSim mimics all the protocol features of DPWS to provide an efficient way to interact with DPWS compatible devices. Key features of DPWSim are: (1) DPWS Protocols Simulation - SOAP Envelop messages and mechanisms such as discovery, service invocation, and eventing are all compliant with DPWS specifications; (2) Graphical User Interface (GUI) - An elegant and intuitive graphical interface is a key to accelerate the development process; (3) Platform Independence - DPWSim is written in Java programming language and can run on any machine equipped with Java Virtual Machine; and (4) Flexibility - There are several ways to define a new DPWSim device ranging from manually creating, importing from a file to automatically generating from a physical device.

The remainder of the article is organized as follows. Section II summarizes DPWS technology. Section III presents DPWSim core components and functionalities followed by Section IV about use cases. Section V exhibits the experiment results of DPWSim, and Section VI concludes the article.

II. Devices Profile for Web Services

DPWS was debuted in 2004 by a consortium led by Microsoft and became an integrated part of Microsoft's Windows Vista and Windows Rally (a group of technologies from Microsoft intended to simplify the setup and maintenance of wired and wireless networked devices). DPWS defines a set of implementation constraints to provide a secure and effective mechanism for describing, discovering, messaging and eventing of services for resource-constrained devices. Many technology giants such as ABB, SAP, Schneider Electric, Siemens, and Thales have participated in several Enropean R&D projects and standardization activities related to the DPWS technology. Result of the ITEA SIRENA project is now available in the Web Service for Devices initiative [8] providing an open-source implementation of DPWS with four

DPWS stacks having been implemented and verified: WS4D-gSOAP, WS4D-uDPWS, WS4D-JMEDS, and WS4D-Android.

There are two types of services in DPWS: *hosting service* and *hosted service*. The former is a special service representing a device to participate in discovery, and to describe other services hosted in it. These services present the functionalities of each device and are called hosted services. DPWS uses SOAP, WS-Addressing [9], and MTOM/XOP [10] for messaging and supports SOAP-over-HTTP and SOAP-over-UDP bindings. It uses WS-Discovery [11] for discovering a hosting service (device), and WSDL to describe the hosted service (device service). It uses Web Services Metadata Exchange (WS-MetadataExchange) [12] to define metadata about the device, Web Services Policy [13] to define a policy assertion to indicate the compliance of the device with DPWS, and WS-Transfer [14] to retrieve service description and metadata information about the device.

In addition to the standardization effort of DPWS, many studies have been carried out to deal with several DPWS technical issues and scenarios. It has been shown that DPWS is a promising technology to seamlessly integrate device functionalities and events into existing resources, services, and applications on the Web. DPWS thus far has been widely used in automation industry, home entertainment, and automotive systems [15] and also applicable for enterprise integration [16]. Encoding and compression issues are discussed and preliminarily put under a careful consideration to improve the performance of SOAP messages in DPWS [17]. Experiments on WS4D-uDPWS stack show that DPWS is able to be implemented into (even) highly resource-constrained devices such as sensor nodes with reasonable ROM footprints [18]. The scalability of service deployment was first exploited in [19] showing a prototype for a dynamic and scalable deployment of DPWS devices. One of the important issues in IoT, the integration of DPWS into IPv6 infrastructure and 6LoWPAN, has been also well investigated in many studies such as [20] and [21]. Han *et al.* propose an extension of DPWS standard by using a REST proxy to provide RESTful Web APIs to developers [22]. The latest version of WS4D-JMEDS offers the use of private keys to encrypt SOAP messages. In addition, real applications adopting DPWS technology have started to gain attention such as the DPWS-based semantic building automation system presented in [23].

III. DPWSim: A DPWS Simulator

DPWSim is a cross-platform simulator of the DPWS standard. It supports the development of IoT applications using DPWS; DPWSim is based on WS4D-JMEDS [8], the Java implementation of DPWS. The core function of DPWSim is to simulate the DPWS protocols by generating DPWS messages and its communication messaging patterns. It simulates DPWS devices, called *DPWSim devices*, which can be discovered on the network and can communicate with other devices or clients via DPWS protocols. Besides, it also simulates environments where DPWSim devices reside in. DPWSim provides many simulation tools for users to create, manage, store, and load simulations with high flexibility. DPWSim GUI

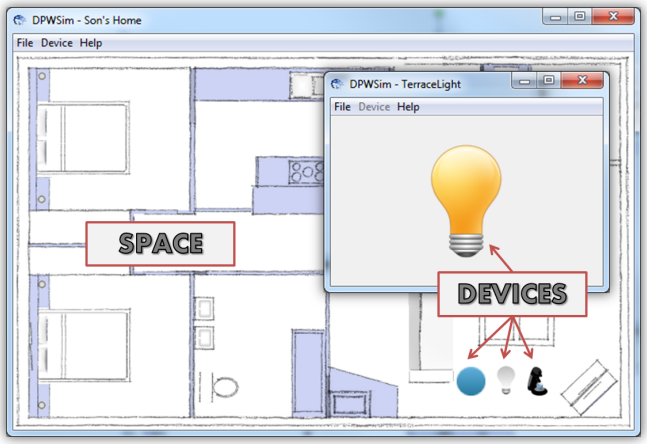


Fig. 1. A home space contains three devices: a generic DPWSim device (blue button), a light bulb, and a coffee maker. A stand-alone device (space with only one device) is a light bulb.

that is based on Java Swing [24] is quite intuitive and easy to use. DPWSim helps developers to prototype, develop, and test DPWS functionalities. The following sub-sections describe the simulation model, core components, functionalities, usage scenarios, and GUI of the simulator.

A. Simulation Model

DPWSim simulates the DPWS devices by modeling them as services that operate according to the input of sensing data (*e.g.*, environmental temperature provided by users) and communication data (*e.g.*, service invocation commands sent from clients). We use a number of hardware including IBM PCs, Raspberry Pi, and Telos B sensor nodes to build real-life devices such as thermostats, motion detectors, and TVs to record how these devices work in several scenarios in order to mimic their behaviors in the simulator. DPWSim builds simulated devices regarding all layers of the TCP/IP networking model [25]. At the network interface layer, the reliable Ethernet link of the host machine is considered to focus on the DPWS protocol messages and mechanisms rather than physical issues (*e.g.*, radio interference). At application layer, each DPWSim device is modeled as a list of services (events and operations) binding to an IP address (internet layer) over UDP (transport layer). Events happen periodically after an interval of time or manually via user interaction; operations are software components receiving input, processing it, and producing output (with its status updated and sent to the invoker). On top of that, device status and outputs of events/operations are modeled as graphical representations. When it comes to modeling and simulating real DPWS systems, DPWSim can support steps involving modeling, designing experiment, and performing analysis of the discrete-event simulation [26].

B. DPWSim Components

DPWSim has four basic components namely Spaces, Devices, Operations, and Events. A space contains several devices; each device has a list of operations and events.

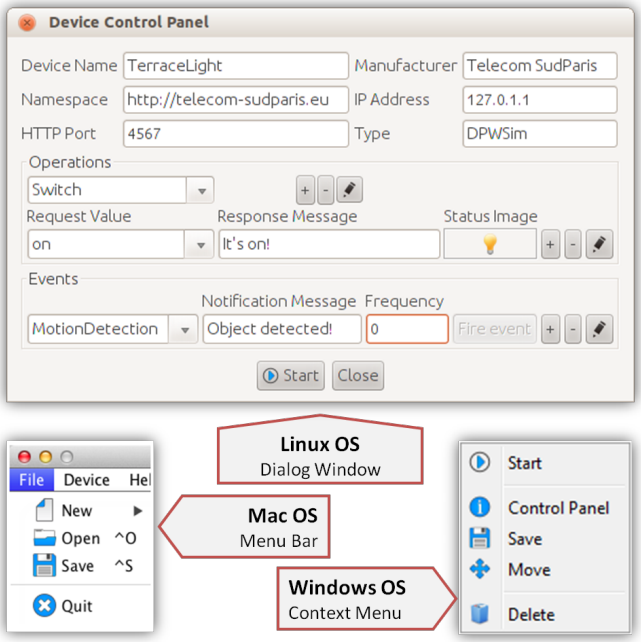


Fig. 2. DPWSim GUI components: a dialog window (Linux OS), a menu bar (Mac OS), and a context menu (Windows OS).

Spaces: A space is a virtual environment representing a real-life setting in which DPWSim devices reside in. It can be a home, an office, a train station, a public space, or simply a stand-alone device. Fig. 1 illustrates a home space containing three devices and a stand-alone device.

Devices: A device refers to both DPWS hosting service and hosted service. Since these two kinds of services, in reality, share similar characteristics, they are used interchangeably in DPWSim for simulation purpose. It contains two different endpoint addresses used for each type of services. For example, when taking part in the discovery, it uses the device endpoint address; when invoking an operation, it uses the service endpoint address.

Operations: Each device contains a list of operations carrying out device functionalities such as switching on and off based on commands received from clients. These operations are described in WSDL descriptions and can be retrieved via service endpoint addresses. Each output of an operation is represented by a graphical status, for example, the light bulb in Fig. 1 will be changed to off status when the corresponding operation is successfully invoked by a client.

Events: An event, similar to an operation, is used for a device functionality related to changes in device state. When the device state changes (or an event happens), it notifies subscribed clients by sending notification messages. An event can happen periodically (*i.e.*, it happens frequently after an interval of time such as sensing CO2 level every 15 minutes) or manually (*i.e.*, it is invoked by users). This property can be set in the Device Control Panel as shown in Fig. 2.

C. DPWSim Core Functionalities

DPWSim provides simulation tools to help researchers and developers to build IoT applications consuming DPWS services. DPWSim can support users to create virtual environments from a simple to a complex one, even a graphically-rich interface like in the Fig. 5 with the aid of external computer graphics software and design skills. DPWSim acts as a dynamic mediator to generate different types of simulation meanwhile maintaining the DPWS functionalities.

New Space/Stand-alone Device: There are two options for creating a virtual environment: stand-alone device and space. These functions can be accessed through File menu or keyboard shortcuts. A space is a composite environment to host several devices. It is created by using a plan image such as office, home, and airport. A stand-alone device is simply a DPWSim device with a hosted service containing operations and events. This kind of virtual environment can be stored in file and re-used in other virtual environments.

New Device: Devices can be created by several ways, each is associated with a submenu of the Device menu in DPWSim: Add New (new user-customized devices), Add Predefined (pre-configured devices by DPWSim), Add From File (importing device from saved device description), and Genereate from Physical Device (creating new device by mapping functionalities from a real device to a simulated one). Users can further customize physical device properties to fit a new device. This capability is especially useful when developers want to focus on designing the business logic of an IoT application rather than the physical performance of devices.

Device Management: Once a device has been created within a virtual environment or as a stand-alone device, it can be queried for DPWS information, re-located, deleted, or saved for future uses. Similarly, a virtual environment including its devices can be saved in the file system for being shared among co-workers. Device services can be changed once created through the Device Control Panel associated to each device as shown in the Fig. 2. It provides an important approach for developers to change device functionalities during the development process without re-creating the device.

D. Usage Scenarios

DPWSim can be used in different phases in the development process of DPWS products and systems. In general, it can be used in three scenarios

Scenario one - Product Integrating: Device manufacturers can pre-provide the DPWSim-compatible **.dpws* file that describes functionalities of upcoming devices to developers. It enables them to test these devices in their real IoT applications before the official release of these products.

Scenario two - Product Protyping: Developers can prototype new devices and new functionalities based on their application requirements without going through the complex manufacturing process. The final design then can be transferred to the manufacturer to work on it.

Scenario three - Resources Sharing: This scenario describes the situation when several teams, at the same time, develop different modules over the same devices. To solve the problem

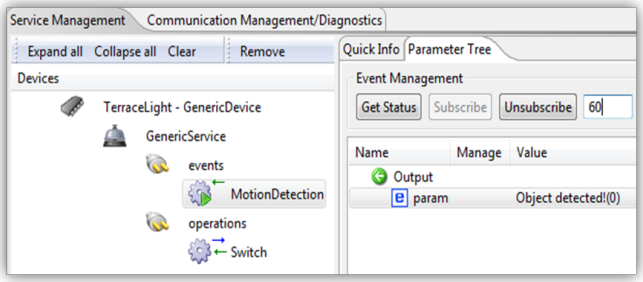


Fig. 3. DPWS Explorer discovers a DPWSim device *TerraceLight* containing an event *MotionDetection* and an operation *Switch*. The green icon next to the *MotionDetection* event indicates that DPWS Explorer is subscribing to the event; once the event occurs, DPWS Explorer will receive the notification, *e.g., Object detected.*

and speed up the development process, a new set of simulated devices is generated by DPWSim to share among developers. The simulation can also be used for demonstration purpose without the loss of the accuracy.

E. Graphical User Interface

DPWSim GUI is built on lightweight Java Swing with a high level of flexibility and the inherent ability to override native host operating system (OS) UI controls. Swing components do not have corresponding native OS GUI components, and every component is free to render itself in any way possible within the underlying graphics GUIs. DPWSim GUI is intuitive to users with the dialog/menu/context menu system. Fig. 2 shows some snapshots of DPWSim GUI in different platforms: Windows OS, Linux OS, and Mac OS.

IV. DPWSim Use Cases

DPWSim has been used in several environments such as DPWS Explorer [8], a Web application, a testbed, and in a number of DPWS studies. The following parts explain each of these experiments on DPWSim and information about the development process.

DPWS Explorer: DPWS Explorer is an analyzing tool for DPWS compliant services. It visualizes various aspects of both hosting and hosted services like metadata or message exchange and provides capabilities to call or subscribe to device services and events. It is used to preview DPWS services during the development process. DPWSim devices can be discovered, their operations can be invoked, and their events can be subscribed from DPWS Explorer. Fig. 3 shows how DPWS Explorer retrieves data and interacts with a DPWSim device.

DPWSim Web: DPWSim Web is a small Web application included in the release of DPWSim to illustrate an use case when a Web application interacts with DPWSim devices. It is a Java Web application running on Apache Tomcat application. It can discover available DPWS devices on the network and retrieve their metadata. Following these data, users can invoke device operations to carry out their tasks. Fig. 4 shows DPWSim Web via its smartphone interface to invoke an operation of a light bulb device *KitchenLight*. Users

Fig. 4. A user can turn on the light bulb KitchenLight by invoking its SwitchOn operation via the smartphone Web interface of DPWSim Web.

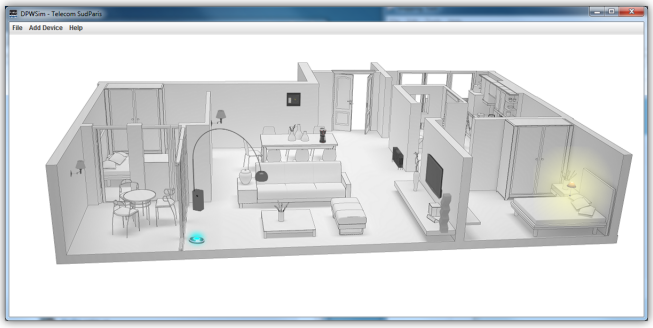


Fig. 5. A virtual home hosting several DPWS devices is designed using DPWSim with the help of a 3D artist (Sa Hoang from École Nationale Supérieure d'Architecture de Paris La Villette - ENSAPLV).

can switch the light bulb on or off from the Web interface by clicking on the buttons.

Research Projects: DPWSim has been used within ITEA Web of Objects (WoO) project (http://www.web-of-objects.com) to support the development of an incident management scenario for testing the contextual object collaboration. DPWSim has been used throughout the development to describe the common interface for the cooperation between devices upon the assigned rights and specific rules imposed in the whole system. An example of the home environment created for the project is shown in Fig. 5. The home consists of several DPWS devices such as a TV, lamps, and a coffee maker. With the help of a 3D artist, it provides an elegant simulation using DPWS protocols.

Besides, DPWSim has been thus far used in several IoT studies such as the semantic building automation system [23], social device networking [27], and REST proxy for DPWS [22]. DPWSim is hosted by WoO project and its source code is maintained on a GitHub repository (http://github.com/sonhan/dpwsim.)

V. DPWSim Experiments

We carry out experiments to evaluate the performance and the compliance of DPWSim to the standard specification by exploring the messaging patterns and exchange messages between DPWSim and clients (*e.g.*, DPWS Explorer) Specifically, the experiments cover following protocols: messaging, dynamic discovery, service description, service invocation, and publish-subscribe eventing. We also evaluate CPU and RAM usages of DPWSim on its host machine.

A. Experiment Setup

A local network is set up with two hosts: a Linux machine with DPWSim and a Windows machine with DPWS Explorer (DPWS client or consumer). DPWSim hosts 10 simulated light devices; each consists of a light bulb, a switch, and a motion sensor (*e.g.*, Passive Infrared) and provides a lighting service that has an operation *Switch* and an event *MotionDetection*. The former is to switch the light bulb on or off; the latter is to notify the subscribed clients about the presence of an object. A packet analyzer Wireshark [1] is deployed on the same machine with DPWS Explorer to track DPWS messages.

B. Messaging

DPWS promotes both interoperability between resource-constrained Web service implementations and interoperability with more flexible client implementations where messages exchanged over the network are SOAP compliant. Exchange messages for DPWS protocols always include SOAP Envelop and transport framing information such as HTTP headers, TCP headers, and IP headers. Listing 1 is a typical SOAP Envelop header of a DPWS message (*Hello* message) generated by DPWSim, which is compliant with DPWS specifications. It complies DPWS, SOAP Envelop, and WS-Addressing namespaces and has a Universally Unique Identifier (RFC 4122) as *urn:uuid:eb6ae010-d47c-11e3-8050-d01059022ad8*.

```
<s12:Envelope xmlns:dpws="http://docs.oasis-open.org
    /ws-dd/ns/dpws/2009/01"
  xmlns:s12="http://www.w3.org/2003/05/soap-envelope
    " xmlns:wsa="http://www.w3.org/2005/08/
    addressing"
  xmlns:wsd="http://docs.oasis-open.org/ws-dd/ns/
    discovery/2009/01">
  <s12:Header>
    <wsa:Action>http://docs.oasis-open.org/ws-dd/ns/
    discovery/2009/01/Hello
    </wsa:Action>
    <wsa:MessageID>urn:uuid:eb6ae010-d47c-11e3-8050-
    d01059022ad8
    </wsa:MessageID>
    <wsa:To>urn:docs-oasis-open-org:ws-
    dd:ns:discovery:2009:01</wsa:To>
    <wsd:AppSequence InstanceId="1399311874"
      MessageNumber="1" />
  </s12:Header>
  ...
</s12:Envelope>
```

Listing 1. SOAP Envelop header

[1] http://www.wireshark.org/

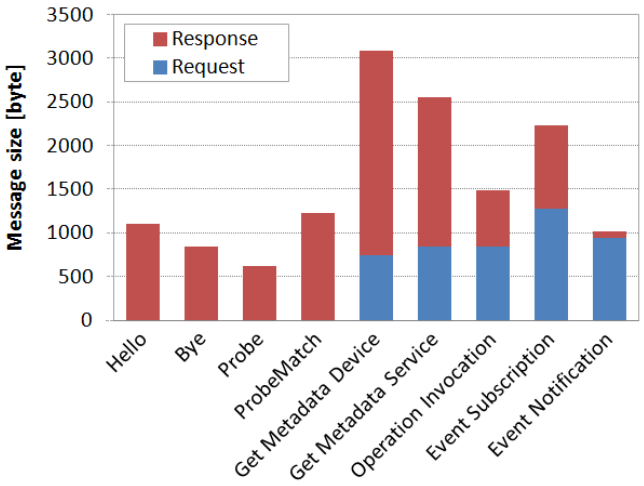


Fig. 6. Sizes of the main DPWS exchange messages in DPWSim with multicast *Hello, Bye*, unicast *Probe, ProbeMatch* and others in the request/response.

There are several types of messages in DPWS including *Hello*, *Bye*, *Probe*, *ProbeMatch*, *Get Metadata Device* (WS-Transfer Get), *Get Metadata Service* (WS-MetadataExchange GetMetadata), *Event Subscription* (HTTP POST), and *Operation Invocation* (HTTP POST), which are frequently exchanged over the network for dynamic discovery, service description, service invocation, and publish-subscribe eventing. DPWSim strictly follows messaging specification for the communication between clients and simulated devices. Fig. 6 shows the size of DPWS messages captured by the Wireshark analyzer.

```
<dpws:Host>
  <wsa:EndpointReference>
    <wsa:Address>
      urn:uuid:1c68cfe0-d475-11e3-803f-d01059022ad8
    </wsa:Address>
  </wsa:EndpointReference>
  <dpws:Types xmlns:wsdp="http://schemas.xmlsoap.org
    /ws/2006/02/devprof" xmlns:i1="http://telecom-
    sudparis.eu">
    dpws:Device wsdp:Device i1:DPWSim
  </dpws:Types>
</dpws:Host>

<dpws:Hosted>
  <wsa:EndpointReference>
    <wsa:Address>
      http://192.168.1.17:4567/D1399308520385Service
    </wsa:Address>
  </wsa:EndpointReference>
  <dpws:Types xmlns:i54="http://telecom-sudparis.eu"
    >
    i54:events i54:operations
  </dpws:Types>
  <dpws:ServiceId>
    GenericService
  </dpws:ServiceId>
</dpws:Hosted>
```

Listing 2. Endpoint address of a hosting service and its hosted service

According to OASIS DPWS specification, a service must at least support WS-Addressing. If a device does not include a transport specific address in *Hello* or *Probe Match* messages,

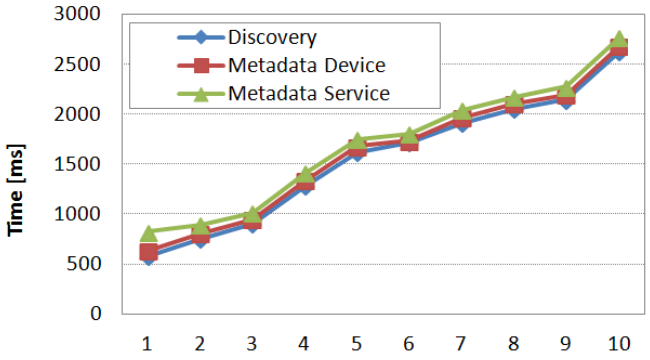


Fig. 7. Discovery time of one to ten DPWSim devices in three scenarios: discovery only (Discovery), discovery and get device metadata (Metadata Device), and discovery and get service metadata (Metadata Service).

the network independent address can be resolved by a UDP multicast *Resolve* message. The response is a *Resolve Match* send unicast through UDP. After finding the transport specific address of a device, the hosted services on the device can be queried. Listing 2 shows endpoint addresses of hosting and hosted services from one DPWSim device.

C. Discovery

The dynamic discovery feature of DPWS is based on WS-Discovery and SOAP-over-UDP specifications with several messages exchanged in the network including *Probe, Hello, Resolve, Device Metadata,* and *Service Metadata*. In the first phase of discovery, multicast *Probe* and *Resolve* messages are sent to the network to discover the target device. After the device is discovered, messages are sent to retrieve device metadata and service metadata before actual communication happens. DPWSim devices also join/leave the network with *Hello/Bye* messages and use the same mechanism for discovering devices. Fig. 7 shows the discovery time of a number of DPWSim devices (randomly created) between one and ten in three cases: discovery only, discovery and get device metadata, and discovery and get service metadata.

D. Description

Service description plays an important role in providing access to a service. Service description in DPWS involves device (hosting service) metadata, relationship with its hosted services, each hosted service WSDL, and WS-Policy. A DPWS client retrieves the device description in form of the XML representation by sending a WS-Transfer Get to the device (hosting service). The response metadata contains characteristics of the device and topology information relating the device to its hosted services. Similarly, the client may also retrieve metadata for individual hosted services by sending a WS-MetadataExchange GetMetadata. WSDL describes the messages a hosted service is capable of receiving and sending. Through WS-Policy, description conveys the capabilities and requirements of a hosted service, particularly the transports over which it may be reached and its security capabilities. DPWSim automatically includes these descriptions in each simulated device, which can be retrieved through any client using DPWS message exchange pattern. Listing 3 from a DPWSim device expresses device metadata that are typically fixed across all devices of the same model from their manufacturer based on *dpws:ThisModel* metadata.

```
1
2  <dpws:ThisModel>
3    <dpws:Manufacturer xml:lang="en-US">Telecom
       SudParis
4    </dpws:Manufacturer>
5    <dpws:ManufacturerUrl>http://telecom-sudparis.eu/
6    </dpws:ManufacturerUrl>
7    <dpws:ModelName xml:lang="en-EN">GenericDevice
8    </dpws:ModelName>
9    <dpws:ModelNumber>1</dpws:ModelNumber>
10   <dpws:ModelUrl>http://telecom-sudparis.eu/</
       dpws:ModelUrl>
11   <dpws:PresentationUrl>http://telecom-sudparis.eu/
12   </dpws:PresentationUrl>
13 </dpws:ThisModel>
```

Listing 3. Device metadata described with *dpws:ThisModel*

E. Service Invocation

Service invocation is the frequent communication between clients and DPWS devices. Listing 4 and 5 illustrate a HTTP request and a response message to invoke an operation from a DPWSim device. The message structure is compliant with SOAP Envelope and WS-Addressing. The request is a HTTP POST with header including endpoint address of the service and *content-type* of *application/soap+xml*. The body of the request is a SOAP Envelope containing the service information. The response is a typical successful HTTP response of code 200 including the return value from the operation. These listings also play as guidelines for developers to handle DPWS communication in any programming language and platform.

```
1  POST /D1399369777882Service HTTP/1.1
2  te: trailers
3  user-agent: JMEDS HTTP Client
4  host: 192.168.1.17:4567
5  content-length: 670
6  content-type: application/soap+xml
7
8  <?xml version="1.0" encoding="UTF-8"?>
9  <s12:Envelope xmlns:dpws="http://docs.oasis-open.org
       /ws-dd/ns/dpws/2009/01"
10   xmlns:s12="http://www.w3.org/2003/05/soap-envelope
       " xmlns:wsa="http://www.w3.org/2005/08/
       addressing">
11   <s12:Header>
12     <wsa:Action>http://telecom-sudparis.eu/
       operations/switch</wsa:Action>
13     <wsa:MessageID>urn:uuid:46932240-d504-11e3-bf6a
       -6eabe38b6788
14     </wsa:MessageID>
15     <wsa:To>http://192.168.1.17:4567/
       D1399369777882Service</wsa:To>
16   </s12:Header>
17   <s12:Body>
18     <i149:param xsi:type="xs:string" xmlns:i149="
       http://telecom-sudparis.eu"
19       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance">on</i149:param>
```

```
  </s12:Body>
</s12:Envelope>
```

Listing 4. Operation Invocation Request

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Date: Tue, 6 May 2014 11:53:30 GMT+0100
Content-Type: application/soap+xml

<?xml version="1.0" encoding="UTF-8"?>
<s12:Envelope xmlns:dpws="http://docs.oasis-open.org
    /ws-dd/ns/dpws/2009/01"
  xmlns:s12="http://www.w3.org/2003/05/soap-envelope
    " xmlns:wsa="http://www.w3.org/2005/08/
    addressing">
  <s12:Header>
    <wsa:Action>http://telecom-sudparis.eu/
    operations/switchResponse
    </wsa:Action>
    <wsa:RelatesTo>urn:uuid:46932240-d504-11e3-bf6a
    -6eabe38b6788
    </wsa:RelatesTo>
  </s12:Header>
  <s12:Body>
    <i53:reply xmlns:i53="http://telecom-sudparis.eu
    ">its on</i53:reply>
  </s12:Body>
</s12:Envelope>
```

Listing 5. Operation Invocation Response

F. Eventing

DPWS has a mechanism for clients to subscribe for device events, *e.g.*, alerts raised by the increasing of the temperature below a specific value, or human presence detected by a video camera. Events are state changes in the device and can be delivered to clients based on the concepts of delivery mode and eventing filter. The former defines how an event is delivered and the latter defines which events are delivered to a client. The PUSH delivery mode defined in DPWS uses for event delivery separated TCP connections to each subscriber. Table I shows details about three types of exchange messages involving eventing mechanism in DPWSim: Subscription, Unsubscription, and Notification.

TABLE I. DPWSIM EVENTING MESSAGES

	Type	Request [byte]	Response [byte]	Total [byte]
Subscription	wse:Subscribe	1279	950	2229
Unsubscription	wse:Unsubscribe	858	611	1469
Notification	NA	940	86	1026
Namespace	xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"			

G. CPU and Memory Usage

The Linux machine hosting DPWSim for the experiment has Intel (R) Core(TM) 2 Duo CPU U7700 1.33 GHz and 2M RAM. Fig. 8 shows the percentage of RAM and CPU consumed by DPWSim with the number of devices between 0 to 10. The CPU usage is measured when the most frequent communication between the client and DPWSim devices,

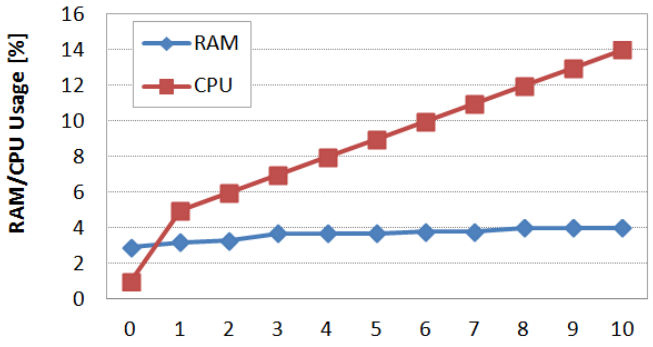


Fig. 8. DPWSim RAM and CPU usage on a Linux Machine of Intel (R) Core(TM) 2 Duo CPU U7700 1.33 GHz and 2M RAM.

invoking a service, takes place. The client sends 1-10 concurrent invocation requests to simulated devices. We observe an approximate linear increase in the CPU usage when the number of requests increases from 1 to 10, consuming up to 14 percent of the host machine CPU.

VI. CONCLUSION

DPWSim is a cross-platform software simulator providing the simulation of DPWS standard. The simulation automatically generates DPWS messages and follows message exchanges pattern in DPWS communication. It can be a transparent, dynamic, and efficient channel between manufacturers and developers for speeding up the development of IoT applications using DPWS technology. To boost the adoption of DPWSim in R&D and industrial context, more experiments with complex scenario, such as dynamic generation of simulated devices, will be further accomplished.

REFERENCES

[1] S. N. Han, G. M. Lee, N. Crespi, V. L. Nguyen, H. Kyoungwoo, M. Brut, and P. Gatellier, "DPWSim: A Simulation Toolkit for IoT Applications Using Devices Profile for Web Services," in *IEEE World Forum on Internet of Things (WF-IoT)*, Mar. 2014, pp. 544–547.

[2] "Devices Profile for Web Services," OASIS Standard, Jul. 2009.

[3] "Web Services Architecture," W3C Working Group Note, Feb. 2004.

[4] "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," W3C Recommendation, Jun. 2007.

[5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," W3C Note, May 2000.

[6] http://uddi.org/pubs/uddi-v3.0.2-20041019.htm.

[7] http://schemas.xmlsoap.org/ws/2004/09/soap-over-udp/.

[8] http://www.ws4d.org/.

[9] http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/.

[10] http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/.

[11] http://schemas.xmlsoap.org/ws/2005/04/discovery/.

[12] http://schemas.xmlsoap.org/ws/2004/09/mex/.

[13] http://schemas.xmlsoap.org/ws/2004/09/policy/.

[14] http://schemas.xmlsoap.org/ws/2004/09/transfer/.

[15] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A Real-Time Service-Oriented Architecture for Industrial Automation," *IEEE Trans. Ind. Informat.*, vol. 5, no. 3, pp. 267–277, 2009.

[16] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, and V. Trifa, "SOA-based Integration of the Internet of Things in Enterprise Services," in *IEEE Int. Conf. on Web Services (ICWS)*, 2009, pp. 968–975.

[17] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski, "Encoding and Compression for the Devices Profile for Web Services," in *IEEE Int. Conf. on Advanced Information Networking and Applications Workshops (WAINA)*, 2010, pp. 514–519.

[18] C. Lerche, N. Laum, G. Moritz, E. Zeeb, F. Golatowski, and D. Timmermann, "Implementing powerful Web Services for highly resource-constrained devices," in *IEEE Int. Conf. on Pervasive Computing and Communications Workshops*, 2011, pp. 332–335.

[19] X. Yang and X. Zhi, "Dynamic Deployment of Embedded Services for DPWS-enabled Devices," in *Int. Conf. on Computing, Measurement, Control and Sensor Network (CMCSN)*, 2012, pp. 302–306.

[20] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, "Beyond 6LoWPAN: Web Services in Wireless Sensor Networks," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 1795–1805, Nov. 2013.

[21] I. Samaras, G. Hassapis, and J. Gialelis, "A Modified DPWS Protocol Stack for 6LoWPAN-Based Wireless Sensor Networks," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 209–217, Feb. 2013.

[22] S. N. Han, S. Park, G. M. Lee, and N. Crespi, "Extending the Device Profile for Web Services (DPWS) Standard using a REST Proxy," *IEEE Internet Comput.*, 2014, Forthcoming.

[23] S. N. Han, G. Lee, and N. Crespi, "Semantic Context-Aware Service Composition for Building Automation System," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 752–761, Feb. 2014.

[24] J. Elliott, R. Eckstein, M. Loy, D. Wood, and B. Cole, *Java Swing*. O'Reilly, 2002.

[25] D. E. Comer, *Internetworking with TCP/IP: Principles, Protocol, and Architectures*. Prentice Hall, 2000.

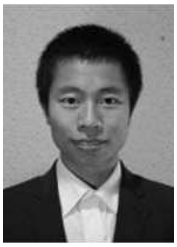
[26] B. L. Nelson, J. S. Carson, and J. Banks, *Discrete-Event System Simulation*. Prentice hall, 2001.

[27] D. Hussein, S. N. Han, X. Han, G. M. Lee, and N. Crespi, "A Framework for Social Device Networking," in *IEEE Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, May 2013, pp. 356–360.

Son N. Han (S'13) received the Dipl.Ing. degree in applied mathematics from the Hanoi University of Technology, Hanoi, Vietnam, in 2006, the M.S. degree in computer science from the University of Seoul, Seoul, Korea, in 2009, and is currently working toward the Ph.D. degree in wireless networks and multimedia services at the Institut Mines-Telecom, Telecom SudParis, Evry, France. From 2009 to 2011, he was a Researcher with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. His research focuses on Web technologies for Internet of Things.

Gyu Myoung Lee (S'02, M'07, SM'12) received the B.S. degree from Hong Ik University, Seoul, Korea, in 1999 and the M.S. and Ph.D. degrees from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2000 and 2007. He is a Senior Lecturer in the Liverpool John Moores University (LJMU), Liverpool, UK since 2014 and an adjunct professor in KAIST since 2012. Prior to joining the LJMU, he worked for Institut Mines-Telecom, Telecom SudParis, France, from 2008 to 2014. Until 2012, he had been invited to work with the Electronics and Telecommunications Research Institute (ETRI), Korea. He was a research professor in KAIST, Korea and a guest researcher in National Institute of Standards and Technology (NIST), USA, in 2007. He has been actively working for standardization in ITU-T, IETF, and oneM2M and currently serves as a Rapporteur and an Editor in ITU-T. His research interests include Internet of Things, future networks, multimedia services, and energy saving technologies.

Noel Crespi (M'07, SM'08) received the Masters degrees from the University of Orsay, Orsay, France, and the University of Canterbury, Christchurch, New Zealand, the Dipl.Ing. degree from Telecom ParisTech, Paris, France, and the Ph.D. and Habilitation degrees from Paris VI University, Paris, France. In 1993, he was with CLIP; Bouygues Telecom, France Telecom R&D, in 1995; and Nortel Networks, in 1999. He joined the Institut Mines-Telecom, Telecom SudParis, Evry, France, in 2002, and is currently a Professor and Program Director, leading the Service Architecture Laboratory. He is a Coordinator for the standardization activities in ETSI and 3GPP. He is also a Visiting Professor with the Asian Institute of Technology and is on the four-person Scientific Advisory Board of FTW,Austria. He has authored/coauthored more than 250 papers and contributions in standardization. His research interests include service architectures, P2P service overlays, future Internet, and Web-NGN convergence.



Nguyen Van Luong (S'13) received the Dipl.Ing. degree in information and communication technology from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2012, the M.S. degree in software engineering for Ambient Intelligence from the Institut Mines-Telecom, Telecom SudParis, Evry, France, in 2014, and is currently a research engineer in Percevio, Paris, France. His research focuses on Web technologies for Internet of Things and WebRTC.

Kyoungwoo Heo received the B.S. degree in electrical communications engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2008. Currently, he is an integrated M.S. and Ph.D. student at the University of Science and Technology (UST) and Electronics and Telecommunication Research Institute (ETRI), Daejeon, Korea. His research interests are multimedia communication and network function virtualization.

Mihaela But is a Project Manager in Theresis Innovation Center, Thales Services S.A., Palaiseau, France. She received Ph.D. degrees in Computer Science in 2008, and in Humanities in 2000. Her works are focused on the areas of knowledge management, semantic web, web of things, ambient intelligence, security infrastructures for public transport, information indexing and retrieval, personalized recommendations, business process automation and interoperability. She has participated in 12 European projects and published more than 40 scientific papers.

Patrick Gatellier is a Research Director at the Theresis Innovation Center, Thales Services S.A., Palaiseau, France. After initial works in graphical systems and compiler developments, he has spent most of his career in Thales Group as a Project Director for projects of French tax administration targeting tens of millions users such as multi-purposes tax authority portal, French National Land Register, and the GAIA citizen global relationship system. His research focuses on object context and representation, multi-modal dialogs and human factors in large systems.