Future Generation Computer Systems 76 (2017) 180–197

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Semantic service provisioning for smart objects: Integrating IoT applications into the web

Son N. Han *, Noel Crespi

*Telecom SudParis, 9 Charles Fourier, 91011 Evry, France*

## HIGHLIGHTS

- Introduce smart objects and related communication technologies to enable an IP-based Internet of Things (IoT) and the vision of IoT applications on Web.
- Propose service provisioning architecture for smart objects to integrate IoT applications into the Web.
- Propose new algorithms and mechanisms for realizing service provisioning for smart objects.
- In-network implementation of the proposed architecture.

## ARTICLE INFO

*Article history:*
Received 25 November 2015
Received in revised form
20 December 2016
Accepted 29 December 2016
Available online 6 January 2017

*Keywords:*
Internet of Things
6LoWPAN
Service provisioning
Smart object

## ABSTRACT

Internet of Things (IoT) applications residing on the Web are the next logical development of the recent effort from academia and industry to design and standardize new communication protocols for smart objects. This paper proposes the service provisioning architecture for smart objects with semantic annotation to enables the integration of IoT applications into the Web. We aim to bring smart object services to the Web and make them accessible by plenty of existing Web APIs in consideration of its constraints such as limited resources (ROM, RAM, and CPU), low-power microcontrollers, and low-bitrate communication links.



## 1. Introduction

The Internet of Things (IoT) is stimulating innovations in virtually all sectors of the economy attracting not only researchers and professionals, but also entrepreneurs, end-users, and even lawmakers. The IoT with its capacity to connect objects to the Internet, blending physical and digital worlds, is going to mark a revolution in how we communicate with other people and everything surrounding us.

Thanks to the advent of IoT technologies, several commercial smart devices improving our everyday life already existed in the market such as Koubachi plant sensor,[1] Alba light bulb,[2] and Luna mattress cover[3] to name just a few. Koubachi plant sensor can measure soil moisture, sunlight, infrared light, and ambient temperature to determine the exact needs of the plants and provide users with highly-specific care advice. Luna mattress cover is able to warm up the bed, track one's sleep, and even wake you up, if necessary. The sensors on the Alba light bulb make it the world's first responsive bulb: its internal sensors allow it to automatically maintain the proper light level, adjust the color of the light according to the time of day, and adapt to the people in the room. What is more, everything surrounding us such as chairs, windows curtains, light bulbs, office equipment, home appliances, and even baby dummies can be turned into Internet-connected smart objects to enhance many application domains (e.g., building automation, healthcare services, smart grids, transportation, and environmental monitoring). A smart object is defined as an item equipped with a form of sensor or actuator, a tiny microprocessor, memory, a communication module, and a power source [1]. They are electronic embedded devices characterized by sensing, processing, and networking capabilities. This can be done by extending the design of

* Corresponding author.
*E-mail address:* son.n.han@outlook.com (S.N. Han).

[1] http://www.koubachi.com/products/outdoor/.

[2] http://stacklighting.com/.

[3] http://lunasleep.com/.

http://dx.doi.org/10.1016/j.future.2016.12.037
0167-739X/

electronic appliances, which fundamentally requires a new set of microelectronic technologies and communication protocols.

To facilitate the smart object connectivity while considering its limited resources (e.g., computing capacity, power, and memory), research and industry have come up over the past decade with a number of advances in low-power microelectronic, radio communication, and corresponding Internet Protocol (IP) networking. IP for decades has effectively supported Internet applications such as email, the Web, Internet telephony, and video streaming. Internet Protocol version 6 (IPv6) is expected to accommodate a huge number of entities, enough for a inconceivably-large number of objects going to be connected to the Internet. These technologies are being engineered by standardization bodies led by Internet Engineering Task Force (IETF) to make them open and accessible to everyone. The objective is for smart objects to consume very low energy, become IP-enabled, and to be an integral part of the services on the Internet. The configuration of smart objects create a new type of networks collectively referred to as IPv6 low-power wireless personal area networks of smart objects (6LoWPANs), which can provide the IP networking infrastructure for the future IoT applications.

6LoWPAN plays an important role in IoT for its benefits of the energy consumption, ubiquitous availability, and the Internet integration of smart objects. First, energy consumption has become an critical issue for modern sustainable development, especially in the time when a huge number of smart objects staying connected to the Internet. The energy used for only maintaining the connectivity of predictably 50 billion objects [2] by current wireless technologies such as Wi-Fi and Bluetooth would account for a considerable large amount the current world energy capacity. Therefore, low-power radio hardware and software protocols are crucial for facilitating a practical IoT ecosystem. Second, more and more wireless devices become available in today's consumer electronics market creating an ubiquitous environment surrounding us which is gradually changing our life style. Advantages to the wireless connectivity are manifold such as the convenience to users, easy deployment, and even for aesthetic aspects that no wires are required. Third, IPv6 with its huge address space is the future for smart objects to seamlessly join the Internet. 6LoWPAN is known under several name such as Low-power Wireless Personal Area Network (LoWPAN) [3], Low-power and Lossy Network (LLN) [4], Constrained Environment [5]. In this paper, 6LoWPAN is used to refer to a network of IPv6, low-power, and wireless smart objects using several IETF standards from working groups including Routing Over Low-power and Lossy Networks (roll), Constrained RESTful Environments (core), and DTLS In Constrained Environments (dice), IPv6 over Networks of Resource-constrained Nodes (6lo), and IPv6 over Low-Power Wireless Personal Area Networks (6lowpan).

On the other hand, the popularity of applications on the Web, along with its open standards and accessibility across a broad range of devices such as desktop computers, laptops, mobile phones, and gaming consoles make the Web an ideal universal platform for future IoT applications. In this future environment, smart objects will be able to offer their functionality via RESTful APIs, enabling other components to interact with them dynamically. The functionality offered by these devices (e.g., temperature sensor data) is referred to as smart object service provided by embedded systems that are related directly to the physical world. Unlike traditional Web services and applications, which are mainly virtual entities, smart object services provide realtime data about the physical world. IoT applications can therefore support a more efficient decision taking process. Hence, smart objects providing their functionality as Web services can be used by other entities such as other Web services, enterprise applications, or even other smart objects. The process of preparing and providing smart object services to the Web is called service provisioning aiming to deliver smart object services to the Web, similar to today's millions of Web services are functioning.

Then comes a new opportunity for truly-intelligent and ubiquitous applications that can incorporate smart object services and conventional Web services using open Web standards. We call these applications *IoT applications on Web*. The arrival of IoT applications on Web also exposes a new opportunity for conventional Internet applications to shift their business model to catch up with this new ecosystem. The concept does not only refer to IoT applications running on Web browser but also to any application residing on the Internet communicating to smart objects and *user agents* using open Web standards via Web Application Programming Interfaces (Web APIs).

Once smart object services reach the Web through communication networks, applications over connected smart objects will go beyond homes, offices, and public spaces to reach the truly global ubiquitous status. The Web then will also undergo the similar evolution to extend their tentacles to the new kids in the block, smart objects, integrating the physical world for more useful and intelligent applications. These applications should be developed in a relatively easy and intuitive way in which developers can use different platforms, frameworks, tools, and programming languages. It is therefore essential to provision services of smart objects in 6LoWPAN to the Web and make them accessible and workable with plenty of existing Web services or APIs. These services also need to catch up with the new trends in the Web world wherein Semantic Web technology (envisioned by Tim Berners-Lee) is predicted to bring more intelligence to the Web. Tim Berners-Lee described Semantic Web as a Web of linked data that can be processed directly by machines allowing applications to automatically infer new meaning from all the information out there [6].

This paper proposes a complete solution to provision smart object services in 6LoWPAN with semantic annotation in order to empower the development and integration of IoT applications into the Web. The solution complies with the constraints of smart objects: limited ROM, RAM, CPU, low-power microcontrollers, and low-bitrate radio.

## 2. 6LoWPAN protocol stack

The IoT aiming to integrate smart objects into the Internet introduces several challenges since many of the existing Internet technologies and protocols were not designed for constrained resources in smart objects. IoT, therefore, has fostered the development of many extensions and adaptations of Internet technologies for the new class of networked objects. This results in a new IP protocol stack for IoT to enable the communication between Internet-connected smart objects and other machines on the Internet. The IoT protocol stack is contributed not only by research results from academia but also from standardization bodies such as Internet Engineering Task Force (IETF), Institute of Electrical and Electronics Engineers (IEEE), and European Telecommunications Standards Institute (ETSI).

Fig. 1 shows the IoT protocol stack along with some common protocols for each layer. It extends four layers of the TCP/IP model (RFC 1122: Link, Internet, Transport, and Application) with the new Adaptation layer, which is required for smart objects to adapt the small frame size of the low-power link layer to the much larger size of IPv6 packets. Adaptation layer defines mechanisms and protocols for header compression/decompression to enable the use of IPv6 on low-power links of smart objects.

<table>
<tr><th colspan="2">IP Protocol Stack</th><th>TCP/IP (RFC 1122)</th><th colspan="2">IoT Protocol Stack</th></tr>
<tr><td colspan="2">HTTP, SOAP, XMPP</td><td>Application</td><td colspan="2">CoAP, DPWS, XMPP, MQTT</td></tr>
<tr><td>TCP, UDP</td><td>ICMP</td><td>Transport</td><td>UDP, TCP, SCTP</td><td>ICMPv6</td></tr>
<tr><td colspan="2">IP</td><td>Internet</td><td colspan="2">IPv6</td></tr>
<tr><td colspan="2" rowspan="2">Ethernet</td><td rowspan="2">Link</td><td colspan="2">6LoWPAN, 6TSCH</td></tr>
<tr><td colspan="2">IEEE 802.15.4, BLE, PLC, TSCH</td></tr>
</table>

6LoWPAN = IPv6 over Low power Wireless Personal Area Networks.
6TISCH = IPv6 over the TSCH mode of IEEE 802.15.4e
BLE = Bluetooth Low Energy
CoAP = Constrained Application Protocol
DPWS = Devices Profile for Web Services
MQTT = Message Queue Telemetry Transport
PLC = Power-line communication
SOAP = Simple Object Access Protocol
TSCH = Time-Slotted Channel Hopping
XMPP = Extensible Messaging and Presence Protocol

**Fig. 1.** Regular IP and IoT protocol stacks in reference to the TCP/IP networking model.

## 3. IoT applications on web

The Internet is a scalable global network of computers that interoperate across heterogeneous hardware and software. On top of the Internet, the Web is an outstanding example of how a set of relatively simple and open standards can be used to build very complex systems while preserving efficiency and scalability. The Web and its underlying open protocols have become a part of our everyday life—something we access at home or on the move, through our laptop computers, phones, tablet, TV, or wearable devices. It has changed the way we communicate and has been a key factor in the way the Internet has transformed the global economy and societies around the world.

Meanwhile, the IoT will allow physical objects to transmit data about themselves and their surroundings, bringing more information about the real world online and help users to better interact with their surroundings. Flowers, for example, can send you an email or a SnapChat[4] photo of your flower when they need watering. Doctors can implant sensors in your body that give you real-time updates about your health updating frequently to a secure online database of your personal data. Even more, IoT data will go beyond the scope of each own service provider to go online and share with other applications and users. We coin the term *IoT Application on Web* to refer to any Web application interacting with smart objects via communication networks using open Web standards. They are IoT applications and they are Web applications identified by:

- Reside on the Web (on Web server/cloud)
- Use open Web standards
- Interact with smart objects
- Be accessed via Web agents.

IoT application on Web is the natural evolution of Web application when Internet is transforming to the Internet of everything to include smart objects in the loop. There can be an application to get access to your Google calendar with the note of cleaning your living room to have your mother visit in few hours. The application then asks your robot cleaner to automatically wake up and do cleaning. Robot cleaner notifies you (by sending an email or a SnapChat message) when it starts working or finishes the work. Another application can let you talk to your devices in the way you talk to your friend with the support of natural language processing engines; this is the new experience of making friendship with your devices. Yet another application can serve you in the airport to update the status of the flight, providing practical information in the airport, connecting to the boarding machine to update you for any delay of boarding time that you can spend more time doing shopping in duty free. Yet another application can synchronize your TV programs and football schedule and also your social network profile to remind you an upcoming match. These applications all require the interactions of existing Web services and new services from smart objects to create new user experience while assuring the seamless transition from developing traditional Web applications to this new type of IoT applications on Web. This is where our work comes in to solve the fundamental problem of such ecosystem, service provisioning.

[4] https://www.snapchat.com/.

## 4. Literature review of service provisioning in IoT

There have been several studies on service provisioning ranging from early-stage models over Radio-Frequency Identification (RFID) and wireless sensor networks, mostly following the concept of Service-Oriented Architecture (SOA) [7], to recent solutions over IP protocol stacks. This section reviews these works on general and SOA-based models of service provisioning in IoT.

### 4.1. General models

Miorandi et al. [8] in their survey paper discussed that the shift from an Internet used for interconnecting end-user devices to an Internet used for interconnecting physical objects that communicate with each other and/or with humans in order to offer a given service encompasses the need to rethink anew some of the conventional approaches customarily used in networking, computing, and service provisioning/management. The arising of IoT provides a shift in service provisioning, moving from the current vision of always-on services, typically of the Web era, to always-responsive situated services, built and composed at runtime to response to a specific need and able to account for the users' context. When a user has specific needs, she will make a request and an ad hoc application, automatically composed and deployed at run-time and tailored to the specific context the user is in, will satisfy them.

The work in [9] aimed to define an IoT ecosystem from the business perspective then identified service provisioning as one of the key fields to realize the vision of the IoT. The defined IoT business ecosystem is a community of interacting companies and individuals along their socio-economic environment. It is where the companies are competing and cooperating by utilizing a common share of core assets, which can be in a form of hardware and software products, platforms or standards that focus on the connected devices, on their connectivity, on application services over this connectivity, or on supporting services. The connectivity is based on common IoT protocol stack as described in the previous section. In order to realize the ecosystem, service provisioning cooperates with other modules such as Developing, Distribution, and Assurances. For example, the end user could acquire various IoT services through a home gateway that supports several technologies. Automated control of lightning, heating and security but also entertainment services could be provisioned through this gateway. With the interoperability issues diminishing, the end user could separately create contracts with network operators and the application service providers, such as a utility company or a content provider. The model here resembles the contemporary Internet service provisioning.

Prasad et al. [10] presented another model called *opportunistic service provisioning* to deal with the variety of situations that users encounter in everyday life. The model came from the fact that in the

real world, a perfectly matching service for a requirement (or tuned to a situation) may not always be available. In these situations, humans try to locate an approximate and an alternative service for the required one that is available and can solve the immediate necessity. For example, a user wants a cup of coffee from a vending machine (with a stack of paper cups), he can locate the coffee machine using his cell phone. Meantime, these coffee cups can be easily used for drinking water, tea, soup or any kind of liquid. The user may use a coffee cup as a pen stand or even as an ashtray. Thus, the service should be able to locate the coffee cup when a pen stand is required. The services now would be based on the non-availability of the exact solution that is not possible to serve a requirement and availability of a close alternative. This work deals with an opportunistic yet an approximate service paradigm in the Internet of the future, especially, in the light of exponential growth of Internet of Things. The authors discussed the characteristics of such a service and also provided the related structure to realize this framework by representing objects in virtual objects and virtual sensing techniques.

Mandler et al. [11] introduced a perspective of Internet of Services within COMPOSE project.[5] The objective is to benefit from the IoT technologies by seamlessly integrating the real and virtual worlds. The ecosystem can be achieved through the provisioning of an open and scalable infrastructure, in which smart objects are associated with services that can be combined, managed, and integrated in a standardized way to easily build innovative applications. Specifically, this study was conducted on specifying and providing a virtual service execution. Moreover, this defined interfaces needed for appropriate services management throughout services lifecycle, creation, upgrade, reconfiguration, resolving security conflicts, rerouting, etc. An accompanying monitoring component oversees security and privacy criteria and Quality of Service guarantees are met. COMPOSE aimed to manage the lifecycle of services in the marketplace and to provide methods for on-the-fly provisioning of service components with better characteristics.

Lee and Chong [12] approached the problem of service provisioning in a user-centric manner wherein services are created efficiently according to the users' competency in their living environments. The approach involves IoT service together with semantic ontology that can support the composition of services suitable to the situation of users, and by the log records it can modify the corresponding happenings. The proposed architecture aims to handle the limitation of user-centric IoT service provision. It is designed to utilize the web based service platform structure that contains versatility and scalability which multiple users or basic environment can easily apply to be a part of the system. The environment requires interoperability, versatility, efficient communication, mobility, intelligence and active functionality to the user-centric IoT service. It is also to give advance management to the system service integration, service management, location management, context management, traffic management, security and privacy management that are all applied to control the faulty operation caused by deficient requirements. The user-centric IoT service and the gathering of information from the scattered object are done by service composition. The web service platform and distributed structure act as the core of the system to handle service provision from Web of Object[6] environment. And the smart gateway manages the devices which are located in the local area of decentralized domain.

[5] http://www.compose-project.eu/.

[6] http://www.web-of-objects.com/.

### 4.2. SOA-based models

Gagnon and Cakici [13] proposed a framework for provisioning and integrating early-stage IoT services (using RFID) to IT infrastructure and business processes. The framework exploits the SOA in two converging technologies, Business Services Network (BSN) and the IoT. RFID tags can embed high value features essential to various industries such as detecting, classifying, and tracking mobile (sensor-less) objects in a surveillance field, monitoring the performance of electro-mechanical components, and controlling manufacturing equipment. They discussed that the integration of SOA and RFID standards was becoming a strategic research priority to leverage mobile business model such as provisioning Web services with pay-per-use, metered, or on demand business. The framework addresses various issues along a typical transaction in business models including: Supplier, Market, Adopter, and Delivery Issues.

The paper [14] presented the architecture of SOA-based IoT including the on-demand service provisioning (along with dynamic network discovery, query, and selection of Web services). They defined real-world device services as functionalities offered by these devices (e.g., the provisioning of online sensor data) because these services are provided by embedded systems that are related directly to the physical world. Unlike traditional enterprise services and applications, which are mainly virtual entities, real-world services provide real-time data about the physical world. Devices providing their functionality as a Web service can be used by other entities such as enterprise applications or even other devices. Authors discussed that services on embedded devices offer rather atomic operations such as obtaining data from a temperature sensor. Thus, the services that the sensor nodes can offer share significant similarities and could be deployed on-demand per developer request. The core mechanism is that on-demand service provisioning first tries to discover service instance on the network that matches the developer's requirements. If this fails, installation of services on suitable devices are carried out.

Li et al. [15] proposed a three-layer service provisioning framework for service-oriented IoT deployments, which is able to represent, discover, detect, and compose services at edge nodes. The purpose is to develop an effective architecture for service operations in the IoT by extending existing architectures over smart things that are connected to the Internet via heterogeneous access networks and technologies (such as sensor networks, mobile networks, and RFID). The framework has three layers: application layer is connected with a business process modeling component for IoT business process; network layer contains several components to provide the functionalities required by services for processing information and for notifying application software and services about events related to the resources and corresponding virtual entities; sensing layer involves the sensing devices such as RID tags and smart sensors which can record, monitor, and process observations and measurements. The network layer can communicate to the sensing layer with device-level APIs.

### 4.3. RESTful service provisioning

Web resources identified by Universal Resource Identifiers (URIs) are considered as the core of modern Web architecture. They are accessed by clients in a synchronous request/response fashion using Hypertext Transfer Protocol (HTTP) methods such as GET, PUT, POST, and DELETE. Resource state is kept only by the server, which allows caching, proxying, and redirection of requests and responses. Web resources may contain links to other resources creating a distributed Web between Internet endpoints, resulting in a highly scalable and flexible architecture. These are

the fundamental concepts of the Web, i.e., Representational State Transfer (REST) [16]. REST has emerged as a predominant Web design model with more than ten thousand RESTful APIs (services) at the time of this article [17].

The RESTful service abstraction advocated by many researchers and professionals is an essential step to provision services in IoT systems. Guinard et al. in several studies [18–23] present a continuous effort to integrate smart objects of different forms ranging from RFID, to WSNs, to embedded systems, to the Web by representing their data and events using RESTful APIs via device gateways. Based on that, authors develop two approaches for mashup: Physical–Virtual and Physical–Physical in a number of applications. Many other studies [24–26] also find their ways to explore this trend over sensor nodes and embedded devices.

### 4.4. Semantic annotation and provisioning

Literature in applying Semantic Web technologies to IoT is focusing on semantically annotating data from smart objects similar to what Semantic Web envisions about the Web of Linked Data. The predominant technique for representing semantics is using Resource Description Framework (RDF) [27], which represents knowledge as triples (subject, predicate, object) (e.g., [*TempSensor803, hasValue, 18*] and [*TempSensor803, locatedIn, Room803*]). A set of triples forms a graph where subjects and objects are vertices and predicates are edges. The advantage of RDF and graph data model is that one can infer new knowledge from existing graph. For example, a system can use domain knowledge to understand that the temperature in Room 803 is 18°, which is transitive property. The domain knowledge is often expressed using Web Ontology Language (OWL) [28], one of the main languages (with RDF schema) to define ontologies on the Web.

To carry out the annotation on smart objects, World Wide Web Consortium has pioneered to establish a working group to gather contributions in this field and to define the first universal ontology for semantic sensor networks (SSNs) [29]. They developed SSN ontology[7] that is an OWL 2 ontology being able to describe sensors in terms of capabilities, measurement processes, observations and deployments. The SSN ontology follows a central Ontology Design Pattern (ODP) [30] depicting the relationships between sensors, stimulus, and observations. The ontology can be seen from four main perspectives: a sensor perspective, with a focus on what senses, how it senses, and what is sensed; an observation perspective, with a focus on observation data and related metadata; a system perspective, with a focus on systems of sensors and deployments; and, a feature and property perspective, focusing on what senses a particular property or observations have been made about a property.

Several studies focused on publishing semantic sensor data. Sense2Web [31], for example is a linked-data platform to publish sensor data and link them to existing resource on the Semantic Web. Sense2Web facilitates the publication of linked sensor data and makes this data available to other Web applications via SPARQL [32] endpoints. Pfisterer et al. [33] introduced the vision of Semantic Web of Things for building semantic applications involving Internet-connected sensors as easy as building, searching, and reading a Web page today. This is done by a crawler periodically scanning the Semantic Web of Things for semantic entities and sensors, downloading metadata and prediction models using their Web APIs, converting this information into RDF triples, and storing them in the triplestore.

The work in [34] is another approach in provisioning semantic annotation for IoT smart objects, similar to the Semantic Web of Things vision. It is about a platform-independent Wiselib RDF Provider to enable the Internet-connected smart objects to act as semantic data providers. They can describe themselves, including their services, sensors, and capabilities, by means of RDF documents. A smart object can auto-configure itself, connect to the Internet, and provide Linked Data without manual intervention. The authors proposed to use a semantic storage for storing RDF documents from smart object data and a data provider responsible for dynamic parts of the RDF documents, such as measurements. It converts sensing data to RDF and inserts it into the semantic storage. Using the Wiselib's callback sensor concept, the data provider gets notified when the value of its associated sensor changes. Another module RDF service broker provides an interface for clients to access and modify the RDF in the storage and to manage subscription from clients.

[35] Bimschas et al. investigated unified concepts, methods, and software infrastructures that support the efficient development of applications across the Internet and the embedded world based on Semantic Web technologies. From an abstract point of view, the main task of IoT application developers is obtaining the data for a specific task. In distributed systems, this requires (1) to identify entities holding the data and (2) to retrieve them. In this paper, authors proposed a methodology to simplify IoT application development. The approach combines technologies from the Internet of Things and the Semantic Web to provide this service efficiently. The central idea is to let entities provide self-descriptions of their type, capabilities, services, etc. in a machine-readable manner.

The paper [36] presented an IoT semantic service model for different components in an IoT framework over physical entities. It is also discussed how the model can be integrated into the IoT framework by using automated association mechanisms with physical entities and how the data can be discovered using semantic search and reasoning mechanisms. The entity constitutes *things* in the Internet of Things and could be a human, animal, car, store or logistic chain item, electronic appliance, or a closed or open environment. The relations between services and entities are modeled as associations. These associations could be static, e.g., in case the device is embedded into the entity or dynamic, e.g., if a device from the environment is monitoring a mobile entity. The semantic modeling and OWL/RDF descriptions solve the interoperability issues within the stakeholders that have agreed and/or provided data using the models.

Kleine argued in [37] that a key indicator for sustainable application development is the reusability of components and data provisioning. The provisioning of sensor readings as CoAP Web services is a straightforward way to integrate the sensors (the physical things) into the Internet and thus makes them part of the IoT. He proposed to divide the data model into three separate parts with Data Provider stay in between Data Origin and Data Consumer. The central component of the Data Provider is the Smart Service Proxy (SSP) which acts as the intermediate device between the client (Data Consumer) and the resource (Data Origin). SSP contains a semantic database as the presentation of data collected from sensor nodes, which is the core of the provisioning process. Since the SSP focuses on semantic service provisioning, the cache is well fitted to semantic content, i.e., triples. This allows Data Consumers not only to retrieve cached resource states but also use SPARQL to find resources with certain properties. The SSP provides an endpoint to run queries on its cached resources via its Web URI.

### 4.5. Literature analysis

We observe several problems in literature about IoT service provisioning as follows:

[7] http://purl.oclc.org/NET/ssnx/ssn.

- Most of the studies focus on the high-level architecture and models for service provisioning without sufficient details about networking protocols at smart object level and about the integration with traditional services at application level. Services from smart objects possess different characteristics then traditional ones as they operate in constrained environments (e.g., low capacity nodes, lossy and low-rate network). It is therefore necessary for service provisioning architecture to consider these properties.
- Current studies have not considered a full IP IoT in service provisioning, which results in the use of protocol gateways to translate non-IP to IP-based communication. Protocol gateways are complex to design, manage, and deploy; their network fragmentation leads to non-efficient networks because of the inconsistent routing, QoS, transport, and network recovery. End-to-end IP architecture is considered suitable and efficient for scalable networks of large numbers of communicating devices such as the IoT.
- Service provisioning in SOA-based IoT using W3C Web Service architecture is facing many difficulties such as the heavyweight of Simple Object Access Protocol (SOAP) messages and the complex parsing XML documents. Web APIs are providing an efficient ways of interacting between Web applications ensuring smooth and simple operation of the Web and coping with the future participation of millions of smart objects. This approach originally aim to IoT application in enterprise solutions which base on business processes of Web services.
- Semantic annotation of smart objects is incorporated within the annotation of sensor data. Whilst, the annotation of functionality (i.e., not data) is also important for these services are present in a great number of smart objects such as services to switch on/off a light bulb and to activate a watering system. The future of IoT is driven by many types of objects that carry not only data but also functionalities. Currently, there are two methods for annotating smart objects (either data or functionality): direct annotation and third-party service. The former incurs large data stored in smart objects and large exchange messages due to the use of XML-based RDF standard. The latter represents a single bottle neck by which the communication stream can be broken or interfered.

In this article, we aim to overcome these problems by proposing a new semantic service provisioning to empower the IoT applications on Web.

## 5. Provisioning requirements

For the integration of IoT applications into the Web in a practical and scalable manner, there still exist several challenges that need to be addressed. We first analyze these problems to establish requirements for service provisioning.

### 5.1. Service discovery

An important issue for developing robust IoT applications is that the applications should be resilient to changes that might occur over time in smart objects (e.g., availability, mobility, and service description) without or with limited need for any external human intervention. Suitable mechanisms for service/resource discovery have been defined. The Constrained Application Protocol (CoAP) [38] defines a procedure used by a client to learn about the endpoints exposed by a CoAP server. A service is discovered by a client by learning the well-known Uniform Resource Identifier (URI) */.well-known/core* (RFC 5785) that contains URIs or links of available services in CoRE Link Format (RFC 6690). CoAP, however, does not specify how a node joining the network for the first time, which can be extended by using multicast communications (RFC 7390). The Devices Profile for Web Services [39] uses WS-Discovery mechanism with multicasting that does not require any central service registry such as Universal Description, Discovery and Integration (UDDI) for Web services. In both cases (DPWS and CoAP), multicast service/resource discovery is applicable when a client needs to locate a service within a local network scope supporting IP multicast. This multicast discovery mechanism operates only within an IP multicast domain and does not scale to larger networks that do not support end-to-end multicast such as the Internet. Centralized approaches could be a solution for service discovery. However, for instance, the resource discovery of the CoAP protocol, suffers from scalability and availability limitations and is prone to attacks such as denial of service (DoS) [40].

### 5.2. Semantic annotation

The Web is becoming more meaningful thanks to the Semantic Web technology that require new systems to be able to understand its language and standards. Semantic annotation of smart objects is incorporated within the annotation of sensor data. Whilst, the annotation of functionality (i.e., not data) is also important for these services are present in a great number of smart objects such as services to switch on/off a light bulb and to activate a watering system. The future of IoT is driven by many types of objects that carry not only data but also functionalities. Currently, there are two methods for annotating smart objects (either data or functionality): direct annotation and third-party service. The former incurs large data stored in smart objects and large exchange messages due to the use of XML-based Resource Description Framework standard [27]. The latter represents a single bottle neck by which the communication stream can be broken or interfered.

### 5.3. Simultaneous requests

The 6LoWPAN design enables smart objects to be accessed directly from Internet using native IP protocols without any protocol translation support. However, smart objects only support a very small number of simultaneous requests due to their resource-constrained nature (memory, processing power, and communication bandwidth) and this issue is also related to the implementation of the networking stack. Although the use of constrained operating systems with a full IoT protocol stack (e.g., Contiki OS) can manage these requests, it can cause the long delay in service response. The delay increases significantly when more requests come to smart objects as can be seen in Fig. 2. A single service request delays at very short time of 50 ms; two or more requests take the smart object several seconds to response; 5 requests create 5 s delay and the figure soars to 35 s in case of 20 simultaneous requests.

### 5.4. Service authorization

When making smart objects available for services on the Internet, beside assuring an interoperable deployment model (i.e., using IP protocols and Web APIs), security measures have to be taken into account that smart objects cannot be hijacked or hacked, making sure access to the smart object is still under controlled by the physical owners. The challenge with service provisioning of smart objects for IoT applications on Web is that the owner of smart objects must give out the access to the applications meanwhile maintaining the secure control of smart objects. If a service provisioning server provides a smart object API to the public or just only to a set of registered third-party developers, it might be possible for developers to misuse the smart objects.

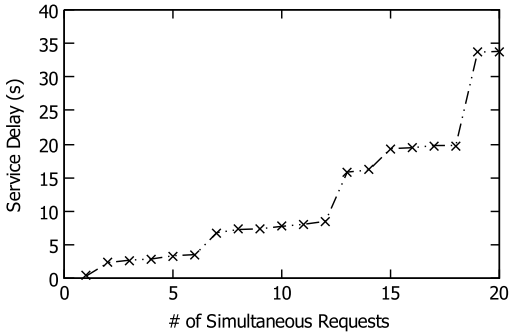



**Fig. 2.** Comparison of service delay when multiple simultaneous requests are sent to one smart object. Results are from the experiment described in Fig. 8 where multiple requests are sent to one Cooja node in a 6LoWPAN network.

### 5.5. Web API generation

Web APIs are specifications that define how to interact with software components, particularly, allow access to remote Web resources via a communication network. The benefit for developers in adopting Web APIs is an easy way to enrich functionality, simple and quick to integration, and leveraging brand strength of established partners. Even in the new platform of smartphone applications, we can already see that the use of Web APIs is prevalent. Shazam,[8] for example, the application that allows users to recognize pieces of music in real-time, integrates Web APIs from many providers such as the Spotify, YouTube, Amazon, iTunes, and radio APIs. Additionally, it allows social sharing, which presumably is realized by using the Web APIs of the various social platforms. The ability to offer smart objects in the form of Web API, therefore, is also an important requirement for the integration of IoT applications into the Web.

## 6. System architecture

This section presents details on our proposed service provisioning architecture including Reference Infrastructure and Functional Block Diagram.

### 6.1. Reference infrastructure

Fig. 3 illustrates the reference architecture in which smart objects are items equipped with sensors or actuators, tiny microprocessors, memory, low-power communication devices, and power sources. Smart objects exist in several real-life facilities such as buildings, houses, and public spaces. Most of them are constrained devices with even few hundred kilobyte memory and is battery-powered. They run low-power operating system implemented with IP-based protocols. These smart objects configure a 6LoWPAN based on low-power physical layer protocols such as IEEE 802.15.4, Bluetooth Low Energy (BLE), and DECT Ultra Low Energy. The 6LoWPAN connects to regular IP networks via a 6LoWPAN Edge Router (6EdR) and beyond to the Internet through a series of other routers across the network. Smart objects are first manufactured with primitive services inside, which can be re-programmed. These services are then provisioned to the IoT applications on Web by the method presented in our proposed architecture. These applications are hosted on the Web servers or cloud and can be accessed via user devices such as laptop computers, smart phones, and tablets.

[8] http://www.shazam.com/.

This reference architecture can be realized in home networks. For example, a home hosts several smart objects including a wireless camera, a wireless LED smart bulb, and an alarm. These objects join the home network via Ethernet coaxial cables (alarm) or wirelessly by BLE (camera, smart bulb). The network connecting to a 6EdR acts as an access point for home Internet connection, and also connects to other devices using full IP capacity such as laptop and TV. A smart phone application can use the Web API provisioned from these smart objects to provide a handy tool for users to remotely control their home with tasks such as switch on or off a light bulb. Another application is a Home Surveillance Web application providing surveillance service for users to remotely track their home environment such as notifying users that their kids are at home.

### 6.2. Functional block diagram

Fig. 4 depicts the proposed service provisioning architecture with functional blocks divided into three subsystems: service communication, service provisioning, and service integration. These functional blocks provide guidelines for implementing relevant IP networking stack in smart objects. IP networking for smart objects is the foundation for facilitating services using application layer protocols doing semantic annotations to these services. It relies on open and standardized protocols mainly from IETF working groups. Service provisioning method for secure, scalable, and reliable services of 6LoWPAN includes: service discovery, scheduling, URI mapping, request handling, authentication, and Web API representation. A method for using provisioned services from smart objects includes steps: retrieving Web API from service providers, requesting authentication tokens, requesting a smart object service, receiving response from smart object, querying and reasoning using an appropriate domain ontology, and mashing up with other APIs.

## 7. Semantic service provisioning

We propose the service provisioning architecture to meet aforementioned five requirements presented in Section 5: service discovery, semantic annotation, simultaneous requests, authorization, and Web API generation. As can be seen from Fig. 4, Resource Management provides a user interface for managing smart objects in the provisioning network as well as granting authorization for IoT applications on Web via Authorization block. Scheduling cooperates with Request Handling to coordinate multiple simultaneous requests to ensure the quality of service. Service Discovery handles native discovery protocols in 6LoWPAN and feed them to Semantic Annotation and to the Web API Generation, which in turn call URI Mapping process to generate API endpoints. Triplestore provides the semantic storage for provisioning services.

### 7.1. Service discovery

This function block resides at the lowest level of provisioning functionality on local network side to directly interact with devices. It is required to discover available services to carry out the provisioning. Web services are usually discovered by querying registries using interfaces such as Universal Description Discovery and Integration (UDDI). While it can be a convenient way to discover services, its centralized nature can lead to many issues such as fault tolerance, performance, and scalability. In DPWS, multicasting-based WS-Discovery does not require any central service registry. When an application tries to locate a device in a network, it sends a UDP multicast message (using the SOAP-over-UDP binding) carrying a SOAP envelope containing a WS-Discovery Probe message with the search criteria, e.g., the name

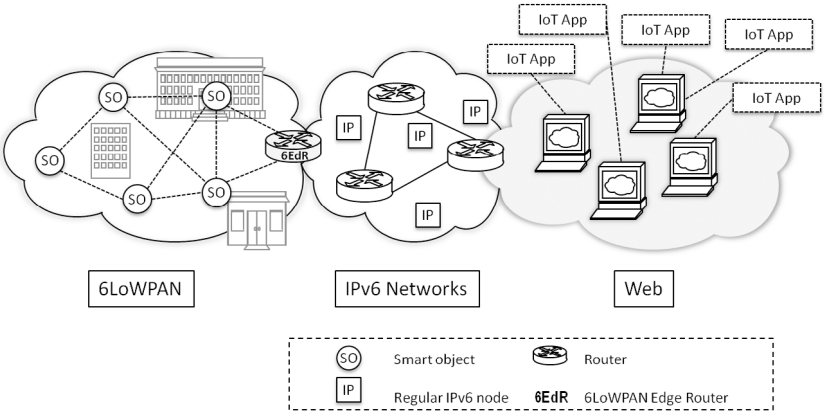



**Fig. 3.** The reference infrastructure.

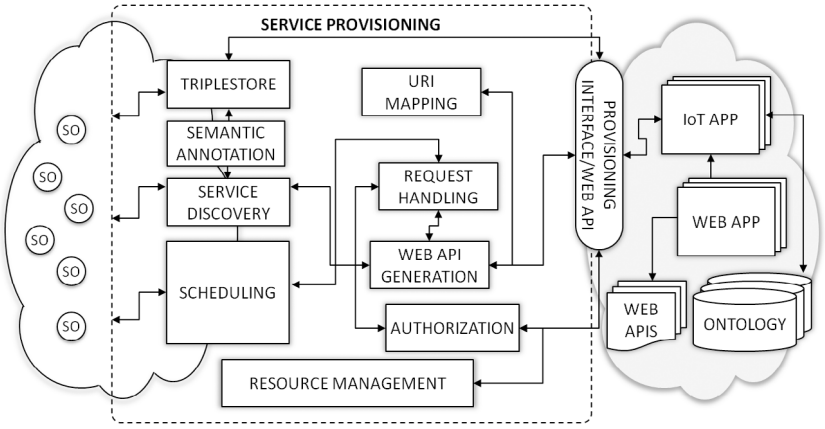



**Fig. 4.** Functional block diagram.

of the device. All the devices in the network (local subnet) that match the search criteria will respond with a unicast WS-Discovery Probe Match message (also using the SOAP-over-UDP binding). To achieve resource discovery, CoAP servers provide a resource description available via a well-known URI /.well-known/core (RFC 5785). This description is then accessed with a GET request on the URI.

```
2.05 Content
</.well-known/core>;ct=40,
</control/led>
 title="LED Red, PUT mode=on|off";rt="control"
</status/temp>
 title="Temperature";rt="status"
```

Listing 1: CoRE Link Format.

```
<?xml version="1.0" encoding="UTF-8"?>
<s12:Envelope
 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/Probe
  </wsa:Action>
  <wsa:MessageID>urn:uuid:3ac5f820-d47d-11e3-80c0-358
     d7a9bbe90
  </wsa:MessageID>
  <wsa:To>urn:docs-oasis-open-org:ws-dd:ns:discovery
     :2009:01</wsa:To>
 </s12:Header>
 <s12:Body>
  <wsd:Probe />
 </s12:Body>
</s12:Envelope>
```

Listing 2: WS-Discovery Probe message.

The Service Discovery provides the same interface to query services regardless of the protocol (e.g., CoAP, DPWS) used in the 6LoWPAN. It is in the form of plugin, when we need to incorporate new protocol we can add in to. This function also plays a role as handling several service discovery functionalities happening at multicasting support provisioning network and making some functionalities possible in global scenario such as dynamic service discovery with DPWS. The approach is to apply URI mapping and API representation directly on underlying discovery mechanism of each protocol. In addition, we use a repository to maintain the list of active devices by carrying out the discovery process periodically or when the traffic is detected low in the 6LoWPAN. For example, a smart object has a temperature sensor and an LED indicator to display the status of room temperature. A client can discover these services by sending a request *GET /.well-known/core* to the smart object, which responses with the content shown in Listing 1. This task can be done with the service provisioning service by using the Web API presented in Table 1. Similarly, instead of using complex WS-Discovery Probe message in Listing 2 for DPWS services, we can discover services of the smart object by the same provisioning APIs. From the content of the response message, two services are discovered and provisioned in two Web APIs (see Table 2).

### 7.2. Scheduling

Limited resources in smart objects result in a problem of supporting simultaneous requests from multiple IoT applications

**Table 1**
Discovery API.

| | |
|---|---|
| GET /[uri]/discovery<br>Search for a smart object with criteria | |
| Arguments | N/A |
| Example | GET http://157.159.103.50/[aaaa::212:7400:13cc:3693 /discovery] |

157.159.103.50 is the provisioning server IP address, 8080 is the port number.
aaaa::212:7400:13cc:3693 is smart object IP address.

**Table 2**
Discovered services: Web APIs.

| | |
|---|---|
| PUT /[uri]/control/led<br>Switch on/off LED indicator in the smart object | |
| Arguments | mode = on/off |
| Example | PUT http://157.159.103.50/[aaaa::212:7400:13cc:3693 /control/led?mode=on] |
| GET /[uri]/status/light<br>Get the current temperature | |
| Arguments | N/A |
| Example | GET http://157.159.103.50/[aaaa::212:7400:13cc:3693 /temp] |

157.159.103.50 is the provisioning server IP address, 8080 is the port number.
aaaa::212:7400:13cc:3693 is smart object IP address.

on Web. Multiple requests can happen frequently for it is a typical case in the interaction between applications and smart objects when they get connected and become an integral part of the Internet. Many smart objects such as sensor nodes only support a very small number of simultaneous connections resulting in an ineffective operation of several real-time applications. We solve this problem by using a scheduling algorithm shown in Listing 3. The algorithm consists of four processes: *RequestHandler*, *Scheduler*, *QuantumAssertion*, and *ResponseObserver*. Two requests are considered to be simultaneous if they come one after another in very short time (less than a threshold denoted by *quantum time*).

```
PROCESS RequestHandler
BEGIN
Initiate requestQueue
Keep track of lastReqestTime
If (requestTime is within lastRequestTime bound)
Begin
  Add new request to requestQueue
  Activate the Scheduling process if it is not active
End
END

PROCESS Scheduler
BEGIN
Every quantumTime
Begin
 If requestQueue is empty
   Stop
 Else
   Remove request from requestQueue
   Add request to sentQueue
   Send request
End
END

PROCESS QuantumAssertion
BEGIN
 If sentQueue is not empty and top of queue is overtime
   Adjust quantumTime
 Else
   Reset quantumTime
END

PROCESS ResponseObserver
BEGIN
 If there is a response
   Remove from sentQueue
   Get client id
   Forward to client
END
```

Listing 3: Scheduling algorithm.

The *RequestHandling* process receives coming HTTP requests via the provisioned Web API and check if each request arrives in a reasonable interval. If a request arrives too fast (less than a *quantum time* after the nearest recorded request), it will be added to a *request queue* (based on a queue data structure [41]). The *Scheduling* process keeps track of the *request queue* and it is activated when there are waiting requests in the queue. When the *Scheduling* process starts, it checks the request queue again, removes the head request (first in the queue), adds this request to another queue called *sent queue*, and sends the request accordingly to the target smart object. The *QuantumAssertion* keeps track of the *sent queue* to see if a request has waited for too long to adjust the *quantum time*. The *ResponseObserver* process forwards the received response messages from smart objects to clients and updates the sent queue.

### 7.3. Semantic annotation

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.it-sudparis.eu/sensor#"
    xmlns:ns0="http://www.w3.org/2000/01/rdf-schema#" >
    xmlns:ns1="http://pur1.oc1c.org/NET/ssnx/ssn#"
  <rdf:Description rdf:about="http://www.it-sudparis.eu/
     sensor#Temp5">
    <ns0:type rdf:resource="http://pur1.oc1c.org/NET/ssnx/
     ssn#Sensor"/>
    <ns1:observedProperty>Temperature</ns1:observedProperty>
    <ns1:hasValue>19.2</ns1:hasValue>
  </rdf:Description>
</rdf:RDF>
```

Listing 4: Temperature sensor smart object RDF/XML format.

Tim Berners-Lee coined the term Semantic Web as an extension of the current Web [6] in which data are consumable and understandable to machines. It brings a new concept of representing data in the meaningful graph database model to improve the communication between human and machine. That means Semantic Web can achieve a certain level of automation on Web [42]. When the IoT paradigm arrives and it is now changing the Web, the Semantic Web concept even fits more to its architecture since smart objects need intelligence and automation

in different level to fulfill their tasks. However, similar to other extensions of Internet and Web protocols originally designed for computers to smart objects such as CoAP to HTTP or DPWS to SOAP, straightforward adoption of semantic annotation to smart objects is impractical. It is because of the complexity of the Semantic Web model with the involvement of ontology, triple, and data presentation following specific requirements.

```
@prefix : <http://www.it-sudparis.eu/sensor#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ns0: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ns1:      <http://pur1.oc1c.org/NET/ssnx/ssn#> .
<http://www.it-sudparis.eu/sensor#Temp5>
    ns0:type ns1:Sensor ;
    ns1:hasValue "19.2" ;
    ns1:observedProperty "Temperature" .
```

Listing 5: Temperature sensor smart object N3 format.

Listing 4, for example, shows an example of RDF representation of temperature data from a sensor of a smart object. It uses 506 bytes to semantically represent the data from the smart object with temperature sensing value is 19.2°. Even with Notation3 (N3) format [43], a textual syntax alternative to RDF, the size of data is still rather large (see Listing 5). The reason is that the semantic annotation for smart object involves a great deal of linking information such as namespace and RDF schema. The size of the semantic data in more complex situation may increase and surpass the maximum buffer size that is provided for resource responses, which must be respected due to the limited IP buffer such as the maximum buffer size for CoAP blocks is typically 1024 bytes.

Literature approaches use third-party semantic services/servers to capture and republish these data. This can solve the problem of limited size for semantic annotation but results in many trade-offs that prevent the adoption of this method. For example, third-party server means the communication stream is broken and can be interfered or the communication is slowed down and semantic server becomes a bottleneck in the communication between applications and smart objects. The ideal way is to have smart objects express semantically expressive based on IP protocols. Our approach is very close to this ideal method in which we unburden most of semantic annotation information from smart objects to the provisioning layer, keeping only core data for transmitting while provisioned services still can be fully annotated. We use following scheme:

1. Service providers provide a domain ontology for each set of smart objects. Ontology for each domain is developed independently by a reliable and consensus decision making process, e.g., Semantic Sensor Network Ontology.[9]
2. Each service in smart object is represented in N3 format without default namespaces, ontology, and application URIs.
3. Ontology and application URI are added accordingly in service provisioning layer based on the information from the service provider for ontology and provisioning server for application URI.

The above temperature sensing data can then be provided by smart object by the format provided in Listing 6 while the actual semantic annotation data can be reached from applications are still the same as shown in Listing 5. The Internet media type passing to Web API calls is denoted as *text/n3*.

```
:Temp5
  a ns:Sensor ;
    ns:hasValue "19.2" ;
    ns:observedProperty "Temperature" .
```

Listing 6: Temperature sensor smart object N3 format.

These semantic data queried from smart objects are store in a Triplestore. A triplestore is the storage for semantic data, in this case, referring to the annotation of smart object data and functionalities. A triple is a data entity composed of *[subject, predicate, object]*, there are three triples in the above data and one more triple about the time stamp is added as shown in following Listing 7.

```
[<http://www.it-sudparis.eu/sensor\#Temp5>
  <http://pur1.oc1c.org/NET/ssnx/ssn\#type>
   <http://pur1.oc1c.org/NET/ssnx/ssn\#Sensor>]
[<http://www.it-sudparis.eu/sensor\#Temp5>
  <http://pur1.oc1c.org/NET/ssnx/ssn\#hasValue> "19.2"]
[<http://www.it-sudparis.eu/sensor\#Temp5>
  <http://pur1.oc1c.org/NET/ssnx/ssn\#observedProperty> "
     Temperature"]
[<http://www.it-sudparis.eu/sensor\#Temp5>
  <http://pur1.oc1c.org/NET/ssnx/ssn\#startTime> "2014:04:24
      14:20"]
```

Listing 7: Four triples from temperature sensor.

Triplestore can be realized by serialization (i.e., using file system) or by third-party solutions such as OpenLink Virtuoso,[10] 3Store,[11] and Apache Jena.[12] All the data in triplestore are associated with a domain ontology indicated by the service provider of the smart objects. The ontology is either available on the Web or newly developed by the service provider depending on the field of the applications.

### 7.4. Authorization with OAuth 2.0

OAuth 2.0 [44] is an authorization framework that enables applications to obtain limited access to resources on the Web on behalf of the resource owner. It has been widely used in many services such as Google, Facebook, and GitHub. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth 2.0 provides authorization flows for Web and desktop applications and mobile devices.

OAuth 2.0 fits the security model of the IoT applications on Web where the resource (smart object) owner can authorize an application to access their smart object functions without having full access on handling the smart object such as terminating its operation. The applications have limited accesses to the smart objects according to the scope of the authorization granted (e.g. read only or update) whilst they still can communicate to the smart objects once having been authorized. We therefore adopt OAuth 2.0 as the core of authentication and authorization framework for our proposed provisioning architecture.

Authorization functional block in our proposed service provisioning architecture refers to an OAuth 2.0 authorization provider functionality, which authenticates the identity of the user, in this case locally within the provisioning network to strengthen the security. It issues access tokens to the interested applications following the confirmation from the user. Any IoT application that wants to access the smart object services must be authorized by the user, and the authorization must be validated by the appropriate Web API endpoints. There are three authorization endpoints in the our proposed service provisioning architecture for this process: Authorization URI (/authorize) is the URI on which users grant the authorization to the interested application; Token URI (/token) is the URI called by client applications when they want to exchange a code for an access token, or a refresh token for a new access token. API URI

[9] http://purl.oclc.org/NET/ssnx/ssn.

[10] https://github.com/openlink/virtuoso-opensource.

[11] http://threestore.sourceforge.net.

[12] http://openjena.org.

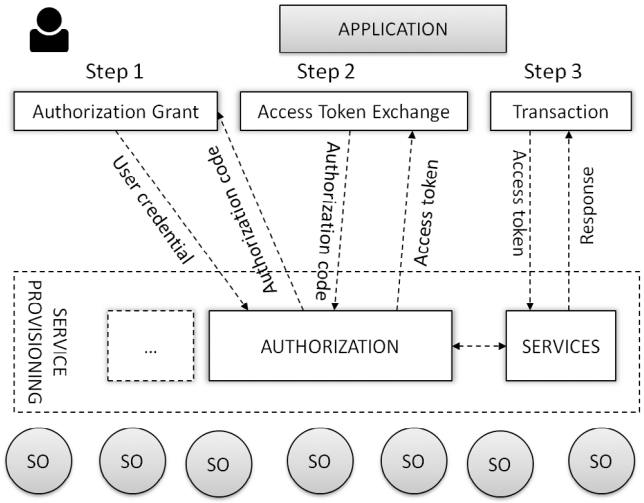



**Fig. 5.** 3-step authorization process for IoT applications on Web.

(/api) is the base URI on which provisioned Web API endpoints are mounted. These Web API endpoints enable a secure communication between IoT applications on Web and 6LoWPAN smart objects. This is done in the three-step mechanism illustrated in the Fig. 5.

1. Step 1: User or the owner of the smart objects gets access to the Resource Management and then goes to the Applications section and looks for the appropriate application to authorize. The user selects the application and click the authorize button to grant the application with Client ID and Redirect URI provided by the application. The Authorization then redirects to the Redirect URI with the authorization code in the URI fragment to transfer it to the application.
2. Step 2: The application requests an access token from the API, by passing the authorization code along with authentication details, including the client secret, to the API token endpoint. If the authorization is valid, the API will send a response containing the access token (and optionally, a refresh token) to the application.
3. Step 3: Now the application is authorized! It may use the token to carry out transactions with real services from provisioning server via the service API, limited to the scope of the access, until the token is expired or revoked.

### 7.5. URI Mapping

We propose two schemes for mapping service URIs to provisioning URIs, which are integral parts of the Web API endpoints. The first scheme is based on the resolved hostnames of smart objects in the network and the second scheme uses IP addresses of smart objects. A thermostat, for example, configured at IP address *aaaa::212:7400:13cc:3693*, has a CoAP service to get the current room temperature binding to its IP address, service port, and service extension: *coap://[aaaa::212:7400:13cc:3693]:5683/status/temp*. The service provisioning service is at address *157.159.103.50*. Then the service URI is mapped to either one of the following provisioning URIs in Table 3.

The first method is straightforward since it does not require any check for address duplication for the IP address is already unique in the network so it is a good candidate for smart object identity. The second method requires the provisioning server to check the hostname duplication. It can be suitable for small homes or offices.

DPWS uses WS-Addressing to assign a unique identification for each smart object (endpoint address), independent from transport specific address. This unique identification is used with a series of message exchanges *Probe/ProbeMatch*, *Resolve/ResolveMatch* to get a transport address and then another series of messages are sent back and forth to invoke an operation. We define a mapping between a pair of DPWS endpoint/transport addresses and a single URI, and then we use the corresponding operation name for each service as the extension of the URI. For example, the aforementioned thermostat has a getTemp() operation implemented in DPWS with the pair of endpoint and transport addresses of *urn:uuid:46932240-d504-11e3-bf6a-6eabe38b6788* and *[aaaa::212:7400:13cc:3693]:4567/thermostat*. Table 4 shows the mapping of these two addresses along with the operation name (*temp*) to a single URI. The mapping is unique for each smart object service, and data are stored in the smart object repository of the proxy. The repository is also updated when there is a change in smart object status and/or periodically when the proxy runs its routine to check all the active smart objects.

### 7.6. Web API generation

Web API Generator is in charge of generating a set of Web API associated to each smart object service. The process is based on above URI mapping scheme. The API consists of endpoints for discovery, subscription, and service calls in Representational State Transfer (REST) architectural style [16]. To generate these RESTful Web APIs, we can extract directly from CoAP URI as CoAP and HTTP basically use the same REST concept. With DPWS, we propose a design constraint on the DPWS implementation for smart objects. It is based on the fact that most smart object services provide relatively simple operations compared to normal Web services with complex input/output data structure. Our proposed constraint follows a simplified CRUD model (“create”, “read”, “update”, “delete”) to map between these services and HTTP methods: DPWS Operation Prefix – CRUD Action – HTTP Method. Specifically, four CRUD actions are applied to map DPWS operations to HTTP methods as in Table 5.

Web APIs are the core of the development of applications on the Web these days providing interfaces for developers to develop applications on the Web. Web APIs are specifications that define how to interact with software components, particularly, allow access to remote Web resources via a communication network. The benefits for developers in adopting Web APIs are: easy to enrich functionality, simple and quick to integration, and leverage brand strength of established partners. Even in the new platform of smartphone applications, we can already see that the use of Web APIs is prevalent. Our provisioning Web API consists of API endpoints represented in the following format (see Table 6), which is used consistently in this paper:

### 7.7. Resource Management

Resource Management functional block is in charge of monitoring and managing the 6LoWPAN and its smart objects. It provides information about the network status such as the number of nodes, network topology, and routing information. It also provides an interface for granting authorization to IoT applications on Web to get access to the provisioned Web API. Resource Management authenticates users by credentials (username/password) via a Web User Interface (Web UI). Fig. 6 shows the Web UI of the Resource Management implemented within ThingsGate provisioning server [45] for a Social IoT application, which is based proposed architecture.

## 8. In-network implementation with DPWS

This section introduces an in-network implementation of the proposed architecture for DPWS protocol. The implementation is in the form of a REST proxy to extend the DPWS standard to better integrate it into the IoT applications on Web while maintaining its

**Table 3**
URI mapping with CoAP.

| | |
|---|---|
| Service URI | coap://[aaaa::212:7400:13cc:3693]:5683/temp |
| Provisioning server | 157.159.103.50 |
| Scheme 1 URI | http://157.159.103.50/thermostat/temp |
| Scheme 2 URI | http://157.159.103.50/[aaaa::212:7400:13cc:3693 /temp] |

**Table 4**
Base URI mapping with DPWS.

| | |
|---|---|
| Endpoint address | urn:uuid:46932240-d504-11e3-bf6a-6eabe38b6788 |
| Transport address | http://[aaaa::212:7400:13cc:3693:4567/thermostat] |
| Service | getTemp() |
| Provisioning server | 157.159.103.50 |
| Scheme 1 URI | http://157.159.103.50/thermostat/temp |
| Scheme 2 URI | http://157.159.103.50/[aaaa::212:7400:13cc:3693 /temp] |

**Table 5**
CRUD operation mapping scheme.

| Prefix | CRUD Action | HTTP Verb |
|---|---|---|
| Get- | Read | GET |
| Set- | Update | PUT |
| Add- | Create | POST |
| Remove- | Delete | DELETE |

**Table 6**
API endpoints format.

| [HTTP-VERB] [URI EXTENSION] [DESCRIPTION] | |
|---|---|
| Arguments | [ARGUMENTS] |
| Example | [EXAMPLE] |

advantages of dynamic discovery and eventing mechanisms. Fig. 7 shows the network topology in two cases of our proposed design and the the original direct DPWS communication.

### 8.1. Devices profile for web services

DPWS is based on Web Service Description Language (WSDL) and SOAP to describe and communicate device services, but it does not require any central service registry such as Universal Description, Discovery and Integration (UDDI) for service discovery. Instead, it relies on SOAP-over-UDP binding and UDP multicast to dynamically discover device services. DPWS offers a publish/subscribe eventing mechanism, WS-Eventing, for clients to subscribe for device events, e.g., a device switch is on/off or sensing when temperature reaches a predefined threshold. When an event occurs, notifications are delivered to subscribers via separate TCP connections.

These features, secure Web services, dynamic discovery, and eventing, are the main advantages of DPWS for event-driven IoT applications. Nevertheless, in fact, developers would face several problems when applying DPWS for IoT applications on Web. The main concern is about the dynamic discovery in which the network range of UDP multicast messages is limited to the local subnet. Therefore, it is impossible to carry out this mechanism in a large network such as the Internet. With WS-Eventing, the establishment of separate TCP connections in case of delivering the same event notification to many different subscribers will generate a global mesh-like connectivity between all devices and subscribers (see Fig. 7). This requires high memory, processing power, and network traffic and thus consumes a considerable amount of energy in devices. Another issue is the overhead due to the data representation in XML format and multiple bidirectional message exchanges. It is not a problem when most DPWS devices currently communicate locally, but in a mass deployment of devices, these messages would generate heavy Internet traffic and increase the latency in device/application communication. Furthermore, W3C Web services use WSDL for service description and SOAP for service communication; the former, despite the fact that it is a W3C standard, requires much effort from developers to process poorly-structured XML data; the latter is mostly common in stateful enterprise applications, whereas recent Web applications are moving toward the core Web

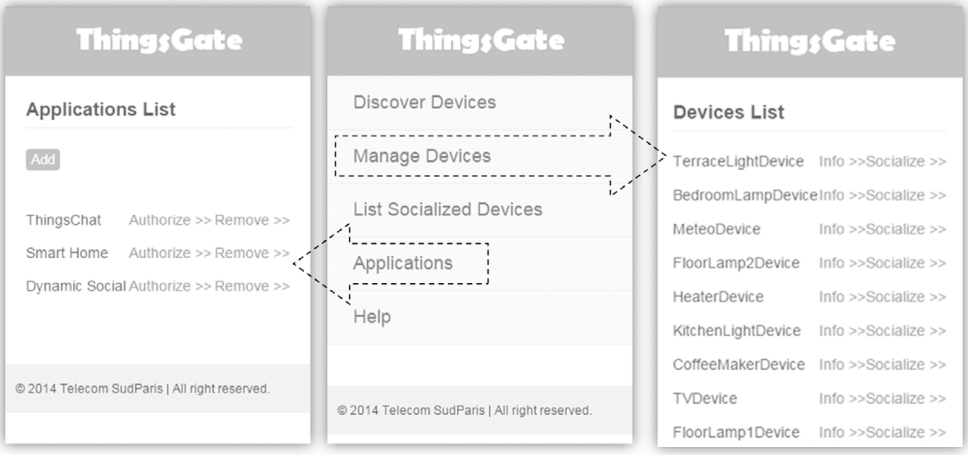



**Fig. 6.** Resource Management Web UI in ThingsGate. Manage Device function/menu shows a list of discovered devices in the 6LoWPAN of home network. User can query detailed information or add social data to each device by Info or Socialize hyperlinks associated to each smart object. Applications function/menu help users to authorize IoT applications on Web to use resources in the 6LoWPAN.

concepts expressed in REST architectural style by offering stateless and unified interfaces of RESTful Web APIs.

To solve these problems, we design a service provisioning mechanism for DPWS using a REST proxy by providing the following features: (1) global dynamic discovery using WS-Discovery in local networks; (2) proxy-based topology for publish/subscribe eventing mechanism; (3) dynamic addressing for DPWS smart objects; (4) RESTful Web APIs; and (5) WSDL caching. The proxy unburdens Internet traffic by processing the main load in local networks. Also, the proxy can extend the dynamic discovery from locally to globally through RESTful Web APIs. Developers do not have to parse complex WSDL documents to get access to service descriptions; they can use RESTful Web APIs to control smart objects.

We will follow an IoT engineer Rosalie's development process to understand what challenges she could encounter when developing, deploying, and interacting the smart object from her IoT application and how the proxy helps her to solve these problems. The following use case illustrates a common situation in several IoT applications when a new smart object joins the network.

### 8.2. Use case

Rosalie would like to make a module for controlling a newly-purchased DPWS heater. The heater is equipped with a temperature sensor, a switch, memory, a processor, and networking media, and is implemented with a hosted Heater service. Heater service consists of eight operations: (1) check the heater status (GetStatus), (2) switch the heater on/off (SetStatus), (3) get room temperature (GetTemperature), (4) adjust the heater temperature (SetTemperature), (5) add (AddRule), (6) remove (RemoveRule), and (7) get (GetRules) available policy rules for defining automatic operation of the heater, and (8) over-heating event eventOverHeat(). She connects the heater to the network and tries to control it from her IoT application.

### 8.3. Global dynamic discovery

When an application tries to locate a smart object in a network, it sends a UDP multicast message (using the SOAP-over-UDP binding) carrying a SOAP envelope that contains a WS-Discovery Probe message with search criteria, e.g., the name of the smart object. All the smart objects in the network (local subnet) that match the search criteria will respond with a unicast WS-Discovery Probe Match message (also using the SOAP-over-UDP binding). In our use case, the heater sends Probe Match message containing network information. At this point, Rosalie realizes that it is impossible for her IoT application to dynamically discover the heater because of the network range limit to local subnet of multicast messages. If a proxy is applied, it allows the application to suppress multicast discovery messages and instead send a unicast request to the proxy. Then, the proxy can representatively send Probe and receive Probe Match messages to and from the network while the behavior of smart objects remains unmodified; they still answer to Probe message arriving via multicast. In networks with many Probe messages, the proxy can significantly unburden the Internet traffic. The proxy provides two RESTful Web APIs to handle the discovery as shown in Table 7.

We also propose a repository in the proxy to maintain the list of active smart objects. The repository is updated when smart objects join and leave the network. In addition, the proxy performs a routine to periodically check the consistency of the repository, says every 30 min. For a proxy with 100 smart objects, the size of the repository is about 600 kb, so it is feasible for unconstrained machines used to host a proxy.

**Table 7**
Discovery API.

| | |
|---|---|
| GET /discovery<br>Search for a smart object with criteria | |
| Arguments | search: search criteria |
| Example | PUT http://157.159.103.50/discovery?search=Heater |
| GET /discovery<br>Get the list of connected smart objects | |
| Arguments | N/A |
| Example | GET http://157.159.103.50/discovery |

157.159.103.50 is the proxy's IP address, and 8080 is the port number.

### 8.4. Publish/subscribe eventing

To receive event notifications, Rosalie can subscribe her application directly to the heater by sending a SOAP envelope containing a WS-Eventing Subscribe message (using the SOAP-over-HTTP binding). The heater responds by sending a *WS-Eventing SubscribeResponse* message via the HTTP response channel. When an event occurs, the heater establishes a new TCP connection and sends an event notification to the subscriber. Therefore, in scenarios with many subscribers, it generates high level of traffic, requiring high resources, and causing smart objects to consume more energy. However, this publish/subscribe mechanism can be done through REST proxy to reduce the overhead of SOAP message exchanges and resource consumption, replacing global mesh-like connectivity by proxy-based topology (see Fig. 7). One RESTful Web API is dedicated for event subscription; instead of sending a WS-Eventing Subscribe message, the application sends an HTTP POST request to the subscription resource (see Table 8).

Fig. 7 shows the network topology in two cases of our proposed design and the original direct DPWS communication. Table 9 shows a list of RESTful Web APIs provided by the proxy for the heater smart object mapping with DPWS operations.

### 8.5. WSDL caching

When an application knows a smart object hosted service (representing smart object functionalities) endpoint address, it can ask that service for its interface description by sending a GetMetadata Service message. The service may respond with a GetMetadata Service Response message including a WSDL document. The WSDL document describes the supported operations and the data structures used in the smart object service. Some DPWS implementations (such as WS4D JMEDS) provide a cache repository to store the WSDL document at runtime. After the application retrieves the WSDL file for the first time, the file can be cached for local usage in the subsequent occurrences within the life cycle of the DPWS framework (start/stop). This kind of caching mechanism would significantly reduce both the latency and the message overhead. Our DPWS proxy can provide WSDL caching not only at runtime but also permanently in a local database. The cache is updated along with the routine to maintain the smart object repository in proxy described in the dynamic discovery section.

## 9. Performance Evaluation

We carry out the experiments with 6LoWPAN set up on Cooja simulator [46]. Experiment results from our previous work [47] allow us to set up a 6LoWPAN network on network simulator with respect to real-life performance. This approach does not lose important properties of smart objects and especially effective to focus on the service integration issues. Cooja can accurately simulate all the constraints in smart objects and 6LoWPAN such as ROM/RAM size, microprocessor instruction set, and IEEE 802.15.4

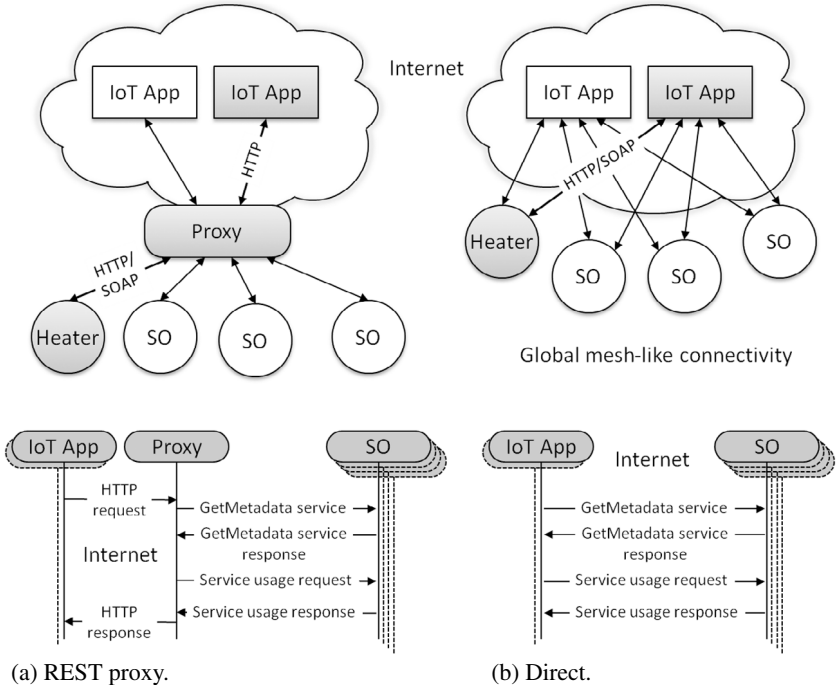
**Table 8**
Event subscription API.

| | |
|---|---|
| POST /[smart object ID]/[event]<br>Subscribe to a smart object event | |
| Arguments | agent: address to send notification messages |
| Example | POST http://157.159.103.50/heater/overheat?agent=157.159.103.63/heating |

157.159.103.50 is the proxy's IP address, 8080 is the port number, 157.159.103.63/heating is the callback endpoint of the application.

**Table 9**
RESTful Web API for the heater.

| RESTful Web API | DPWS operations | Argument | Description |
|---|---|---|---|
| GET /discovery | Discovery | | List smart objects |
| PUT /discovery | | search | Search for smart objects |
| POST /heater/overheat | eventOverHeat() | | Subscribe to an event |
| GET /heater | GetStatus() | | Get heater status |
| PUT /heater | SetStatus(String) | status | Set heater status |
| GET /heater/temp | GetTemp() | | Get room temperature |
| PUT /heater/temp | SetTemp() | temp | Adjust heater temperature |
| POST /heater/rules | AddRule | rule | Add new rule |
| GET /heater.rules | GetRules() | | List of rules |
| DELETE /heater/rules/[ruleID] | RemoveRule() | ruleID | Delete a rule |




(a) REST proxy. (b) Direct.

**Fig. 7.** Network topology in two cases: (a) our proposed design configures a proxy-based topology with local HTTP/SOAP binding, (b) the original smart objects Profile for Web Services (DPWS) communication configures global mesh-like connectivity for HTTP/SOAP binding. Consequently, the original DPWS introduces higher latency and overhead.

radio environment. Fig. 8 shows the 6LoWPAN with 10 random nodes. The longest distance to the 6EdR (node 1) is 3-hop (nodes 1-2-3-4). TX/RX success ratio is set at 98% as suggested in Packet Delivery Ratio test in [47]. Each node is implemented with a CoAP service enriched with the proposed semantic annotation. We aim to test the performance of service provisioning server to see how the proposed algorithms and mechanisms perform in term of transparency and efficiency. The provisioning service is deployed in the simulator host machine, which creates a local network with 6EdR in its Ethernet interface. A Web application is developed in a Web service of the same local network with the provisioning server (the deployment of the same application on a server on Web does not change the nature of the IP communication with the involvement of a number of routers).

### 9.1. Transparency

First of all, the consistent use of IP stack in smart objects as well as in provisioning is aligned with common network infrastructure, which ensures a transparency of communication in the network. 6EdR is an important node in the IP networking model to assure the smooth communication. This can first verified by *ping6* command from a regular IP node to a 6LoWPAN node (see Listing 8). We further examine the transparency of the service provisioning against the implementation of our proposed algorithms, especially for the scheduling. We carry out a single request to a service of node 2 from our IoT application with and without scheduling module. Fig. 9 shows that the service request delay remains stably equal in both cases, meaning that our algorithm does not affect non-

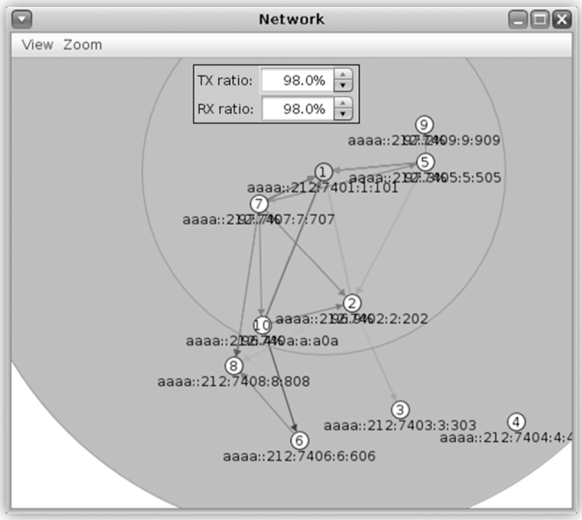



**Fig. 8.** A 6LoWPAN in Cooja with 10 nodes and 3-hop distance from the edge router (node 1). All nodes are implemented with Contiki and uIP stack. The screenshot shows the network if self-configuring with traffic exchanged between nodes.

simultaneous requests while improving the performance when multiple simultaneous requests come to a service.

```
1
2 64 bytes from aaaa::212:7403:3:303: icmp_seq=24 ttl=62 time
      =352 ms
3 64 bytes from aaaa::212:7403:3:303: icmp_seq=25 ttl=62 time
      =355 ms
4 64 bytes from aaaa::212:7403:3:303: icmp_seq=26 ttl=62 time
      =369 ms
5 64 bytes from aaaa::212:7403:3:303: icmp_seq=27 ttl=62 time
      =347 ms
6 64 bytes from aaaa::212:7403:3:303: icmp_seq=28 ttl=62 time
      =334 ms
7 64 bytes from aaaa::212:7403:3:303: icmp_seq=29 ttl=62 time
      =336 ms
8 64 bytes from aaaa::212:7403:3:303: icmp_seq=30 ttl=62 time
      =353 ms
9 64 bytes from aaaa::212:7403:3:303: icmp_seq=31 ttl=62 time
      =372 ms
10 64 bytes from aaaa::212:7403:3:303: icmp_seq=32 ttl=62 time
      =343 ms
11 64 bytes from aaaa::212:7403:3:303: icmp_seq=33 ttl=62 time
      =354 ms
12 ^C
13 --- aaaa::212:7403:3:303 ping statistics ---
14 33 packets transmitted, 26 received, 21% packet loss, time
      32060ms
15 rtt min/avg/max/mdev = 308.008/350.727/411.389/21.042 ms
16 user@instant-contiki:~$
```

Listing 8: Ping command from a regular IP node to 2-hop node 3 in 6LoWPAPN (aaaa::212:7403:3:303).

### 9.2. Scheduling: simultaneous requests handling

We carry out an experiment to test the situation when multiple requests come to the same smart object service. To recap, two requests are considered simultaneous if they happen within a small interval of time, for example as we observe with MTM-CM5000-MSP TelosB motes,[13] the value is about 100 ms. As seen from Fig. 10, the scheduling algorithm significantly improves the delay of service request in all cases with the number of requests

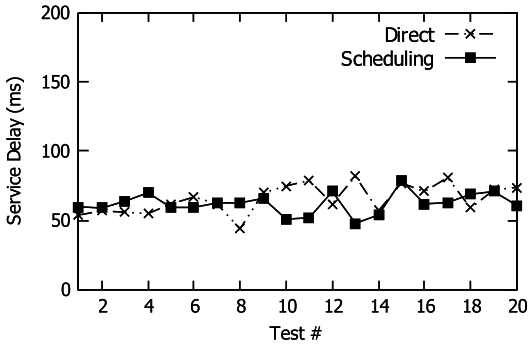



**Fig. 9.** Scheduling algorithm is transparent as it does not affect a single request. Its purpose is to improve the delay when there are multiple simultaneous requests coming to one smart object.

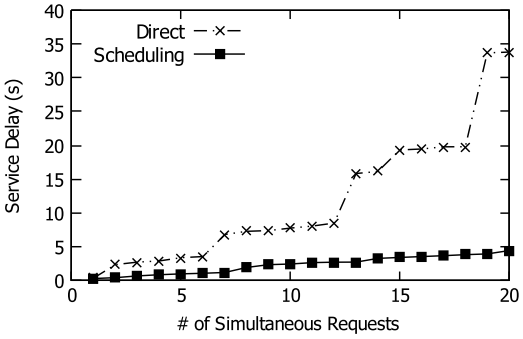



**Fig. 10.** Comparison of service delay when multiple simultaneous requests are sent to one smart object service.

ascending from 1 to 20. Especially when more simultaneous requests sent to the same service, scheduling can be considered to virtually eliminate the bottleneck in the network. Delay with scheduling algorithm also shows the stability with respect to the capacity of smart objects, that would not adversely affect user experience on application side.

### 9.3. Scheduling: energy consumption

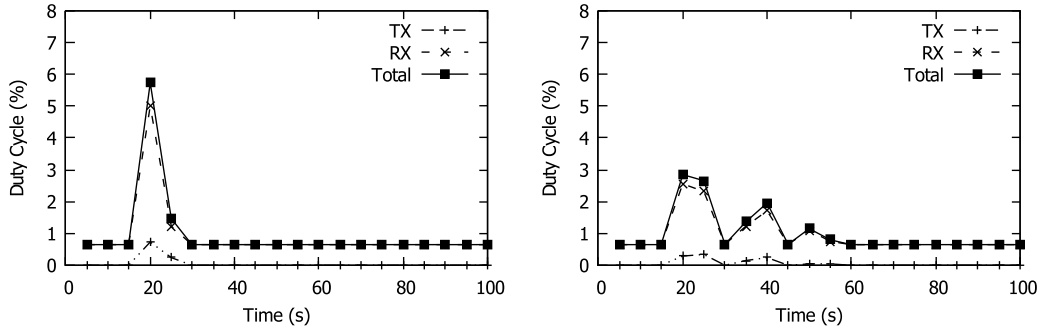
We observe the duty cycle and energy consumption of the smart object hosting the requested service over the period of 100 s when the smart object handling 20 simultaneous requests in the previous experiment. We use the power profile Energest [48] in Contiki OS to record the energy consumption in a target object. Energest uses power state tracking to estimate system power consumption and a structure called energy capsules to attribute energy consumption to activities including CPU in active mode (CPU), CPU in standby mode low-power mode (LPM), packet transmissions (TX), and receptions (RX). The duty cycle and power for each activity is calculated by following Formulas (1) and (2):

$$\frac{Energest\_TX + Energest\_RX}{Energest\_CPU + Energest\_LPM} \quad (1)$$

$$\frac{Energest\_Value \times Current \times Voltage}{RTIMER\_SECOND \times Runtime} \quad (2)$$

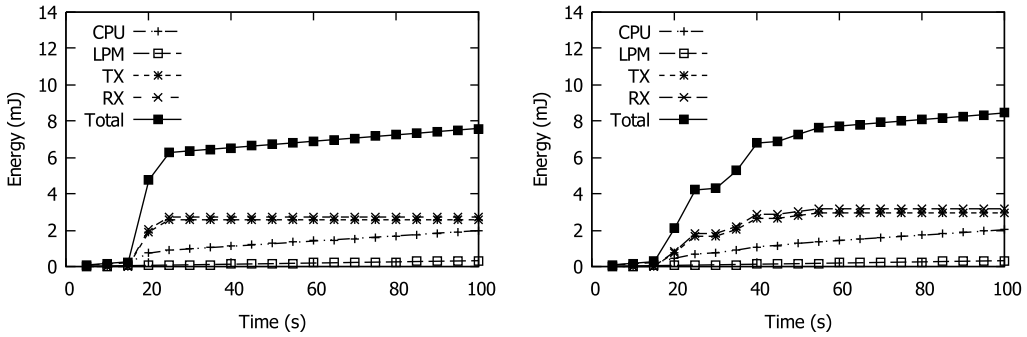
where Energest_value is the value of Energest profile tracked in each activity. Current is the current consumption, which, according to the datasheets of TI CC2420 transceiver and TI MSP430F1611 microcontroller, is 330 μA, 1.1 μA, 18.8 mA, and 17.4 mA for CPU, LPM, TX, and RX respectively. Voltage is the supply voltage, in this case, 3 V for two AA batteries. RTIMER_SECOND is the number of ticks per second for the RTIMER in Contiki OS, which is 32768. Runtime is the runtime between two Energest track points. Note

13 1 http://www.advanticsys.com/shop/mtmcm5000msp-p-14.html.




(a) Using scheduling algorithm. (b) Direct request.

**Fig. 11.** Comparison of radio duty cycle when multiple simultaneous requests are sent to one smart object.




(a) Using scheduling algorithm. (b) Direct request.

**Fig. 12.** Comparison of energy consumption when multiple simultaneous requests are sent to one smart object.

that ContikiMAC [49] radio duty cycling mechanism is enabled in smart objects. It aims to keep their radio transceivers off as much as possible to reach a low power consumption, but wake up often enough to be able to receive communication from their neighbors. Duty cycles are estimated as the percentage of Energest ticks in radio transmission (Energest_TX) and reception (Energest_RX) over the total ticks of the microcontroller in CPU and LPM modes (Energest_CPU, Energest_LPM) over a period of time (10 s).

Fig. 11 shows the duty cycling pattern in two cases. As we notice, by applying scheduling, the smart sensor keeps radio on during a shorter time about 20 s compare to 45 s when there is no scheduling. Although, radio duty cycle peaks at nearly 6% in case of using scheduling but overall energy consumption of the smart object with support of scheduling is slightly lower than without scheduling (see Fig. 12).

### 9.4. Semantic annotation

Our approach in annotating semantics to smart object service is to break down the RDF data into two parts, the core data are stored in smart object service and the additional linking data are added in service provisioning phrase. The annotation in smart object is represented in N3 format, delivered in media type request of *text/n3*. With the richness of semantic annotation for smart service data, our proposed mechanism significantly reduces the size of the messages compared to straightforward annotation and eliminate of using a third-party service for re-describing the services. We consider a typical data representation from a smart object service with the annotated information of type, source, and value. Fig. 13 shows the data sizes in difference cases: no semantic annotation, annotation in RDF format, annotation in N3 format, and the proposed method. Our proposed method ensures that the semantic annotation remains at reasonable bytes that can fit in constrained IP stack such as uIP and CoAP.

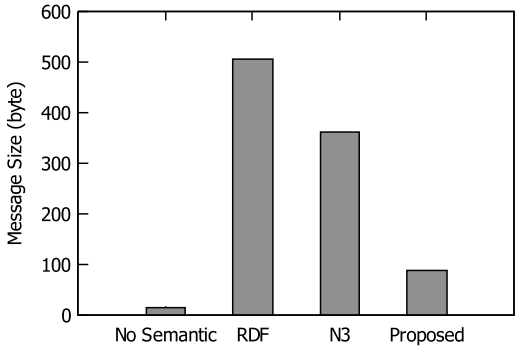



**Fig. 13.** Scheduling algorithm is transparent as it does not delay a single request. Its purpose is to improve the delay when there are multiple simultaneous requests coming to one smart object.

### 9.5. REST proxy message overhead and latency

We set up an experiment to evaluate latency and overhead in two different scenarios: the first one uses our proposed proxy (Fig. 7(a)), and the second one uses the direct DPWS communication (Fig. 7(b)). In both cases, an IoT application communicates with a DPWS smart object (a heater) to invoke its hosted service (heater functionalities). To replicate a realistic deployment of the IoT application, we deployed it on a server running Tomcat[14] that used a public Internet connection and was located about 30 km away from the smart objects. We implemented the heater with a hosted service *SmartHeater* providing eight operations, as in Table 9. We implemented a REST proxy in Java using the Jersey library on Tomcat[15] to generate

[14] http://tomcat.apache.org.
[15] http://jersey.java.net.

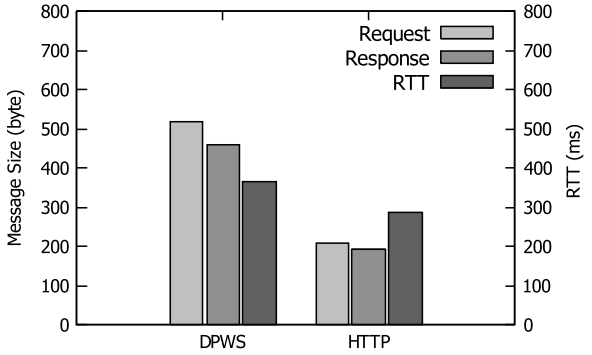



**Fig. 14.** CoAP, DPWS, and HTTP message overhead and latency.

heater Web API. The IoT application either uses the API provided by the REST proxy or directly communicates with the heater (using the WS4D JMEDS library) to carry out the DPWS heater's four functionalities: checking heater status, setting heater status, adding a new rule, and deleting a rule.

```
GET /proxy/heater HTTP/1.1
User-Agent: Java/1.7.0
Host: 157.159.103.50
Accept: text/html
Connection: keep-alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html
Transfer-Encoding: chunked
Date: Fri, 26 Jul 2013 21:46:48 GMT

[1374820483967] ON
```

Listing 9: Request and response messages for obtaining the status of the heater using the proxy Web API expose relatively simple in HTTP format.

Fig. 14 shows the message sizes of the request and response messages and the mean round-trip time (RTT) in the communication between the application and the *SmartHeater*. We use two methods: the RESTful Web API from the proxy and the original DPWS operations. The latency when using proxy is 25% lower than when using DPWS. In many pervasive IoT scenarios requiring high responsiveness, reasonable delay would improve system performance and the user experience. Message overhead improves significantly when we apply the proxy. For real deployments of applications and smart objects in original DPWS communication, nearly full-mesh connectivity (Fig. 7(b)) is unavoidable compared to the linear increments of HTTP traffic in the proxy scenario (Fig. 7(a)). Listings 9 and 10 show the details of request and response messages for an operation using the proxy and DPWS.

```
<?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/
    getstatus</wsa:Action>
    <wsa:MessageID>urn:uuid:46932240-d504-11e3-bf6a-6
    eabe38b6788
    </wsa:MessageID>    <wsa:To>http://[aaaa::212:7400:13cc
    :3693]:4567/Heater</wsa:To>
  </s12:Header>
  <s12:Body/>
</s12:Envelope>

<?xml version="1.0" encoding="UTF-8"?>
<s12:Envelope 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/
    getstatusResponse</wsa:Action>
    <wsa:RelatesTo>urn:uuid:46932240-d504-11e3-bf6a-6
    eabe38b6788</wsa:RelatesTo>
  </s12:Header>
  <s12:Body>
    <i53:reply xmlns:i53="http://telecom-sudparis.eu">ON</
    i53:reply>
  </s12:Body>
</s12:Envelope>
```

Listing 10: Request and response messages for obtaining the status of the heater using DPWS expose the complex XML-based messages in SOAP format.

## 10. Conclusion

This paper has proposed the semantic service provisioning architecture for smart objects including its related algorithms and mechanisms. The proposed architecture presents a secure, scalable, and reliable method to power IoT applications on Web. We have carried out empirical evaluation by means of several prototypes and applications and on different environments: the IoT testbed consisting of MTM-CM5000-MSP TelosB sensor nodes and the Contiki Cooja simulator. Overall, the results demonstrate that the proposed semantic service provisioning architecture for smart objects can cope with several challenges and enhance the experience for the development and integration of IoT applications on Web.

## References

[1] J.-P. Vasseur, A. Dunkels, Interconnecting Smart Objects with IP: The Next Internet, Morgan Kaufmann, 2010.
[2] D. Evans, The Internet of things how the next evolution of the Internet is changing everything, White paper, Cisco (2011).
[3] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler, Transmission of ipv6 packets over ieee 802.15. 4 networks, Tech. Rep. RFC 4944, IETF, 2007.
[4] T. Winter, P. Thubert, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, Rpl: Ipv6 routing protocol for low power and lossy networks, Tech. Rep. RFC 6550, IETF, 2012.
[5] Z. Shelby, Constrained restful environments (core) link format, Tech. Rep. RFC 6690, IETF, 2012.
[6] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Sci. Am. (2001) 29–37.
[7] M.N. Huhns, M.P. Singh, Service-oriented computing: Key concepts and principles, IEEE Internet Comput. 9 (1) (2005) 75–81.
[8] D. Miorandi, S. Sicari, F.D. Pellegrini, I. Chlamtac, Internet of things: Vision, applications and research challenges, Ad Hoc Networks 10 (7) (2012) 1497–1516.
[9] O. Mazhelis, E. Luoma, H. Warma, Defining an Internet-of-things ecosystem, in: S. Andreev, S. Balandin, Y. Koucheryavy (Eds.), Internet of Things, Smart Spaces, and Next Generation Networking, in: Lecture Notes in Computer Science, vol. 7469, Springer, Berlin, Heidelberg, 2012, pp. 1–14. http://dx.doi.org/10.1007/978-3-642-32686-8_1.
[10] R.V. Prasad, C. Sarkar, V.S. Rao, A.R. Biswas, I. Niemegeers, Opportunistic service provisioning in the future Internet using cognitive service approximation, in: 28th WWRF Meeting, Athens, Greece, 2012.
[11] B. Mandler, F. Antonelli, R. Kleinfeld, C. Pedrinaci, D. Carrera, A. Gugliotta, D. Schreckling, I. Carreras, D. Raggett, M. Pous, C. Villares, V. Trifa, Compose – a journey from the Internet of things to the Internet of services, in: 2013 27th International Conference on Advanced Information Networking and Applications Workshops, WAINA, 2013, pp. 1217–1222. http://dx.doi.org/10.1109/WAINA.2013.116.
[12] S. Lee, I. Chong, User-centric intelligence provisioning in web-of-objects based iot service, in: 2013 International Conference on ICT Convergence, ICTC, 2013, pp. 44–49. http://dx.doi.org/10.1109/ICTC.2013.6675303.
[13] S. Gagnon, K. Cakici, Integrating business services networks and the Internet of things: A new framework for mobile software as a service, in: V conference of the Italian chapter of AIS, itAIS 2008, Paris, France, 2008.
[14] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, D. Savio, Interacting with the soa-based Internet of things: Discovery, query, selection, and on-demand provisioning of web services, IEEE Trans. Serv. Comput. 3 (3) (2010) 223–235.
[15] S. Li, G. Oikonomou, T. Tryfonas, T. Chen, L.D. Xu, A distributed consensus algorithm for decision making in service-oriented Internet of things, IEEE Trans. Ind. Inf. 10 (2) (2014) 1461–1468. http://dx.doi.org/10.1109/TII.2014.2306331.
[16] R.T. Fielding, R.N. Taylor, Principled design of the modern web architecture, ACM Trans. Internet Technol. 2 (2) (2002) 115–150.

[17] Programmableweb. URL http://www.programmableweb.com/.
[18] D. Guinard, V. Trifa, T. Pham, O. Liechti, Towards physical mashups in the web of things, in: Proceedings of INSS 2009, IEEE Sixth International Conference on Networked Sensing Systems, Pittsburgh, USA, 2009.
[19] D. Guinard, V. Trifa, Towards the web of things: Web mashups for embedded devices, in: Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web, MEM 2009, in proceedings of WWW, Intl. World Wide Web Conferences, Madrid, Spain, 2009.
[20] D. Guinard, Mashing up your web-enabled home, in: Current Trends in Web Engineering, Springer, 2010, pp. 442–446.
[21] D. Guinard, M. Mueller, J. Pasquier-Rocha, Giving rfid a rest: building a web-enabled epcis, in: Internet of Things, (IOT), IEEE, 2010, pp. 1–8.
[22] D. Guinard, C. Floerkemeier, S. Sarma, Cloud computing, rest and mashups to simplify rfid application development and deployment, in: Proceedings of the Second International Workshop on Web of Things, ACM, 2011, p. 9.
[23] D. Guinard, M. Mueller, V. Trifa, Restifying real-world systems: A practical case study in rfid, in: REST: From Research to Practice, Springer, 2011, pp. 359–379.
[24] D. Zhiquan, Y. Nan, C. Bo, C. Junliang, Data mashup in the Internet of things, in: 2011 International Conference on Computer Science and Network Technology, Vol. 2, (ICCSNT), IEEE, 2011, pp. 948–952.
[25] E. Avilés-López, J.A. García-Macías, Mashing up the Internet of things: a framework for smart environments, EURASIP J. Wirel. Comm. Netw. 2012 (1) (2012) 1–11.
[26] K. Kenda, C. Fortuna, A. Moraru, D. Mladenić, B. Fortuna, M. Grobelnik, Mashups for the web of things, in: Semantic Mashups, Springer, 2013, pp. 145–169.
[27] RDF Primer, W3C recommendation, W3C (Feb. 2004).
[28] OWL 2 web ontology language document overview, W3C recommendation, W3C (Oct. 2009).
[29] M. Compton, P. Barnaghi, L. Bermudez, R. GarcíA-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, et al., The ssn ontology of the w3c semantic sensor network incubator group, Web Semant.: Sci. Serv. Agents World Wide Web 17 (2012) 25–32.
[30] A. Gangemi, Ontology design patterns for semantic web content, in: Y. Gil, E. Motta, V. Benjamins, M. Musen (Eds.), The Semantic Web - ISWC 2005, in: Lecture Notes in Computer Science, vol. 3729, Springer, Berlin, Heidelberg, 2005, pp. 262–276. http://dx.doi.org/10.1007/11574620_21.
[31] P. Barnaghi, M. Presser, K. Moessner, Publishing linked sensor data, in: CEUR Workshop Proceedings: Proceedings of the 3rd International Workshop on Semantic Sensor Networks, SSN, Vol. 668, 2010.
[32] Sparql 1.1 query language, W3c recommendation, W3C (Mar. 2013).
[33] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kroller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant, R. Richardson, Spitfire: toward a semantic web of things, IEEE Commun. Mag. 49 (11) (2011) 40–48.
[34] H. Hasemann, A. Kroller, M. Pagel, Rdf provisioning for the Internet of things, in: 2012 3rd International Conference on the Internet of Things, (IOT), IEEE, 2012, pp. 143–150.
[35] D. Bimschas, H. Hasemann, M. Hauswirth, M. Karnstedt, O. Kleine, A. Kröller, M. Leggieri, R. Mietz, A. Passant, D. Pfisterer, K. Römer, C. Truong, Semantic-service provisioning for the Internet of things, Electron. Commun. EASST 37 (2011).
[36] S. De, P. Barnaghi, M. Bauer, S. Meissner, Service modelling for the Internet of things, in: 2011 Federated Conference on Computer Science and Information Systems, FedCSIS, 2011, pp. 949–955.
[37] O. Kleine, Integrating the physical world with the Internet – a concept evaluation, in: 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications, SOCA, 2013, pp. 323–327. http://dx.doi.org/10.1109/SOCA.2013.42.
[38] Z. Shelby, K. Hartke, C. Bormann, The constrained application protocol (coap), Tech. Rep. RFC 7252, IETF, 2014.
[39] Devices Profile for Web Services Version 1.1, Tech. Rep., OASIS, 2009.
[40] S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone, L. Veltri, A scalable and self-configuring architecture for service discovery in the Internet of things, IEEE Internet Things J. 1 (5) (2014) 508–521.
[41] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, 2001.
[42] H. Chen, Z. Wu, P. Cudré-Mauroux, Semantic web meets computational intelligence: State of the art and perspectives [review article], IEEE Comput. Intell. Mag. 7 (2) (2012) 67–74.
[43] T. Berners-Lee, D. Connolly, Notation3 (N3): A readable rdf syntax, W3C team submission, W3C (Mar. 2011).
[44] The oauth 2.0 authorization framework, Tech. Rep. RFC 6749, IETF, 2012.
[45] D. Hussein, S.N. Han, G.M. Lee, N. Crespi, Social cloud-based cognitive reasoning for task-oriented recommendation, IEEE Cloud Comput. 2 (6) (2015) 10–19.
[46] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, T. Voigt, Cross-level sensor network simulation with cooja, in: Proceedings 2006 31st IEEE Conference on Local Computer Networks, 2006, pp. 641–648.
[47] S.N. Han, Q.H.C.B. Alinia, N. Crespi, Design, implementation, and evaluation of 6lowpan for home and building automation in the Internet of things, in: 2015 IEEE/ACS 12th International Conference on Computer Systems and Applications, AICCSA, 2015.
[48] A. Dunkels, F. Osterlind, N. Tsiftes, Z. He, Software-based on-line energy estimation for sensor nodes, in: Proceedings of the 4th Workshop on Embedded Networked Sensors, ACM, 2007, pp. 28–32.
[49] A. Dunkels, The contikimac radio duty cycling protocol, Swedish Institute of Computer Science report, 2011.

**Son N. Han** received M.S. degree in Computer Science from The University of Seoul in 2009 and Dip.-Ing. degree in Applied Mathematics from Hanoi University of Technology in 2006. He has just finished his Ph.D. study at the Department of Wireless Networks and Multimedia Services of Telecom SudParis, Institut Mines-Telecom. His research includes Embedded Networking, Internet of Things, Semantic Web, and Web-based Communication. Previously, he worked as a researcher for Korea Electronics and Telecommunications Research Institute (ETRI) from 2009 to 2011.

**Noel Crespi** received the Master's degrees from the University of Orsay, and the University of Canterbury, the Dipl.Ing. degree from Telecom ParisTech, and the Ph.D. and Habilitation degrees from Paris VI University. 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 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.