Semantic service provisioning for 6LoWPAN : powering internet of things applications on Web

Ngoc Son Han

DOCTORAT EN CO-ACCREDITATION
TÉLÉCOM SUDPARIS - INSTITUT MINES-TÉLÉCOM
ET L'UNIVERSITÉ PIERRE ET MARIE CURIE - PARIS 6

Spécialité : Informatique et Réseaux

École doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

HAN Ngoc Son

Semantic Service Provisioning for 6LoWPAN: Powering Internet of Things Applications on Web

Soutenue le 08/07/2015 devant le jury composé de:

Noël Crespi	Professeur HDR, Telecom SudParis	Directeur de thèse
Roch H. Glitho	Professeur, Concordia University, Canada	Rapporteur
Yacine Ghamri-Doudane	Professeur HDR, Université de La Rochelle	Rapporteur
Guy Pujolle	Professeur HDR, UPMC (Paris 6)	Examinateur
Mika Ylianttila	Professeur, University of Oulu, Finland	Examinateur
Hélia Pouyllau	Ph.D., Ingénieure de recherche, Thales R&T	Examinatrice
Emmanuel Bertin	Ph.D., Senior service architect, Orange Labs	Examinateur

Thèse numéro : 2015TELE0018

Résumé

L'Internet des objets (IoT) implique la connexion des appareils embarqués tels que les capteurs, les électroménagers, les compteurs intelligents, les appareils de surveillance de la santé, et même les lampadaires à l'Internet. Une grande variété d'appareils intelligents et en réseau sont de plus en plus à la disposition de bénéficier de nombreux domaines d'application. Pour faciliter cette connexion, la recherche et l'industrie ont mis un certain nombre d'avancées dans la technologie microélectronique, de la radio de faible puissance, et du réseautage au cours de la dernière décennie. L'objectif est de permettre aux appareils embarqués de devenir IP activé et une partie intégrante des services sur l'Internet. Ces appareils connectés sont considérés comme les objets intelligents qui sont caractérisés par des capacités de détection, de traitement, et de réseautage. Les réseaux personnels sans fil à faible consommation d'IPv6 (6LoWPANs) jouent un rôle important dans l'IoT, surtout sur la consommation d'énergie (de faible puissance), la disponibilité omniprésente (sans fil), et l'intégration d'Internet (IPv6).

La popularité des applications sur le Web, aux côtés de ses standards ouverts et de l'accessibilité à travers d'une large gamme d'appareils tels que les ordinateurs de bureau, les ordinateurs portables, les téléphones mobiles, les consoles de jeu, fait que le Web est une plateforme universelle idéale pour l'IoT à l'avenir. Par conséquent, quand de plus en plus d'objets intelligents se connectent à l'Internet, l'IoT est naturellement évolué pour la provision des services des objets intelligents sur le Web, comme des millions de services Web d'aujourd'hui. Puis vient une nouvelle opportunité pour des applications vraiment intelligentes et omniprésentes qui peuvent intégrer des objets intelligents et des services Web conventionnels en utilisant des standards Web ouverts. Nous appelons ces applications les applications IoT sur le Web.

Cette thèse propose une solution complète pour la provision de 6LoWPAN avec une annotation sémantique pour pousser le développement d'applications IoT sur le Web. Nous visons à offrir des services d'objets intelligents pour le Web et les rendre accessibles par beaucoup d'API Web qui existe en considérant des contraintes de 6LoWPAN comme les ressources limitées (ROM, RAM et CPU), la faible puissance, et la communication à faible débit.

Il y a quatre contributions: (i) La première contribution est sur l'architecture globale de la provision sémantique de services pour les applications IoT sur le Web qui comprennent trois sous-systèmes: le système de communication des services, le système de provision des services, et le système d'intégration des services. (ii) La deuxième contribution étudie un modèle d'interconnexion entre les réseaux 6LoWPAN et les réseaux IPv6 réguliers par la conception, la mise en œuvre et l'évaluation de la performance d'un 6LoWPAN qui constitué des MTM-CM5000-MSP TelosB motes pour les objets intelligents, et le Raspberry Pi pour un routeur de bordure. (iii) La troisième contribution présente en détails de l'architecture, des algorithmes et des mécanismes pour la provision des services des objets intelligents fiables, évolutifs et sécurisés en respectant des contraintes de ressources limitées; (iv) La quatrième contribution est composée de deux applications innovantes IoT sur le Web pour l'intégration des services dans lesquels nous appliquons l'architecture proposée: un système d'automatisation de la construction (SamBAS) et une plateforme Social IoT (ThingsChat).

Abstract

The Internet of Things (IoT) involves connecting embedded devices such as sensors, home appliances, smart meters, health-monitoring devices, and even street lights to the Internet. With about 10 to 15 billion microcontrollers being shipped every year, each of which can potentially be connected to the Internet, a huge variety of intelligent and networked devices are becoming available to benefit many application domains. To facilitate this connection, research and industry have come up over the past decade with a number of advances in microelectronic technology, low-power radio, and networking. The objective is for embedded devices to become IP-enabled and an integral part of the services on the Internet. These connected devices are referred to as smart objects characterized by sensing, processing, and networking capabilities. They usually configure an IPv6 low-power wireless personal area network (6LoWPAN), which plays an important role in IoT, especially on account of energy consumption (low-power), ubiquitous availability (wireless), and Internet integration (IPv6).

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. Hence, when more and more smart objects are getting connected to the Internet, it is the natural evolution of the IoT to provision smart object services to the Web, similar to today's millions of conventional Web services. There is a new opportunity of truly intelligent and ubiquitous applications that can incorporate smart objects and conventional Web services using open Web standards, denoted by *IoT applications on Web*.

This dissertation proposes a complete solution to provision 6LoWPAN services with semantic annotation that enables the development of IoT applications on Web. We aim to bring smart object services to the Web and make them accessible by plenty of existing Web APIs in consideration of 6LoWPAN constraints such as limited resources (ROM, RAM, and CPU), low-power, and low-bitrate communication links. There are four contributions: (i) The first contribution is about the overall architecture of the semantic service provisioning for IoT application on Web consisting of three subsystems: service communication, service provisioning, and service integration. (ii) The second contribution studies the internetworking model between 6LoWPAN and regular IPv6 networks by a design, implementation, and performance evaluation of a 6LoWPAN consisting of MTM-CM5000-MSP TelosB motes with TI MSP430F1611 microprocessors and CC2420 IEEE 802.15.4 radio transceivers for smart objects, and Raspberry Pi for an edge router; (iii) The third contribution presents the detailed architecture, algorithms, and mechanisms for provisioning reliable, scalable, and secure smart object services with respect to its resource-constrained requirements; (iv) The fourth contribution is in application domain for service integration in which we apply the proposed architecture on two innovative IoT applications on Web: a building automation system (SamBAS) and a Social IoT platform (ThingsChat).

Acknowledgments

I first would like to thank my supervisor, Prof. Noel Crespi, who gave me the opportunity to do this research and has been providing me a myriad of help, suggestions, and encouragement. Next, thanks go to Prof. Gyu Myoung Lee who was mentoring me for the first two (important) years of my Ph.D. and still offers me valuable advice. We have discussed a lot and I really appreciate as well as enjoy the time working with him.

I must thank my colleagues and friends in Service Architecture Lab for the joint cross-topic work we have done together that in its own way made a difference to this research. Many ideas in this research have their origins in countless discussions with Dina Adel. She inputted innovative scenarios for the proof-of-concept prototypes developed in this research; I owe her creativity. Imran Khan shared with me many ideas in wireless sensor network virtualization and helped to refine this work; Soochang Park fortified the networking foundation of the research; Xiao Han gave her in-sight analysis from a different perspective of the Internet of Things, data science. I am more grateful to everyone in the lab for the fun atmosphere they have created, for everyday coffee breaks that helped me to go through the endless days on campus. Thanks especially go to the lovely Valerie Mateus for her beautiful nature of helping me (and everyone) with administrative paperworks; she has made it easy for everything.

I had chance to work with excellent people in industry for several European projects. I warmly thank David Excoffier from Sogeti and Helia Pouyllau from Thales who helped me to approach practical designs presented in this research. I would like to thank Vladimir Vukadinovic and colleagues at Disney Research Zurich with whom I had a good time developing a very interesting prototype using IoT protocol stack. I learned a lot about the IoT technology and the beauty of the art twisted in it.

Thank you Tim Berners-Lee for inventing the World Wide Web, volunteers at Internet Engineering Task Force and many scientists that I cannot list all here for creating this connected world. Also, I am very much inspired by the work carried out by Distributed Systems Group at the ETH Zurich which significantly influenced this research.

Thank you my beloved wife Genie. Had it not been for her enormous love and support, I couldn't have done this.

Thanks to all of you!

July 2015
HAN Ngoc Son

to Genie and Trang Mi

Table of contents

Chapter 1

Introduction

Contents

1.1 Motivation

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's more, everything surrounding us such as chairs, windows curtains, light bulbs, office equipment, home appliances, and even baby dummies can be turned into

[1] http://www.koubachi.com/products/outdoor/
[2] http://stacklighting.com/
[3] http://lunasleep.com/

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 electronic appliances to these objects, 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 names such as Low-power Wireless Personal Area Network (LoWPAN, RFC 4944), Low-power and Lossy Network (LLN, RFC 6550), Constrained Environment (RFC 6690). In this dissertation, 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). The user agent can be a Web browser, a smart phone application, computer software, or even a game console firmware. 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 [4], 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.

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

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 [3].

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

1.2 Contributions

This dissertation has four contributions solving fundamental problems for a secure, scalable, and reliable semantic service provisioning to enable the development IoT applications on Web, as follows:

- The first contribution is about the overall service provisioning architecture for 6LoWPAN to enable the development of IoT applications on Web. The architecture covers the full development cycle taking into account object, network, and application levels. We first explain about the key requirements and the concept of IoT applications on Web, and then propose the architecture consisting of three subsystems delineated in different views: Reference Infrastructure, Multilayer Architecture, Functional Block Diagram, and Provisioning Workflow.

- The second contribution provides the networking foundation and studies the performance of the internetworking model between 6LoWPAN, regular IPv6 networks, and the Internet. It includes the design, implementation, and performance evaluation of a 6LoWPAN consisting of MTM-CM5000-MSP TelosB motes equipped with Texas Instruments MSP430F1611 microprocessors, CC2420 IEEE 802.15.4 radio chips, and Contiki OS 3.x as smart objects, and a Raspberry Pi as the 6LoWPAN edge router.

- The third contribution is about the detailed architecture, algorithms, and mechanisms for realizing the proposed semantic service provisioning. It solves the problems of service discovery, scheduling, semantic annotation, authorization, Web Uniform resource identifier (Web URI) mapping, and Web API presentation. We also provide an in-network implementation of the proposed architecture.

- The forth contribution is in application domain in which we apply the proposed architecture in two innovative IoT applications on Web: a Social IoT platform (ThingsChat) enabling an online social network for humans and objects and a building automation system (SamBAS) using semantic technology to offer intelligence in smart environments. These applications illustrate how the architecture can be applied in various application scenarios.

1.3 Dissertation Outline

The remainder of this dissertation is structured as follows. Chapter 2 reviews the literature of service provisioning in IoT. Chapter 3 proposes the novel service provisioning architecture consisting of three subsystems (service communication, service provisioning, and service integration) presented in the following Chapter 4, Chapter 5, and Chapter 6 respectively. Chapter 7 concludes the dissertation and discusses the future work.

Chapter 2

Literature Review

Contents

This chapter provides a literature review on the IoT protocol stack, service provisioning in IoT, and semantic annotation for smart objects. The IoT protocol stack extending TCP/IP networking model for smart object communication is playing the foundation role supporting many innovative contributions in the IoT research and development.

2.1 IoT 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).

The IoT protocol stack 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 2.1 summaries common protocols for each of the five layers which are elaborated more in the following sub sections.

Table 2.1: IoT Networking Protocol Stack

Layer	Protocols
Application	HTTP, CoAP, DPWS, XMPP, MQTT, AMQP, CoSIP
Transport	TCP, UDP, SCTP, ICMP, DTLS
Internet	IPv6, RPL
Adaptation	6LoWPAN, 6TiSCH, IPv6-over-foo
Link	IEEE 802.15.4, BLE, PLC, DECT, Low-power Wi-Fi, ITU-T G.9959

2.1.1 Link and Adaptation Layers

IPv6 resides at the center of the IoT protocol stack for the interconnection between smart objects and existing services on the Internet. IPv6 with its inconceivably-large address space is foreseen to be available on a wide variety of different Link layer technologies meeting a wide variety of communication requirements such as wired or wireless, short or long range, and high or low data throughput. Almost all types of communication links can support IP-based communication, therefore potentially operable for smart objects where the low-power requirement is the key for designing the networking models. There are several link layer technologies that are being developed for smart objects such as IEEE 802.15.4, Bluetooth Low Energy (BLE), Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy, and ITU-T G.9959. Each of these link protocol has its corresponding adaptation layer technology, for example IPv6 over Low power Wireless Personal Area Networks (RFC 4944) for IEEE 802.15.4, IPv6 over Bluetooth Low Energy (draft-ietf-6lo-btle) for BLE, Transmission of IPv6 Packets over DECT Ultra Low Energy (draft-ietf-6lo-dect-ule-01) for DECT, and IPv6 packets over ITU-T G.9959 Networks (draft-ietf-6lo-lowpanz) for ITU-T G.9959.

The IPv6 protocol has a high overhead and restrictions that make it unsuitable for low-power or constrained networks such as IEEE 802.15.4 networks. For example, considering the limited space available for the Medium Access Control (MAC) payload

in an 802.15.4 MAC Protocol Data Unit (MPDU), the use of a 40-byte IPv6 header would be too excessive. The IETF *6lowpan WG*, therefore was formed to work on the IPv6 protocol extensions required for such networks where hosts are interconnected by IEEE 802.15.4 radios. Similarly, the IETF *6lo WG* aims to connect smart objects running a number of different link layer technologies to the Internet. The results of these efforts will be a number of *IPv6-over-foo* adaptation layer specifications similar to RFC 4944. Thus far, the working group has adopted four Internet drafts that define the adaptations for IPv6 over BLE (draft-ietf-6lo-btle), DECT Ultra Low Energy (draft-ietf-6lo-dect-ule), MS/TP (master-slave/token-passing) networks (draft-ietf-6lo-6lobac), and G.9969 networks (draft-ietf-6lo-lowpanz). IETF *6tisch WG* is another working group aiming to bring IPv6 to a specific link layer technology, IEEE 802.15.4e in this case. The IEEE 802.15.4e Timeslotted Channel Hopping (TSCH) is a recent amendment to the MAC portion of the IEEE 802.15.4 standard. As a result the 802.15.4e timeslotted channel hopping MAC differs fundamentally from the Carrier Sense Multiple Access (CSMA) MAC found in standard 802.15.4. In short, TSCH allows for more controlled and deterministic network access as opposed to CSMA, while also offering increased resiliency to interference via channel hopping. TSCH MAC protocols are therefore commonly used in industrial applications.

2.1.2 Internet Layer: Routing

Due to the distinctive characteristics of 6LoWPAN (e.g., low energy availability, throughput, reliability, availability, and processing capabilities), it has specific routing requirements (RFC 5867, RFC 5826, RFC 5673, and RFC 5548) that differ from those found in traditional IP networks. The IETF *roll WG* focuses on building routing solutions for 6LoWPANs as the result of the evaluation of existing routing protocols like Open Shortest Path First (OSPF), Intermediate System to Intermediate System (IS-IS), Ad hoc On-Demand Distance Vector (AODV), and Optimized Link State Routing (OLSR) indicating that they do not satisfy all of the specific routing requirements (draft-ietf-roll-protocols-survey). The working group focuses on an IPv6 routing architectural framework while also taking into account high reliability in presence of time varying loss characteristics and connectivity with low-power operated smart objects with limited memory and CPU in large scale networks. The main realization of this working group is the design of Routing Protocol for Low-Power and Lossy Networks (RPL) which provides a mechanism to support multipoint-to-point traffic from smart objects inside 6LoWPAN towards a central control point as well as point-to-multipoint traffic from the central control point to the smart objects inside the 6LoWPAN. Within the constrained parts of the network, the RPL offers a uniform and efficient method for realizing multihop networks.

2.1.3 Transport Layer

The Transport layer is responsible for providing end-to-end reliability over IP based networks. Transmission Control Protocol (TCP) sustains the traffic on the Internet and provides reliability thanks to the control overhead introduced for each transmitted packet. Reliable transport protocols over LLNs are being studied but the amount of information for traffic control and reliability are expensive in terms of number of transmitted packets and end-to-end packet confirmation which directly maps to energy consumption. The use of User Datagram Protocol (UDP) and retransmission control mechanisms at application layer are demonstrating a good trade-off between energy cost and reliability. UDP is a datagram oriented protocol that provides a procedure for application to send messages to other applications with a minimum of protocol mechanism and overhead. In addition, the IETF *dice WG* focuses on supporting the use of Datagram Transport Layer Security (DTLS) transport-layer security in constrained environments. DTLS is the UDP adaptation of TLS (hence the name Datagram TLS) that provides end-to-end security between two applications. Stream Control Transmission Protocol (SCTP) is also used in IoT with some works focusing on Constrained Session Initiation Protocol (CoSIP) for smart objects [4].

2.1.4 Application Layer

Regardless of the specific link layer technology to deploy the IoT network, all the end-devices should make their data available to the Internet. This can be achieved by using several application layer technologies tailored for smart objects. On top of the IPv6 Internet, constrained smart objects are able to reap the benefits of a lightweight application protocols. The Constrained Application Protocol (CoAP) [5] is designed exclusively for smart objects to replace Hypertext Transfer Protocol (HTTP) and can be easily translated to HTTP for a transparent integration with the Web, while meeting the smart object requirements such as multicast support, very low overhead, and publish/subscribe model. The OASIS Devices Profile for Web Services (DPWS) [6] standard is a lightweight version of W3C Web Service [7] providing a secure and effective mechanism for describing, discovering, messaging, and eventing services for resource-constrained smart objects. The Message Queue Telemetry Transport (MQTT) [8] is an asynchronous publish/subscribe protocol that runs on top of the TCP. Publish/subscribe protocols better meet the IoT requirements than request/response since clients do not have to request updates resulting in the decrease in the network bandwidth the need for using computational resources. The Extensible Messaging and Presence Protocol (XMPP, RFC 3920) was designed for chatting and message exchanging. It was standardized by the IETF over a decade ago and is a well-proven protocol that has been used widely all over the Internet. Recently, XMPP has gained attention as a suitable

communication protocol for the IoT. The Advanced Message Queuing Protocol (AMQP) [9] is a protocol that arose from the financial industry. AMQP provides asynchronous publish/subscribe communication with messaging. It can utilize different transport protocols but it assumes an underlying reliable transport protocol such as TCP. Its main advantage is its store-and-forward feature that ensures reliability even after network disruptions. CoSIP [4] is a constrained version of the Session Initiation Protocol (SIP) to allow smart objects to instantiate communication sessions in a lightweight and standard fashion. Session instantiation can include a negotiation phase of some parameters which will be used for all subsequent communication.

2.2 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) [10], to recent solutions over IP protocol stacks. This section reviews these works on general and SOA-based models of service provisioning in IoT.

2.2.1 General Models

Miorandi et al. [11] 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 [12] 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. [13] 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. [14] introduced a perspective of Internet of Services within COMPOSE project [1]. 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.

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

Lee and Chong [15] 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 [2] environment. And the smart gateway manages the devices which are located in the local area of decentralized domain.

2.2.2 SOA-based Models

Gagnon and Cakici [16] 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 [17] 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

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

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. [18] 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.

2.2.3 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) [19]. REST has emerged as a predominant Web design model with more than ten thousand RESTful APIs (services) at the time of this article [20].

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 [21, 22, 23, 24, 25, 26] 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 [27], [28], [29] also find their ways to explore this trend over sensor nodes and embedded devices.

Besides, many IoT platforms have been developed to support the development of IoT applications tend to approach RESTful service provisioning of smart objects. As can be seen in the Table 2.2, these platforms mainly aim at integrating smart objects of different types into the Web through RESTful APIs. These platforms provide mid-point services to encapsulate underlying heterogeneous smart objects into Web interfaces that can further integrate into modern Web infrastructures such as cloud and platform-as-a-service (PaaS). These approaches expose some difficulties to scale IoT systems since each platform has to handle routing discrepancy and protocol translation.

Table 2.2: IoT Platforms

Platform	Smart Objects	Service Abstraction
BUGswarm [30]	IP networked devices	RESTful APIs
Carriots [31]	Web-enabled devices	RESTful APIs
EVRYTHNG [32]	Web-enabled devices	RESTful APIs (EVRYTHNG Engine)
GroveStreams [33]	Web-enabled devices	RESTful APIs
Nimbits [34]	Sensors	RESTful APIs
Open.Sen.se [35]	General physical objects	RESTful APIs JSON, XML
Paraimpu [36]	Web-eanbled devices	RESTful APIs
NanoService [37]	Embedded PCs, Mobile devices Embedded devices	Nano Service Platform RESTful APIs
SensorCloud [38]	MicroStrain WSNs Android phones/tablets iOS phones/tablets NI CompactRIO	SensorCloud OpenData APIs
ThingSpeak [39]	Web-connected devices Sensors	RESTful APIs
ThingWorx [40]	General connected devices (Not specified)	RESTful APIs Sockets, MQTT, AlwaysOn
Xively (Pachube) [41]	Multiple hardware Multiple platform	RESTful APIs Sockets, MQTT
Yaler [42]	Embedded systems (Arduino, BeagleBone Netduino, Raspberry Pi)	RESTful APIs SSH Service

2.3 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) [43], 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 degree, which is transitive property. The domain knowledge is often expressed using Web Ontology Language (OWL) [44], 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) [45]. They developed SSN ontology [3] 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) [46] 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 [47], 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 [48] endpoints. Pfisterer et al. [49] 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 [50] 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,

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

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.

[51] 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 [52] 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.

Klaine argued in [53] 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.

2.4 Literature Analysis

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

- 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 dissertation, we aim to overcome these problems by carrying out empirical study of 6LoWPAN performance, requirement analysis, and then propose a new semantic service provisioning to empower the IoT applications on Web.

Chapter 3

System Architecture

Contents

This chapter presents the key requirements of service provisioning for 6LoWPAN followed by the proposed system architecture. The architecture aims to provision smart object services in 6LoWPAN using open standards to power IoT applications on Web connecting smart objects with existing Web services in a scalable, secure, and reliable manner. We provide the architecture in different perspectives including reference infrastructure, data model, multilayer architecture, functional block diagram, and provisioning

workflow. The architecture is based on all IP protocol and networking principles realized by service communication subsystem in the functional block diagram. On top of that, service provisioning uses a scheduling algorithm, OAuth 2.0 authorization framework, semantic annotation, and URI mapping schemes to generate Web APIs to be used in IoT applications on Web by mechanisms explained in the service integration subsystem.

3.1 Requirements

Smart objects and 6LoWPANs are similar to any IP-based computer network, but they carry many different characteristics that need to be taken into account. This section presents the core requirements of service provisioning in 6LoWPAN.

3.1.1 Open Standards and Interoperability

The Internet as we see today is based on a plenty of open and non-proprietary standards. They are the key for a huge number of devices, services, and applications across the global to exchange data in a wide and dispersed network of networks. Some international groups are behind the development of these standards such as the Internet Engineering Task Force (IETF), the Internet Research Task Force (IRTF), and the World Wide Web Consortium (W3C). These organizations are all open, transparent, and rely on a consensus-based decision making process to develop standards. They are experts around the world working together to create freely-accessible specifications that available online at no charge, thus to foster the adoption of them.

Open standards lie at the core of the success of today's Internet and Internet-related technologies. While the Internet continues to grow to the next evolution with the arrival of new actors, smart objects, to create the new ecosystem of the Internet of everything or IoT, it is critical that new technologies continue to be developed based on open principles and processes. When it comes to system design such as service provisioning, using open standards does not only provide the interoperability, but also can backward promote the development of the Internet technologies.

Interoperability is the key characteristic of the Internet where the information being exchanged across a wide network of heterogeneous systems and devices. It is about the ability of a system to work with or use the functions of other systems. It has been one of the key requirements for Internet applications, which are based on the communication of different hardware and software infrastructure. In IoT, the heterogeneity of the systems and devices become even larger where smart objects, which are limited in resources, cannot operate in a full-fledged manner with other networks. Therefore, for new IoT systems to be an integral part of the Internet, they must be able to exchange data and subsequently present that data such that it can be consumed by existing systems on the

Internet. Interoperability requires standards on several levels. It is necessary to have uniform mechanisms in what is being exchanged (data elements), how to structure data for exchange (record schemas and record syntaxes), and how to actually exchange it (protocol transactions and messages and profiles).

Service provisioning therefore has to provide data format, protocol messages, and data schema that can be used in other systems on the Web using open Web standards Web design principles such as Representational State Transfer (REST) [19] and Semantic Web [3] are the key to enable the interoperability for IoT applications on Web.

3.1.2 Low Energy Consumption

Energy consumption has been at the center of any discussion for the sustainable development these days, especially when the digital revolution has happened recently. Machines are driven by electronic parts, and electronics need power. This fact brings the energy issue even up to a more critical level when the IoT is happening very fast with billions of personal electronic are predicted to be available in coming years. Today, the most common power source is a battery, but there are also several other possibilities such as solar cells, piezoelectricity, radio-transmitted energy, and other forms of power scavenging. Power scavenging is a technique in which devices harvest power from the physical environment. Solar cells represent the most common form of power scavenging. They harvest their power from the ambient and direct light hitting the smart object. While static energy sources are limited, for those powered by power scavenging, energy is not always assured and difficult to be stored for extended periods of time.

For this reason, both the hardware and the software of the smart object must be designed to meet stringent power requirements. To achieve this, low-power hardware such as microprocessors and radio chips have been developed. Low-power radio hardware, which is the most critical part of consuming energy in smart objects to maintain the connectivity, however it is still not sufficient. Existing low-power radio transceivers (though optimized) use too much power to provide long lifetime on batteries. For example, the CC2420 radio transceiver, used in the MTM-CM5000-MSP TelosB and Z1 motes, use approximately 60 milliwatt of power when listening for radio traffic and a similar figure for data transmission. By that power, radio operation depletes 2 AA batteries equipped for these motes in a matter of days. Therefore in addition to hardware, software design of protocols and system architecture play an important role on improving the energy consumption of smart objects. For example, radio duty cycling mechanisms (e.g., ContikiMAC) aim to deal with this problem by keeping the radio turned off as much as possible while providing enough rendezvous points for two smart objects to communicate.

3.1.3 Reliability

The trade-off of low-power design for smart objects and 6LoWPAN comes with the less reliable communication link due to the use of low-power and low bitrate protocols. There are fundamentally two factors in 6LowPAN leading to the low reliability of the network: constrained processing power in smart objects and lossy and low bitrate in the communication. To fulfill the energy requirement, smart objects are equipped with limited processing power, memory, and energy; they are also in many cases battery-operated or energy scavenging that leads to the data processing capacity is subject to a limit. Besides, smart objects are interconnected by lossy links, typically supporting only low data rates that are usually unstable with relatively low packet delivery rates. This may result in the loss of packets. There have been efforts to handle the issue such as new routing protocol RPL considering the loss nature of the link. Even when mechanisms for dealing with high packet loss rate are applied, long delay between service requests and responses is anticipated due to the limited resource on the smart objects side. In many applications, the delay may be not tolerable for practical uses. Therefore, to effectively integrate 6LoWPAN into the Internet environment, IoT architects have to take into account this requirement as one the core values.

3.1.4 Security and Privacy

Security in IoT is becoming a critical issue with millions of devices getting connected to the Internet. The IoT means that everyday objects going online, being connected, and talking to each other without human being's involvement to, for example, carry out many of our tedious routines. For what we are witnessing today, the first wave of the IoT is already around with several tracking devices on the market such as activity trackers that record your movements and geographical position, baby monitors that measure breathing and skin temperature, and smart Wi-Fi light bulbs that can be programmed via a smartphone. The question is how these personal data are being handled and by whom. The autonomy of devices comes with more concern about our privacy and when more objects expose themselves to the Internet, more security issues come to our software systems where these objects are also connected. The critical problem is these smart objects are getting *smarter* to intervene in users' privacy. It is possible that once you bought a television, turn it on and while it serves you with new smart services taking into account of your preferences and use contents on the Internet, it could be listening to your private conversations and sharing them over the Internet.

Another issue with security and privacy for smart objects is that they have the owner-object relationship to their owners and the owners have several preferences for setting up their devices. Smart objects, different to other resources on the Web, have limited resources and a special degree of privacy because they belong to individuals

with their privacy to protect and have limited resources. Therefore, the consumption of smart object services should be well-managed in a secure and stable manner. Therefore, provisioning smart objects services has to come with appropriate authorization to use.

3.1.5 Scalability

Today's Internet is a giant global network of networks based on IP-based protocols thanks to its inherent scalability. No other networking technology in the history has ever been deployed and tested at such an immense scale and with such a large number of devices. As smart objects will connect to the Internet in even a larger number, scalability is a primary concern and should be staying at the core of designing new system. To assure the continuous development of the Internet, smart objects and 6LoWPANs as new actors are required to work in the similar scalable manner. Service provisioning must be efficient and practical when applied to large-scale situations either in the scale of 6LoWPANs or of IoT applications on Web in addition to the assurance of system performance when the network expands. Also, at network deployment level, the installation of smart objects network in any facility (home, office, public space, etc.) is required to be fast and integral part with the existing communication infrastructure.

This requirement leads to a direct suggestion of using IP-based protocols for IP has proven itself a stable and highly scalable communication technology that supports both a wide range of applications, devices, and underlying communication technologies. End-to-end IP architecture is considered suitable and efficient for scalable networks of large numbers of communicating devices such as the IoT. The next generation Internet protocol, IPv6, expands the address space from 32 bits to 128 bits. Such a large address space has been estimated enough for billions of smart objects going online in the near future. The adoption of IP standards can be carried out at low level of smart objects themselves to avoid using protocol translation gateways, which prevent the scalable deployment of 6LoWPAN and IoT systems.

3.2 IoT Application 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 [1] 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 through 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.

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

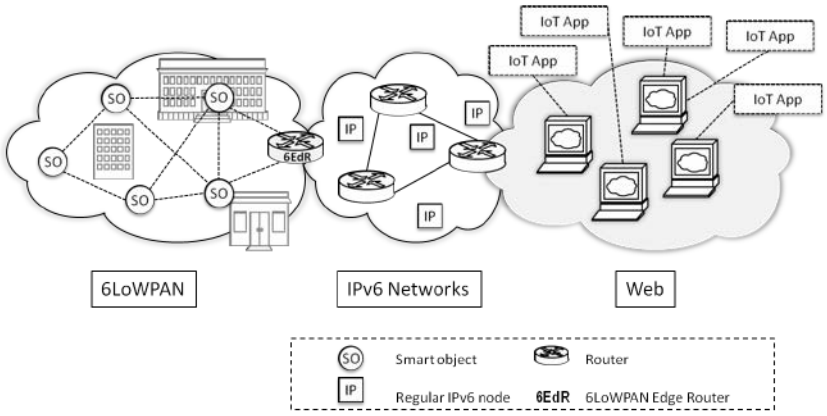


Figure 3.1: The reference infrastructure.

3.3 System Architecture

This section presents details on our proposed service provisioning architecture including Reference Infrastructure, Data Model, Multilayer Architecture, and Functional Block Diagram. We also provide a brief introduction on each component of the architecture, which will be elaborated in the following chapters.

3.3.1 Reference Infrastructure

Figure 3.1 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 and stacks. These smart objects configure a 6LoWPAN based on low-power physical layer protocols such as IEEE 802.15.4, 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.

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 Bluetooth Low Energy (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

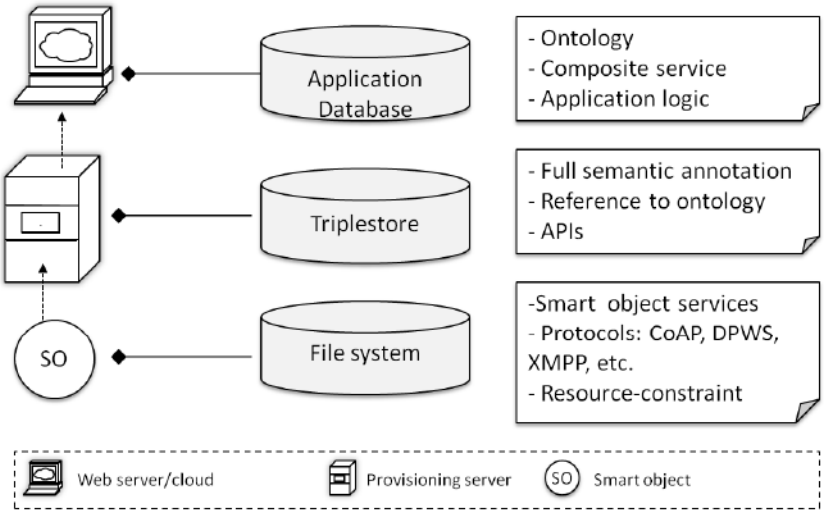


Figure 3.2: Data model in the hierarchical scheme.

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.

3.3.2 Data Model

Every smart object in the 6LoWPAN is provisioned with data in which information about low level resources (sensors, actuators, memory, energy, etc.) and high level resources (semantic services and contextual services) are stored in different locations. These data create data model for our provisioning architecture.

Data are organized according to the reference architecture to store different information about the smart objects, 6LoWPAN, and services. There are three kinds of data: object data, provisioning data, and application data. Object data, which store information about primitive services directly provided from smart objects, cover the physical resource status and contextual data such as temperature and humidity. They can be implemented in the smart objects using resource-constrained protocols such as CoAP, DPWS, and XMPP and accessed via protocol messages. Provisioning data store information about services provided by 6LoWPAN nodes enriched by semantic annotation with reference to the domain ontology. Application can get access to this data by calling its Web API. Application data store high-level information about single services and composite services provided by 6LoWPAN that meet the requirement of each application. Besides, application logics are stored in this repository. By this data model, IoT applications on Web collect and store semantic data, consulting to online ontology to handle many context-aware scenario.

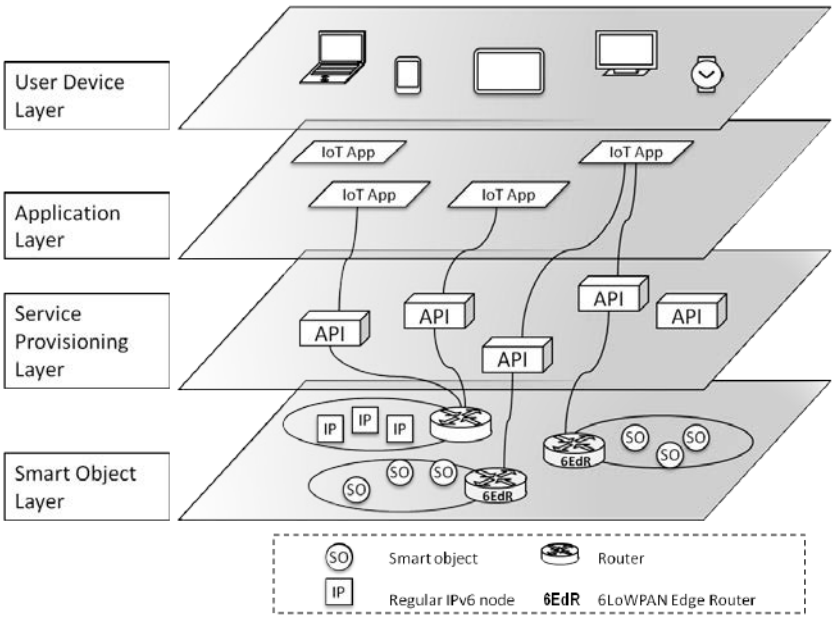


Figure 3.3: Multiayer architecture.

The organization schema of our data model is shown in Figure 3.2. The repository on the top of the hierarchy, in the form of application database, is located in the Web server or cloud which combines smart objects services with conventional Web services to create novel composite services. Ontology is separately developed by many providers who have expertise on each domain. The provisioning data repository is a triplestore located in a provisioning server, which can be located in local network or on the Web. Object data repository is located in smart object with its file system or dynamically in program logic.

3.3.3 Multilayer Architecture

Figure 3.3 shows a high-level view of the architecture in the form of multilayer architecture. Therein, the Smart Object Layer is the lowest layer where services are implemented on physical entities including smart objects and conventional computers. This layer consists of 6LoWPANs and regular IP networks. Each 6LoWPAN connects to an edge router 6EdR which carries out the routing function between the 6LoWPAN to regular IP networks assuring the consistence in routing, Quality of Service, transport, and network recovery for the entire system. Regular IP networks also reside in this layer providing services to the Web in a similar way to 6LoWPANs. The second layer is the Service Provisioning Layer representing the interface between smart objects and applications on the Internet. There will be a Web API associated to each 6LoWPAN, which can be used in multiple applications and mashed up with other Web APIs of smart objects and conventional Web APIs. The third layer is Application Layer of IoT applications on Web. This layer exhibits the mechanism of how an application use

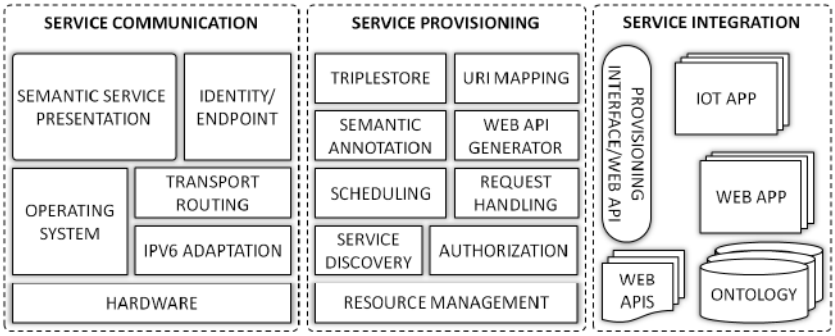


Figure 3.4: Functional block diagram.

6LoWPAN Web APIs to interact with smart objects, and how these applications can carry out semantic reasoning by retrieving semantic annotation from smart object and querying from available ontology in the domain.

3.3.4 Functional Block Diagram

Figure 3.4 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 stacks 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.

3.3.4.1 Service Communication

Starting with Service Communication system, the functional blocks suggest that smart object is implemented with low-power operating systems supporting constrained IP stacks such as Contiki OS [2], TinyOS [3], and RIOT OS [4]. Common IoT protocols, e.g., 6LoWPAN Adaptation, IPv6, RPL, TCP, and UDP can be used to provide the networking functionality of smart objects. On top of that, services are implemented using an application layer protocol such as CoAP, DPWS, and XMPP. These services can

[2]http://www.contiki-os.org/
[3]http://www.tinyos.net/
[4]http://www.riot-os.org/

communicate with applications on the Web and interact with sensor/actuator hardware in smart objects to, for example, collect contextual information and activate a routine task of smart objects. Smart objects are also enriched by simple semantic annotation based on domain ontology previously published or available on the Web. Each service is identified by an identity, which plays a role of an endpoint address for service communication. Also this identity is reserved for security purposes. This domain enables the consistent communication between smart object networks and normal IP networks also facilitate smart object functionalities using open application standards.

3.3.4.2 Service Provisioning

The Service Provisioning subsystem, based on the IP infrastructure in 6LoWPAN, encapsulates constrained protocols and interfaces into useful Web APIs that can be used in multiple IoT applications on the Web. One of the common characteristics of smart object services is the ability of dynamic discovery, mostly using multicasting. Service Discovery functional block is deployed to either relay the multicast messages to the applications or forward the messages in and out the local network. URIs discovered in discovery process are sent to URI Mapping to apply the mapping rules. Developers also can maintain the service cache by providing discovered data to a cache module. Provisioning Interface in the form of Web API contains descriptions of smart object services that can be consumed by IoT applications on the Web. It also provides the ability for applications to mashup smart object services with other Web services.

Each API request consisting of a HTTP verb (GET, POST, PUT, and DELETE) on an URI (retrieved from URI Mapping) and an authentication token comes to the Request Handling for preprocessing. Request Handling parses the token to get the permission of using the services and first looks into the cache for available resource to bypass the discovery step. If the resource is not found in the cache, Request Handling with send a discovery request to the target smart object to check for its availability and updated information. Thereafter, the request is put into the Scheduling queue waiting for interacting with the target smart object. This Scheduling block is to ensure that the constrained environment can reply to a maximum number of requests. When the request is process, data go through a dispatcher for communicating with the target smart object.

3.3.4.3 Service Integration

In Service Integration subsystem, each IoT application on Web can use smart object Web APIs in the same way as other Web APIs and carry out inference from semantic data. Provisioning Interface provided by the Service Provisioning subsystem to be used by IoT applications on Web in the same way that conventional Web applications use

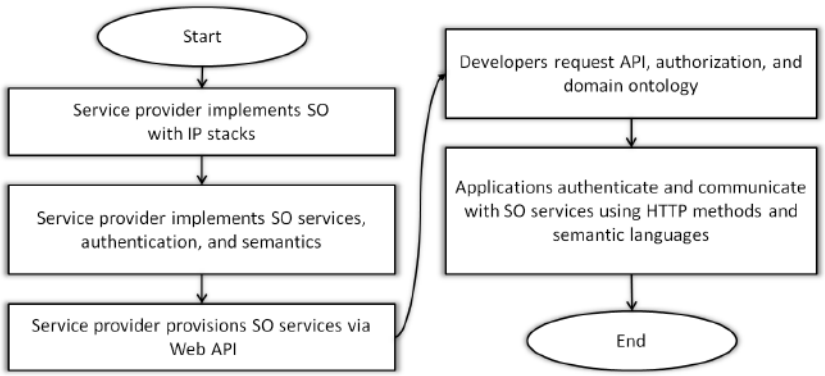


Figure 3.5: Provisioning workflow.

Web APIs such as Google API, Twitter API, and Yahoo API. Semantic data received from smart object services can be associated with the domain ontology to carry out inference or intelligence for the application. All of these functional blocks can be used in standard programming language and tools provided by several development platforms.

3.3.5 Provisioning Workflow

Figure 3.5 illustrates an overall workflow of provisioning smart object services including three subsystem of the architecture. At the first step, service providers such as building owners and third-party service companies provide smart objects with pre-manufactured operating systems supporting IP stacks for developers or third-party IoT technology firms to implement IP stacks. Next, service providers describe and implement smart object functionality in the form of services, authentication schemes, and semantic annotations. The final step involves service provider to provision smart object services to the applications on the Web. Developers can start to use a set of smart object by acquiring their API, authentication tokens to get access to the API, and domain ontology to use reasoning with semantic data retrieved from smart object, depicted in the next step. Thereafter, applications can communicate with smart objects via HTTP methods over Web API, mashup with other Web APIs and doing reasoning based on semantic tools.

3.4 Summary

This chapter has given the overall architecture for semantic service provisioning of 6LoWPAN. In the next chapters, we will introduce the details of each subsystem to realize this architecture. Starting with Chapter 4, we present the design, implementation, and evaluation of 6LoWPAN with which, the results act as the foundation to build up the provisioning architecture described in the next Chapter 5. Chapter 6 describes two innovative applications applying the proposed architecture.

Chapter 4

Design and Performance Study of 6LoWPAN

Contents

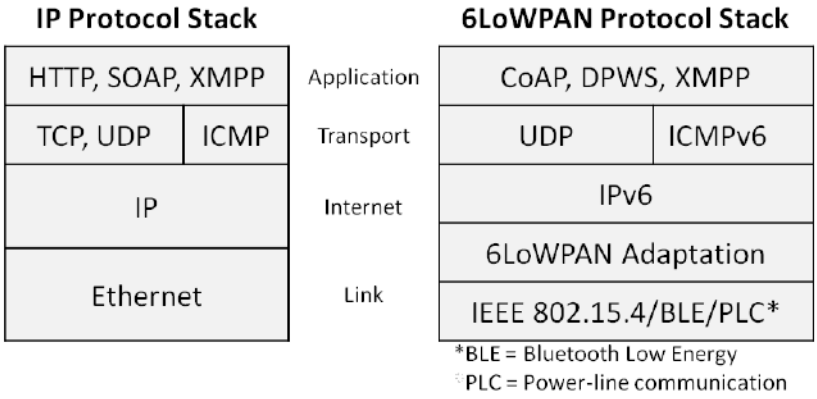


Figure 4.1: IP and 6LoWPAN protocol stack in reference to layers of the TCP/IP networking model.

Since IPv6-enabled low-power wireless personal area networks of smart objects (6LoWPANs) play an important part in the IoT, especially on account of the Internet integration (IPv6), energy consumption (low-power), and ubiquitous availability (wireless), this chapter presents the design and a study on 6LoWPANs providing the networking foundation for the proposed provisioning architecture. The design realizes Smart Object layer in the multilayer architecture and Service Communication subsystem in the block diagram presented in Chapter 3. The performance study contains a comprehensive analysis on several internetworking characteristics between 6LoWPANs and regular IPv6 networks including energy consumption on nodes, network performance, and service communication.

Figure 4.1 shows a comparison between typical networking stacks of regular IP networks and 6LoWPAN following 4-layer TCP/IP model (RFC 1122): Link, Internet, Transport, and Applications. The key difference lies at 6LoWPAN adaptation layer, which adds a specific layer and IPv6 header compression before forwarding to regular IPv6 destination. This technology gives the efficient extension of IPv6 into the 6LoWPAN domain, thus enabling end-to-end IP networking features for a wide range of IoT applications.

While these technologies are gaining stable status, how they affect the design of many potential intelligent and ubiquitous IoT applications is still rather an island for new discoveries. In this chapter, we present our design, implementation, and performance evaluation of 6LoWPAN based on open IoT standards provided by IETF in CoRE, ROLL, and 6LoWPAN working groups. We implement the design on a set MTM-CM5000-MSP TelosB motes (CM5000) [1] for smart objects and a Raspberry Pi (RPi) [2] for an edge router, some laptop computers for hosts in regular IPv6 network. All

[1] http://www.advanticsys.com/shop/mtmcm5000msp-p- 14.html
[2] https://www.raspberrypi.org/

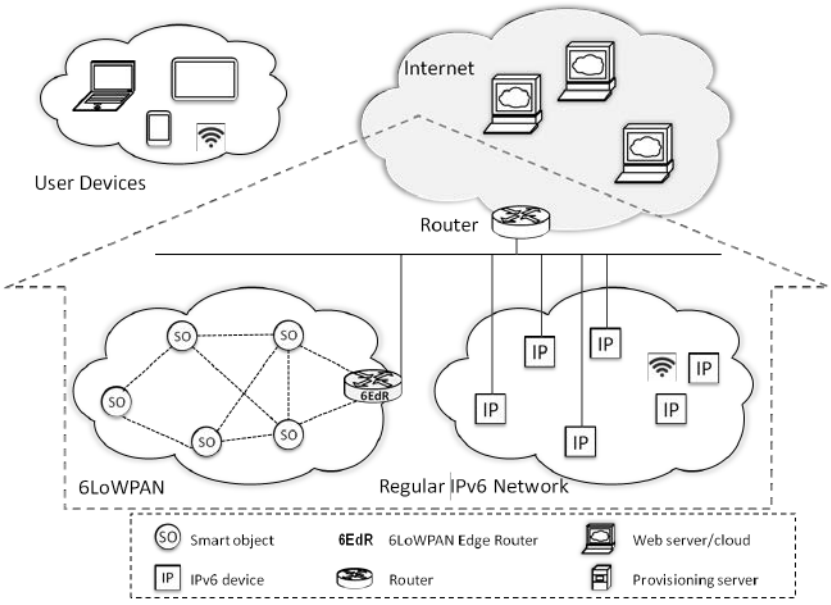


Figure 4.2: 6LoWPAN Internetworking Architecture.

are connected to the backbone network of the building. The performance evaluation exhibits how these new networking technologies operate in real-life deployments and where to adapt them to different scenarios.

4.1 6LoWPAN Design

4.1.1 Internetworking Architecture

The 6LoWPAN internetworking architecture is made up of 6LoWPANs, regular IP networks (IPv4, IPv6), and routers. The overall architecture is presented in Figure 4.2 in which 6LoWPAN is an IPv6 subnet of smart objects sharing a common IPv6 address prefix (the first 64 bits of an IPv6 address). These smart objects can play the role of a host or a router to create a mesh network. 6LoWPAN is connected to regular IP networks through an edge router. The edge router forwards data packets between the 6LoWPAN and backbone IPv6, while handling IPv6 compression and neighbor discovery.

Communication between 6LoWPAN smart objects and IP hosts in other networks happens in an end-to-end manner, just like between any regular IP nodes. Each 6LoWPAN smart object is identified by a unique IPv6 address, and is capable of sending and receiving IPv6 packets. In Figure 4.2, the 6LoWPAN smart objects can communicate with either of the regular IPv6 hosts, servers on the Internet, or personal users' devices. Smart objects support ICMPv6 traffic (*ping*), and use the UDP as a transport. Since the payload and processing capabilities of smart objects are extremely limited to save energy, application protocols are designed to use a simple binary format over UDP such as DPWS and CoAP.

4.1.2 6LoWPAN Edge Router

In order to connect 6LoWPAN networks to other IP networks, we use 6LoWPAN Edge Routers (6EdRs). These edge routers are located at the border of the 6LoWPAN performing two essential tasks: adaptation between 6LoWPAN and regular IPv6 networks and routing the IP traffic in and out of the 6LoWPAN. This transformation is transparent, efficient and stateless in both directions.

Figure 4.3 presents our 6EdR architecture consisting of several layers: Network Interfaces (regular IPv6, e.g., Ethernet and low-power, e.g., IEEE 802.15.4), 6LoWPAN Adaptation, Neighbor Discovery, IPv6, IPv6 Routing, Network Management, and Proxy. 6LoWPAN Adaptation Layer is for decompressing frames received from the low-power link (RFC 4944) using known information about the network and compressing regular IPv6 frames from the regular network interface. This step could be performed in the wireless interface or the edge router driver. Neighbor Discovery is responsible for several configuration tasks such as auto-configuration of nodes, discovery of other nodes on the link, and maintaining reachability information about the paths to other active neighbor nodes. It includes both IPv6 Neighbor Discovery Protocol (NDP, RFC 4861) and 6LoWPAN Neighbor Discovery (6LoWPAN-ND, RFC 6775). The interface or driver should take care of configuring the stack or adapting relevant neighbor discovery messages between 6LoWPAN-ND and NDP. IPv6 Routing maintains route entries between its interfaces belonging to two different routing domains where most traffic flows are coming from the Internet towards one or more 6LoWPAN nodes, or from LoWPAN nodes towards the Internet. Network Management is one of the core features of any network deployment for managing smart objects on 6LoWPAN. It may use Simple Network Management Protocol (SNMP, RFC 6353). Proxy further adds application layer translation models for transferring request in and out 6LoWPAN.

4.2 6LoWPAN Implementation

4.2.1 Hardware

We use CM5000 motes equipped with three LEDs, a temperature sensor, a humidity sensor, two light sensors, and button sensor as generic smart objects to set up a 6LoWPAN. With several sensors and LEDs, CM5000 can represent many home and building appliances such as a light sensor, a light bulb, a thermostat, a switch, and even a motion sensor. We use a RPi for the edge router with the built-in Ethernet as an IPv6 interface and a CM5000 mote connecting to RPi USB port as a 6LoWPAN interface (IEEE 802.15.4). Some laptop computers are used to deploy a regular IPv6 networks with Ethernet interfaces connecting to the same router with the edge router.

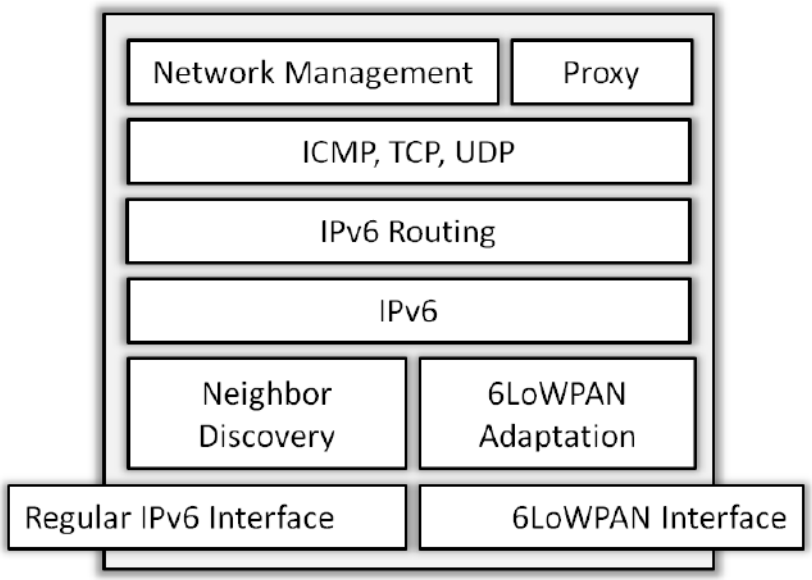


Figure 4.3: 6EdR Design.

4.2.2 Software

We use the latest update of Contiki OS 3.x [3] (by the time of writing this manuscript) with μIP protocol stack to implement IPv6 networking functionalities for the 6LoWPAN nodes. TI MSP430 toolchain on Ubuntu is used to compile the programs for CM5000 motes. These programs all configure the smart objects to use radio channel 26, Contiki-MAC [54] for duty cycling mechanism, *router mode* to create a mesh network, and MAC addresses to auto generate their IPv6 addresses (e.g., MAC 00:12:74:00:13: cb:2d:a6 for IPv6 aaaa::212:7400:13cb:2da6 address). On top of that, several modules are developed to provide different functionalities to the smart objects such as energy profiling (using Energest power profile [55]), UDP server, CoAP server (Erbium library [56]), and DPWS server (uDPWS library [4]). Figure 4.4 shows the real hardware configuration of the edge router with two interfaces: IEEE 802.15.4 and Ethernet.

Raspbian OS, a Debian-based OS is provided as the platform for the edge router. Its IEEE 802.15.4 interface communicates with the edge router via USB port using Serial Line Internet Protocol (SLIP). We create a network TUNnel (TUN) virtual interface to simulate a network device operating on Internet layer. This TUN interface works with SIP to apply 6LoWPAN adaptation. We also configure the Raspbian OS as an IPv6 router between two network interfaces Ethernet and TUN. By that, traffic from 6LoWPAN comes to the edge router with IEEE 802.15.4 frames adding compression and 6LoWPAN adaptation in the software, passed to TUN interface and then routed to Ethernet interface to reach regular IPv6 network. For example, when a 6LoWPAN

[3] http://www.contiki-os.org/, version 2015/02/16
[4] http://www.ws4d.org/

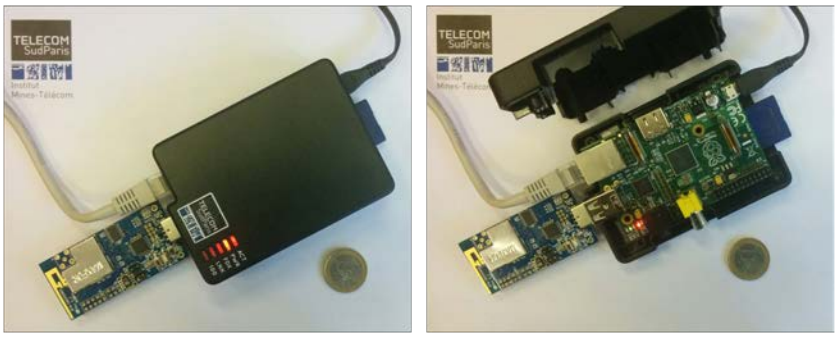


Figure 4.4: 6EdR hardware: Raspberry Pi with an Ethernet interface and a CM5000 mote as an IEEE 802.15.4 interface.

packet is forwarded to the IPv6 network, edge router removes its 6LoWPAN adaptation layer, uncompresses its header, and ensures that global IPv6 source address is used for the outgoing packets. For incoming packets to the 6LoWPAN, edge router adds 6LoWPAN specific adaptation layer and possibly 6LoWPAN IPv6 header compression mechanism and then forwards them to the 6LoWPAN.

4.3 Performance Evaluation

We carry out the experiments on the communication between a 6LoWPAN and a regular IPv6 network to observe the quality of the link in several aspects under a real-life deployment. The deployment takes place in an L-shape office building floor. We deploy the 6LoWPAN with a 6EdR and a number of nodes and a simple IPv6 network with one host. The 6EdR is deployed in one office along with a laptop computer (as a regular IPv6 host), both connected to the same local network via a home and building router. Three nodes are put in 1-hop, 2-hop, 3-hop positions to the edge router as shown in Figure 4.5. There estimates about 10 Wi-Fi devices operating at the time of the experiment. A screen capture from a Wi-Fi analyzer indicates which channels wireless networks are on and how strong they are. We notice that only *eduroam* and *eduspot* are busy on channel 1, other in mild status which would not affect much on the experiment nodes operating on IEEE 802.15.4 radio channel 26. Network configuration is as follows:

Building Router Linksys E1200, IPv6-enabled

6LoWPAN aaaa::/64

6EdR (Raspberry Pi and CM5000)

- Ethernet: fde5:d6db:6ff6::1 (connected to E1200)
- IEEE 802.15.4: aaaa::212:7400:13cb:44
- Virtual TUN: aaaa::1

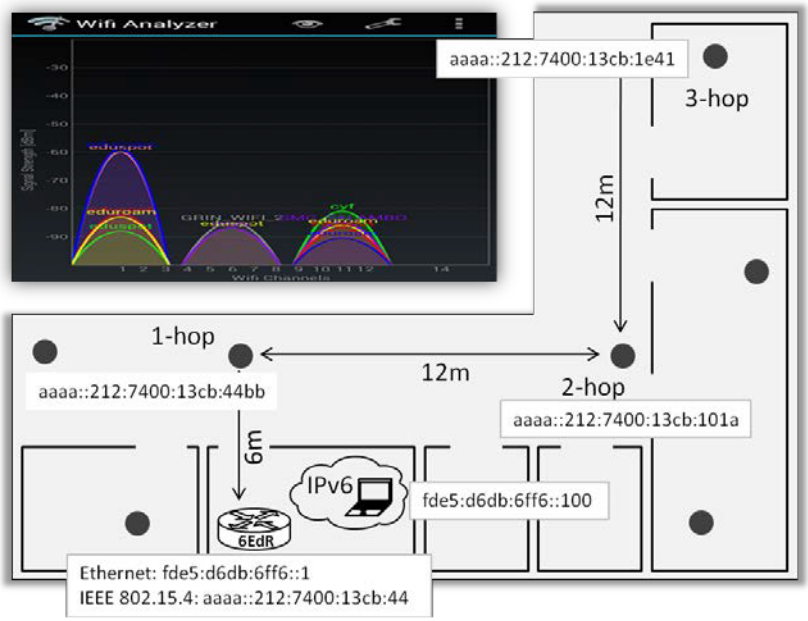


Figure 4.5: 6LoWPAN home network setting.

Smart objects

- 1-hop node: aaaa::212:7400:13cb:44bb
- 2-hop node: aaaa::212:7400:13cb:101a
- 3-hop node: aaaa::212:7400:13cb:1e41

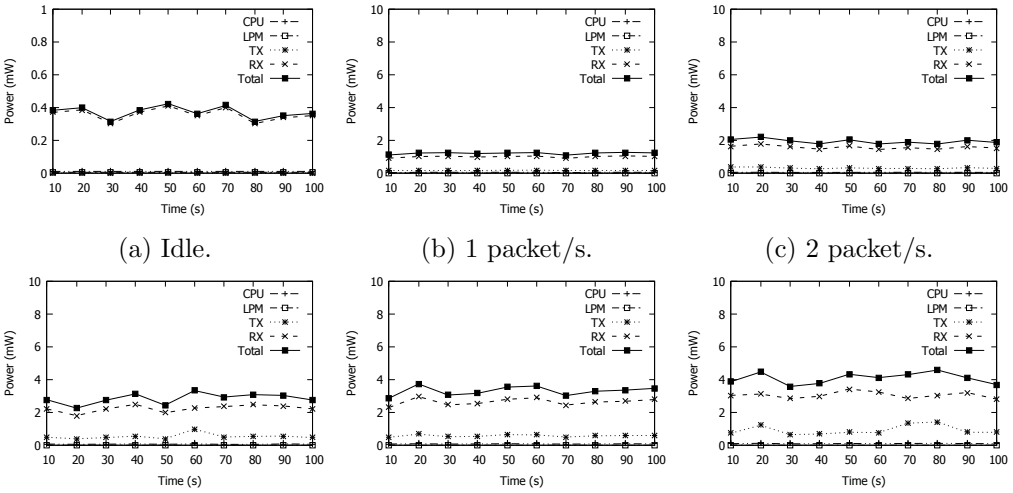
IPv6 host

- Ethernet: fde5:d6db:6ff6::100 (connected to E1200)

4.3.1 Energy Consumption

The first experiment is about energy consumption. We examine how each node in the network consumes energy in two modes (host and router) with five different data rates (1 to 5 packet/s). A smart object is considered to be in *router mode* if it forwards the traffic between other nodes. The *host mode* is when the smart object only communicates with other objects or the edge router without doing any traffic forwarding. We use the power profile Energest [55] 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 power for each activity is calculated by following Formula 4.1:

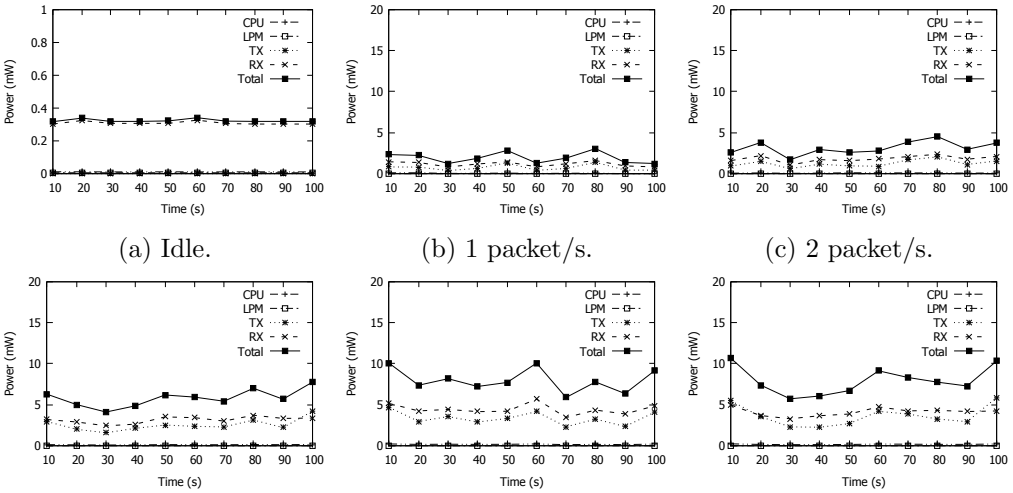
$$\frac{Energest\_Value \times Current \times Voltage}{RTIMER\_SECOND \times Runtime} \tag{4.1}$$



(a) Idle. (b) 1 packet/s. (c) 2 packet/s.

(d) 3 packet/s. (e) 4 packet/s. (f) 5 packet/s.

Figure 4.6: Energy consumption in *host mode*.

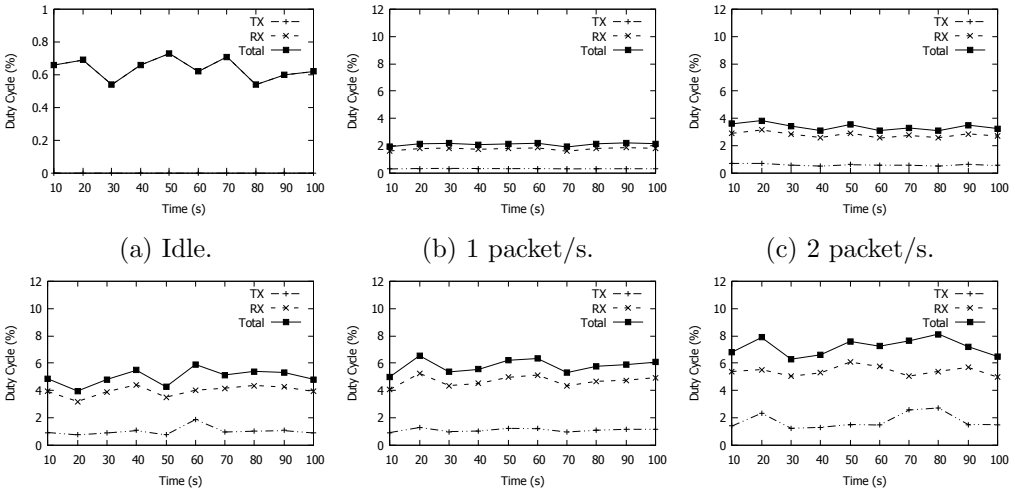


(a) Idle. (b) 1 packet/s. (c) 2 packet/s.

(d) 3 packet/s. (e) 4 packet/s. (f) 5 packet/s.

Figure 4.7: Energy consumption in *router mode*

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. The results are shown in Figure 4.6 for smart objects in *host mode* and in Figure 4.7 for objects in *router mode*. As can be seen from the graph, the power remains low at 0.4 mW when smart objects are idle, and increases proportionally to the data rate in both *host* and *router mode*.



(a) Idle. (b) 1 packet/s. (c) 2 packet/s.

(d) 3 packet/s. (e) 4 packet/s. (f) 5 packet/s.

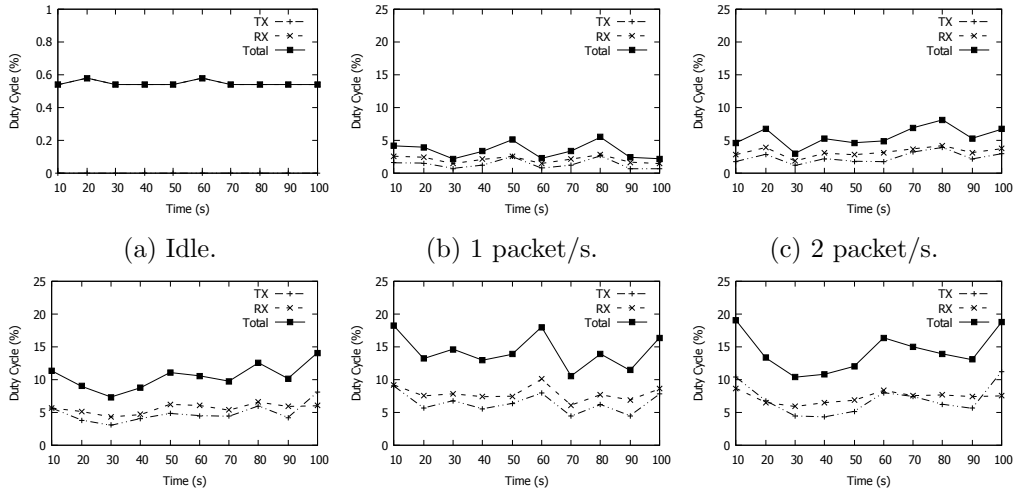
Figure 4.8: Duty cycle in *host mode*

4.3.2 Duty Cycle

The second experiment is to explore radio duty cycle in each 6LoWPAN node. Similar to recording energy consumption, we also use Energest power profile to estimate the duty cycle of each smart object. ContikiMAC 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 seconds) by following Formular 4.2:

$$\frac{Energest\_TX + Energest\_RX}{Energest\_CPU + Energest\_LPM} \tag{4.2}$$

Figure 4.8 and Figure 4.9 depict duty cycles of a smart object in *host mode* and *router mode* with 5 different data rates of 1, 2, 3, 4, and 5 packet/s. In general, duty cycle of a smart object in *host mode* is lower and more stable than in *router mode*. Forwarding data packets apparently requires radio to be more waken-up then only receiving data. When smart objects are idle (or in sleep mode, but still wake up frequently enough to maintain the connectivity), the duty cycle remains fairly low about 0.3 percent in the *host mode* and 0.6 percent in the *router mode*. Duty cycle increases constantly over the change of data rate from 1 to 5 packet/s.



(a) Idle. (b) 1 packet/s. (c) 2 packet/s.

(d) 3 packet/s. (e) 4 packet/s. (f) 5 packet/s.

Figure 4.9: Duty cycle in *router mode*

4.3.3 Network Performance

In the third experiment, we send 100 Internet Control Message Protocol version 6 (ICMPv6, RFC4443) packets from a regular IPv6 host to different smart objects in the 6LoWPAN and wait for the echo responses to record some network parameters such as packet loss, round-trip time, and time-to-live to calculate packet delivery ratio (PDR), end-to-end delay, and data transfer rate. The experiment is to send three sets of packets to three types of 6LoWPAN nodes: 1-hop, 2-hop, and 3-hop. Each set is carried out in 5 different data rates from 1 to 5 packet/s.

4.3.3.1 Radio Signal Strength

We first carry out a supplementary experiment between two CM5000 motes to measure Received Signal Strength (RSSI) and Link Quality Indication (LQI) between two nodes to access the IEEE 802.15.4 signal strength in CC2420 transceivers. The CM5000 devices are programmed to transmit and receive 802.15.4 wireless beacons. We maintain the connection between two nodes and record RSSI and LQI over distances ranging from 3 and 21 m with steps of 3 m. Figure 4.10 shows the experiment results indicating RSSI in a good condition within the range of 21 m and LQI remains stable at 108.

4.3.3.2 Packet Delivery Ratio

As illustrated in Figure 4.11, PDR gets very high rate of 98 percent for 1-hop nodes and slightly drops to 86 percent when data rate reaches the highest rate among the tests of 5 packet/s. For 2-hop and 3-hop nodes, PDR is lower through out the experiment fluctuating from 45 to 80 percent.

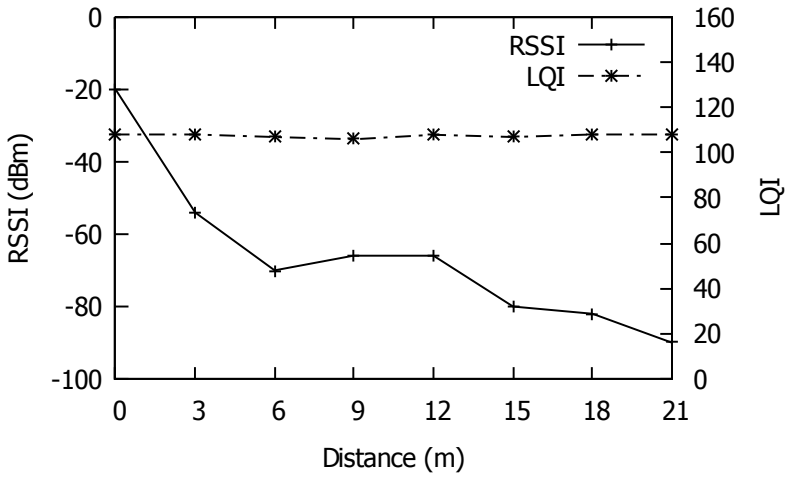


Figure 4.10: Radio RSSI and LQI.

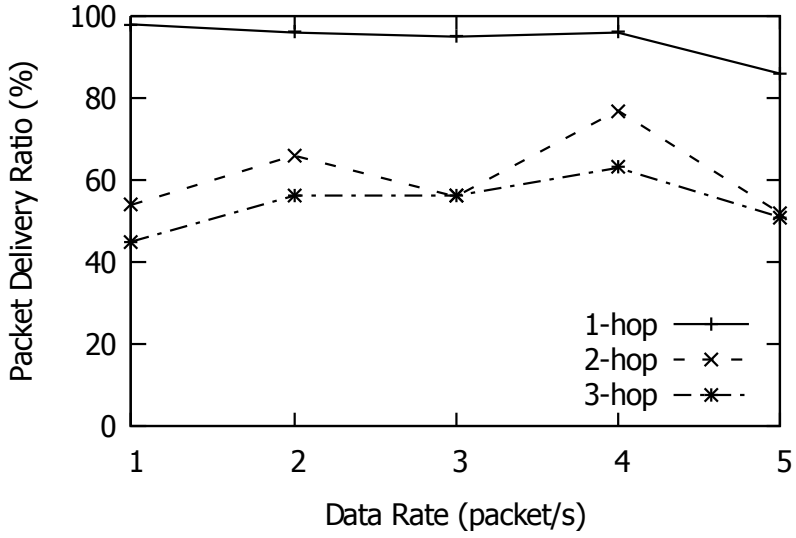


Figure 4.11: Packet Delivery Ratio.

4.3.3.3 End-to-End Delay

Figure 4.12 shows the end-to-end delay slightly increases when more packets come back and forth between nodes, it however remains very low and not much diverse between different types of nodes (1-hop, 2-hop, and 3-hop), ranging from 30 to 60 ms. These figures are considered transparent to the communication.

4.3.3.4 Data Transfer Rate

Similarly, data transfer rate shows a similar pattern with 25 kbit/s, 15 kbit/s, and 10 kbit/s for 1-hop, 2-hop, and 3-hop nodes respectively (see Figure 4.13).

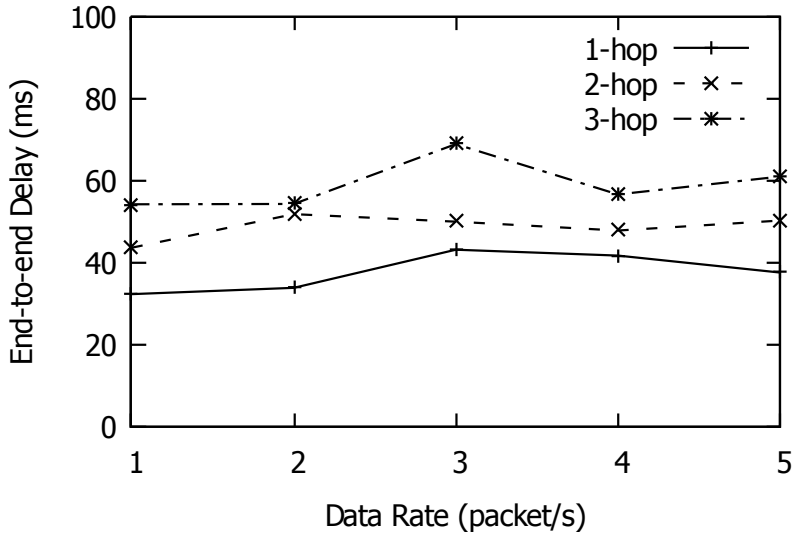


Figure 4.12: End-to-end Delay.

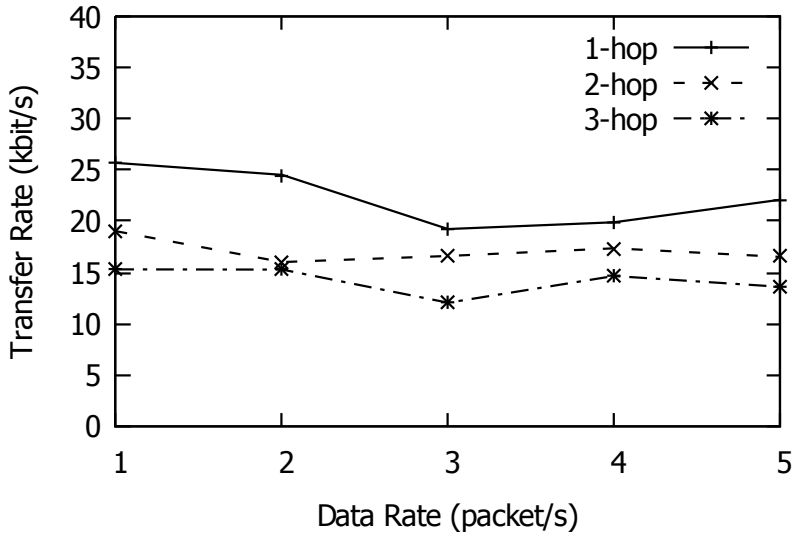


Figure 4.13: Transfer rate.

4.3.4 Service Communication

When it comes to the development of IoT applications, apart from the IP networking infrastructure, provisioning smart object services is an essential issue. Developers expect APIs that they can integrate new features and create new functionalities to their applications. Notably, CoAP is designed exclusively for smart objects to replace HTTP and can be easily translated to HTTP for a transparent integration with the Web while meeting the smart object requirements such as multicast support, very low overhead, and publish/subscribe model. Since CoAP is not native to the Web protocols, a CoAP/HTTP proxy is a common approach to provide HTTP-based APIs for CoAP services.

DPWS is another application protocol for smart objects. It brings W3C Web services technology into the IoT by defining specifications that provide a secure and effective mechanism for describing, discovering, messaging, and eventing services for resource-constrained devices. DPWS uses WSDL to describe a device, Web Services Metadata Exchange [5] to define metadata about the device, and WS-Transfer [6] to retrieve the service description and metadata information. The messaging exchange occurs via SOAP, WS-Addressing [7], and the Message Transmission Optimization Mechanism/XML-Binary Optimized Packaging [8] with SOAP-over-HTTP and SOAP-over-UDP bindings. It uses WS-Discovery [9] for discovering a device (hosting service) and its services (hosted services) and the Web Services Policy [10] to define a policy assertion and indicate the device compliance with DPWS. Secure Web services, dynamic discovery, and eventing features are the main advantages of DPWS for event-driven IoT applications.

We carry out the fourth experiment on message overhead and latency of service communication between a Web application and smart objects using CoAP, DPWS, and HTTP protocols (via CoAP/HTTP proxy). We use Java, CoAP Californium library [57], and WS4D-JMEDS [58] to implement the Web application. Figure 4.14 presents the request/response message sizes and latency of CoAP, DPWS, and HTTP transactions. CoAP messages apparently smaller than HTTP messages due to the the use of simplified headers compared to HTTP headers though the difference is at hundred kb. DPWS, meanwhile, shows a significant overhead compared to the other two protocols. The round-trip time of CoAP and HTTP communications are not much different and considered to be transparent to user's experience. DPWS round-trip time is still low but much greater than CoAP and around twice more than of HTTP.

4.4 Discussion and Lessons Learned

4.4.1 Energy Consumption

Based on the average capacity of an AA battery is 2500 mAh and nominal voltage is 1.5 V, we can estimate the battery life for smart objects to maintain the connectivity (using duty cycling) as following Formula 4.3

$$\frac{2500mAh \times 1.5V \times 2}{0.37mW \times 24h \times 365days} = 2.304years \tag{4.3}$$

where 0.37mW is the average power of smart objects in *host mode* when idle.

[5] www.w3.org/TR/ws-metadata-exchange/
[6] www.w3.org/Submission/WS-Transfer/
[7] www.w3.org/Submission/ws-addressing/
[8] www.w3.org/TR/soap12-mtom/
[9] http://schemas.xmlsoap.org/ws/2005/04/discovery/
[10] www.w3.org/Submission/WS-Policy/

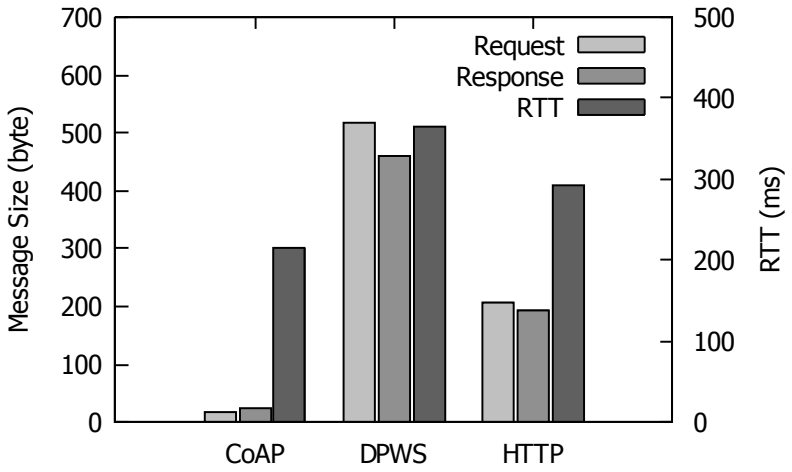


Figure 4.14: CoAP, DPWS, and HTTP message overhead and latency

.

2.3 years can be considered to be the very long for just only two AA batteries to maintain the connectivity. Similarly, Table 4.1 illustrates the estimated battery life of smart objects in some cases: idle (duty cycling) and continuously sending packets with the data rate from 1 to 5 packet/s in two modes (*host* and *router*). With the average duty cycle remains lower than 0.5 percent when objects are idle, the connectivity of 6LoWPANs is still maintained meanwhile energy consumption is kept minimal. Even in case of continuously sending data with very high data rate of 5 packet/s, two AA batteries can provide enough power for about 2.5 months.

Table 4.1: Battery powered smart object lifetime for IP connectivity.

Data rate	**Lifetime (year)**	
(packet/s)	**Host**	**Router**
Idle	2.303757013	2.64475527
1	0.7060081564	0.442350067
2	0.4414037119	0.271677592
3	0.3000770222	0.146784071
4	0.2581720706	0.107174921
5	0.209452065	0.107788434

4.4.2 Contiki OS 3.x and Network Performance

In 1-hop communication, the PDR show very high value of 98 percent, almost at theoretical PDR of IEEE 802.15.4 radio. With more than 1-hop communication, there's an obvious trend of much lower PDR. This is identified as the result of the RPL routing protocol. More investigation is expected to figure out the cause of the packet loss. The data transfer rate at around 25 kbit/s is considered low-rate due to the sacrifice of hard-

ware for the sake of energy consumption. Many IoT devices such as home appliances and sensor nodes only transfer control data with few bytes then this rate is adequate for most IoT applications containing relatively simple service communication. When considering new application ideas, system designers are expected to take into account the transfer rate to make a right choice for the network deployment.

The operation of uIP networking stacks in Contiki OS 3.x appears reliable in our intensive experiment with packets sending for a period of 24 hours. Compared to previous releases, our experience with Contiki OS 3.x indicates that IP performance has been improved considerably. Besides, there are several useful libraries with Contiki OS such as Erbium CoAP, Web server, file system, and Shell. Contiki OS programming experience is very effective with protothreads for multi-threading and event-driven applications. Our experience suggests Contiki OS is very robust and can be the universal operating system for smart objects.

4.4.3 Current IPv4 Infrastructure

Even though IPv6 is an ideal addressing space for future Internet but the shift to IPv6 is still happening at a slow pace accounting for only 5 percent of the worldwide Internet traffic, according to Cisco 6lab [11]. Smart objects have just arrived but already bear a full support of IPv6 rather than IPv4 (there is apparently no IPv4 adaptation layer for IEEE 802.15.4 alike 6LoWPAN). A backward integration appears to be a temporary problem during the transition time from IPv4 to IPv6. Some basic transition mechanisms between IPv4 and IPv6 systems have been proposed and applied throughout the Internet (RFC 4213). However, the use of such techniques for smart objects and 6LoWPAN may costly and double the effort to use IP technologies for smart objects. Furthermore, in contrast to conventional computer networks on the Internet providing several services such as e-mail, telephony, and Web, smart object services tend to use in ubiquitous applications that require application level interface rather than raw IP services. Therefore, proxy can be a fair solution on current Internet infrastructure that doesn't change the backbone of the network and provides a seamless interface for IoT applications.

4.4.4 Web Services

There are several candidate protocol for application layer in IoT including HTTP, CoAP, DPWS, XMPP, MQTT, and AMQP. Among which, DPWS and CoAP are mostly close to common Web architecture aiming to bring functionalities of smart objects (data and events) to the Web in the form of services. By following Web design principles (REST, SOA), these services can acquire open Web standards to enable them to understand the Web languages and protocols, denoted as smart object services.

[11] http://6lab.cisco.com/

CoAP follows REST architectural style, compromising a minimal subset of REST along with mechanisms of resource discovery, subscription/notification, and security measures for smart objects. It is similar to HTTP and can be easily translated to HTTP for a transparent integration with the Web, while having very low overhead. It also supports multicast and publish/subscribe model. The CoAP protocol provides a technique for discovering and advertising resource descriptions via CoAP endpoints using CoRE Link Format (RFC 6690) of discoverable resources. As standardized by IETF, CoAP is showing suitable for smart objects as well as getting attention from the community. There are many CoAP implementations available not only for smart objects (e.g., Erbium [12] for Contiki OS, *libcoap* for TinyOS, and SMCP [13] for embedded systems) but also for powerful servers (e.g., Java Californium [14]), Web browser (e.g., Copper [15]), and mobile platform (e.g., nCoAP). The Erbium implementation, according to our experiments, exposes very low overhead and supports well multicast as well as publish/subscribe model. This protocol is showing an excellent choice to meet event-driven requirements from IoT application. A secure mechanism for CoAP transaction is expected to explored more to make it widely usable in real-life applications.

DPWS, on the other hand, is the lightweight version of W3C Web Service [7] in addition to new features such dynamic discovery and event notification. Even though DPWS use XML-based SOAP envelopes (something considered bulky), our experiment shows that it can be implemented on top of IP protocol stack to (even) highly resource-constrained smart object such as sensor nodes (thanks to uDPWS [16] and Contiki OS). The request and response messages are relatively large compared to HTTP or CoAP but still well operate on very limited node. Besides, in smart objects with higher computing power and memory such as home appliances and office equipments, DPWS can perform in its best to enable secure translations between smart objects and applications.

4.4.5 Deployment

From the experiment results, we look into some deployment issues such as how large 6LoWPAN coverage can be in typical premises and how difficult the deployment can be when it comes to mass production.

IEEE 802.15.4 Radio Range

Since the radio signal is considerably strong at 15 m in reality, the range of the network is considered sufficient to several homes and buildings. 1-hop 6LoWPANs which only

[12] http://people.inf.ethz.ch/mkovatsc/erbium.php
[13] https://github.com/darconeous/smcp/
[14] http://people.inf.ethz.ch/mkovatsc/californium.php
[15] http://people.inf.ethz.ch/mkovatsc/copper.php
[16] http://ws4d.org/udpws/

consist of smart objects in the radio range of the edge router can cover the area of 707 m2 in good radio signal. That area can comfortably cover typical 2-storey houses. Table 4.2 shows more details about the estimated ranges in different facilities that 6LoWPAN can give healthy radio coverages. In most cases of average houses and offices, IEEE 802.15.4 can comfortably maintain a stable connectivity.

Table 4.2: IEEE 802.15.4 Radio Range.

Node Type	**Range (m)**	**Area (m2)**	**Typical facilities**
1-hop	15	707	large, two-storey houses
2-hop	30	2827	medium building floors
3-hop	45	6362	large building floors

Installation

IoT application thus far is frequently considered high cost and difficulty to deployment. However, with the presented design, the deployment of 6LoWPAN such as for home and building networks appears to be easy and intuitive, in the same way to conventional IP networks. The edge router hardware and software can be developed very fast and at low-cost using current advances in micro-electronics and radios (equivalent to a single computer board plus a 802.15.4 radio module). Besides, services of smart objects can seamlessly reside on the Web by implementing application protocols such as CoAP, DPWS, and protocol proxy for HTTP. By that, the development model from developers' point of view virtually remains the same, which will stimulate more the adoption of IoT applications. Furthermore, the installation of smart objects in 6LoWPAN is zero-configuration, which doesn't require any additional commissioning device (e.g., a laptop computer). In other words, a smart object can obtain an address and join the 6LoWPAN on its own, without human intervention.

4.5 Summary

We have presented our design of 6LoWPANs using open standards with a real-life implementation for home and building networks. The present study on networking performance of 6LoWPANs exhibits several positive results on the deployment and on the perspective of using IP protocols for smart objects for end-to-end communication with services/applications on the Internet. This study is the fundamental for us to develop service provisioning mechanisms presented in the next chapter to power the IoT applications on Web. They also provide essential data to set up simulation environments in Contiki OS Cooja and our own DPWSim simulators.

Chapter 5

Semantic Service Provisioning

Contents

We in the previous Chapter 4 have successfully designed and implemented 6LoWPANs for smart objects. Experiment results show that even highly-constrained objects can communicate effectively (energy, round-trip time, messages, etc). with IP protocols. 6EdR routers provide a transparent traffic between smart objects and regular IPv6 nodes, and to the Internet. The 6LoWPAN eliminates the protocol translation that is complex to design, manage, and deploy and its 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. The deployment of 6LoWPANs is relatively intuitive and easy to carry out and virtually in the same way as installing regular IP networks. Besides, application protocols such as DPWS and CoAP enable the use of smart objects services in IoT applications. However, for IoT applications on Web to use these services in practical and scalable scenarios, there still exist several problems that need to be addressed, as presented in following Section 5.1.

5.1 Provisioning Issues

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. CoAP 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). DPWS 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) [59].

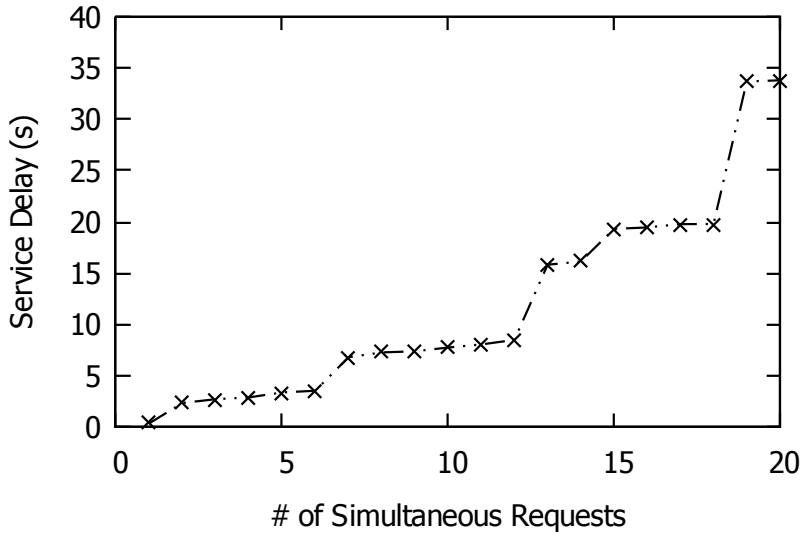


Figure 5.1: Comparison of service delay when multiple simultaneous requests are sent to one smart object.

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 stacks. 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 Figure 5.1. 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 seconds delay and the figure soars to 35 seconds in case of 20 simultaneous requests.

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.

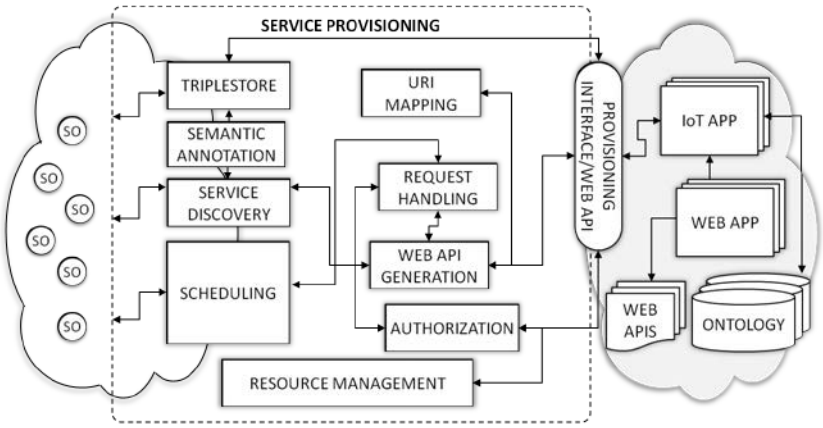


Figure 5.2: Semantic service provisioning architecture.

This chapter presents the Service Provisioning subsystem of the proposed architecture (see Figure 3.4) to address aforementioned problems while meeting requirements of service provisioning for 6LoWPAN (open standards, interoperability, low energy consumption, and reliability). In addition, we propose new schemes on other issues related to provisioning including Semantic Annotation, URI Mapping, and API Representation. The following section elaborate functional blocks and related algorithms and mechanisms for a secure, scalable, and reliable service provisioning to power IoT applications on Web.

5.2 Service Provisioning

Figure 5.2 shows nine functional blocks in our proposed service provisioning system to handle five main issues: service discovery, semantic annotation, simultaneous requests, authorization, and Web API generation. In which, Resource Management provides a user interface for resources (6LoWPAN and smart objects) management in 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.

5.2.1 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 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 5.1: 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-11e3-80c0-358d7a9bbe90
    </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 5.2: WS-Discovery Probe message.

The Service Discovery provides the same interface to query services regardless of the protocol (e.g., CoAP, DPWS, or XMPP) 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 5.1. This task can be done with the service provisioning service by using the Web API presented in Table 5.1. Similarly, instead of using complex WS-Discovery Probe message in Listing 5.2 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 5.2).

Table 5.1: Discovery API

GET /[uri]/discovery 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 5.2: Discovered services: Web APIs

PUT /[uri]/control/led 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 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	

5.2.2 Scheduling

Limited resources in smart objects result in a problem of supporting simultaneous requests from multiple IoT applications 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 Listing5.3). 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 5.3: 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 recored request), it will be added to a *request queue* (based on a queue data structure [60]). 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.

5.2.3 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 5.4: Temperature sensor smart object RDF/XML format.

Tim Berners-Lee coined the term Semantic Web as an extension of the current Web [3] 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 [61]. 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 5.5: Temperature sensor smart object N3 format.

Listing 5.4, 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 degree. Even with Notation3 (N3) format [62], a textual syntax alternative to RDF, the size of data is still rather large (see Listing 5.5). 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 re-publish these data. This can solve the problem of limited size for semantic annotation but results in many tradeoffs 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 [1].

2. Each service in smart object is represented in N3 format without default namespaces, ontolgy, 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 5.6 while the actual semantic annotation data can be reached from applications are still the same as shown in Listing 5.5. The Internet media type passing to Web API calls is denoted as *text/n3*.

```
1 :Temp5
2   a ns:Sensor ;
3     ns:hasValue "19.2" ;
4     ns:observedProperty "Temperature" .
```

Listing 5.6: Temperature sensor smart object N3 format.

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

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 5.7.

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

Listing 5.7: 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 [2], 3Store [3], and Apache Jena [4]. 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.

5.2.4 Authorization with OAuth 2.0

OAuth 2.0 (RFC 6749) 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.

[2] https://github.com/openlink/virtuoso-opensource
[3] http://threestore.sourceforge.net
[4] http://openjena.org

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 (/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 Figure 5.3.

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.

5.2.5 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 5.3:

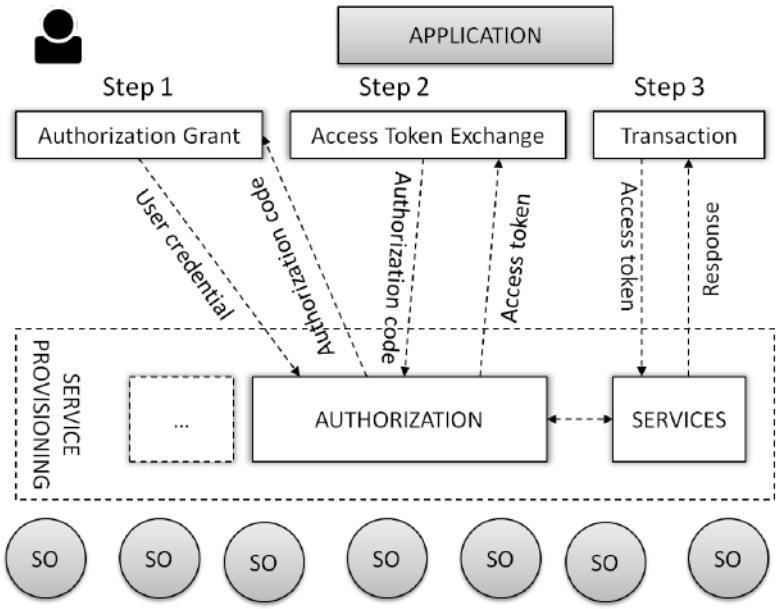


Figure 5.3: 3-step authorization process for IoT applications on Web.

The first method is straightforward since it doesn't 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 5.4 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.

Table 5.3: 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 5.4: 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

5.2.6 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 [19]. 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 5.5

Table 5.5: CRUD operation mapping scheme

Prefix	CRUD Action	HTTP Verb
Get-	READ	GET
Set-	UPDATE	PUT
Add-	CREATE	POST
Remove-	DELETE	DELETE

Web APIs are the core of the development of applications on Web these days providing interfaces for developers to develop applications on 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 5.6), which is used consistently in this dissertation:

Table 5.6: API endpoints format

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

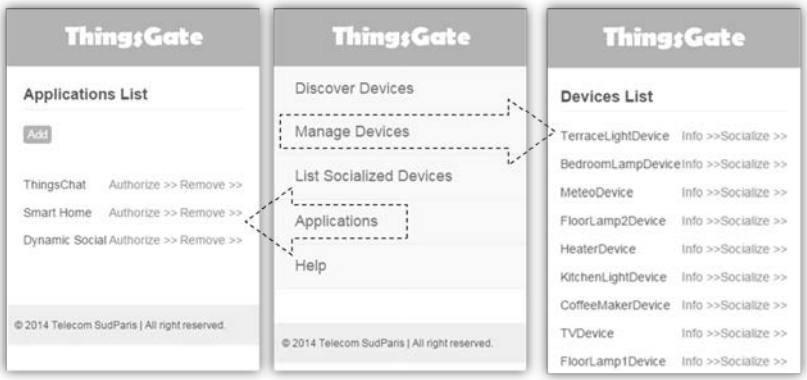


Figure 5.4: Resource Management Web UI in ThingsGate. Manage Device function/-menu shows a list of discovered devices in the 6LoWPAN of home network. User can query detailed information or add social data to each device by Info or Socialize hyperlinks associated to each smart object. Applications function/menu help users to authorize IoT applications on Web to use resources in the 6LoWPAN.

5.2.7 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). Figure 5.4 shows the Web UI of the Resource Management implemented within ThingsGate provisioning server for the Social IoT application presented in Chapter 6.

5.3 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 advantages of dynamic discovery and eventing mechanisms.

5.3.1 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 Figure 5.5). 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 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.

5.3.2 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.

5.3.3 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 5.7

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 minutes. 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 5.7: Discovery API

GET /discovery Search for a smart object with criteria	
Arguments	search: search criteria
Example	PUT http://157.159.103.50/discovery?search=Heater
GET /discovery 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.

5.3.4 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 Figure 5.5). 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 5.8).

Table 5.8: Event subscription API

POST /[smart object ID]/[event] 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

Figure 5.5 shows the network topology in two cases of our proposed design and the the original direct DPWS communication. Table 5.9 shows a list of RESTful Web APIs provided by the proxy for the heater smart object mapping with DPWS operations.

5.3.5 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

Table 5.9: RESTful Web API for the heater

RESTful Web API	DPWS operations	Argument	Discription
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

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.

Figure 5.5 shows the network topology in two cases of our proposed design and the the original direct DPWS communication. Figure 5.5 shows the network topology in two cases of our proposed design and the the original direct DPWS communication.

5.4 Performance Evaluation

We carry out the experiments with 6LoWPAN set up on Cooja simulator [63]. Experiment results from Chapter 4 allow us to set up 6LoWPAN network on network simulator with respect to real-life performance. This approach doesn't 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 radio environment. Figure 5.6 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 percent as suggested in Packet Delivery Ratio test in Chapter 4. Each node is implemented with a CoAP service enriched with the proposed semantic annotation. We aim to test

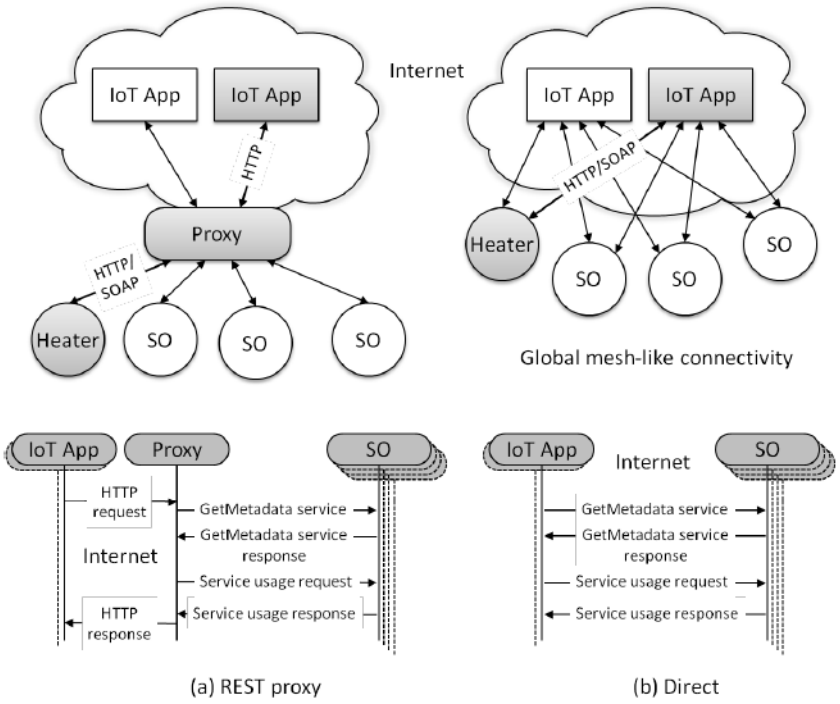


Figure 5.5: Network topology in two cases: (a) Our proposed design configures a proxy-based topology with local HTTP/SOAP binding, (b) The original smart objects Profile for Web Services (DPWS) communication configures global mesh-like connectivity for HTTP/SOAP binding. Consequently, the original DPWS introduces higher latency and overhead.

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 doesn't change the nature of the IP communication with the involvement of a number of routers).

5.4.1 Transparency

First of all, the consistent use of IP stacks 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 5.8). 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. Figure 5.7 shows that the

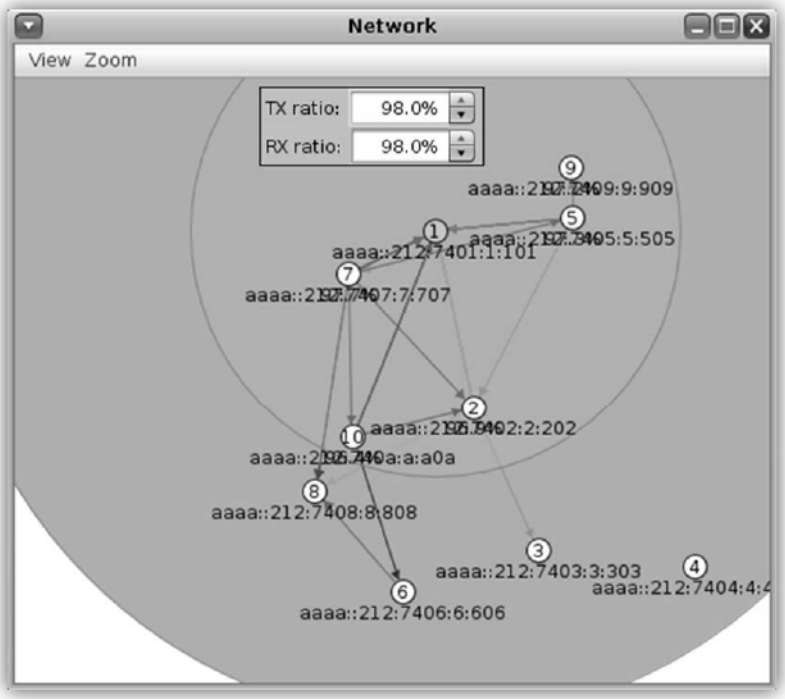


Figure 5.6: A 6LoWPAN in Cooja with 10 nodes and 3-hop distance from the edge router (node 1). All nodes are implemented with Contiki and uIP stacks. The screenshot shows the network if self-configuring with traffic exchanged between nodes.

service request delay remains stably equal in both cases, meaning that our algorithm doesn't affect non-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 5.8: Ping command from a regular IP node to 2-hop node 3 in 6LoWPAPN (aaaa::212:7403:3:303).

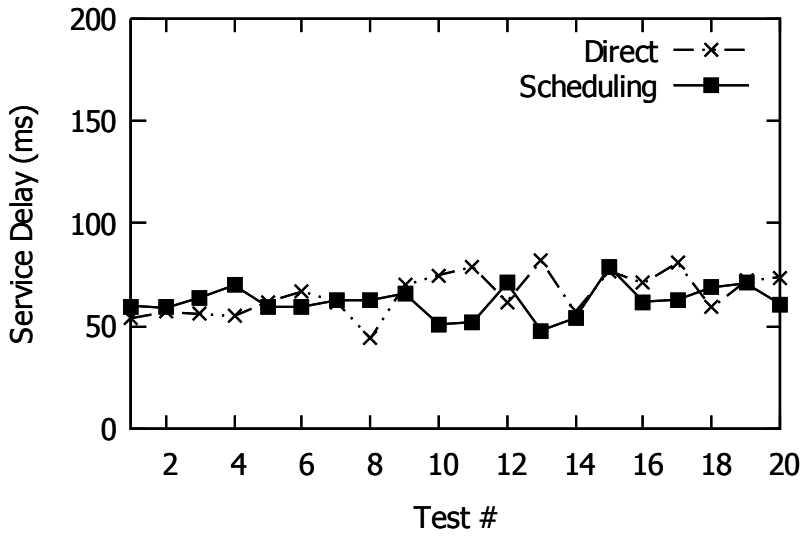


Figure 5.7: Scheduling algorithm is transparent as it does not affect a single request. Its purpose is to improve the delay when there are multiple simultaneous requests coming to one smart object.

5.4.2 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 CM5000 motes, the value is about 100 ms. As seen from Figure 5.8, the scheduling algorithm significantly improves the delay of service request in all cases with the number of requests 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.

5.4.3 Scheduling: Energy Consumption

We observe the duty cycle and energy consumption of the smart object hosting the requested service over the period of 100 seconds when the smart object handling 20 simultaneous requests in the previous experiment. Figure 5.9 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 seconds compare to 45 seconds when there is no scheduling. Although, radio duty cycle peaks at nearly 6 percent in case of using scheduling but overall energy consumption of the smart object with support of scheduling is slightly lower than without scheduling (see Figure 5.10).

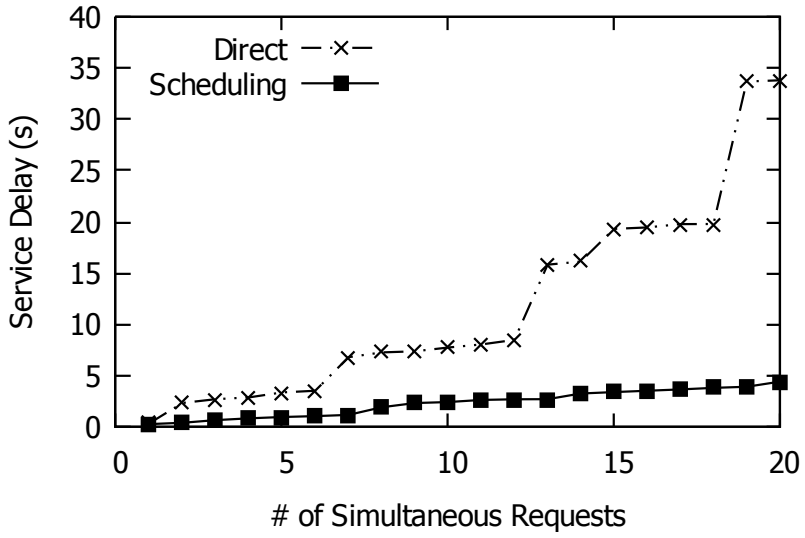
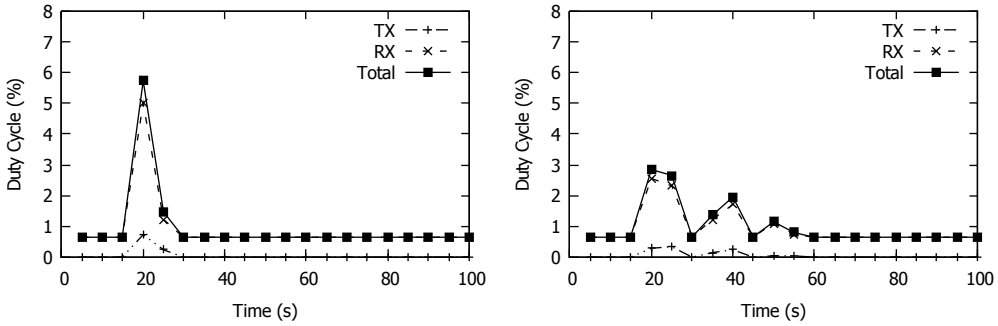
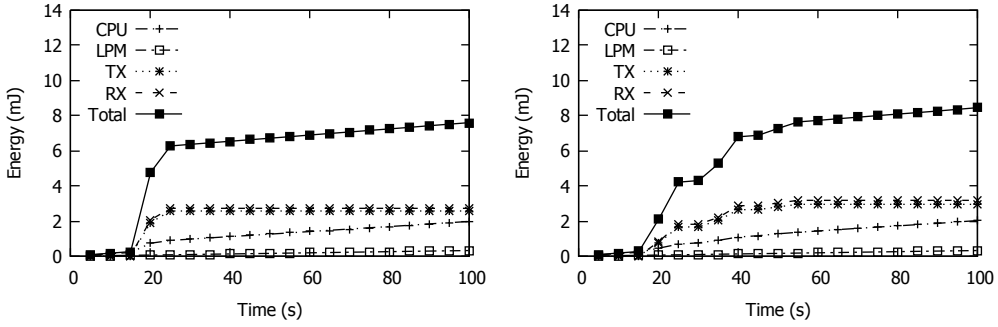


Figure 5.8: Comparison of service delay when multiple simultaneous requests are sent to one smart object service.



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

Figure 5.9: Comparison of radio duty cycle when multiple simultaneous requests are sent to one smart object.



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

Figure 5.10: Comparison of energy consumption when multiple simultaneous requests are sent to one smart object.

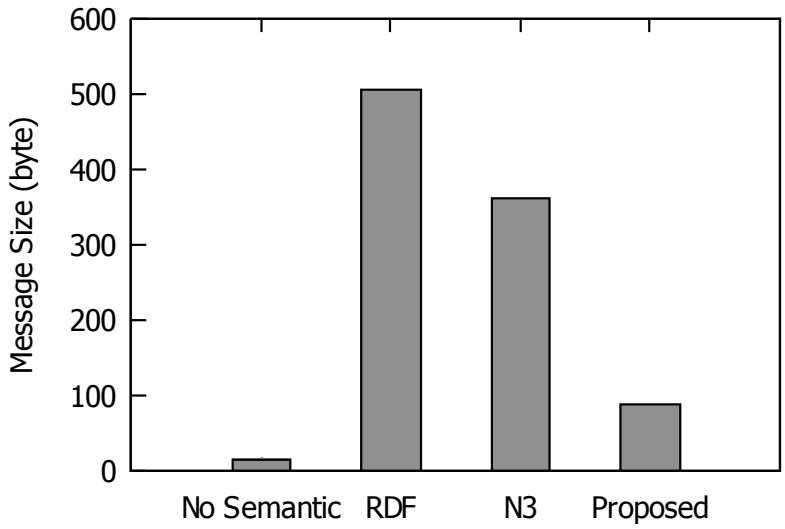


Figure 5.11: Scheduling algorithm is transparent as it does not delay a single request. Its purpose is to improve the delay when there are multiple simultaneous requests coming to one smart object.

5.4.4 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. Figure 5.11 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 stacks such as uIP and CoAP.

5.4.5 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 (Figure 5.5a), and the second one uses the direct DPWS communication (Figure 5.5b). 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 [5] that used a public Internet connection and was located about 30 km away

[5]http://tomcat.apache.org

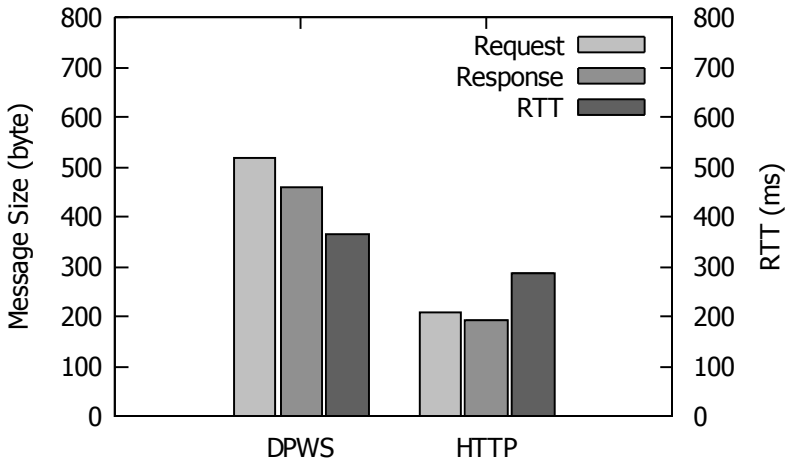


Figure 5.12: CoAP, DPWS, and HTTP message overhead and latency.

from the smart objects. We implemented the heater with a hosted service *SmartHeater* providing eight operations, as in Table 5.9. We implemented a REST proxy in Java using the Jersey library on Tomcat [6] to generate 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.

```
1 GET /proxy/heater HTTP/1.1
2 User-Agent: Java/1.7.0
3 Host: 157.159.103.50
4 Accept: text/html
5 Connection: keep-alive
6
7 HTTP/1.1 200 OK
8 Server: Apache-Coyote/1.1
9 Content-Type: text/html
10 Transfer-Encoding: chunked
11 Date: Fri, 26 Jul 2013 21:46:48 GMT
12
13 [1374820483967] ON
```

Listing 5.9: Request and response messages for obtaining the status of the heater using the proxy Web API expose relatively simple in HTTP format.

Figure 5.12 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 percent lower than when using DPWS. In many pervasive IoT scenarios requiring high responsiveness, reasonable

[6] http://jersey.java.net

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 (Figure 5.5b) is unavoidable compared to the linear increments of HTTP traffic in the proxy scenario (Figure 5.5a). Listings 5.9 and 5.10 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-6eabe38b6788
    </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-6eabe38b6788</wsa:
    RelatesTo>
  </s12:Header>
  <s12:Body>
    <i53:reply xmlns:i53="http://telecom-sudparis.eu">ON</i53:reply>
  </s12:Body>
</s12:Envelope>
```

Listing 5.10: Request and response messages for obtaining the status of the heater using DPWS expose the complex XML-based messages in SOAP format.

5.5 Summary

This chapter has presented the functional blocks along with several algorithms and mechanisms for realizing the proposed service provisioning architecture for IoT applications on Web. The main goal is to solve several problems associated with provisioning smart object services to the Web such as service discovery, multiple simultaneous requests, and service authorization in order to generate a friendly interface for developers to effectively and smoothly integrate smart objects into IoT applications on Web.

Chapter 6

Case Studies: IoT Applications on Web

Contents

This chapter presents the Service Integration process in our proposed architecture with two case studies of developing IoT applications using service provisioning and open Web standards. These are innovative IoT applications in the domains of Social Internet of Things (SIoT) and Building Automation System (BAS). The first application is ThingsChat, a SIoT platform facilitating the social relationship between human and smart objects in the similar way with between human and human in traditional social

networks. To do that, we extend our proposed semantic service provisioning by adding a Device Socializing module, creating a Socialized Web API to automate the communication between human and devices over the Social IoT platform. The second application, SamBAS, focuses on exploiting the potential of semantic data model for developing a novel building automation system with the adoption of IoT technologies for the device communication. Both the applications use DPWSim [64] (see Appendix A), a DPWS simulator developed within this research, to simulate home and office environments hosting several smart objects using IoT protocol stack. We call smart objects as devices in these contexts to align with end-user's point of view since a device or appliance is more popular than a smart object.

6.1 Devices Profile for Web Services

DPWS defines a set of implementation constraints to provide secure and effective mechanisms for service describing, discovering, messaging, and eventing for resource-constrained devices. Since its debut in 2004 by a consortium led by Microsoft, DPWS has become part of Microsoft's Windows Vista and Windows Rally (a set of technologies from Microsoft intended to simplify the setup and maintenance of wired and wireless networked devices), and has been developed in several research and development projects under the European Information Technology for European Advancement (ITEA) and Framework Programme (FP): SIRENA (02014 ITEA2), SODA (05022 ITEA2), SOCRATES (FP6), and on-going IMC-AESOP (FP7) and WOO (10028 ITEA2). Many technology giants such as ABB, SAP, Schneider Electric, Siemens, and Thales have been participating in these projects. As they have large market shares in electronics, power, automation technologies as well as enterprise solutions, their promotion of the DPWS technology promise a wide range of the future DPWS/IoT products. Schneider Electric and Odonata pioneered the implementation of DPWS leading to the early and open-source release of software stacks implementing DPWS in C and Java available at Service-Oriented Architecture for Device Website [1]. Web Services for Devices initiative [2] reinforces the implementation by providing and maintaining a repository to host several open-source stacks and toolkits for DPWS. In addition, many studies have been recently carried out to complete the technology. Experiment results show that DPWS is able to be implemented into (even) highly resource-constrained devices such as sensor nodes with reasonable ROM footprints [65]. Other technical issues of DPWS have also been explored such as encoding and compression [66], the integration with IPv6 infrastructure and 6LoWPAN [67, 68], the scalability of service deployment [69], and the security in the latest release of WS4D DPWS stacks.

[1] http://soa4d.org/
[2] http://ws4d.org/

DPWS thus far has been widely used in automation industry, home entertainment, and automotive systems [70] and also applicable for enterprise integrations [71]. It satisfies many requirements for IoT applications such as resource-constrained, event-driven, and dynamic discovery; In the meantime, it can maintain the integration with the Internet and enterprises infrastructures. In addition, the strong support from the community is another reason to make it a promising technology for the future IoT. WS4D has been developing several DPWS standard implementations in different languages and platforms as summarized in the Table 6.1. DPWS-gSOAP provides C/C++ toolkits for deploying Web services consumers and providers. It is multi-platform implementation supporting Windows-native, Windows-cygwin, Linux, and Embedded Linux. DPWS-uDPWS is DPWS implementation in C language designed for Embedded Linux, Contiki and especially for highly resource-constrained devices such as sensor nodes. DPWS-JMEDS is Java framework for DPWS supporting different Java editions. The latest release of DPWS-JMEDS hosts the feature of Android OS which paves the way for implementing services on Android devices.

Table 6.1: DPWS Implementation.

Version	Language	Operating System
DPWS-gSOAP	C	Linux, Windows, Embedded Linux
DPWS-uDPWS	C	Embedded Linux, Contiki
DPWS-JMEDS	Java	Java Virtual Machine
DPWS-Android	Java	Android

6.2 ThingsChat: A Social Internet of Things Platform

Online Social Network (OSN) has emerged as an inter-connectivity forum encouraging people to establish and expand their network of friends/acquaintances for social interacting and sharing ideas as well as various resources in textual and other multimedia formats. The OSNs aggregates users' interests, preferences, groups of friends, and activities to form rich user profiles. The concept of *content mashup* has emerged to encourage and support users' customization of their own OSN by adding services to expand the functionalities already provided or adding feeds from other OSN. These values of OSN has been changing social interaction over the Internet, from enhancing the way we reach information to enhancing the way we reach for each other. In the meantime, the IoT is gradually penetrating into our daily life with dozens of appealing products are filling up the shelves. These devices, thanks to the efforts from research activities, can now be facilitated with inexpensive sensors, low-power wireless communication protocols to sense and transmit the status of physical world to Internet. A new generation of applications on this connected ecosystem is being developed excitingly, not only to interact with single device or service but also to use the concept of mashup and composition with other

Web services to create new experiences. However, the best story has yet to come when the idea keeps flying higher and further by offering these smart and connected devices a new attribute of being *social* to benefit OSNs over Internet-connected and socialized devices. This new paradigm is called Social Internet of Things (SIoT). Industry and academia since then have been following up this trend and come up with some models and prototypes of SIoT [72] but mainly in the conceptual level and preliminary data models for the device-to-device social relationship.

Our vision is to further enhance the social interaction by bringing connected devices to a new level of being able to have social relationship with other devices and with people. To achieve that objective, we extend our proposed service provisioning architecture to facilitate the devices with social ability in the form of Web APIs that can be used by OSNs to interact with devices. Furthermore, we add to the core OSN functionalities new capabilities of profiles, intelligence, recommendation, and Natural Language Processing (NLP) to inherit all the features of OSN and IoT. This design results in a universal OSN of everything, people and devices, called ThingsChat.

6.2.1 System Architecture

ThingsChat system architecture aims to minimize the discrepancy between device and human profiles in the social network structure. In other words, ThingsChat treats devices alike human in a way that devices can make decision and communicate with human users. The architecture covers a network of people and devices with a Service Provisioning subsystem magnified by a Device Socialization module to connect devices to the social networking platform in a similar way that human users connect to it. There are two main subsystems communicating via the Socialized Web API: Service Provisioning and Social Network. The former extends our proposed service provisioning architecture not only to bring device services to the Web but also socialize these services by adding abilities (API) such as talking and making friend to human users. The latter is based on the social networking core (e.g., Elgg [3], phpBB Social Network [4], and Oxwall [5]) which features a full-fledged OSN with a Web-based User Interface (WUI). A Device Profiles database is added to store the device profiles inheriting all the properties of user profiles but containing some additional information to interact with the socialized devices such as endpoint address referring to the base URI of Web API related to the corresponding device. In addition, Device Adapter module acts as the interface for the communication between the social network platform and socialized devices. The other modules Recommender, Semantic Reasoner, and NLP Interface are in charge of realizing the human-like intelligence and recommendation functionalities for devices.

[3] http://elgg.org/
[4] http://phpbbsocialnetwork.com/
[5] http://www.oxwall.org/

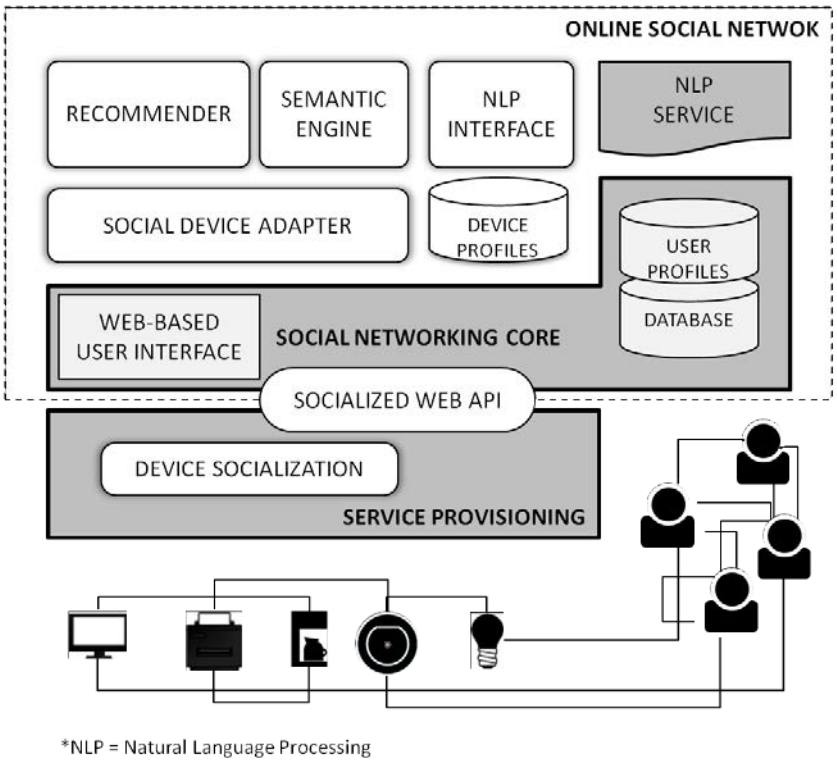


Figure 6.1: ThingsChat architecture.

6.2.2 Socialized Web API

We extend our proposed service provisioning to include the social characteristics (e.g., communication and decision-making) by adding the Device Socialization module to existing service provisioning framework. This creates a new Socialized Web API to facilitate not only the communication between devices and the SIoT platform but also between devices and human users. The service provisioning server is implemented in the form of a home gateway ThingsGate, which extends the in-network implementation of our proposed architecture presented in Section 5.3 to discover, store, and transmit device services to ThingsChat. It provides an interactive ThingsGate WUI in Resource Management (a service provisioning functional block, details at 5.2.7) for users to grant authorization to ThingsChat and initialize devices with socialized functionalities, turning them into social entities. Table 6.2 shows the main endpoints of the Socialized Web API from ThingsGate.

In this application, ThingsGate also plays another role as the mediator for an important step called *socializing device*. It involves user or device owner in the loop to authorize and customize the device to fit in the SIoT platform. Figure 6.2 illustrates the step of socializing a robot cleaner via Resource Management WUI. ThingsGate discovers a DPWS device with an ID *RobotCleanerDevice* (a robot cleaner with details shown in Figure 6.3) in the network and automatically generates Web API endpoints for provi-

Table 6.2: ThingsGate Socialized Web API.

Web API Endpoint	Arguments	Description
GET /social/device-list	N/A	List socialized devices
POST /social/register	device-name device-username password	Register a social device profile
POST /social/chat-to-device	device-id post-id content	Send new post to a user
POST /social/friend-request	device-id user-id	Send new post to a user
GET /social/nlp	content	Get device code translated to natural language

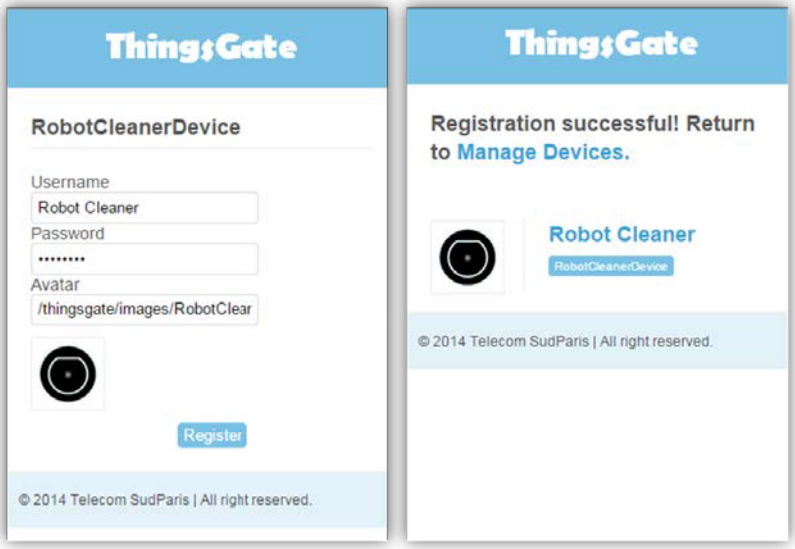


Figure 6.2: ThingsGate Resource Management: Socializing a device.

sioning its service. In this step, the device is already available for communicating to IoT applications on Web but not yet ready for interacting with users on SIoT platform. An user logs in to the system via ThingsGate WUI to see the list of discovered/provisioned devices and select *RobotCleanerDevice* to socialize it. ThingsGate than redirects the user to a registration interface that she can customize the device with some social characteristics such as user name and avatar. The user clicks Register button to finish the registration, and if ThingsGate successfully registers the device in the SIoT platform User Profiles, it will generate device's Socialized Web API endpoints and add the device into the socialized list, which can be seen later by user (see Figure 6.3)

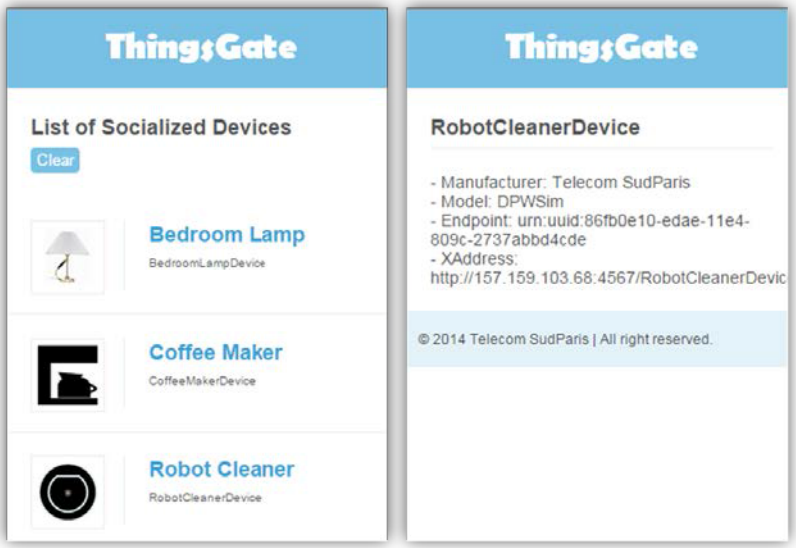


Figure 6.3: ThingsGate Resource Management: List of socialized devices.

6.2.3 ThingsChat Platform

In addition to the core functionalities of an OSN, ThingsChat has the Device Adapter module to interact with the Socialized Web APIs. This module has its own set of API endpoints (see Table 6.3) that can cooperate with the Socialized Web API to fulfill the duplex communication between users and devices. For example, a device service can call an API endpoint in the request shown in Listing 6.1 to talk to user Nadia in a previously-established conversation with post-id = 375. Device Adapter is supported by Device Profiles database that stores device profiles, and extension of User Profiles in the core social networking database. The extension includes API endpoints and ownership of the devices by which ThingsChat can notify devices in the same way that it notifies human users (via notification messages). NLP Interface pre-process natural language messages from human users into a list of machine readable commands (tokens) and vice versa. Semantic Engine is in charge of processing semantic data received from devices in N3 format. It creates a data model out of the triples received from devices and carries out reasoning to extract more information that can be used in the Recommender module to make recommendation to human users or other devices.

```
1 POST /socialnet/chat-to-user.php HTTP/1.1
2 Host: thingschat.com
3 Accept: text/n3
4
5 device-id=Robot Cleaner
6 post-id=375
```

Listing 6.1: API call from device side to talk to user Nadia.

Table 6.3: ThingsChat Device Adapter Web API

Web API Endpoint	Arguments	Description
POST /socialnet/chat-to-user.php	device-id post-id	Send message to a user
POST /socialnet/confirm-friend.php	device-id user-id	Confirm making-friend request
GET /socialnet/nlp.php	content	Get natural language translated to device code

6.2.4 Prototype and Experiment

ThingsChat application consists of four components: Virtual Home (powered by DPWSim), provisioning server ThignsGate, SIoT platform ThingsChat, and an external NLP Service. Virtual Home is created by DPWSim to precisely generate DPWS protocol messages for each devices. DPWSim also helps creating rich graphical user interfaces for the simulation environments (see Figure 6.4 for an example of such interface with the help of an 3D artist). DPWS devices can be discovered and communicated by DPWS clients following DPWS standards. ThingsGate is a Java Web application running on an Apache Tomcat server [6]. ThingsChat is based on phpBB Social Network Engine providing all basic features of an OSN such as profiles, friends, and sharing. It also has a WUI to allow people and devices to talk. An NLP Service uses Apache OpenNLP [7] to provide a tokenization function for natural language text. All the machines/servers DPWSim, ThingsGate, ThingsChat, and NLP Service are deployed in the same local network for testing purpose.

To illustrate the application, we explain two use cases of Coffee Maker in office and Robot Cleaner at home (see Figure 6.4). Nadia is living in an apartment (Virtual Home) with several DPWS appliances such as a TV, lamps, coffee makers, and heater. She can easily install these devices in the home network from her mobile phone by using the Resource Management module. ThingsGate allows Nadia to detect available devices and then socialize them by simple touches on the WUI. She can do the same procedure in her office to install new devices, probably with the help of a network administrator. When she finishes setting up things, she can talk with her devices anywhere through ThingsChat in a natural way. In the morning, when Nadia is on the way to office, Coffee Maker based on her profile offers a coffee at 09:00, but Nadia has an early meeting at that time so she asks the Coffee Maker to make it few minutes earlier at the same time she is talking with her friend on ThingsChat. Coffee Maker receives her request and update to status that her favorite coffee has set to be ready at 08:55, it also knows how to reply when Nadia say thanks. In another use case, it has been three days that

[6] http://tomcat.apache.org/
[7] http://opennlp.apache.org/

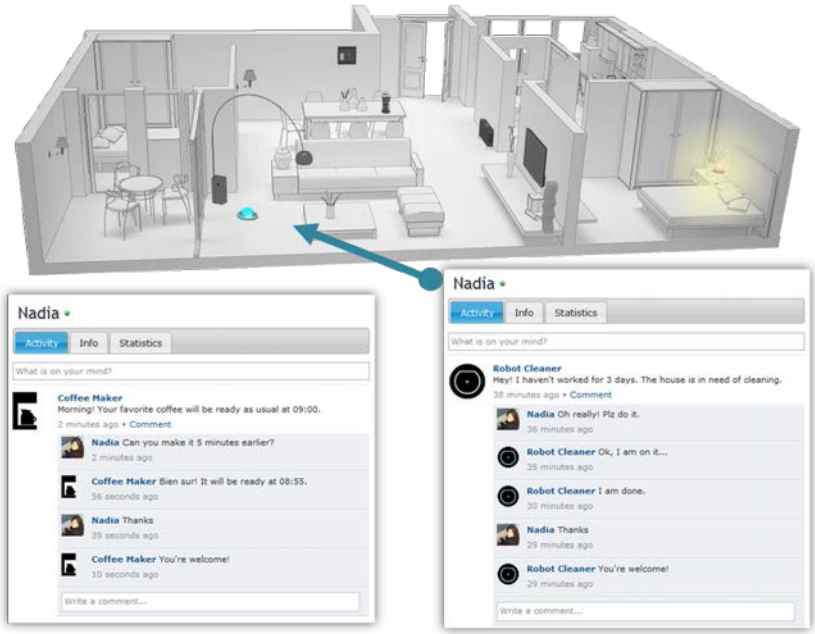


Figure 6.4: Coffee Maker in office and Robot Cleaner at Virtual Home recommend to Nadia according to her profile and can understand her requests.

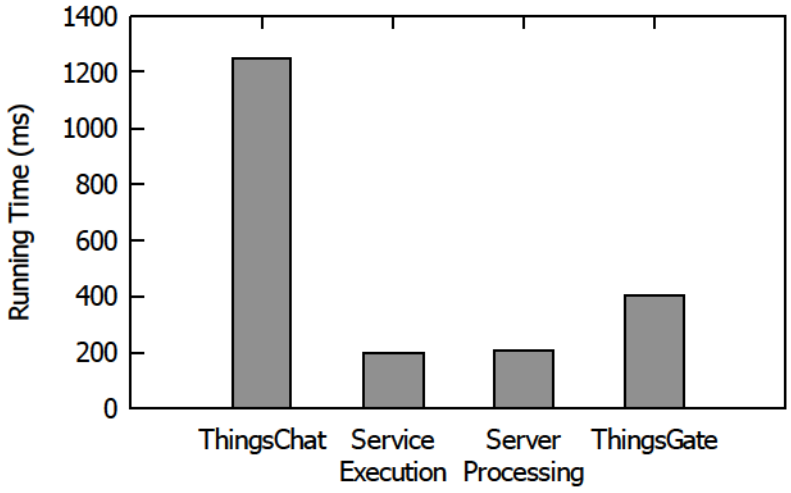


Figure 6.5: Performance of the Socialized Web API of ThingsGate and Device Adapter Web API of ThingsChat.

Robot Cleaner hasn't cleaned the house. Robot Cleaner, based on Nadia's profile for here preference of cleaning frequency, reminds her to have the house cleaned and she is glad to know that and asks the Robot Cleaner to do it.

We aim to evaluate the performance of the Socialized Web API in provisioning server (ThingsGate) and Device Apdapter Web API in SIoT server (ThingsChat). The experiment exhibits a typical interaction flow between users and devices: user asks device to do some jobs, device carries out the requested jobs and replies back to user. This conversa-

tion involves two API endpoints, one from ThingsGate (POST [thingsgate]/social/chat-to-device) [8] and the other from ThingsChat (POST [thingschat]/socialnet/chat-to-user.php) [9]. There are also other modules used in the conversation such as natural language processing, machine tokens conversion, semantic data processing, and recommendation, however we focus on evaluating APIs and these modules in minimal workload. We break down the job on ThingsGate into two parts: Service Execution for invoking the requested device service and API Processing for server to process the API request. Figure 6.5 shows the running time for each task in the above conversation: request to [thingschat]/socialnet/chat-to-user.php,

ThingsGate request takes average of 404 ms to complete consisting 199 ms for service execution and 205 ms for API processing, that is reasonably low for a service provisioning server in a local network deployment. ThingsChat request, however, takes just over 1 second for processing and transmitting data back and forth between ThingsGate and ThingsChat. It is because that ThingsChat API mainly deals with querying the database system for handling user and device profiles.

6.3 SamBAS: A Building Automation System

The idea of smart house or smart building has been around for many years receiving much expectation. A building automation system, residing at the heart of such smart environments, interacts with its components including hardware, software, and the communication among them. It involves in several disciplines such as electronics, informatics, automation, or control engineering. BAS, since its debut, has been developed and promoted by a community of developers, technologists, and scientists with plenty of impressive prototypes and products. These products bring in comforts and conveniences to daily life, freeing people from tedious house-works or office-works. Use cases vary from very simple ones, e.g., automatically turn on/off the lights to complex and critical situations, e.g., security surveillance. Furthermore, BAS also provides value-added services by offering intelligent services such as customer tracking in shopping malls or elderly people healthcare services. All of those make it a very promising business attracting attention of the community to target not only organization customers but also individual end-users.

Industry and academia have been developing many new technologies for building automation such as communication protocols, data management, data bus systems, software components, and/or new hardware devices which can be integrated in the new systems. Thanks to all those efforts, building automation has advanced over the last decades with several communication protocols and a variety of BAS products from many

[8] [thingsgate]: ThingsGate server address
[9] [thingschat]: ThingsChat server address

different vendors. A comprehensive overview of communication protocols in building automation can be found in [73] with different BAS products. Traditionally, equipments in BASs are interconnected by proprietary communication protocols such as LonWorks [74], Building Automation and Control Network (BACnet) [75], or KNX [76]. These protocols have been used to cover all the features of building automation, including Heating, Ventilating and Air Conditioning (HVAC), lighting, and alarming. There are also many other standards for BAS. HomeConnex (Peracom Networks), for example, is a home entertainment network which unites PCs, TVs, audio/video components and set-top devices into an integrated system. X-10 (X10 Inc.) is another industry standard using power line and radio for communication among electronic devices used for home automation. Other proprietary standards include Easy-Radio (Low Power Radio Solution Inc.), No New Wires (Intellon Corp.), Sharewave (Sharewave Inc.), SoapBox (VTT Electronics), and Z-wave (Zensys).

Even though, the BAS market is very active with plenty of appealing BAS solutions but consumers are well aware of the value of such smart systems. However, it is not difficult to recognize the reluctance among customers in adopting available BAS products on the market. The main reasons are identified as the cost and the scalability of these proprietary systems. This normally leads to the suspension or partially deployment of several on-the-table building renovation projects.

The arrival of the IoT paradigm has opened up new approaches in the building automation domain with the availability of new devices and communication protocols which are open, light-weight, low-cost, and interoperable. IoT open standards, both in software and hardware have brought building automation in a new perspective that is never more realistic and affordable. This case study, therefore, aims to provide a new solution for BASs using open Web standards and IoT communication technologies such as 6LoWPAN and DPWS. We focus to solve the two fundamental problems of BASs: the first one is to enable the system to quickly adapt to the dynamic changes in user and environment context; the second one is to coordinate devices in order to serve the diverse and complex user's needs involving not only one but several services at the same time. To solve these problems, first, we semantically model the user and environment context using RDF and from DPWS communication. Then, we apply service composition over semantic data from device services and predefined semantic policy rules to select, bind, and execute appropriate services. The proposed solution, SamBAS is to use composite service plans to describe users' requirements using the proposed Composition Plan Description Language (CPDL). We design a Building Ontology containing the description of concepts and relationships in building environment for the reference schema of storing graph data in the triplestore database. Context information is modeled, processed and passed to service composition engine to coordinate appropriate devices/services based on predefined policy rules and five-step composition process.

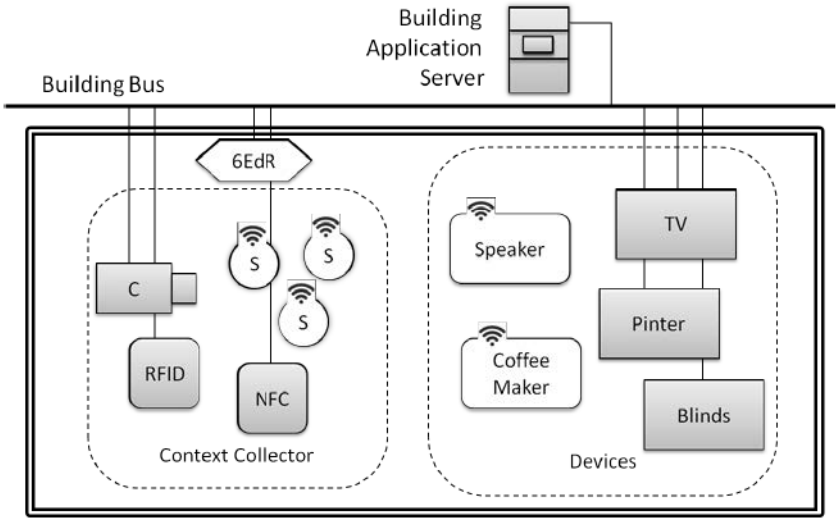


Figure 6.6: System configuration.

6.3.1 System Architecture

System configuration shown in Figure 6.6 depicts a typical setup of devices inside a room of a building. There are DPWS devices consisting of a wide range of building equipment (e.g., TV, printer, and light bulbs) and context collectors (e.g., sensors, RFID, NFC readers). These devices are connected to the building network via wireless or wired connections with IP stack and low power wireless protocols. Sensors are implemented by uDPWS over the Contiki OS. Equipment with larger memory and processing power run on Embedded Linux or Andoid OS, their functionalities are developed using DPWS-gSOAP, some can be connected directly to the regular IP network, some join the network via home access point or 6EdR. Sensors with their sensing capacity can monitor the environment RFID readers, NFC readers, or camera can identify users. All hardware components get connected to the Building Application Server that hosts the core functionalities of the system.

System architecture shown in Figure 6.7 consists of a Service Provisioning module based on our proposed service provisioning and several other functional blocks to use DPWS services in building automation. In which, COMPOSITION subsystem resides at the center of the architecture with its five-step composition process helps to realize and deliver appropriate composite services to user based on the user and environment context. The subsystem can be functionally divided into selecting services, binding services and executing services which are reflected in three components: Service Selector, Service Binder, and Service Executor respectively. In addition, Composition Plan Creator has access to Composition Plan database and provides functionalities for users to create, modify, and delete composition plans. Composition Broker decides whether to call the COMPOSITION or not via a decision-making process based on the received context information. Context Processor receives and process semantic context data from

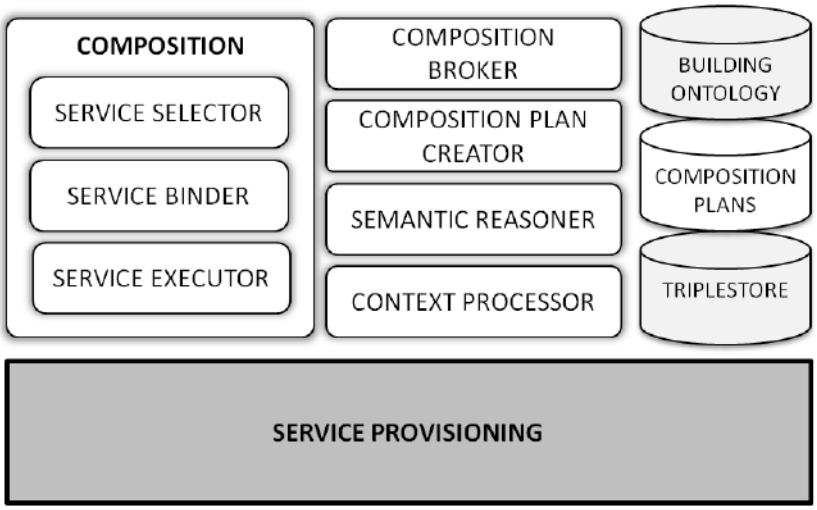


Figure 6.7: System architecture consists of four main subsystems DATABASE, COMMUNICATION, DISCOVERY and COMPOSITION and four other modules Composition Plan Creator, Semantic Reasoner, Composition Broker and Context Processor.

context collectors, and then sends them to the Composition Broker.

6.3.2 Building Ontology and Graph Database

Building Ontology defines concepts and relationships between entities within the building environment. It provides a schema to build up semantic database in the form of graph data. This is a new concept of database for Semantic Web which consumes RDF to present the domain knowledge. RDF is a common acronym within the semantic web community as it creates one of the basic building blocks for forming the Web of semantic data. A graph consists of resources related to other resources, with no single resource having any particular intrinsic importance over another. RDF database includes of RDF statements, or sometimes called an RDF triples. The term triple is used to describe the components of a statement with three constituent parts: subject, predicate, and object of the statement.

The primary purpose of this ontology is to classify things in terms of semantics, or meaning and especially for describing policies used in composition process. A class in OWL [44] is a classification of individuals into groups which share common characteristics. If an individual is a member of a class, it tells a machine that it falls under the semantic classification given by the OWL class. This provides the meaning of the data that helps reasoning engine to draw inferred information from the database. Listing 6.2 shows a part of Building Ontology document in OWL by Protégé-OWL editor [10]. It consists of document header and the declaration of the class *Policy* with two properties of *applyFor* and *hasCondition*. These properties also reflex the relationship of class *Policy*

[10]http://protege.stanford.edu/

```
<rdf:RDF xmlns=''http://www.it-sudparis.eu/bas_ont#''
     xml:base=''http://www.it-sudparis.eu/bas_ont''
     xmlns:rdfs=''http://www.w3.org/2000/01/rdf-schema#''
     xmlns:owl=''http://www.w3.org/2002/07/owl#''
     xmlns:xsd=''http://www.w3.org/2001/XMLSchema#''
     xmlns:rdf=''http://www.w3.org/1999/02/22-rdf-syntax-ns#''
     xmlns:xml=''http://www.w3.org/XML/1998/namespace''>
    <owl:Ontology rdf:about=''http://www.it-sudparis.eu/bas''/>
...
<!-- http://www.it-sudparis.eu/bas_ont#Policy -->
<owl:Class rdf:about='' http://www.it-sudparis.eu/bas Policy''>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=''applyFor''/>
                <owl:someValuesFrom rdf:resource=''Building''/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=''applyFor''/>
                <owl:someValuesFrom rdf:resource=''User''/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=''hasCondition''/>
                <owl:someValuesFrom rdf:resource=''Condition''/>
            </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
...
```

Listing 6.2: Building Ontology Document.

with other classes including *Building*, *User* and *Condition*. Figure 6.8 shows the classes of Building Ontology and their hierarchical relationship. An example of the hierarchy between classes of *User* and *Director* can be seen in the figure with the arrow starting from *User* pointing to *Director* which means *Director* is a subclass of *User* and inherits all the properties of *User*.

6.3.2.1 Context-Awareness

Context-awareness plays an important role in the pervasive computing architectures to enable the automatic modification of the system behavior according to the current situation with minimal human intervention. Since appeared in [77], context has become a powerful and longstanding concept in human-machine interaction. As human beings, we can more efficiently interact with each other by fully understanding the context in which the interactions take place. It is difficult to enable a machine to understand and use the context of human beings. Therefore the concept of context-awareness becomes critical and is generally defined by those working in ubiquitous/pervasive computing,

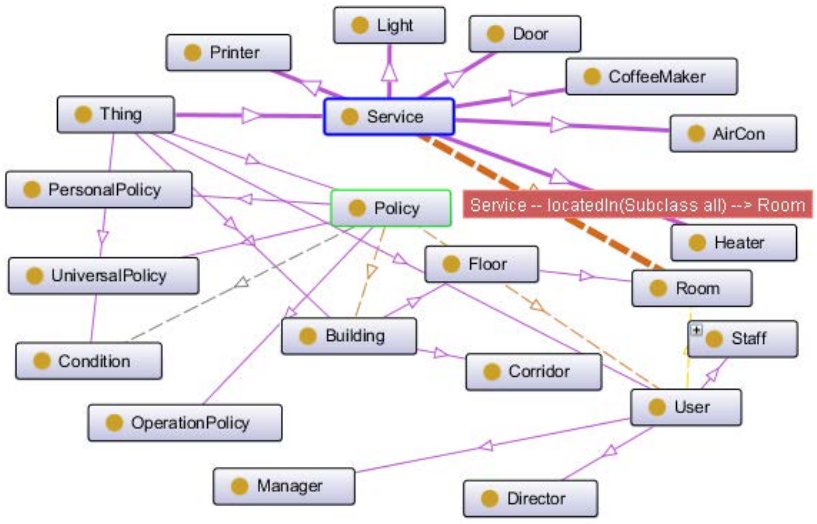


Figure 6.8: Building Ontology graph. The highlighted blocks in the graph show the hierarchy among class *Service* and its subclasses. The dotted line with a label presents a property called *locatedIn* which takes class *Room* as object meaning a service is located in a room.

where it is a key to the effort of bringing computation into daily lives. One major task in context-aware computing is to acquire and utilize information about the context of participating entities of a system in order to provide the most adequate services. The service should be appropriate to the particular person, place, time, event, etc. where it is required. In the scope of this building automation, user, device, and environment context are considered in order to bring more efficient service composition. These context data are sent to the system by using DPWS events implemented in context collectors and devices.

```
@prefix : <http://www.it-sudparis.eu/bas_data#> .
@prefix bdg: <http://www.it-sudparis.eu/bas_ont#> .

:Context1430042727831
    a bdg:UserContext ;
    bdg:happenIn :Room803 ;
    bdg:hasActor :Jennifer ;
    bdg:time "2015:04:26 12:05" .

:Context1430043339256
    a bdg:EnvironmentContext ;
    bdg:happenIn :Room803 ;
    bdg:hasActor :TempSensor803 ;
    bdg:time "2015:04:26 12:15" .

:Context1430056199063
```

```
17      a bdg:DeviceContext ;
18      bdg:happenIn :Room803 ;
19      bdg:hasActor :CoffeeMaker803 ;
20      bdg:time "2015:04:26 15:49" .
```

Listing 6.3: *Context* Data.

Listing 6.3 illustrates three pieces of context data for each type of context: User Context, Device Context, and Environment Context. The context data then are sent to Composition Broker which plays the role as a composition decision maker. It decides whether to call the COMPOSITION or not based on data defined in policy. For example, if the context information of room temperature is over 10 degree Celsius, no composition will be carried out otherwise Composition Broker checks the temperature with current status of the system to launch the COMPOSITION in case the situation is labeled as context change.

6.3.2.2 Policy

A policy is represented by *Policy* class in Building Ontology, which applies to a user, device, or location and contains a condition (Condition class) for representing the policy. There are three types of policies: Operation Policy, Universal Policy, and Personal Policy. Listing 6.4 illustrates a piece of data containing a policy called *UniversalHeatingPolicy* which is an instance of *OperationPolicy* (Building Ontology class). It applies for all users, instances of *User* (Building Ontology class) and has condition *HeatingCondition* (data). *HeatingCondition* is later on described as an instance of *Condition* (Building Ontology class) with "Heating" type and taking the value 10. Previously, two name spaces were defined at the header, one for the data and the other for the ontology.

```
1  @prefix : <http://www.it-sudparis.eu/bas_data#> .
2  @prefix bdg: <http://www.it-sudparis.eu/bas_ont#> .
3
4  :UniversalHeatingPolicy
5      a bdg:OperationPolicy ;
6      bdg:applyFor bdg:User ;
7      bdg:hasCondition :HeatingCondition .
8  :HeatingCondition
9      a bdg:Condition ;
10     bdg:conditionType ''Heating'' ;
11     bdg:conditionValue 10 .
```

Listing 6.4: *HeatingCondition* Rule Data.

6.3.2.3 Reasoning

Graph database built around the Building Ontology enables Semantic Reasoner to infer additional information from existed data and relationship. We exlain a simple example of

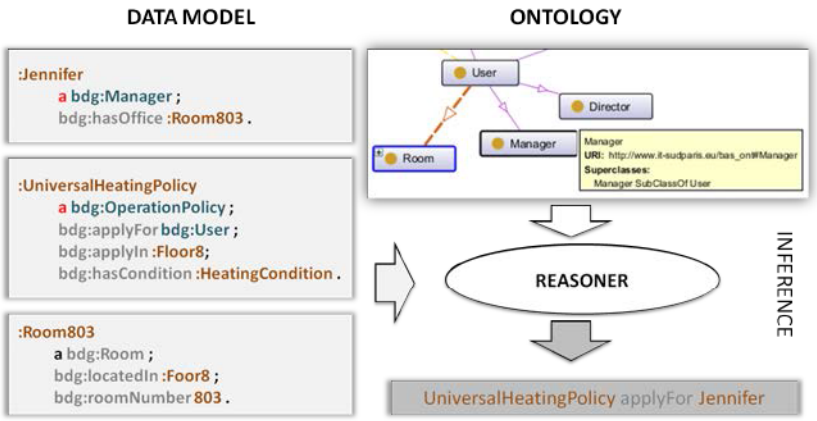


Figure 6.9: Reasoning example.

the reasoning from the data shown in the Listing 6.4 to infer that *UniversalHeatingPolicy* applies for user Jennifer. The reasoning process is depecited in Figure 6.9 in which a fact is stated as the *UniversalHeatingPolicy* rule applying for instances of *User* class. A reasoner with basic capacity can be used to demonstrate the use case, e.g., Apache Jena [11] natively-supported reasoner. An inference model is created which takes the reasoner, Building Ontology and the Graph Database as input parameters. Data in the form of resources and properties are then created from database. A simple code line can be used to generate an entailed relationship. Specifically, user *Jennifer* who is an instance of *Director* (Building Ontology class, subclass of class *User*) would be imposed by the *UniversalHeatingPolicy* rule as well. This reasoning model helps to reduce the database size and quickly collect all related data of an event or user which are all necessary for the service composition process.

6.3.3 Semantic Context-aware Service Composition

Residing at the heart of the proposed BAS, the COMPOSITION subsystem is in charge of answering composition requests from Composition Broker with regard to collected context information. It then gets access to all related resources to coordinate appropriate devices/services to serve the request. Previously, Building Ontology and Graph Database have been discussed to provide the semantic database. Also, context information processed by the Context Processor is passed to the composition process as the input data. In addition, a description language is designed to describe the composition plans and a five-step composition process is proposed to efficiently and accurately carry out service composition.

[11] http://jena.apache.org/

6.3.3.1 Composition Plan Description Language (CPDL)

A language called Composition Plan Description Language (CPDL) has been designed to describe composition plans associating with each context. An example of a CPDL document is shown in the Listing 6.5. This document describes a composition plan related to user *Jennifer* with the context of when she comes in her office (room 803) in the morning. It defines the composite service in that context consisting of four component services *Window*, *Light*, *CoffeeMaker*, and *Heater*. The actual execution of this plan depends on the context, user, localtion, and policies.

```
<?xml version=``1.0'' encoding=``UTF-8'' ?>
<CSDL xmlns:xsi=``http://it-sudparis.eu/bas''>
  <plan user="Jennifer" location="Room803" context="MorningCheckin">
    <service status="on">Window</service>
    <service status="on">Light</service>
    <service status="on">CoffeMaker</service>
    <service status="on">Heater</service>
  </plan>
</CSDL>
<xml>
```

Listing 6.5: Composition Plan Description Language (CPDL).

6.3.3.2 Service Composition

five-step service composition process is shown in Figure 6.10 which visually depicts six phases of the composition as follows:

- Step 1: Collect and process context information
- Step 2: Query related policies, make decision to call COMPOSITION
- Step 3: Query related services, select services
- Step 4: Bind services to their operations
- Step 5: Execute operations of services

The process starts with an event notified from context collectors when they detect changes in context and send that information in to the Context Processor. This information can be one of the three types of context: User, Device, and Environment. Context Processor processes and represents this information in semantic data which are sent to the Composition Broker to decide whether to move on by calling the COMPOSITION or not. In case no action needs to be carried out, the system switch to the sleep mode, otherwise the COMPOSITION is called. Then, resources are collected in the database to support the composition process. Service Selector uses provided context information,

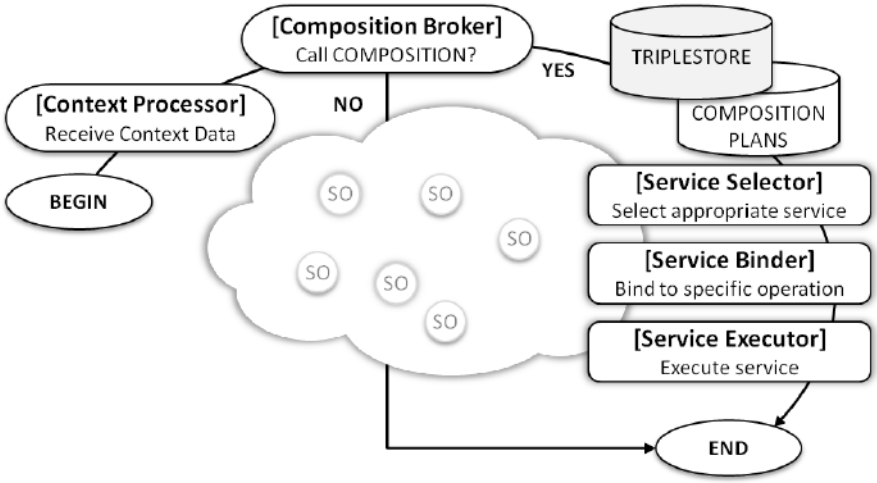


Figure 6.10: Five-step composition process.

CPDL data of the user at that context and inferred policies from the Semantic Reasoner to select appropriate services and create a concrete description of the required composite service. Service Binder follows up by binding with operations of selected services and Service Executor gets access to Service Cache to execute that operations.

6.3.4 Prototype and Experiments

We develop SamBAS prototype to illustrate the operation of the proposed system and to test the feasibility and scalability of the system. The prototype uses DPWSim for simulating DPWS devices in an office building. A Graphical User Interface on top of the devices representing an office plan along with its actors: office equipment and a user who can move around the office space to change her context as shown in Figure 6.11. Context changes in environment and devices are activated by user by firing device events provided in DPWSim. The SamBAS consists all the system components discussed previously. Building Ontology is developed using Protégé-OWL editor, graph database is represented in N3 format, and the COMPOSITION modules are developed in Java programming language on an Building Application Server with Intel processor 2.6 GHz, 6 GB RAM. It uses Jena library for semantic data manipulation and Jena integrated reasoner for inference functionalities.

Figure 6.11 illustrates a use case when a user Jennifer comes to her office located in the room 803 in the morning. When she enters her office, she uses her RFID keycard to check on the RFID reader located on her office door. This RFID reader, functioning as a context collector, sends a context-change notification to the Composition Broker to check with associated policies whether to call up the COMPOSITION or not. In this scenario, it is YES. The system uses the reasoner to collect all the policies constrained to the user to create a concrete appropriate composite service based on the user's composition plan,

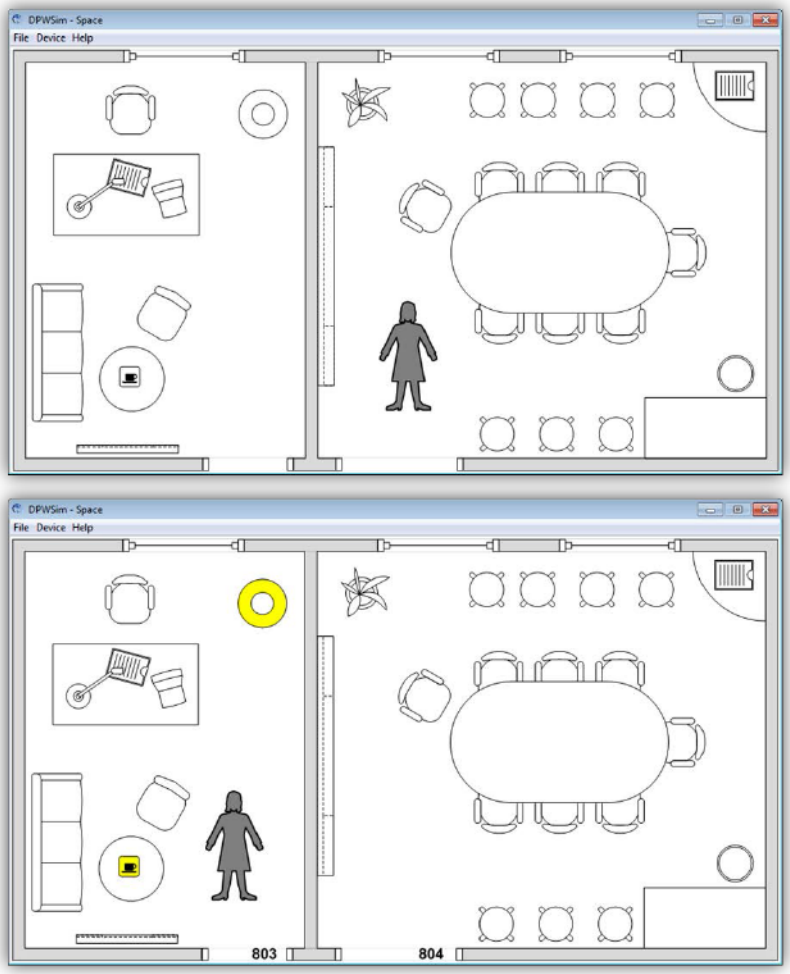


Figure 6.11: DPWSim office simulation demonstrates the service of user by the context. The composite service consists of two component services *Light* and *CoffeeMaker* is executed when the user is present in her office.

which, in this case, consists of two component services *CoffeeMaker* and *Light*. Then the two concrete context-based services *CoffeeMaker803* and *Light803* are selected and bound to their operations and finally executed by Service Executor to serve the user.

We carry out the experiment to measure the running time of the COMPOSITION process against the size of the services in the building varying from 50 to 500. The composition plan used in this experiment has 10 component services (size = 10). Results from Figure 6.12 show that the semantic model performs efficiently with the composition time remains very low even with the data size of 500 services, which is estimated for medium building with about 50 rooms.

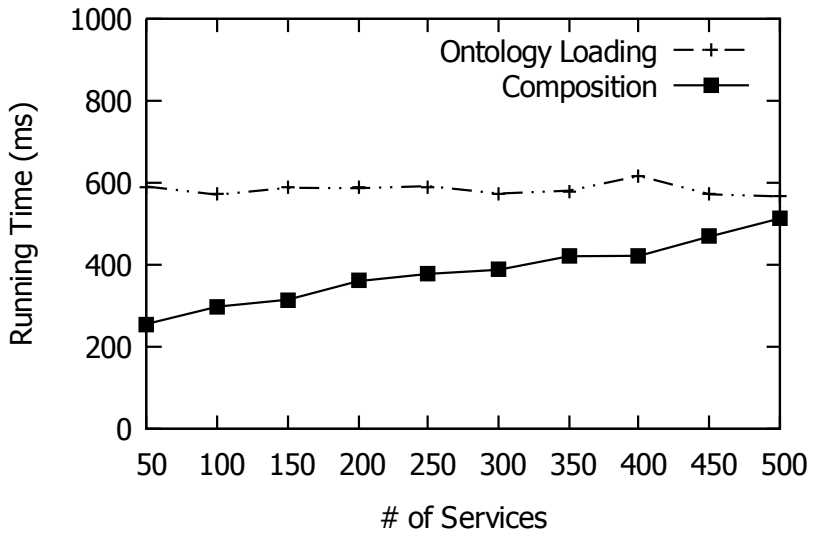


Figure 6.12: Service selection. Composition time of service selection process as the number of devices increases from 500 to 5000.

6.4 Implementation Remarks

The success of these applications shows that the proposed architecture is highly flexible and applicable in different application scenarios proving again that end-to-end IP architecture is an ideal choice for IoT. According to the developers of these applications, the software development experience is very positive and the integration of smart object services into traditional Web applications is easy and transparent. For example, ThingsChat is based on an existing online social network (phpBB Social Network) using PHP programming language, which is one of the most common server-side languages for Web applications. In order to build a new social network (SIoT), developers are required to master PHP language. In the meantime, they are not expected to have knowledge on IoT protocol stack. However, software developers encounter no problem with working on these interface even without any knowledge on the underlying IoT protocols and the architecture only provides the interface (Web API) using open standards.

Chapter 7

Conclusion and Future Work

Contents

7.1 Conclusion

This dissertation has proposed a new architecture of semantic service provisioning for 6LoWPAN including a design of 6LoWPAN internetworking model with regular IPv6 network, a study on networking performance of 6LoWPAN, algorithms and mechanisms for service provisioning, and two innovative proof-of-concept IoT applications on Web illustrating the integration of the proposed architecture in different application domains. The design and study on networking performance of the 6LoWPAN has shown that end-to-end IP communication is possible for real-life deployment of smart objects. The results suggest that the IP-based IoT protocol stack can be used for even with highly resource-constrained such as sensor nodes with a few Kb of memory. Operating systems such as Contiki OS are providing effective platforms to enable the communication of smart objects. The proposed service provisioning architecture presents a secure, scalable, and reliable method to power IoT applications on Web. The architecture was verified by two IoT applications of ThingsChat and SamBAS on Social IoT and Building Automation domains. For each domain, we evaluated the implementation empirically by means of several prototypes and applications and on different environments: the IoT testbed consisting of MTM-CM5000-MSP TelosB sensor nodes, the Contiki Cooja simulator, and DPthWSim – the open-source DPWS simulator developed within this research. Overall, the results demonstrate that the proposed semantic service provisioning architecture can cope with several challenges and enhance the experience for the development of IoT applications on Web.

The work has profound impact on two large-scale European projects: ITEA2 Web of Objects (WoO) [1] and ITEA2 Social Internet of Things - Applications by and for the Crowd (SiTAC) [2]. The WoO project addressed specific issues relating to the increasing integration of Internet-connected devices in existing business applications, proposing a modular solution kit to enable the development of industrial and consumer applications with smart objects as actors, across multiple layers from objects to Web-based user applications. The SiTAC project exploits the social networking paradigm in order to facilitate and unify interactions both between people and devices and between devices. It provides a distributed framework for enabling the Web-based service representation of smart spaces and the objects they include.

7.2 Future Work

The proposed architecture with its profound impact on both academia and industry is a starting point for several future directions for IoT research. In this section, we look at some of the future work directly extending the results from this dissertation.

Service Composition

The question in IoT is not only how to make smart objects be able to communicate over the Internet through provisioning, but also how smart objects services can be used in multiple application in serendipitous ways to create new and creative applications. To answer this, we can be use service composition, one of the core principles of the Service-Oriented Architecture. Advanced functionalities can then be created by combining a set of atomic services in the form of composite services. These composite services can be used in different scenarios to meet various user requirements. The true value of the IoT and new opportunities to create a smarter world will become apparent when data and events from an increasing number of smart objects can be easily and dynamically composed to create novel applications. Service composition has been extensively studied in the context of Web services and business processes [78]. A number of standards have been developed and are being used in real-world deployments to support the service composition. However, the characteristics of IoT systems, such as resource-constraints and data/event-driven devices render some of the techniques devised for traditional Web service composition inadequate. Therefore, new composition models with respect to new requirements of IoT systems are expected. We continue with the the future full-IP IoT to apply the service composition to further expand the proposed service provisioning architecture to give more innovation on the IoT domain.

[1] http://www.web-of-objects.com/, 2012 - 2015
[2] http://sitac.wp.tem-tsp.eu/, 2012 - 2015

IoT Protocol Stack

The IEEE 802.15.4 has been proven to be an excellent standard for low-power smart objects to carry out end-to-end IP communication. It goes with a set of supported standards such as 6LoWPAN adaptation and uIPv6 implementation. direction that we plan to investigate in coming time is for other link layer technologies rather than IEEE 802.15.4 such as BLE and PLC. IETF 6lo WG is working on several standards related to these links and can provide adaptation protocols for these technology such as four Internet drafts that define the adaptations for IPv6 over BLE (draft-ietf-6lo-btle), DECT Ultra Low Energy (draft-ietf-6lo-dect-ule), MS/TP (master-slave/token-passing) networks (draft-ietf-6lo-6lobac), and G.9969 networks (draft-ietf-6lo-lowpanz). Especially, we focus on the recent update of IEEE to the MAC portion of the IEEE 802.15.4 standard, 802.15.4e TSCH for the communication link. Besides, there are several room to improve the performance of the entire networking protocol stack including routing protocols for 6LoWPANs and other efficient messaging protocol for applications layers such as XMPP and AMQP.

Smart Grid

The proposed architecture also one of the core communication technology of the new European project in the domain of smart energy management: Future Unified System for Energy and Information Technology [3] (FUSE-IT). The project has just started by the time of this manuscript. We are planing to to extend the proposed architecture to the smart grid applications with a large-scale testbed in Barcelona city. The aim of the project is to develop a smart secured building system, incorporating secure shared sensors, actuators and devices strongly interconnected through not only information networks but trusted energy networks, including a core building data processing & analysis module, a smart unified building management interface, and a full security dashboard.

[3] http://www.itea2-fuse-it.com/, 2014 - 2017

Appendix A

DPWSim: A DPWS Simulator

Contents

DPWSim is a cross-platform simulator of the DPWS standard. It supports the development of IoT applications using DPWS; DPWSim is based on WS4D-JMEDS [1], the Java implementation of DPWS. The core function of DPWSim is to simulate the DPWS protocols by generating DPWS messages and its communication messaging patterns. It simulates DPWS devices, called *DPWSim devices*, which can be discovered on the network and can communicate with other devices or clients via DPWS protocols. Besides, it also simulates environments where DPWSim devices reside in. DPWSim provides many simulation tools for users to create, manage, store, and load simulations with high flexibility. DPWSim GUI that is based on Java Swing [79] is quite intuitive and easy to use. DPWSim helps developers to prototype, develop, and test DPWS functionalities. The following sub-sections describe the simulation model, core components, functionalities, usage scenarios, and GUI of the simulator.

A.1 Simulation Model

DPWSim simulates the DPWS devices by modeling them as services that operate according to the input of sensing data (e.g., environmental temperature provided by users)

[1] ws4d.org/jmeds/

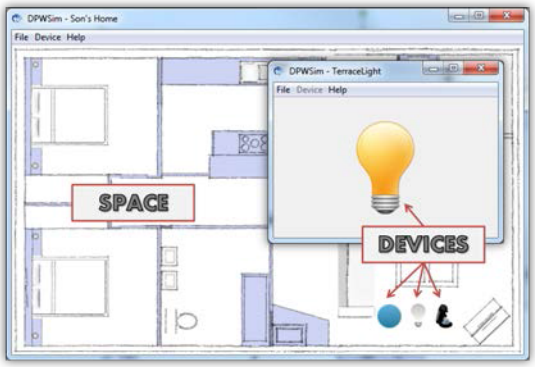


Figure A.1: A home space contains three devices: a generic DPWSim device (blue button), a light bulb, and a coffee maker. A stand-alone device (space with only one device) is a light bulb.

and communication data (e.g., service invocation commands sent from clients). We use a number of hardware including IBM PCs, Raspberry Pi, and Telos B sensor nodes to build real-life devices such as thermostats, motion detectors, and TVs to record how these devices work in several scenarios in order to mimic their behaviors in the simulator. DPWSim builds simulated devices regarding all layers of the TCP/IP networking model [80]. At the network interface layer, the reliable Ethernet link of the host machine is considered to focus on the DPWS protocol messages and mechanisms rather than physical issues (e.g., radio interference). At application layer, each DPWSim device is modeled as a list of services (events and operations) binding to an IP address (internet layer) over UDP (transport layer). Events happen periodically after an interval of time or manually via user interaction; operations are software components receiving input, processing it, and producing output (with its status updated and sent to the invoker). On top of that, device status and outputs of events/operations are modeled as graphical representations. When it comes to modeling and simulating real DPWS systems, DPWSim can support steps involving modeling, designing experiment, and performing analysis of the discrete-event simulation [81].

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

Spaces

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

Devices

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

Operations

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

Events

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

A.2 DPWSim Components

A.3 DPWSim Core Functionalities

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

New Space/Stand-alone Device

There are two options for creating a virtual environment: stand-alone device and space. These functions can be accessed through File menu or keyboard shortcuts. A space is a

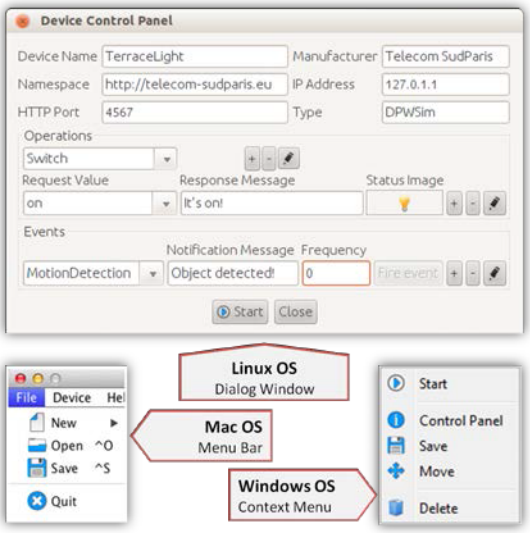


Figure A.2: DPWSim GUI components: a dialog window (Linux OS), a menu bar (Mac OS), and a context menu (Windows OS).

composite environment to host several devices. It is created by using a plan image such as office, home, and airport. A stand-alone device is simply a DPWSim device with a hosted service containing operations and events. This kind of virtual environment can be stored in file and re-used in other virtual environments.

New Device

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

Device Management

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

A.4 Usage Scenarios

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

Scenario one - Product Integrating

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

Scenario two - Product Protyping

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

Scenario three - Resources Sharing

This scenario describes the situation when several teams, at the same time, develop different modules over the same devices. To solve the problem and speed up the development process, a new set of simulated devices is generated by DPWSim to share among developers. The simulation can also be used for demonstration purpose without the loss of the accuracy.

A.5 Graphical User Interface

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

A.6 DPWSim Use Cases

DPWSim has been used in several environments such as DPWS Explorer [2], a Web application, a testbed, and in a number of DPWS studies. The following parts explain each of these experiments on DPWSim and information about the development process.

[2] http://ws4d.org/dpws-explorer/

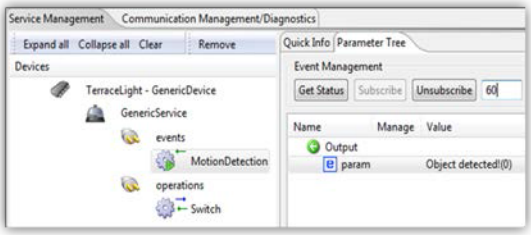


Figure A.3: DPWS Explorer discovers a DPWSim device *TerraceLight* containing an event *MotionDetection* and an operation *Switch*. The green icon next to the *MotionDetection* event indicates that DPWS Explorer is subscribing to the event; once the event occurs, DPWS Explorer will receive the notification,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

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

DPWS Explorer

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

DPWSim Web

DPWSim Web is a small Web application included in the release of DPWSim to illustrate an use case when a Web application interacts with DPWSim devices. It is a Java Web application running on Apache Tomcat application. It can discover available DPWS devices on the network and retrieve their metadata. Following these data, users can invoke device operations to carry out their tasks. Figure A.4 shows DPWSim Web via

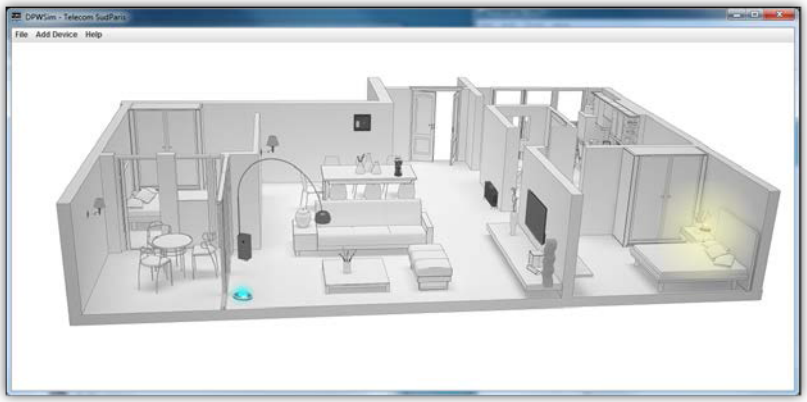


Figure A.5: A virtual home hosting several DPWS devices is designed using DPWSim with the help of a 3D artist (Sa Hoang from École Nationale Supérieure d'Architecture de Paris La Villette - ENSAPLV).

its smartphone interface to invoke an operation of a light bulb device *KitchenLight*. Users can switch the light bulb on or off from the Web interface by clicking on the buttons.

Research Projects

DPWSim has been used within ITEA2 Web of Objects (WoO) project to support the development of an incident management scenario for testing the contextual object collaboration. DPWSim has been used throughout the development to describe the common interface for the cooperation between devices upon the assigned rights and specific rules imposed in the whole system. An example of the home environment created for the project is shown in Figure A.5. The home consists of several DPWS devices such as a TV, lamps, and a coffee maker. With the help of a 3D artist, it provides an elegant simulation using DPWS protocols.

Besides, DPWSim has been thus far used in several IoT studies such as the semantic building automation system [82], social device networking [83], and REST proxy for DPWS [84]. DPWSim is hosted by WoO project and its source code is maintained on a GitHub repository (http://github.com/sonhan/dpwsim.)

Bibliography

[1] J.-P. Vasseur and 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," Cisco, White Paper, 2011.

[3] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, pp. 29–37, May 2001.

[4] S. Cirani, M. Picone, and L. Veltri, "Cosip: a constrained session initiation protocol for the internet of things," in *Advances in Service-Oriented and Cloud Computing*. Springer, 2013, pp. 13–24.

[5] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)," IETF, IETF Internet Draft – work in progress 18, Jun. 2013.

[6] "Devices Profile for Web Services Version 1.1," OASIS, Tech. Rep., Jul. 2009.

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

[8] A. Stanford-Clark and H. L. Truong, "Mqtt for sensor networks (mqtt-sn) protocol specification," IBM, Tech. Rep., Nov. 2013.

[9] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, Nov 2006.

[10] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *IEEE Internet Computing*, vol. 9, no. 1, pp. 75–81, 2005.

[11] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[12] O. Mazhelis, E. Luoma, and H. Warma, "Defining an internet-of-things ecosystem," in *Internet of Things, Smart Spaces, and Next Generation Networking*, ser. Lecture Notes in Computer Science, S. Andreev, S. Balandin, and Y. Koucheryavy, Eds. Springer Berlin Heidelberg, 2012, vol. 7469, pp. 1–14.

[13] R. V. Prasad, C. Sarkar, V. S. Rao, A. R. Biswas, and I. Niemegeers, "Opportunistic service provisioning in the future internet using cognitive service approximation," in *28th WWRF Meeting, Athens, Greece*, 2012.

[14] B. Mandler, F. Antonelli, R. Kleinfeld, C. Pedrinaci, D. Carrera, A. Gugliotta, D. Schreckling, I. Carreras, D. Raggett, M. Pous, C. Villares, and 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)*, Mar. 2013, pp. 1217–1222.

[15] S. Lee and I. Chong, "User-centric intelligence provisioning in web-of-objects based iot service," in *2013 International Conference on ICT Convergence (ICTC)*, Oct. 2013, pp. 44–49.

[16] S. Gagnon and 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.

[17] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services." *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 223–235, Jul. 2010.

[18] S. Li, G. Oikonomou, T. Tryfonas, T. Chen, and L. D. Xu, "A distributed consensus algorithm for decision making in service-oriented internet of things," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1461–1468, May 2014.

[19] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002.

[20] "Programmableweb," ProgrammableWeb. [Online]. Available: http://www.programmableweb.com/

[21] D. Guinard, V. Trifa, T. Pham, and 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, Jun. 2009.

[22] D. Guinard and 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.

[23] D. Guinard, "Mashing up your web-enabled home," in *Current Trends in Web Engineering*. Springer, 2010, pp. 442–446.

[24] D. Guinard, M. Mueller, and J. Pasquier-Rocha, "Giving rfid a rest: building a web-enabled epcis," in *Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[25] D. Guinard, C. Floerkemeier, and 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.

[26] D. Guinard, M. Mueller, and V. Trifa, "Restifying real-world systems: A practical case study in rfid," in *REST: From Research to Practice*. Springer, 2011, pp. 359–379.

[27] D. Zhiquan, Y. Nan, C. Bo, and C. Junliang, "Data mashup in the internet of things," in *2011 International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 2. IEEE, 2011, pp. 948–952.

[28] E. Avilés-López and J. A. García-Macías, "Mashing up the internet of things: a framework for smart environments," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, no. 1, pp. 1–11, 2012.

[29] K. Kenda, C. Fortuna, A. Moraru, D. Mladenić, B. Fortuna, and M. Grobelnik, "Mashups for the web of things," in *Semantic Mashups*. Springer, 2013, pp. 145–169.

[30] "BUGswarm," BUGswarm. [Online]. Available: http://developer.bugswarm.net/

[31] "Carriots," Carriots. [Online]. Available: https://www.carriots.com/

[32] "Evrythng," EVRYTHNG. [Online]. Available: http://www.evrythng.com/

[33] "Grovestreams," GroveStreams. [Online]. Available: https://grovestreams.com/

[34] "Nimbits," Nimbits. [Online]. Available: http://www.nimbits.com/

[35] "Open.Sen.se," Open.Sen.se. [Online]. Available: http://open.sen.se/

[36] "Paraimpu," Paraimpu. [Online]. Available: http://paraimpu.crs4.it/

[37] "Sensinode," NanoService. [Online]. Available: http://www.sensinode.com/

[38] "SensorCloud," SensorCloud. [Online]. Available: http://www.sensorcloud.com/

[39] "Thinkspeak," ThingSpeak Community. [Online]. Available: https://www.thingspeak.com/

[40] "Thingworx," ThingWorx. [Online]. Available: http://www.thingworx.com/

[41] "Xively," Xively (Pachube). [Online]. Available: https://xively.com/

[42] "Yaler," Yaler. [Online]. Available: https://yaler.net/

[43] "RDF Primer," W3C, W3C Recommendation, Feb. 2004.

[44] "OWL 2 web ontology language document overview," W3C, W3C Recommendation, Oct. 2009.

[45] 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 Semantics: Science, Services and Agents on the World Wide Web*, vol. 17, pp. 25–32, 2012.

[46] A. Gangemi, "Ontology design patterns for semantic web content," in *The Semantic Web - ISWC 2005*, ser. Lecture Notes in Computer Science, Y. Gil, E. Motta, V. Benjamins, and M. Musen, Eds. Springer Berlin Heidelberg, 2005, vol. 3729, pp. 262–276.

[47] P. Barnaghi, M. Presser, and K. Moessner, "Publishing linked sensor data," in *CEUR Workshop Proceedings: Proceedings of the 3rd International Workshop on Semantic Sensor Networks (SSN)*, vol. 668, 2010.

[48] "Sparql 1.1 query language," W3C, W3C Recommendation, Mar. 2013.

[49] 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, and R. Richardson, "Spitfire: toward a semantic web of things," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 40–48, 2011.

[50] H. Hasemann, A. Kroller, and 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.

[51] D. Bimschas, H. Hasemann, M. Hauswirth, M. Karnstedt, O. Kleine, A. Kröller, M. Leggieri, R. Mietz, A. Passant, D. Pfisterer, K. Römer, and C. Truong, "Semantic-service provisioning for the internet of things," *Electronic Communications of the EASST*, vol. 37, 2011.

[52] S. De, P. Barnaghi, M. Bauer, and S. Meissner, "Service modelling for the internet of things," in *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sep. 2011, pp. 949–955.

[53] 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)*, Dec. 2013, pp. 323–327.

[54] A. Dunkels, "The contikimac radio duty cycling protocol," 2011.

[55] A. Dunkels, F. Osterlind, N. Tsiftes, and 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.

[56] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power coap for contiki," in *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011)*, Valencia, Spain, Oct. 2011.

[57] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: Scalable cloud services for the internet of things with coap," in *Proceedings of the 4th International Conference on the Internet of Things (IoT 2014)*, 2014.

[58] "Web Service for Devices Initiative," Web Service for Devices Initiative. [Online]. Available: http://www.ws4d.org/

[59] S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone, and L. Veltri, "A scalable and self-configuring architecture for service discovery in the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 508–521, 2014.

[60] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[61] H. Chen, Z. Wu, and P. Cudré-Mauroux, "Semantic web meets computational intelligence: State of the art and perspectives [review article]," *IEEE Computational Intelligence Magazine*, vol. 7, no. 2, pp. 67–74, 2012.

[62] T. Berners-Lee and D. Connolly, "Notation3 (N3): A readable rdf syntax," W3C, W3C Team Submission, Mar. 2011.

[63] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Proceedings 2006 31st IEEE Conference on Local Computer Networks*, Nov 2006, pp. 641–648.

[64] S. N. Han, G. Lee, N. Crespi, N. Luong, K. Heo, M. Brut, and P. Gatellier, "Dpwsim: A devices profile for web services (dpws) simulator," *IEEE Internet of Things Journal*, vol. 2, no. 3, pp. 221–229, Jun. 2015.

[65] C. Lerche, N. Laum, G. Moritz, E. Zeeb, F. Golatowski, and D. Timmermann, "Implementing powerful web services for highly resource-constrained devices," in *2011 IEEE International Conference on Pervasive Computing and Communications Workshops*, 2011, pp. 332–335.

[66] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski, "Encoding and compression for the devices profile for web services," in *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2010, pp. 514–519.

[67] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, "Beyond 6lowpan: Web services in wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 1795–1805, Nov. 2013.

[68] I. Samaras, G. Hassapis, and J. Gialelis, "A modified DPWS protocol stack for 6lowpan-based wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 209–217, Feb. 2013.

[69] X. Yang and X. Zhi, "Dynamic deployment of embedded services for DPWS-enabled devices," in *2012 Int. Conf. on Computing, Measurement, Control and Sensor Network (CMCSN)*, 2012, pp. 302–306.

[70] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A real-time service-oriented architecture for industrial automation," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 267–277, 2009.

[71] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, and V. Trifa, "SOA-based integration of the internet of things in enterprise services," in *IEEE International Conference on Web Services (ICWS 2009)*, 2009, pp. 968–975.

[72] L. Atzori, A. Iera, G. Morabito, and M. Nitti, "The social internet of things (SIoT) - when social networks meet the internet of things: Concept, architecture and network characterization," *Computer Networks*, vol. 56, no. 16, pp. 3594–3608, 2012.

[73] D. Dietrich, D. Bruckner, G. Zucker, and P. Palensky, "Communication and computation in buildings: A short introduction and overview," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3577–3584, Nov. 2010.

[74] EN 14908-x (1-6), *Open Data Communication in Building Automation, Controls and Building Management - Control Network Protocol*. European Committee for Standardization, Brussels, Belgium, 2005-2010.

[75] ISO 16484-5, *Building automation and control systems – Part 5: Data communication protocol*. International Organization for Standardization, Geneva, Switzerland, Jul. 2012.

[76] ISO/IEC 14543-4-1, *Information technology – Home electronic system (HES) architecture – Part 4-1: Communication layers – Application layer for network enhanced control devices of HES Class 1*. International Organization for Standardization, Geneva, Switzerland, Jun. 2008.

[77] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, ser. WMCSA '94, Washington, DC, USA, 1994, pp. 85–90.

[78] S. Dustdar and W. Schreiner, "A survey on web services composition," *Intl. Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1–30, 2005.

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

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

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

[82] S. N. Han, G. Lee, and N. Crespi, "Semantic context-aware service composition for building automation system," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 752–761, Feb. 2014.

[83] D. Hussein, S. N. Han, X. Han, G. M. Lee, and N. Crespi, "A framework for social device networking," in *2013 IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, May 2013, pp. 356–360.

[84] S. N. Han, S. Park, G. M. Lee, and N. Crespi, "Extending the device profile for web services (dpws) standard using a rest proxy," *IEEE Internet Computing*, vol. 19, no. 1, pp. 10–17, Jan. 2015.

Acronym

6EdR	6LoWPAN Edge Router
6LoWPAN	IPv6 Low-power Wireless Personal Area Network
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BAS	Building Automation System
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CPDL	Composition Plan Description Language
CSMA	Carrier Sense Multiple Access
DECT	Digital Enhanced Cordless Telecommunications
DPWS	Devices Profile for Web Services
DTLS	Datagram Transport Layer Security
ETSI	European Telecommunications Standards Institute
FP	European Framework Programme
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv6	Internet Protocol version 6
IRTF	Internet Research Task Force
ITEA	Information Technology for European Advancement
LLN	Low-power and Lossy Network
LoWPAN	Low-power Wireless Personal Area Network
LPM	Low-power Mode
LQI	Link Quality Indication
MQTT	Message Queue Telemetry Transport
N3	Notation3
NDP	Neighbor Discovery Protocol
NLP	Natural Language Processing

ODP	Ontology Design Pattern
OSN	Online Social Network
OWL	Web Ontology Language
RDF	Resource Description Framework
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RPL	Low-Power and Lossy Networks
RSSI	Received Signal Strength
RTT	Round-trip Time
RX	Reception
SIoT	Social Internet of Things
SLIP	Serial Line Internet Protocol
SNMP	Simple Network Management Protocol
SOA	Service-Oriented Architecture
SSN	Semantic Sensor Network
TCP	Transmission Control Protocol
TSCH	Timeslotted Channel Hopping
TUN	Network TUNnel
TX	Transmission
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
W3C	Word Wide Web Consortium
WSDL	Web Service Description Language
WUI	Web-based User Interface
XMPP	Extensible Messaging and Presence Protocol

Publications

1. **S. N. Han**, G. M. Lee and N. Crespi, "Towards Automated Service Composition Using Policy Ontology in Building Automation System," *2012 IEEE 9th International Conference on Services Computing (SCC)*, pp. 685-686, June 2012.

2. **S. N. Han**, G. M. Lee and N. Crespi, "Context-aware Service Composition Framework in Web-enabled Building Automation System," *2012 16th Intl. Conf. on Intelligence in Next Generation Networks (ICIN)*, pp. 128-133, October 2012.

3. D. Hussein, **S. N. Han**, X. Han, G. M. Lee and N. Crespi, "A Framework for Social Device Networking," *2013 IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 356-360, May 2013.

4. **S. N. Han**, G. Lee and N. Crespi, "Semantic Context-Aware Service Composition for Building Automation System," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 752-761, February 2014.

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

6. A. Ortiz, D. Hussein, S. Park, **S. N. Han** and N. Crespi, "The Cluster Between Internet of Things and Social Networks: Review and Research Challenges," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 206-215, June 2014.

7. **S. N. Han**, S. Park, G. M. Lee and N. Crespi, "Extending the Device Profile for Web Services (DPWS) Standard using a REST Proxy," *IEEE Internet Computing*, vol. 19, no. 1, pp. 10-17, January 2015.

8. D. Hussein, S. Park, **S. N. Han** and N. Crespi, "Dynamic Social Structure of Things: A Contextual Approach in CPSS," *IEEE Internet Computing*, vol. 19, no. 3, pp. 12-20, May 2015.

9. **S. N. Han**, G. Lee, N. Crespi, N. Luong, K. Heo, M. Brut and P. Gatellier, "DPWSim: A Devices Profile for Web Services (DPWS) Simulator," *IEEE Internet of Things Journal*, vol. 2, no. 3, pp. 221-229, June 2015.