Exploring Major Architectural Aspects of the Web of Things

Iván Corredor Pérez and Ana M. Bernardos Barbolla

Abstract A number of technological research lines around the feasibility and applicability of the Internet of Things are currently under discussion. Some of those research issues are dealing with the deployment of friendly smart spaces made of smart objects which are digitally augmented by means of RFID tags or embedded wireless sensor and actuator devices. The advantages of deploying heterogeneous ecosystems of smart things through web technologies have lead to the so called Web of Things paradigm, that relays on the Internet of Things principles. The opportunity of developing to develop new services and applications has driven research towards proposals that integrate isolate islands of networks based on the Internet of Things into the Web. In this context, the contribution of this chapter is twofold. On the one hand, it aims at detailing a Web of Things Open Platform for deploying and integrating smart things networks into the Web. The purpose of this platform is to expose the functionalities of sensor and actuator devices and their involved information model as a set of RESTful services to be retrieved from the Web. On the other hand,the Chapter describes a Resource-Oriented and Ontology-Driven development methodology, which enables easier the development and deployment of smart spaces. The feasibility of this holistic approach is demonstrated through a fully-implemented case study.

1 Introduction: Towards Smart Spaces for Human Beings

The first decade of XXI century have been very prolific in research focusing on making real Weiser's vision [1] on Pervasive Computing. Those early works led to

I. Corredor Pérez (✉) · A. M. Bernardos Barbolla
Grupo de Procesado de Datos y Simulación, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, Madrid, Spain
e-mail: ivan.corredor@grpss.ssr.upm.es

A. M. Bernardos Barbolla
e-mail: abernardos@grpss.ssr.upm.es

S. C. Mukhopadhyay (ed.), *Internet of Things*, Smart Sensors, Measurement and Instrumentation 9, DOI: 10.1007/978-3-319-04223-7_2,

a number of exciting research results which have contributed to get more and more tiny devices connected to each other and to the Internet. Different technologies of very diverse fields have been involved on enhancing the communication capabilities of smart objects, e.g. RFID/NFC tags, wireless sensor and actuator embedded devices, low-energy communication protocols, etc. All these technologies are subsumed behind a concept that encompasses a lot of research subjects: the Internet of Things, which was conceived with the promising idea of improving the connectivity of constrained and embedded devices to the Internet.

Originally, the application of Internet of Things (IoT) was focused on managing information through RFID tags [2, 3] such as goods tracking, management of everyday objects, automatic payments in markets and military applications. The evolution of Micro-Electro-Mechanical (MEMS) technologies have fostered the evolution of the Internet of Things by means of the miniaturization of hardware components, i.e. wireless transceivers, sensors, actuators, microcontrollers, etc. Those achievements have motivated many proposals for improving the software of embedded devices in several aspects, e.g. protocols, operating systems, architectures, etc., which have been essential in order to optimize Machine-to-Machine (M2M) interactions, providing a much more proactive character to IoT-based applications. In this sense, current paradigmatic IoT applications are designed to create smart spaces which are made of thousands with of smart things autonomous capabilities.

While the implementation of the Internet of Things is exponentially growing, the Internet of Services is already a reality. One the pillars of the Future Internet is focused on how to integrate *real-world* services provided by the Internet of Things into mash-ups of traditional services. This new Internet order will be addressed under two viewpoints specifying two interaction dimensions, *horizontal* and *vertical* approaches, involving Machine-to-Machine and Human-to-Machine (H2M) interaction, respectively. The second one will also involve interactions among Machines and processes which could be running on external entities connected to other networks. Recently, concerns have raised on how to optimize the management of such vertical interactions which will generate enormous amounts of information in terms of *heterogeneity*, *reusability*, *scalability* or *accessibility*.

The major problem that hinders to deal with those issues is the wide use of *ad hoc* and monolithic designs that difficult the convergence of different applications through a generic and open solution. To make feasible and reusable IoT-based technologies, an open infrastructure is needed, which allows accessing heterogeneous sensors and actuators devices following the Web and the Internet standards. Future applications for IoT-based networks will require a significant improvement in reusability of deployment resources in order to be available for many high-level entities, e.g. smart phones apps, Web sites, expert systems, etc.

Solid proposals to deal with the above mentioned challenges have arisen behind the *open platform* concept. This concept involves a set of specific issues that can fairly facilitate rapid prototyping and deployment of large IoT-based networks. Although its major feature is the openness of interfaces (i.e. well documented and public interfaces), it involves other ideas that facilitate reusability, extension and reimplementation of functionalities for which the platform has not been initially designed.

It is important to highlight that this concept does not necessarily imply open source results, but a set of public operations that are specified in an API (Application Programming Interface). Multiple implementations of a same API can be offered to be used for different technological platforms (e.g. different mobile operating systems) or programming languages (e.g. JAVA, C, C++, etc.). A new trend in open platforms is to provide some interesting tools to integrate IoT networks with well-known Web technologies by means of a category of cloud computing services: the Platform as a Service (PaaS).

The PaaS model allows service consumers to easily develop and deploy value-added services by using workbench tools and/or libraries provided by a vendor; this helps the costumer to spend few resources for management and maintenance of large infrastructures of hardware and software. PaaS-based solutions have commonalties and specific characteristics that differentiate each other. Common features are, among other, the provisioning of a deployment environment, monitoring dashboard, as well as communication mechanisms to exchange information between the platform and the client. Computing resources transferred to client accounts usually depends on the scalability and Quality of Service required by the applications using the platforms.

The conjunction of the open platform with the PaaS paradigm has gained popularity among researchers who are focused on the integration of the IoT into the Web. PaaS is strongly related to the Web of Things (WoT). Generally speaking, the WoT tries to bridge the gap between embedded devices and the Web in order to integrate very different and isolated IoT networks with users environments through Web technologies, as well as with each other. In order to reach the major objective of the WoT, it is needed to adapt traditional Web technologies to create open interfaces, representation formats, discovery and communication protocols, or information retrieving mechanisms that facilitate integrating real-world entities including on the IoT. For instance, a common aspect of these approaches is the use of the Representational State Transfer (REST) architectural style [4] and the JavaScript Object Notation (JSON) for representation formats. In REST-based approaches, real world entities are identified by Unique Resource Identifiers (URIs) and their associated information can be accessed by means of invoking HTTP methods. These features (among others) make easier the development of IoT-based mashups aiming at gathering, processing or aggregating massive amounts of data from sensors and other virtual data sources. The design of approaches under these precepts should motivate the emerging of open platforms focused on abstracting every concept related to IoT in order to build the pillars for the future WoT.

1.1 Relevant Contributions of this Research in the Fields of the IoT and the WoT

Obviously, the emerging of novel approaches in the WoT needs to tackle with new and efficient implementations that move theoretical proposals into practical solutions in a short term of time. This Chapter compiles our recent research work [5–7], focused on providing a holistic solution that facilitates the design, development and deployment

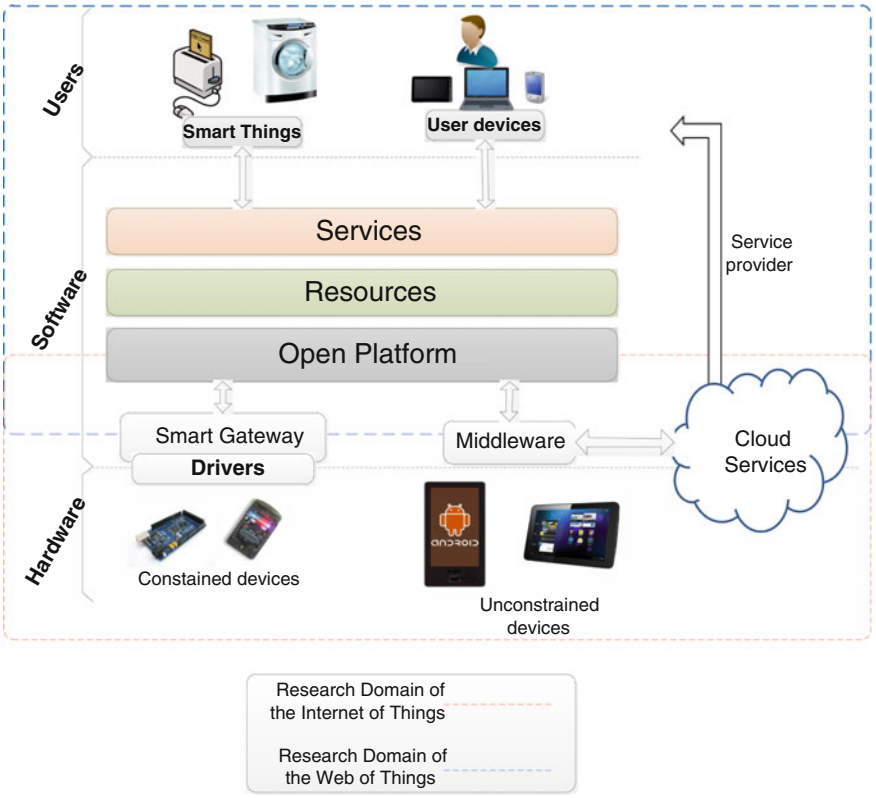

Fig. 1 The proposed reference architecture for the IoT and WoT research domains

of enriched smart environments according to the IoT and WoT paradigms, specially on the latter.

In order to efficiently address this issue, we defined a reference architecture which establishes a principle guide for our design decisions. This reference architecture gives a general perspective of the major areas to be developed in the research domains of the IoT and the WoT, i.e. a set of unified concepts and their relationships. It is important to highlight that a reference model has to be independent of issues as standards, technologies or implementations [8]. The proposed reference architecture for the IoT and the WoT research domains is shown in Fig. 1.

The architecture in Fig. 1 is defined according to three major layers. The lowest layer is composed of the hardware ecosystem. This layer is classified into two groups of devices: (i) *constrained devices*, i.e. those devices that have limited features and, thus, they have to outsource some processes to be run by external devices (usually Smart Gateways) which expose their functionalities to clients; (ii) *unconstrained devices*, i.e. those devices that have enough resources to run the necessary processes or middleware components to provide directly functionalities to clients through a

platform or third part cloud services. Concepts managed in this layer exclusively belongs to the Internet of Things domain. The next layer is the software layer which has to be supported by an open platform. The role of this platform is important since it has to provide a mechanism to set up the functionalities of the underlying hardware as resources (e.g. sensor, actuator, processing capabilities, etc.) and finally, to compose and orchestrate them in order to build simple or complex services. The software layer aims at implementing the research domain of the Web of Things, overlapping with the Internet of Things domain in some low-level issues as protocols, drivers or communication standards. Finally, the user layer is composed of clients which are going to consume the services exposed by the open platform. Those clients can be human using user devices (e.g. smart phones, tablets, laptops, etc.) or machine users as smart objects that might request services to perform high or low level tasks (e.g. to turn on an ambient light or analyze the capabilities of peer entities to compose services for making complex smart environments).

As commented previously, the proposal described here is focused on achieving a holistic solution for the IoT and the WoT. The research works that led us to our approach can be summarized into the following points:

- *The Web of Things Open Platform (WoTOP)*: The first research contribution included in this Chapter describes a novel open platform, called the Web of Things Open Platform. WoTOP allows interconnecting smart objects, business process and users. This platform is designed on the well-known Web technologies which enable vertical interaction. From a general perspective, our proposal addresses the major challenges to achieve a consistent business method for deploying services over an underlying IoT ecosystem consisting of networks of embedded sensors and actuators devices.
- *Methodology for Rapid Deployment of Smart Spaces*: Currently, people who deal with the WoT approaches need to work with specific software and hardware platforms as well as web technologies. The Do-it-Yourself movement is motivating the design of *creation environments* to enable no-technicians in the development and deployment of WoT-based smart spaces composed of embedded sensor and actuator devices and distributed business logic. We have designed a development methodology in order to alleviate the development process of complex and large smart spaces; our objective has been twofold: (i) to facilitate the integration of sensing and actuating functionalities into everyday objects and (ii) to enable a heterogeneous ecosystem of devices to integrate into the Web. The result has been the Resource-Oriented and Ontology-Driven Development (ROOD) methodology which is based on the Model Driven Architecture (MDA). This methodology aims at enabling the development of smart spaces through a set of modeling tools and semantic technologies that support the definition of the smart space and the automatic code generation for a specific hardware platform.
- *Prototyping of Real Smart Spaces*: Prototyping is an essential stage that allows proving, validating and consolidating theoretical concepts through their application in real deployments. For that purpose, we designed and deployed a smart shop which involved a number of different entities to offer enriched services to

hypothetical users. This smart space was used to validate most of the concepts and tools proposed in this Chapter.

The rest of the Chapter is organized as follows. Section 2 discusses open research challenges related to the IoT and the WoT, highlighting proposals that address the problem from both industry and academical perspectives. Section 3 and 4 compiles the major research contributions of this Chapter, i.e. the Web of Things Open Platform and the Resource-Oriented and Ontology-Driven, respectively. Section 5 describes the case study deployed for validating our contributions that were theoretically explained in previous sections. Conclusions of the Chapter and a outlook to research challenges are included in Sect. 6.

2 Open Research Challenges and Background

2.1 Summary of Research Challenges

As was commented in the previous Section the convergence of IoT, WoT and PaaS present different challenges when integrating constrained networks into the Internet by using internetworking techniques and middleware architectures. These challenges can be classified as following:

A. *Integration of heterogeneous IoT-based networks*: Usually IoT-based networks need to link ecosystems of technologically heterogeneous devices. This diversity makes difficult the integration of things into the Web. Some IoT platforms as Cosm [9] or Paraimpu [10] have designed their own strategics to integrate data streams from open source devices as Arduino [11]. Other platforms, as Sensinode [12] or SmartThings [13], have created their own hardware products which are compatible with their respective IoT platforms. The most appropriate solutions would be those offering high-level mechanisms to integrate additional types of embedded devices, based on both proprietary and open source hardware. The objective is to create a larger scene attracting developers specialized in most popular sensor and actuator embedded technologies (e.g. Crossbow, Zephyr, Sun SPOT, Arduino, etc.). To this aim, it is necessary to simplify the mechanism to integrate things as much as possible by hiding to developers the technical aspects of the platform.
B. *Adaptive scalability*: Typically, WoT platforms are provided through cloud services with high capabilities to manage milliards, even thousands, of things per user. This kind of services can scale a lot and they are only limited to the available resources of the servers supporting them. Apart from this, WoT platforms for homes and small businesses are being more and more popular since they allow users to take full control over security issues [13]. Thus, this is an important property for services managing sensitive information, e.g. those based on eHealth or home security. The latter type of platforms is deployed to work at smaller scale using *hubs* (also called *Smart Gateways*) through which any necessary devices

can be connected to a smart space. However, these Smart Gateways are limited in hardware resources, so their scalability is also limited. Thus, it is needed to explore a trade-off for small and medium scale deployments (e.g. in factors as number of deployed sensors/actuators or rate of generated events) in order to optimize the performance of these Smart Gateways.

C. *Data filtering and alert notification*: When managing a plethora of things generating big amounts of data it is necessary to implement mechanisms to filter raw data in order to optimize the processes that will store and dispatch information to the clients. Simple filtering mechanisms are typically defined by means of first-order rule engines or data source combination (e.g. aggregation or fusion of data sources) by applying functions to aggregate them. Data filtering techniques are very often related to the detection of events. Other approaches take advantage of Semantic Web techniques in order to disseminate data by performing complex inferences [14].

   Ideally, the design of open platforms should include an event-driven subsystem based on the *publisher/subscriber* communication paradigm. For example, Cosm provides an unsophisticated filtering mechanism based on simple rules in order to detect events and send notifications to a Twitter account. This functionality should be implemented through generic event-driven mechanisms in order to dispatch notifications to any subscribed application or service.

D. *Sharing of knowledge resources*: Sharing knowledge resources is the cornerstone of collaborative networks with different objectives, e.g. scientific, technological or social. Consequently, this aspect has to be taken into account when designing WoT open platforms. These platforms should provide mechanisms to share resources in collaborative networks or just to distribute such resources among users that may be interested in them.

   A very accepted proposal to offer resources sharing is the called *API key* [15]. These API keys allow accessing specific resources (e.g. sensor streams, actuator control or monitoring tools) only for trusted users. Additionally, those resources can be available both permanently and for a period of time. A recent proposal [16] tries to take advantage of social networks (e.g. Twitter, Facebook, LinkedIn, etc.) and their open Web APIs in order to share smart things among trusted users. This proposal is based on an authentication proxy, called Social Access Controller (SAC), which allows publishing and sharing smart things among user groups registered in some social network with specific credentials.

E. *Development and deployment of services*: The own nature of an open platform is to provide a public Application Programming Interface (API) that allows quick prototyping of complex applications by creating mashups consisting of a variety of both sensor data sources and actuator control points. Ideally, open platforms have to provide natively common services for many applications (e.g. data visualization or localization services). However, an enriched API should offer a set of development tools to application developers to create their own additional applications for processing and visualizing data. Blackstock et al. [17] propose WoTKit. This is an holistic solution, that offers a WoT architecture and a graphical toolkit allowing rapid development of IoT mashups by means of well known web technologies.

New trends are focusing on mapping APIs for IoT services into RESTful services in order to take advantage of Web techniques. Simon et al. [18] propose a toolkit that facilitates the integration of IoT-based smart things into the Web by defining RESTful services, which are mapped over embedded sensor functionalities. Zhenyu Wu et al. [19] work on the concept of Gateway as a Service (GaaS) in order to design a framework which seamlessly integrates third party embedded devices into the Internet by modeling RESTful services and mapping Web Services over them. Ideally, a development methodology can be offered in order to facilitate to developers, even those with no technical skills, to develop advance application and services from mashups of models of IoT-based environments. For example, a development methodology based on the Model Driven Engineering has already proposed by authors of this work [5]. The methodology offers development tools for rapid prototyping of WoT-based complex systems.

After the previous review of open challenges for IoT and WoT platforms, next Section provides an overview of the initiatives and projects that are coordinating their efforts to gather and conduct research results in different fields to reach standards and recommendations.

2.2 *Background: Standardization Initiatives and Other Industrial and Academical Projects*

From five years to date, many approaches have emerged with the aim at implementing technology agnostic solutions related to IoT or M2M research fields. This fact has raised the need of managing hardware and software interoperability. Many companies behind this IoT or M2M services belong to IPSO Alliance that was launched in 2008 as a non-profit organization to coordinate an initiative to establish the IP as standard network protocol for connecting smart things by means of carrying out common marketing efforts. Currently, the IPSO Alliance is composed of around 50 companies, including Google, Cisco, Ericsson or Alcatel. Additionally, there are successful commercial solutions that have taken advantage of the standards and recommendations specified by the partners of IPSO Alliance, for instance ThingWorx [20], AirVantange [21], Axeda [22], or SmartThing [13].

Aside from commercial approaches, academia is actively contributing to IoT-related research. In fact, the concept of Internet of Things was conceived in the MIT Auto-ID center. Auto-ID labs, which are spread around the world in seven countries (US, UK, Switzerland, China, Korea, Australia and Japan), work on architecting the IoT together with EPCglobal [23]. In Europe, part of the academic research work focused on IoT has been developed within projects funded by the EU Comission (under the Framework Programmes, ITEA or Artemis initiatives). Many of these projects are coordinated by the European Research Cluster on the Internet of Things (IERC) [24]. The IERC was founded within the FP7 in order to manage the wide range of results and applications from the European projects on IoT, and to coordinate

the on-going activities facilitating knowledge sharing, not only at European level but also at a global level. More than 30 EU-funded projects (some are still in execution) have been involved on the IERC initiative, among them AMI-4-SME, SENSEI, IoT-i, IoT-a or DiYSE [25].

Recently, some initiatives based on web-centric open platforms have come up contributing with interesting mechanisms intended to integrate isolated IoT-based networks into the Internet through cloud services. For instance, Cosm [9], EVRYTHNG [26], Paraimpu [10] or ThingSpeak [27] provide services to integrate any sensor and actuator device, or tagged object (e.g. with a RFID tag or QR code) into each other and into the Internet. These platforms usually provide RESTful interfaces that enable access to the information gathered from the mentioned sources through URIs that identifies each data stream, usually called *feeds* or *channels*. The most usual set of services provided by these platforms comprise processing, sharing, mashuping and visualization of sensors and actuators. Each one of these platforms are characterized by different features that contributes to establish the pillars of the current and the future Web of Things. However, any of the mentioned projects have proposed an integral solution which tackle the management of the Internet of Things, providing adaptive communication and information models for different business cases, as well as tools and methodologies to develop and deploy rapidly services involving smart things.

3 The Web of Thing Open Platform

As it was commented in Sect. 1, our contribution merges several research concepts (IoT, WoT, PaaS and development methodologies) which are certainly interrelated among them. We have made the most of this situation in order to reach a holistic solution to build modern smart spaces, enabling the integration of different interaction methods and based on the paradigm of Pervasive Computing (e.g. lightweight protocols, context-aware systems or localization-aware services). The latter is changing the way in which we currently understand Internet. The Future Internet will provide a wide range of *real-world* services supported by networked smart things setting up complex smart spaces on a global scale.

Our first approach aims at providing an open platform to provide developers with rapid and easy prototyping tools to plug smart things into the Web. This proposal is called the *Web of Things Open Platform* (WoTOP). The architectural model of the WoTOP is based on five design principles taking into account the research challenges mentioned in Sect. 2.1:

(i) Integration of isolated IoT-based networks composed of embedded devices implementing specific stack protocols that are not compatible with IP/TCP (*Challenge A*). The integration process is supported by a set of synchronized Smart Gateways which implements both IP/TCP stack and specific protocols depending on the embedded devices plugged to it.

(ii) Discovery, configuration and management of heterogeneous ecosystems of smart things, i.e. things equipped with sets of sensors and actuators as well as processing capabilities that are capable of performing inferences from its own contextual information (*Challenge A and B*). All those features are exposed to other entities in the smart space (peers, external or even humans).
(iii) Usage of an information model and standardized protocol based on the Representational State Transfer (REST) in order to offer a public API accessible to as many clients as possible (*Challenges D and E*). This API provides access in a uniform way without concerning neither communication issues nor data representation formats.
(iv) Support for different communication modes to deliver information to clients according to their information consumption requirements (*Challenge C*). The most typical communication modes are supported: *on-demand* and *event-driven*. Both communication modes will be accessible through API mentioned before.
(v) Management of massive amounts of data generated by large ecosystems of Smart Objects (*Challenge D*). This design principle is supported by an information model that enable persistence techniques that aims at optimizing the massive storage of data generated in smart spaces.

The following Section describes the WoTOP architecture supporting the above mentioned design principles.

3.1 Platform Architecture Overview

The WoTOP is based on a layered architecture composed of three layers interconnected among them through well-defined interfaces (see Fig. 2). In turn, every layer of the architecture includes subsystems and components that fulfil specific objectives. These subsystems and components are explained in the following.

A. *Internet of Things Ecosystem Layer*: The Internet of Things Ecosystem Layer enables the WoTOP to connect with *things* coexisting in the *real-world*. These things are usually *smart*, and they are characterized according to different natures and objectives within the *smart space* they belong to. Physically, a smart thing consists of one or more embedded devices equipped with sensors and/or actuators that enable it to play a role in the smart space, coexisting with other smart things to fulfil a given objective.
Smart spaces are usually composed by a wide variety of smart things. In order to deal with this technological heterogeneity issue, the Internet of Things Ecosystem Layer implements a mechanism based on the *Plug&Play* paradigm. Through this mechanism, devices of very different technologies can be plugged to the WoTOP and discovered *on the fly*, in order to expose smart thing capabilities to the higher layers of the architecture. The components implementing drivers for any technology are called *adapters* , and they are stored in a repository.

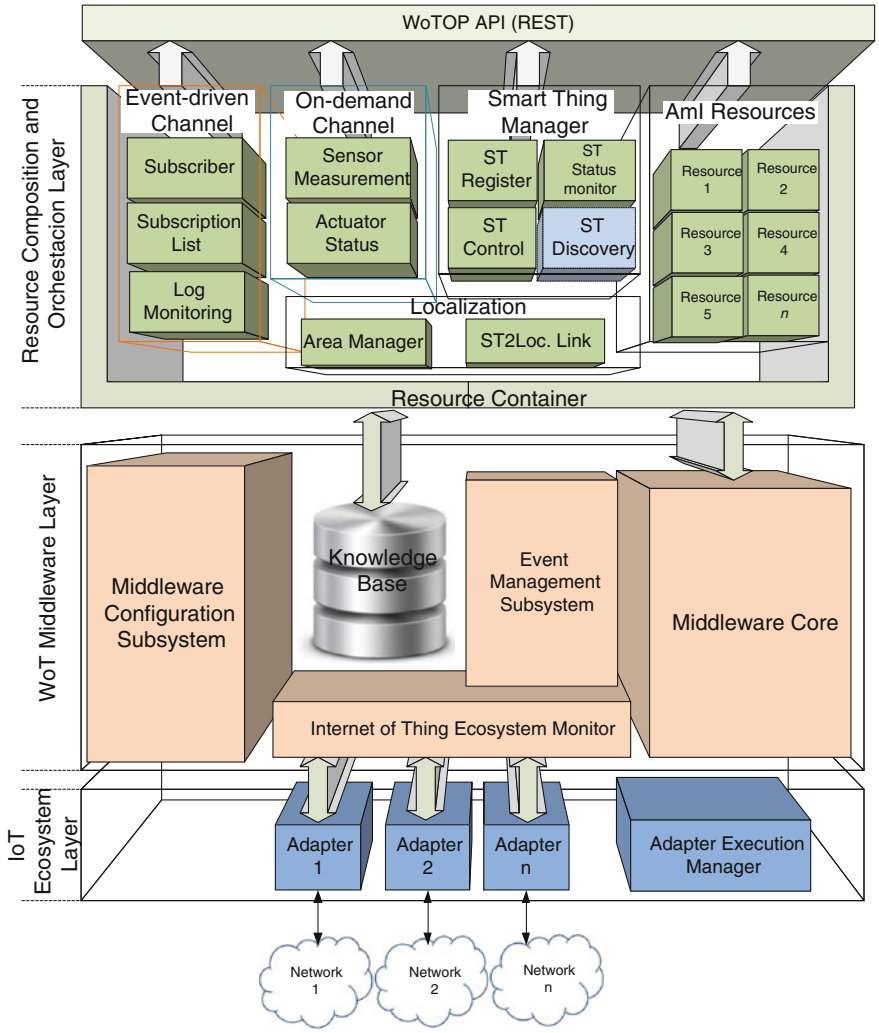

Fig. 2 The layered architecture of the Web of Things Open Platform

This mechanism is supported by an OSGi framework that allows scheduling the life-cycle of adapters, i.e. load, unload, run and stop. To integrate the adapters with OSGi, they are implemented as *bundles* that are stored in a OSGi Bundle Repository (OBR). When a new device is plugged to a WoTOP Gateway, the needed bundle is retrieved and loaded automatically.

B. *The Web of Things Middleware Layer*: The functionalities of this layer allow designers and developers of WoT services to model, implement and deploy complex smart spaces and expose them according to the WoT paradigm. The Web of Things Middleware Layer is composed of the following subsystems:

(1) *Internet of Things Ecosystem Monitor*: This component aims at listening data from the adapters deployed on the Internet of Things Ecosystem Layer. Basically, it registers every device connected to the WoTOP as well as its own context information and events generated by them. From the information collected from all those devices, this component creates and updates a registry called *Alive Smart Thing Register* (ASTR). The ASTR consist of a table of devices' identifiers associated to smart things, including relevant information about its status (i.e. battery level, current localization or the owner) and a cache with historical information related to the environment context in which they are deployed (i.e. sensor measurement, high level context information or actuator status). The information temporally stored in that cache is periodically dumped into a *Knowledge Base* which manages its persistency. The process commented before is shown in the Fig. 3.
The information stored in the cache of the ASTR is also processed by the Event Management Subsystem in order to detect events, as it is explained below.

(2) *Event Management Subsystem*: This subsystem was designed to enable WoTOP to manage asynchronous communications for those clients that need to consume context information according to specific requirements. This subsystem includes different mechanisms to send notifications or events to the clients. Two *event-driven* mechanisms are supported: (i) *condition-based*: events are dispatched to the clients when context information gathered from smart things matches some condition set up by the clients, e.g. temperature in room A $>$ 22 °C AND humidity in room A $<$ 35 RH; (ii) *contract-based*: If the context information available in ASTR's cache is relevant to the client and said information is updated, then events are dispatched to clients periodically, e.g. the location of a smart thing is notified to a client every 2 s. Both types of mechanisms can be initialized by clients through subscriptions that are defined and sent to the WoTOP accordingly to their interests.
Once the client has sent a subscription expressing an interest, a *Webhook* is instantiated in the client application . The *Webhook* allows opening an entry to receive and handle messages from WoTOP. A *Webhook* consists of defining a HTTP callback to deliver messages asynchronously to the client; it is identified by an URI that will be used by the *Event Management Subsystem* in order to POST events to the client interested in them. The Event Management Subsystem keeps two subscription tables, one per type of communication mode supported by it: (i) *Condition-based subscription table* and, (ii) *Contract-based subscription table*. The former is intended to store subscriptions to events that have to be sent if a condition occurs. The latter is intended to store subscriptions to events that have to be sent according to contract method. Processes and components involved in event dispatching are different for each type of event. Theses processes are shown in the Fig. 4.
As Fig. 4 shows, the data to be processed by the *Event Management Subsystem* always have the same origin, the *Internet of Things Monitoring Sybsystem* which gather them in a orderly manner. Then, the data are temporary buffered before being processed by the corresponding module of the *Event Management*

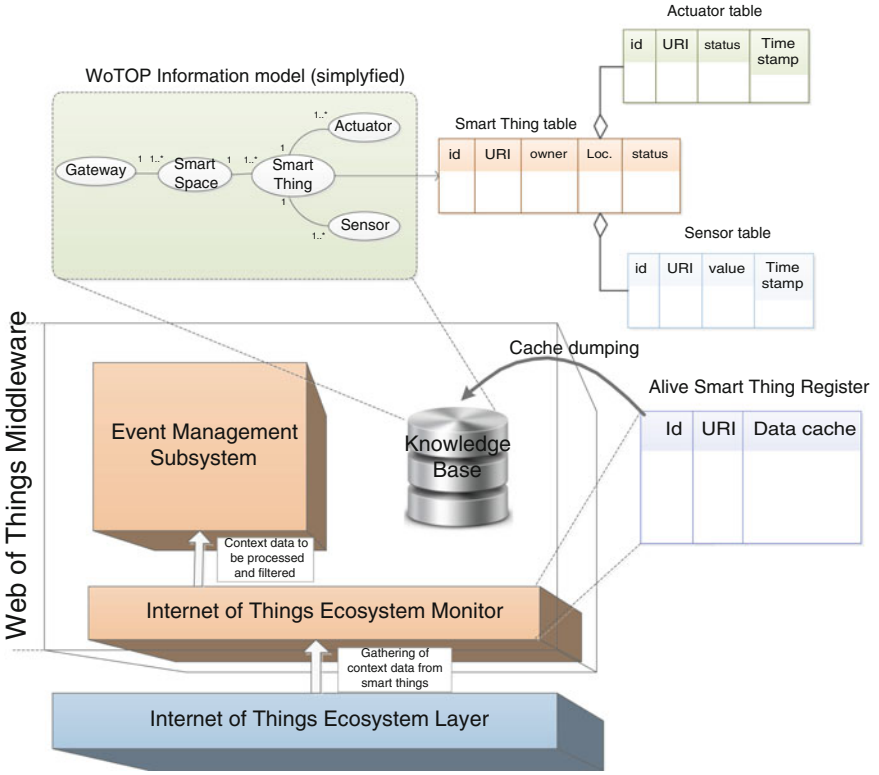

Fig. 3 Processing of the context data gathered through *the Internet of Things Ecosystem Monitor*

Subsystem. On the one hand, the buffered data are filtered by the *Rule Analyzer* (see Fig. 4a) which checks subscriptions that were configured according to the conditional mechanism. If one or more subscriptions match some Context data, then an event is detected and delivered to the client that is interested in it. On the other hand, the buffered data are periodically polled by the *Scheduler of Context Data Notifications* (see Fig. 4b) according to the subscriptions based on the contract mechanism. If more recent data are found and there is any subscriber to them, then they are dispatched to the subscribers.

(3) *Middleware Configuration Subsystem*: This subsystem aims at preparing and configuring the execution environment of the Web of Things Middleware. An optimized configuration depends on the features of the device that is going to run the middleware. The standard parameters that can be configured through this subsystem are related to the location and credentials to access the knowledge base, maximum number of clients that can access to WoTOP services simultaneously or maximum number of subscriptions that can be established in the subscription table.

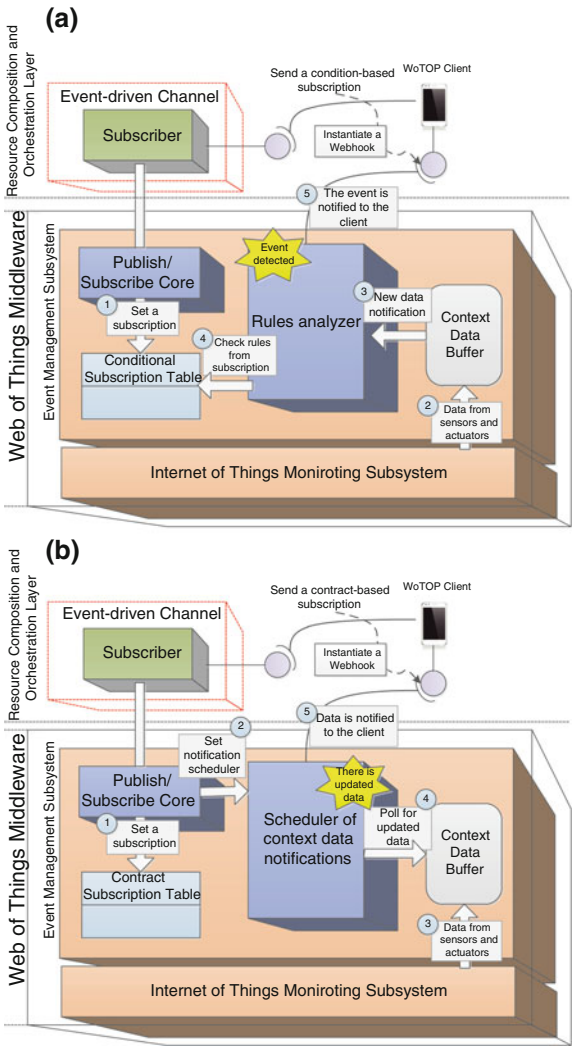

Fig. 4 **a** Event-driven communication based on the condition technique; **b** Event-driven communication based on the contract technique

(4) *Middleware Core*: This is an essential module of the Web of Things Middleware, whose major objective is to expose the interfaces of the components building the WoTOP's architecture, in order to facilitate the necessary communication among them. This kind of communication is based on OSGI services. Additionally, the Middleware Core can dynamically create REST end-points according to the components deployed on the *Resource Composition and Orchestration Layer*. The latter functionality is a cornerstone within the WoTOP architecture since it facilitates exposing RESTful services to the clients.

Table 1 REST interface of the Event-driven channel of the WoTOP

Access method and input message	Description

• **Create a subscription**
`POST /{gw-id}/eventmanager /subscriber`
`Content-type: application/json`
• **Update a subscription**
`PUT /{gw-id}/eventmanager /subscriber/{subs-id}`
`Content-type: application/json`
- **Conditional**
`{`
`"consumer": {"URI": <String>, "Id": <String>},`
`"producer": {"URI": <String>, "Id": <String>},`
`"type":<String>, //Contitional (1)`
`"end":<long>, //End time of the subscription`
`"event": [{ "value": <String>,`
`"key": <String>,`
`"unit": <String>,`
`"operator": <String>}, ...]`
`}`
- **Contract**
`{`
`"consumer": {"URI": <String>, "Id": <String>},`
`"producer": {"URI": <String>, "Id": <String>},`
`"type":<String>, //Contract (2)`
`"start":<long>, //Start time of the subscription`
`"end":<long>, //End time of the subscription`
`"samplingPeriod":<long>,`
`"event": { "key": <String>,`
`"unit": <String>}`
`}`
	These requests create (or update) a subscription for a specific event. The dispatching mechanism of to the client will depend on the type of subscription done previously (by condition or by contract).
	
	The reply to this request will content a unique subscription code which is needed for update or unsubscribe any subscription.
	
	`consumer` and `producer` parameters indicate the root URI (`URI`) and their id (`Id`) identifying uniquely the event consumer and producer, respectively. Event consumers must enable a RESTful handler in order to manage event callbacks which are based on POST messages (Webhooks).
	The format of an event is the following:
	`{`
	`"producer": {"URI": (String), "Id": (String)},`
	`"type": (String), //Conditional (1) or Contract(2)`
	`"timestamp": (long), //When the event was generated`
	`"payload": [ {"key": (String), "unit":(String),`
	`"value": (String) }, …]`
	`}`
	
	The event payload contains useful information of the event (structured in triples):
	-Key: Concept of the event.
	-Unit: Unit or type of the measured value.
	-Value: Value of the measurement that trigger the event.

• **List subscriptions of a consumer**
`GET /{gw-id}/eventmanager/subscriber?consumer= <Consumer URI>`
	This request returns a list of subscriptions related to an event consumer.
	The format of the returned messages is an array of JSON documents as follows:
	`[{`
	`"subscriptionCode": <String>,`
	`"subscription": <subscription format>`
	`}, …]`

• **Delete a subscription (unsubscription)**
`POST /{gw-id}/eventmanager/unsubcriber`
`Content-type: application/json`
`{`
`"consumer": {"URI": <String>, "Id": <String>},`
`"eventType":<string>`
`"subscriptionCode":<string>`
`}`
	A POST request has to be sent to the specific URI identifying a subscription which has to be deleted form the subscription table. This method attaches a JSON document that indicates the URI and Id of the event consumer, the type of the event and the code of the specific subscription to be deleted.

C. *Resource Composition and Orchestration Layer*: As mentioned in the previous point, the Resource Composition and Orchestration Layer is a strategic subsystem of the WoTOP architecture since this layer enables WoTOP to provide enriched RESTful services to manage smart spaces. Resources provided by this layer are RESTful services which are implemented by components. Those components are deployed on the *Resource Container* which facilitates the management of their lifecycle. The lifecycle of the components deployed in the *Resource Container* is planned in four stages: initializing, running, stopping and destroying. The first (initializing) and the fourth (destroying) stages allocate and free memory for the operation of the component, respectively. The second (running) and the third (stopping) stages are mainly focused on opening and closing the REST end-points that facilitates the access to the resources that are offered by the components. Those resources can be accessed through one or more HTTP methods according to the REST paradigm, i.e. GET, PUT, POST or DELETE. As an example, the Table 1 shows the REST interface of the Event-driven channel.

The components provide atomic services that can be offered to the clients without

needing support from other components. Additionally, the resource container provides mechanisms to perform composition and orchestration of atomic resources in order to provide complex resources involving two or more atomic resources. As shown in Fig. 2, WoTOP deploys some resources by default; these are classified according to the following groups: *Event-driven channel, On-demand channel, Smart Thing Manager* and *Localization*. These resources are deployed with the aim of exposing essential services of the platform that allow accessing middleware functionalities. The Event-driven and On-demand channels provide resources to the clients to allow them to consume information and to handle events which are generated by smart things and stored according to the information model shown in Fig. 3. The *Smart Thing Manager* resource provides functionalities for integral management of smart things composing a smart space. For instance, functionalities are offered to register a smart thing in the ASTR or to monitor the status of smart things registered in the ASTR through RESTful interfaces.

Additionally, WoTOP provides the *Localization* resource, which is deployed to provide positioning information that may be necessary to manage a given smart space. The major functionalities offered by this resource allow dividing the smart space in areas and linking smart things to those areas by means of semantic annotations. This resource could be potentially used by clients or external services using WoTOP.

Finally, the *Resource Composition and Orchestration Layer* allocates part of the *Resource Container* to deploy additional resources, so called *Ambient Intelligent Resources*. These resources are designed to extend the capabilities of the WoTOP according to the open nature of its architecture, by developing and deploying resource components that may take advantage of the available functionalities in the platform.

3.2 Development and Deployment of Smart Spaces using WoTOP

As shown in Fig. 5, WoTOP is hosted by one or more Smart Gateways that can be synchronized with each other in order to create domains of Smart Gateways. A domain of Smart Gateways aims at sharing knowledge resources (e.g. information collected from smart things or other AmI services) to offer them to application clients through a Platform as a Service (PaaS) paradigm. This paradigm allows providing services uniformly and seamlessly regardless of the heterogeneity and extension of the underlying smart space in terms of hardware and functionalities. According to the openness of WoTOP, some procedures were designed to facilitate the development and deployment of smart spaces.

In this subsection, we provide a brief introduction of the steps to follow to create a varied ecosystem composed of sensor and actuator devices, logic entities and human users. Specifically, three aspects are tackled in this subsection: (i) integration of additional sensor and actuator devices into WoTOP, (ii) development of new resources to offer new functionalities to client applications, and (iii) developmentof

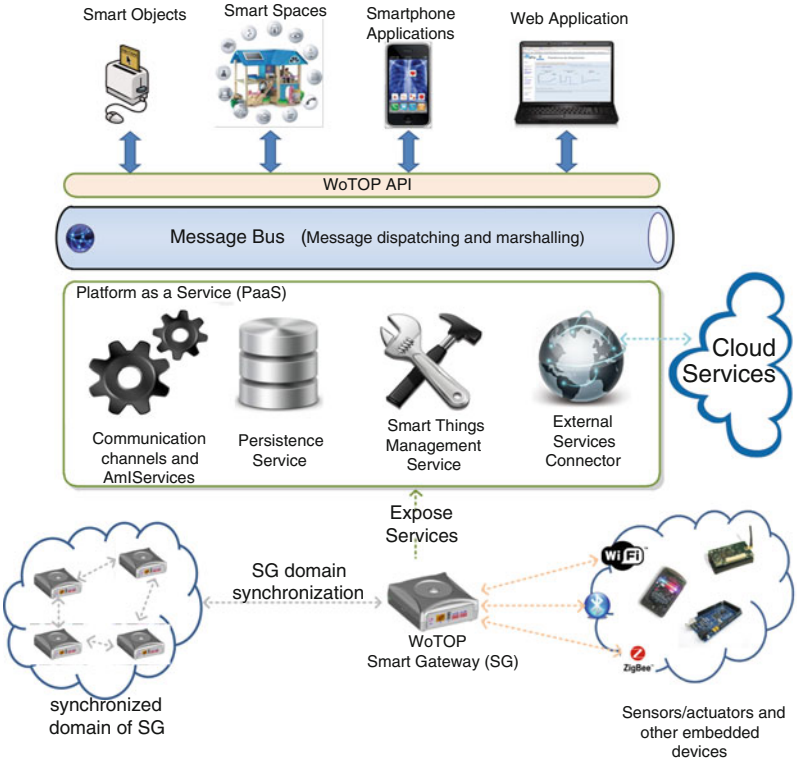

Fig. 5 Infrastructure, entities and services involved in WoTOP

client applications with capabilities to access WoTOP's resources. In order to explain those aspects, let us take as a reference the model shown in Fig. 5.

(i) *Integration of additional sensor and actuator devices*

There are two general methods to integrate additional sensors and actuators. The first one allows connecting those devices directly to some Smart Gateways hosting an instantiation of WoTOP. Firstly, this requires that the Smart Gateway is compatible with the devices to be connected at hardware level and that the specification of its communication protocol is open for the developer community. Let us assume that such preconditions are fulfilled thus, those devices can be physically connected to the Smart Gateways. Secondly, it will be necessary to deploy a specific adapter for those devices into the WoTOP. As commented in Sect. 3, every adapter has to be wrapped by OSGi bundles which are stored in an OBR. These bundles have to implement a specific protocol in order to enable bidirectional communication among WoTOP and devices. Additionally, every adapter must use the interface of the *Internet of Things Ecosystem Monitor*, which is the subsystem in charge of registering every device in WoTOP, as well as collecting and handling data received or sent from/to

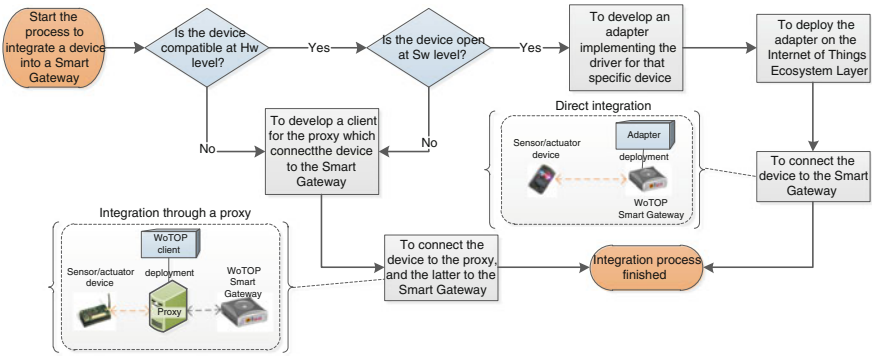

Fig. 6 The stages of the process for integrating devices to WoTOP

these devices. Whenever a device of that type is plugged into a smart gateway, the corresponding bundle will be automatically loaded to deal with it.

Only if the device to be integrated is not compatible at hardware level or its communication protocol is proprietary, its integration into WoTOP will depend on the framework provided by the manufacturer (composed of hardware and/or software elements). That framework will be used to create a kind of *proxy* among the device and WoTOP; this proxy will have to implement a client using the WoTOP API to get access to every resource deployed on the *Resource Composition and Orchestration Layer*. Specifically, that client would have to use the REST methods provided by the *Smart Thing Manager* resource in order to register or unregister the device to the WoTOP by means of the *ASTR* managed by the *Internet of Things Ecosystem Monitor*.

Figure 6 depicts a flow diagram showing the steps to be followed to integrate devices into WoTOP according to the two alternative methods explained before.

(ii) *Development and deployment of new resource components*

As commented previously in this Section, the *Resource Composition and Orchestration Layer* is the cornerstone to expose RESTful services into the Web. This layer not only manages several components by default to provide essential platform functionalities, but it can also deploy additional resource components that aggregate new platform services with the aim of improving and extending the usability of WoTOP for a wide range of scenarios. These components are called *Ambient Intelligence Resources*. Every developer is responsible of developing and deploying their own components inside the *Resource Composition and Orchestration Layer*; this layer is based on the RESTlet framework thus resources components must accomplish some requirements related to the internal work of that framework.[1] Those components can

[1] It implies that every resource component extends, at least, the super class *ResourceServer* to specify the accepted REST methods as well as the format of the message body. Then, it will be necessary to attach every component to a URI in order to forward every request to the right component. The latter is carried out during initialization sequence.

access every service provided by other resource components or by the *Web of Thing Middleware Layer* in order to reuse functionalities already implemented.

The idea behind building resource components which can perform atomic processes or integrate between each other to build complex services, is to create communities of developers that can share components to customize their own smart gateways.

(iii) *Development of WoTOP's application clients*

First of all, it is important to highlight that the purpose of *application clients* is completely different to that for *clients* used for implementing proxies that connect devices to WoTOP. Specifically, application clients are intended to create applications using the services offered by the WoTOP. They get access to those services through a message bus which hides the complexity of service. Such message bus provides a uniform interface that allows interacting with a smart space, i.e. to consume information generated by that smart space or to send messages to it in order to cause a specific behaviour. The most common way of consuming information consist of accessing the two information channels provided by default: On-demand and Event-driven channels, both provided by resource components at the *Resource Composition and Orchestration Layer*.

4 The Resource-Oriented and Ontology-Driven Methodology

4.1 Introduction

As explained in the previous Section, the WoTOP provides functionalities to integrate the lower layer of the reference architecture (see Fig. 1) into the Web by delivering mechanisms that facilitate the development and deployment of heterogeneous ecosystems for smart spaces. This WoTOP's feature fulfills a relevant requirement of the WoT research domain specified in the reference architecture, i.e. the ubiquitous and seamless interaction among client applications and smart spaces providing *real-world* services. However, there are other open challenges. Those challenges are going to arise in a near future, when smart spaces composed of hundreds, even thousands, of Web-enabled smart objects, will become daily omnipresent entities providing real-world services at global scale. There will be a need for sound methodologies that improve the links in the development chain of IoT and WoT, mostly the mapping of physical things into RESTful services in order to optimize the cost of deploying smart spaces.

Recent trends [6, 28–30] have adopted the concept of the Resource-Oriented Architecture (ROA) proposed by Fielding at the beginnings of 2000s [31]. The purpose of those proposals is to facilitate rapid developments and deployments in the context of the IoT and the WoT taking advantage of simplicity and versatility of ROA. Despite this promising characteristics, existing platforms do not still fully decouple lower layers (focused on hardware and protocols) from the higher layers

(mostly dedicated to the information management and the provision of services to application clients).

So far, any developer who faces the development and deployment of RESTful services over any platform or framework, still needs to have strong knowledge on general and specific technology aspects as communication protocols, specific programming languages or platforms of wireless sensor and actuator networks, among others. For this reason, some research works are addressing domain-specific development frameworks based on patterns and models since it is a solution to reduce costs with deployment time in large deployments of smart spaces. A recent trend in this field bets on the Model Driven Engineering (MDE) principles [32–34]. MDE-based methodologies simplify the process of design and development by using models and design patterns. Additionally, those approaches can increase the communication among participants working on the system development via standardization of languages and terminology, e.g. by means of a domain ad-hoc language specified by a work group in order to manage the life-cycle of a software product. Furthermore, MDE-based approaches can be very well tailored to the IoT and WoT, as this paradigm:

(i) Abstracts every part of the system through high-level models independently of the underlying software and hardware technologies.
(ii) Decouples consumers and providers of context resources (mostly sensor, actuators and logic processes), enabling a feasible reuse of model artifacts and software components.
(iii) Provides a model-based development framework to facilitate rapid and agile prototyping of complex deployments even for non-expert developers and users.

4.2 Architecture Principles of the ROOD Methodology

According to the points commented before, the research contribution included in this Section is focused on a MDE-based methodology which specifies a common abstraction model, semantically expressive, which provides the tools to define all the important elements of a smart space. A complete review of this contribution can be found in a previous work [5]. The initial motivation of this research was to provide a versatile solution to facilitate the development of different smart spaces composed of heterogeneous sensors, actuators and logic processors interacting among them through a variety of mechanisms. To address such a challenge, we propose the Resource-Oriented and Ontology-Driven Development (ROOD) methodology, which takes advantage from traditional MDE-based tools and improves them by providing semantic expressiveness. The ROOD was designed to be platform-agnostic, i.e. it can be used on almost every software and hardware platform without concerning the underlying software or hardware technology.

ROOD methodology is based on a OMG's (Object Management Group) MDE architecture: the Model-Driven Architecture (MDA). The MDA is a layered

architecture composed of three interrelated kinds of models: (i) Computational Independent Model (CIM), (ii) Platform Independent Model (PIM) and (iii) Platform Specific Model (PSM). These models have to be machine-readable so that they are successively transformed into other models, code stubs, schemas, test harnesses, and deployment scripts for diverse platforms [35]. OMG provides standardized tools to perform the MDA development methodology, particularly the Unified Modelling Language (UML) [36]. Domain Specific Modelling Languages (DSML) can be designed by means of a profile mechanism, provided by UML 2, with enough expressiveness and precision for almost any technological domain. In order to design a canonical MDA approach, we specified a novel DSL to model different aspects of smart spaces. On the one hand, this DSL allows defining the behaviour and contextual activities of smart objects from a high level point of view. On the other hand, they allow using modelling tools to define functional aspects of business processes from a low level point of view, associated to the previously modelled behaviours.

4.3 The Smart Space Modeling Language

ROOD methodology addresses the development of smart spaces from two different perspectives: (a) *the context activities*, which specify the behavior of resources (sensor, actuator, and interfaces for human interactions) used within a smart space and the relationships among them, and (b) *the smart object*, which provides a deployment perspective of the system involving information and processing models characterizing sensor and actuator entities within the smart space and its association with RESTful services. The ROOD methodology includes models related to both levels that encompass the mentioned features: (a) the *Environment Context Model* (ECM), and (b) the *Smart Object Model* (SOM). These models are instances of a DSL, the Smart Space Modeling Language (SsML) that was designed as an UML profile. Additionally, the modeling processes concerning those models are enriched through semantic technologies; concepts represented both in ECM and SOM are aligned to semantic contents that are stored in Knowledge Bases (KB) and defined according to an ontology called Smart Space Ontology (SSO). In that way, the ROOD methodology takes advantage from ontological resources to verify the completeness and consistence of ECM and SOM models according to the semantic description of the domain system; consequently the verification mechanism optimizes the model-to-model transformation processes from ECM to SOM.

The definition of SsML depends on the MDA architecture that is stratified in four abstraction levels (M0 through M3). The objectives of these levels are the following: (i) M0 contains instances of data for a specific platform; (ii) M1 is where the systems models are defined; (iii) M2 specifies the DSLs that take part in the definition of models at M1; (iv) Finally, M3 defines the Meta-Object Facility (MOF), that establishes the basis for different modeling languages.

Figure 7 shows the logical position of the SsML in the MDA architecture. As it can be seen, the SsML is in M2 layer and extends the UML metamodel; SsML uses

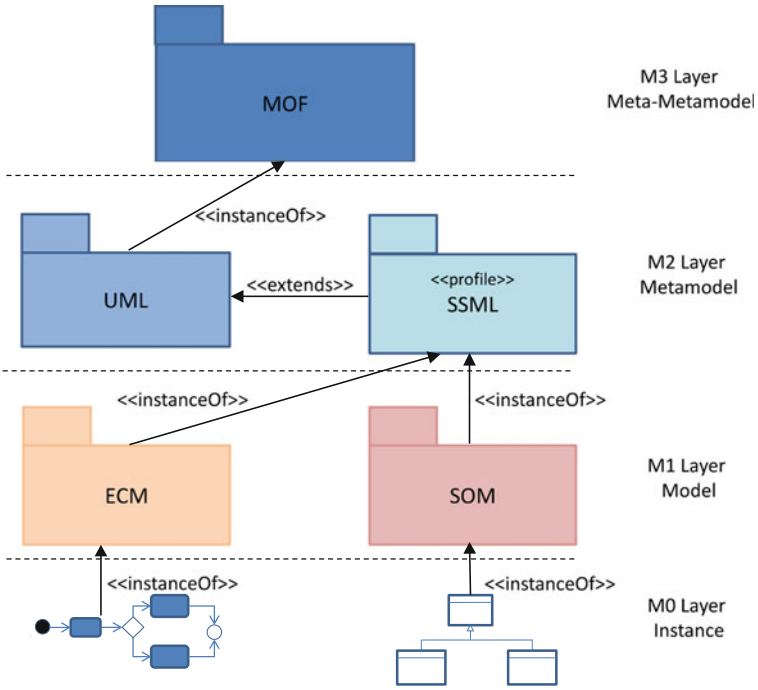

Fig. 7 The SsML according to the MDA architecture

the extension mechanisms defined in the UML 2 specification in order to create an own profile that defines every necessary element (entities, relations and interfaces) to model the smart space.

4.4 Stages of the ROOD Methodology

This Section presents the different stages of the ROOD methodology involving the elements of the architecture. This guideline focuses on the traceability between the concepts presented in ECM and SOM metamodels, as well as the mechanisms to verify models delivered in each stage.

As discussed in the previous section, any development methodology based on MDA consists of three main phases: (i) Computation Independent Model (CIM); (ii) Platform Independent Model (PIM); (iii) Platform Specific Model (PSM). Along these phases, MDA manages to separate the conceptual design (focusing on functional requirements of the system) from the platform features (defining no functional and technological aspects of the underlying architecture).

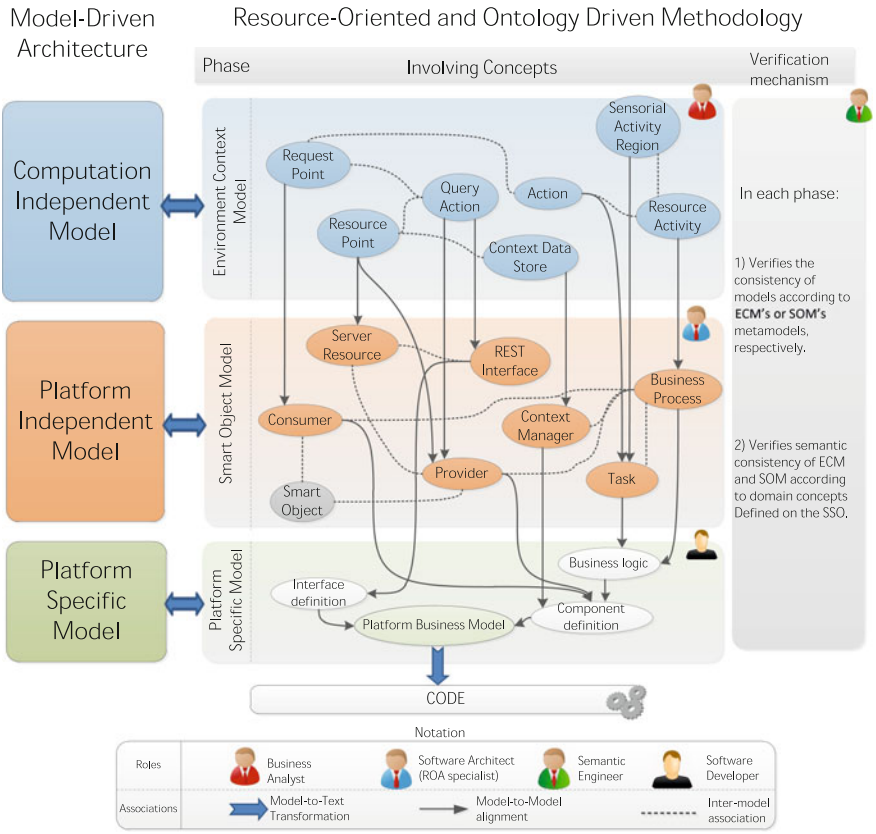

Fig. 8 The ROOD methodology according to the concepts managed in every phase, as well as model transformations and verifications between phases

Firstly, we identified a set of traceability relations between concepts in the different models. A full map of the mentioned traceability can be consulted in [5]. In Fig. 6, it is represented the major concepts that have to be modelled in each RODOs stage as well as the traceability between ECM and SOM elements. Additionally, the processes for model verification are included.

The concepts shown in Fig. 8 are defined in the ECM and SOM models. Such models set restrictions of use for different entities in each stage of the ROOD methodology, as well as their relationships. The entities specified in SSO are projected on concepts of ECM and SOM models which facilitates the verification of integrity and completeness of models in relation to the domain information stored in Knowledge Bases.

It is important to remark that the ROOD methodology is flexible and its stages are loosely coupled. By means of this feature, a wide range of professionals are enabled to work together, collaborating and working jointly in the development

chain, e.g.: (i) Business analyst; (ii) Software architect, specialized in ROA; (iii) Semantic engineer; and (iv) Software developer.

Before starting the first stage of the methodology, each element of the smart space, involving the specification and development of smart objects, will have to be analyzed. Meanwhile, every smart object in such smart space is supported by one or more platforms with a set of sensor and actuator devices. The behavior of the smart objects will be conditioned to the business processes and underlying tasks (e.g. those managing sensor measurements) that are performed by the platforms associated to them.

In ROOD, there are two different agents managing the business logic: consumers or providers. Finally, services exposed by the smart objects, to be consumed by other entities belonging to the smart space, are encapsulated into resources, which are defined accordingly to the REST architectural style. The ROOD methodology follows the most accepted REST implementation in which, for each resource, it is assigned a Unique Resource Identifier (URI), one or more HTTP method (GET, PUT, POST or DELETE) and specific formats for input and output messages (usually XML, RDF or JSON).

Once the Knowledge Base of the smart space is instantiated, the modeling phase of the ROOD methodology can start. Firstly, business analysts have to model the smart space using the elements defined in the ECMs metamodel. In this stage, only behavioral information of the smart space is represented without concerning the underlying platform, e.g. activity threads, actions, transitions between actions, and smart objects involved in that activity context.

The information contained in ECM models is used to partially generate the models in the next stage, SOM. This stage should be managed by software architects specialized in resource-oriented architectures, who will take advantage of the information from the ECM to model agents (providers or consumers of services), business processes and tasks. Moreover, it will be needed to set up the information system to manage context information that will have an influence in the current and future behavior of the smart objects. Finally, the involved software architects will design the necessary architectural elements that will offer the smart object resources as RESTful services. This step will provide the key piece to integrate the smart space into a WoT paradigm.

The models created in previous steps will be subjected to a verification process to check the consistency and integrity of the entities, relationships and other semantic information represented in them, according to the domain information stored in a scenario Knowledge Base.

The final step consists of generating program code from SOM models. For this aim to be achieved, the elements represented in SOM models are filtered through a model-to-text transformation mechanism. The percentage of generated code can vary depending on the underlaying platform but in any case, it will be totally generated. Therefore, a software developer is required to complete existing gaps in the code (e.g. configuration parameters for hardware peripherals or specific information to integrate devices into a communication infrastructure).

It is important to highlight that the transformation specified in the ROOD methodology (ECM-to-SOM and SOM-to-PSM/code) is conducted by mapping rules that in some cases are almost automatically generated and only partially automated in the remainder cases.

5 Case-Study Through Real Prototyping: Smart Shop

Smart spaces are designed to adapt to their inhabitants, at the same time that they can configure their operation in an efficient way. Combining WoTOP and the ROOD methodology, it is feasible to easily deploy and configure a smart space and its related services. In this Section, we describe how to do it to in a particular scenario.

5.1 Motivation Scenario

Let us consider a public space such a smart shop, in which visitors can wander about and explore its commercial offer. In this smart shop:

- Clients are able to augment the objects on sale, retrieve personalized offers or browse customized information about external services through their smart devices.
- Technical assistants are able to control the environmental conditions of the space. They can receive alerts and notifications in case that humidity, temperature or occupancy thresholds are exceeded. They may also establish the shop configuration that optimizes energy consumption while maximizing clients comfort.
- Shop managers may configure specific atmospheres, e.g. through virtual contents and lighting.

To enable these services, it is necessary to configure an infrastructure of sensors and actuators, which connected to WoTOP will be able to provide the needed information to the consumer services. In particular, to deploy the services above, the following infrastructure is required:

- An accurate positioning system, providing centimeter error.
- A network of humidity, temperature and light wireless sensors.
- A network of wireless actuators (e.g. capable of managing the air conditioning, the lights and the blinds).

And consumer services include:

- An augmented reality service with a backend, which allows managers to configure the virtual offering of contents. These contents will be delivered taking into account the users preferences and context.
- An environmental monitor with a configuration board for alerts.

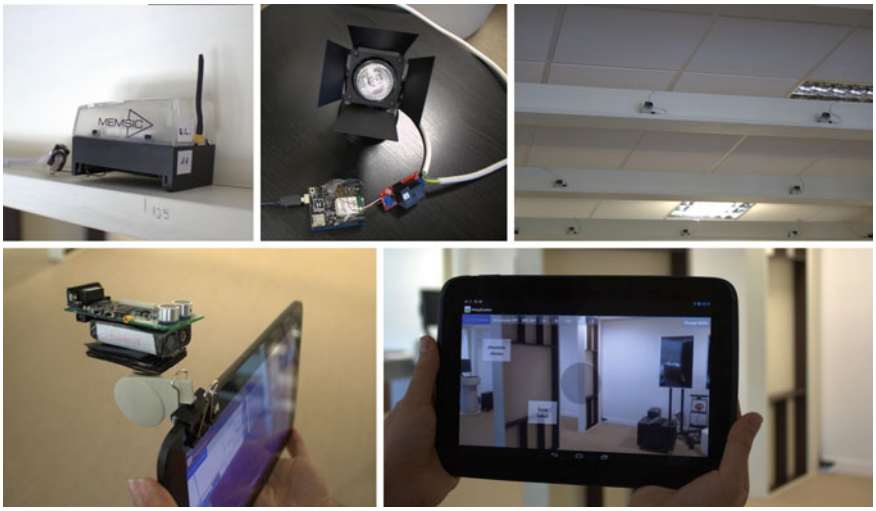

Fig. 9 Infrastructure deployed to simulate the Smart Shop. From *top to bottom* and *right to left*: **a** MicaZ node with a sensor board integrated, fixed to the ceiling; **b** Arduino Uno connected to a LED lamp; **c** Cricket-based network deployed strategically on a ceiling structure; **d** Cricket node attached to a tablet PC; **e** A tablet PC running a AR application

- An environmental optimizer, to control the actuators while maximizing comfort and minimizing energy consumption.

We have prototyped this service scenario in our Experience Lab, a testing environment located in the Montegancedo Campus of the Universidad Politecnica de Madrid. In this space, it is feasible to customize technical infrastructure, furniture and décor to simulate different environments. A sample of the variety of elements deployed for our study case is shown in Fig. 9. In the next subsections, we detail how the infrastructure sensors and actuators have been deployed and connected to WoTOP.

5.2 Deployment Infrastructure

The necessary deployment infrastructure for the smart shop is shown in the Fig. 10. The core of the infrastructure is a Smart Gateway running a WoTOP's instance. The role of this Smart Gateway is twofold. On the one hand, it is responsible of gathering data from smart things and dispatch them to the application clients according to their interests in consuming them. On the other hand, the Smart Gateway can forward messages from client applications to actuators whenever they consider.

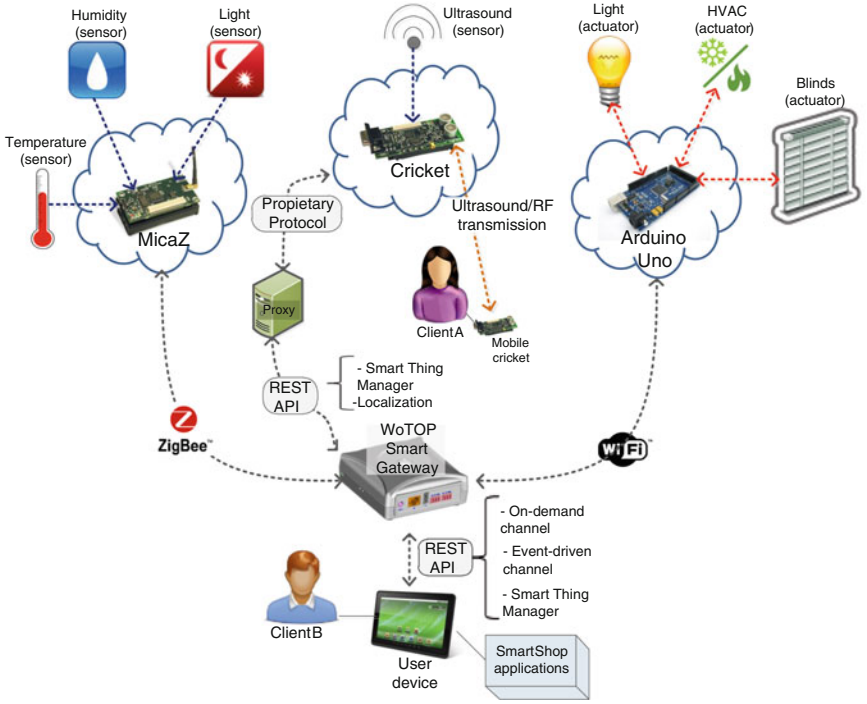

Fig. 10 The deployment infrastructure necessary for the proposed smart shop and the users involved on it

Basically, the sensing and actuating infrastructure is based on wireless embedded nodes of different technologies: Crossbow MicaZ,[2] Crossbow Cricket[3] and Arduino Uno.[4] Each one has different roles within the scenario that are collected in the Table 2.

Most of this infrastructure is deployed on specific facilities of the smart shop, i.e. they are fixed nodes associated to those places. Exceptionally, there will be one or more mobile nodes based on the Cricket platform. The mobile node is used to transmit two concurrent message based on the combination of RF and ultrasound technologies. These messages are listened by several beacon nodes to calculate the localization information attached to the mobile node.[5] Note that the Cricket network is connected to the Smart Gateway through a proxy according to the method explained in Sect. 3. The rest of embedded nodes are directly integrated into the Smart Gateway.

Beyond the networks of embedded sensor and actuator nodes, the infrastructure is also composed of user devices that are provided to the clients of the smart shop. These devices have a significant role within the smart shop since they make possible

[2] http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[3] http://www.willow.co.uk/Cricket_Datasheet.pdf

[4] http://arduino.cc/en/Main/arduinoBoardUno

[5] For more information about the localization algorithm, visit: http://cricket.csail.mit.edu/.

Table 2 Classification of the deployed smart things according to their sensors and actuators, data inputs and outputs, as well as services provided by them

Smart thing	Sensor/ Actuator	Data input/ Output	Main functionality and associated services
Crossbow MicaZ	**Sensors**: • Temperature • Humidity • Light	**Output**: temparature, humidity and light measurements both in event-driven and on demand modes	**Funct.**: to dispatch temperature humidity and light measurements **Services**: saving energy and ambiance configuration
Crossbow Cricket	**Sensors**: • Ultrasound	**Input**: RF signal + ultrasound signal **Output**: difference between reception timestamp of RF and ultrasound signals in the listeners	**Funct.**: to calculate accurately the localization attached to mobile users within specific areas **Services**: AR, saving energy and recommendation of products and services
Arduino Uno	**Actuators**: • Blind • Led lamps	**Input**: Command message: turn on/off for lamps; up/down for blinds	**Funct**: To activate actuators according to commands received from context-aware agents running on client applications **Services**: AR, saving energy and ambiance configuration

the interaction between human users and real world services which are provided through the Smart Gateway. For a reasonable performance of client applications and other agents running on those devices, it is necessary an appropriate configuration and usage of the services provided by WoTOP. This requires to identify the producers and consumers of information, as well as the *method* used to dispatch information between producers and consumers. The next section includes a brief overview of the configuration of the WoTOP's services for our smart shop, including the most important parameters.

5.3 Configuration of WoTOP's Services for the Smart Shop

As mentioned above, the first stage for reaching a suitable configuration of the WoTOP's services consist of identifying producers and consumers of information. The information producers of the smart shop were gathered in Table 2. In general they correspond with embedded sensor nodes, but also the Localization resource is identified as an information producer. The latter can provide preprocessed information from specific coordinates generated by Cricket-based network, e.g. areas of the building in which users are located at some point.

The information consumers are mostly client application running on user devices, for instance tablets and smartphones, that will process such information to offer products or services to the shop users. In other occasions, consumers will enable autonomous responses from the actuators in the smart space. Previously, a method to dispatch information between producers and consumers of information has to be set up. As said in Sect. 3, WoTOP provides two different modes to transmit information to consumers: event-driven and on-demand.

Let us illustrate the configuration of the smart shop through the saving energy service. This service is associated to the *sustainable consumption agent* which consumes concrete events to minimize energy consumption. Specifically, these agents will perform *conditional-based* subscriptions for events that are generated from temperature, humidity and light sensor nodes to manage situations in a untended and smart way according to the received sensor values, e.g. to configure automatically the HVAC system, pull up/down blinds and set up light intensity of the LED lamps of a room/area of the shop, depending on the energy efficiency rating needed for the building.

Additionally, the *sustainable consumption agent* can make *contract-based* subscriptions in order to collect periodical localization information from clients, in order to improve even more the energetic efficiency (e.g. to perform fine-grained configuration of the HVAC and LEDs lamps depending on the accurate localization of the clients and staff in an area of the shop).

The *ambience configuration agent*, that is associated to the smart ambiance configuration service, also needs to make *contract-based* subscriptions for localization of users to offer interaction functionalities to control some parameters of the environment, e.g. to control actuators on-demand mode by pointing the camera of the user device (e.g. tablet or smart phone) to the specific actuator.

Table 3 shows a list of producers and consumers of information associated to the services deployed in the smart shop. Every producer and consumer is related to a URI that, in the case of consumers, identify the agent which has to receive a callback message, i.e. a POST message with an event in the body. The format of the messages encapsulated into the event payload (JSON documents) is also shown in Table 3.

From the information collected in Table 3, subscriptions of different types have to be done. The number and type of those subscriptions will depend on the number of consumers being interested in consuming specific types of events. Table 4 shows examples of two type of subscriptions that could be sent to the Smart Gateway from different agents.

It is important to highlight that interactions between agents running on user devices and actuators deployed on the smart shop (e.g. between *ambiance configuration agent* and HVAC, blinds or LEDs lamps) are carried out by means of typical request/response methods, i.e. using on-demand communication mode of WoTOP. Those requests are sent to the Smart Gateway by means of a PUT method to the specific resource and, from there, they are forwarded to a specific actuator. The same communication mode is used to obtain sensor values instantly but, in this case, it is sent a GET request to a resource associated to a sensor.

Table 3 Event configuration for services provided by client applications in the smart shop

Smart Shop Service	Information management		Event type*	Payload format
	Producer	Consumer		
• Saving Energy • Smart Ambience Configuration	**Temperature sensor** URI: `http://[gw-id]/→ →/…/sensor/temp`	• **Sustainable consumption agent** URI: `http://[IP]:[port]/→ →/agent/energysav` • **Ambience config. agent** URI: `http://[IP]:[port]/→ →/agent/ambience`	1	`{ "key": "temp", "unit":"celsius", "value": (String) }`
	Humidity sensor URI: `http://[gw-id]/→ →/…/sensor/hum`		1	`{ "key": "hum", "unit":"RH", "value": (String) }`
	Light sensor URI: `http://[gw-id]/→ →/…/sensor/ligth`		1	`{ "key": "temp", "unit":"lux", "value": (String) }`
	Localization (coordinate) URI: `http://[gw-id]/→ →/…/loc/coordinate`		2	`[ {"key": "x", "unit":"double", "value": (String)}, … , {"key": "z", "unit":"double", "value": (String)} ]`
Augmented Reality: • Recommendation of products and services • Smart Ambience Configuration	**Localization (coordinate)** URI: `http://[gw-id]/→ →/…/loc/coordinate`	• **Location-aware recommendator agent** URI: `http://[IP]:[port]/→ →/agent/recommedator`	2	`[ {"key": "x", "unit":"double", "value": (String)}, … , {"key": "z", "unit":"double", "value": (String)} ]`
	Localization (area) URI: `http://[gw-id]/→ →/…/loc/area`	• **Location-aware recommendator agent** URI: `http://[IP]:[port]/→ →/agent/ambience`	1	`{ "key": "area", "unit":"integer", "value": (String) }`

*) Type of event according to the two supported communication modes : *contract* (1) and *conditional* (2).

Table 4 Subscription examples in JSON format: on the left, conditional-based subscription from a saving energy agent to temperature values in a MicaZ node; on the right, a contract-based subscription from an ambience agent to localization information of a mobile Cricket node

```
{
    "consumer": {"URI": "http://[IP]:[port]/",
                 "Id": "/agent/energsav"},
    "producer": {"URI": "http://[gw-id]",
                 "Id": "/micaz/1/sensor/temp"},
    "type": "1", //Contitional (1)
    "end": "86400000", //24 hours
    "event": [{ "key": "temperature",
                "unit": "Celsius",
               "value": "25",
                "operator": "GREATER"}, ...]
}
```

```
{
    "consumer": {"URI": "http://[IP]:[port]/",
                 "Id": "/agent/ambience"},
    "producer": {"URI": "http://[gw-id]",
                 "Id": "/cricket/1/loc/coordinate"},
    "type":"2", //Contract (2)
    "start":"20000",//Start time (ms) of the subscription
    "end":"300000", //End time (ms) of the subscription
    "samplingPeriod":"1500", //Sampling period (ms)
    "event": [{ "key": "x",
                "unit": "centimeter"},
              { "key": "y",
                "unit": "centimeter"}
               ]
}
```

To conclude, although it was not mentioned in this Section, the configuration of WoTOP's services, focused on managing the information exchange between the entities of the smart shop, can be carried out by means of the ROOD methodology. Currently, we are working on a completely functional prototype of a modeling tool based on ROOD that will be able to be used to model every entity involved on a smart space taking into account their roles (consumer and/or producers), as well

as the major characteristics of their interactions with other entities belonging to the same smart space (communication modes, message format, subscription parameters, etc.) among other characteristics mentioned in Sect. 4.

6 Conclusions

In this Chapter, some important aspects of the Web of Thing were analysed, together with our approaches to deal with them. In particular, we firstly presented the Web of Things Open Platform (WoTOP) that was designed keeping in mind an essential principle: to bring the gap among the physical world and human users by developing smart spaces which interconnect seamlessly smart things, business process and users. WoTOP takes advantage of well-known Web technologies as well as of one of its most popular approaches in the pervasive computing field, the WoT paradigm, in order to hide the underlying technological heterogeneity. Such heterogeneity comes from a IoT-based ecosystem, composed of a number of different sensor and actuator devices which can be connected to WoTOP's smart gateways by means of two different methods: (i) the first one enables to connect devices directly to those Smart Gateway and, (ii) the second one, to connect devices indirectly through a proxy, that is ad-hocly developed according to its technological features (e.g. protocols and hardware).

The second contribution, briefly introduced in this Chapter, was the Resource-Oriented and Ontology-Driven (ROOD) methodology which is based on the Model-Driven Architecture (MDA) of the OMG. This methodology aims at facilitating the rapid and friendly deployment of WoT-based smart spaces, built on heterogeneous ecosystems and involving different producers and consumers of resources. For this purpose, visual modelling languages have been specified by extending the UML 2 profile so that no-technicians could develop and deploy smart things belonging to a smart space. Moreover, every stage of ROOD methodology is enhanced with validation mechanisms that autonomously analyses the models to validate them both syntactically and semantically.

Finally, the feasibility of the major research contributions presented in this Chapter were demonstrated through a case study. The motivation scenario used for our case study was a smart shop. That smart space was partially deployed in a simulated environment, using a variety of technologies based on wireless networks of embedded sensor and actuator devices, and some multimedia devices running specific applications to interact with the smart shop. The core of the smart space is a Smart Gateway hosting an instance of WoTOP through which every entity (physical or logical) of the smart space can be connected among each other rapidly and seamlessly, accordingly to the requirements of the smart space.

6.1 Outlook to Future Research Challenges

During 2000's decade technical and scientific disciplines related to Pervasive Computing progressed exponentially, opening novel and exciting research challenges which have provided new basis for integration of computers in the everyday life. Particularly, the Internet of Things and the Web of Things paradigms provide a wide amount of service possibilities.

The major idea behind IoT is to achieve a network layer for constrained devices highly compatible with Internet environment through IP-based protocols. In this research field, the most important initiative is 6LoWPAN which is an Internet Engineering Task Group (IETF) Working Group founded with the purpose of designing of protocols and mechanisms to integrate seamlessly networks of wireless and resource-constrained devices into Internet keeping in mind the major features of both types of networks (e.g. routing, discovery, security or memory footprint). Although interesting results have been reached in form of RFCs, there is still a lot of work to be done before reaching an standard that is agreed by industry and academia.

Beyond network layer, interesting research challenges have risen in the edge between the IoT and WoT research domains. Those initiatives have been focused on an application perspective consisting of tackling a direct integration of resource-constrained devices into the Web [37]. Among proposals in this field, the Constrained Application Protocol (CoAP) is being widely accepted in the IoT community. CoAP is a RESTful protocol which minimizes the complexity of mapping with HTTP enabling resource-constrained devices to deploy tiny Web servers that are characterized by low footprints and workload. The IETF Constrained RESTful environments (CoRE) Working Group is carrying out the major standardization work for CoAP. Despite CoAP is in final phases of standardization, current Web browsers lacks capabilities to use CoAP-based devices seamlessly without requiring cross-proxies. To the best of our knowledge, only one proposal [38] has addressed the issue of integrating CoAP connectivity capabilities into browsers to facilitate the communication between human users with resource-constraint devices like if they were conventional Web servers.

Additionally, the management of bi-directional data streams is still an issue under discussion. HTTP is a stateless protocol which works adequately for *request-response* operations initiated by clients (e.g. read/write data from/to a embedded sensor or actuator) but lacks of mechanisms to manage data streams in event-driven scenarios based on publisher/subscriber paradigm. In this kind of scenarios, information is asynchronously generated by embedded devices, thus it must be dispatched to specific clients (only the interested ones in that information) as soon as possible. In this Chapter, we have detailed an event-driven subsystem, which was implemented for WoTOP. This subsystem is based on Webhooks to send information from smart things to clients. We could prove that Webhooks works correctly under controlled local environments (e.g. LANs or VPNs). However, a proper working of Webhooks is not guaranteed when clients are located at external networks since the call-back POST messages, that encapsulates events, would have to be transmitted through firewalls that could

discard that type of traffic. Alternatives to Webhooks have been proposed as the use of Extensible Messaging and Presence Protocol (XMPP) or Comet-based mechanisms. The former is an open protocol based on real-time messages in XML format, widely used in instant messaging applications (e.g. Google Hangouts). The latter is a mechanism that enables Web servers to push data back to clients based on AJAX programming. Despite both proposals would accomplish requirements for dispatching information to the clients asynchronously, they are not suited for embedded devices since they use heavyweight and verbose formats as XML. Moreover, they usually are optimized for Web browsers and plugins to be run on them, restricting their capabilities to implement customized application clients.

Other alternatives propose extending Atom and RSS protocols to create publication/subscription brokers which can manage feeds of information. Clients can subscribe to that feeds through a message broker in order to receive data via callbacks when they are published. For example, the PubSubHubbub[6] initiative provides the mentioned mechanisms using the PuSH protocol. Nonetheless, this type of solutions could be unsuitable for scenarios in which sensitive information is managed (e.g. logistic, eHealth or finances) since every event have to be dispatched by means of a message broker depending on a third part which could not provide necessary Quality of Service level (in terms of timeliness or reliability), or sufficient security parameters.

Other research topics that are being considered are those related to the integration of smart things into the Semantic Web. Recently, some promising approaches have arisen with the purpose of semantically annotating smart things for different applications, for instance to collaborate among devices to reach a common goal by means of reasoners [39]. It will be interesting to observe how these approaches will evolve to support complex and large smart spaces in the future, when semantics will be necessary to search information and create mashups of services related to the physical world.

To conclude, it is certainly important to mention that one of the pillars of the Future Web of Things will lie on the existence of methodologies to prototype, deploy and maintain large smart spaces, rapidly and easily, in order to reduce costs as much as possible. As was proposed in ROOD methodology, a major characteristic of this type of methodologies consist of providing friendly tools (essentially based on visual elements) to facilitate no-technicians to deploy smart space abstracting from low level issues. The Model-Driven Architecture (MDA) is a step forward in this direction.

Acknowledgments This work has being supported by the Government of Madrid under grant S2009/TIC-1485 (CONTEXTS). The authors also acknowledge related discussions within the THOFU initiative, funded by the Spanish Center for the Development of Technology.

[6] https://code.google.com/p/pubsubhubbub/

References

1. Weiser, M.: The computer for the 21st century. IEEE Pervasive Comput. **99**(1), 19–25 (2002)
2. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**, 2805 (2010)
3. Besbes, M.A., Hamam, H.: An intelligent RFID checkout for stores. In: 2011 International Conference on Microelectronics (ICM), pp. 1–12 (2011)
4. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. In: Proceedings of the 2000 International Conference on Software Engineering, pp. 407–416 (2000)
5. Corredor, I., Bernardos, A., Iglesias, J., Casar, J.R.: Model-driven methodology for rapid deployment of smart spaces based on resource-oriented architectures. Sensors **12**(7), 9286–9335 (2012). URL http://www.mdpi.com/1424-8220/12/7/9286
6. Corredor, I., Martínez, J., Familiar, M.: Bringing pervasive embedded networks to the service cloud: a lightweight middleware approach. J. Syst. Architect. **57**(10), 916–933 (2011)
7. Corredor, I., Martínez, J., Familiar, M., López, L.: Knowledge-aware and service-oriented middleware for deploying pervasive services. J. Netw. Comput. Appl. **35**(2), 562–576 (2012)
8. Brown, P., Estefan, J., Laskey, K., McCabe, F., Thomton, D.: Reference Architecture Foundation for Service Oriented Architecture Version 1.0. Technical Report, OASIS (2012). URL http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.pdf
9. Cosm–Internet of Things Platform Connecting Devices and Applications for Real–Time Control and Data Storage (2013). https://cosm.com/
10. Paraimpu–The Web of Things is more than Things in the Web. http://paraimpu.crs4.it/
11. Arduino Website (2013). arduino.cc/.
12. Sensinode Ltd. (2013). https://www.sensinode.com/
13. SmartThings–Make your world smarter (2013). http://smartthings.com/
14. Gomez-Goiri, A., de Ipina, D.L.: Assessing data dissemination strategies within triple spaces on the web of things. In: 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 763–769 (2012)
15. Farrell, S.: API keys to the kingdom. IEEE Internet Comput **13**(5), 91–93 (2009)
16. Guinard, D., Fischer, M., Trifa, V.: Sharing using social networks in a composable Web of Things. In: 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 702–707 (2010). ID: 1
17. Blackstock, M., Lea, R.: IoT mashups with the WoTKit. In: 2012 3rd International Conference on the Internet of Things (IoT), pp. 159–166 (2012)
18. Mayer, S., Guinard, D., Trifa, V.: Facilitating the integration and interaction for real-world services for the web of things. In: Urban Internet of Things: Towards Programmable Real-Time Cities (UrbanIOT 2010); Workshop at the Internet of Things 2010 Conference (IoT 2010) (2010)
19. Wu, Z., Itala, T., Tang, T., Zhang, C., Ji, Y., Hamalainen, M., Liu, Y.: Gateway as a service: a cloud computing framework for web of things. In: 2012 19th International Conference on Telecommunications (ICT), pp. 1–6 (2012)
20. ThingWorx–M2M and Internet of Things Application Development Platform (2013). https://www.thingworx.com/
21. AirVantage–M2M Platform (2013). http://www.sierrawireless.com/airvantage
22. Axeda–M2M cloud service (2013). http://www.axeda.com/
23. Thiesse, F., Floerkemeier, C., Harrison, M., Michahelles, F., Roduner, C.: Technology, standards, and real-world deployments of the EPC network. IEEE Internet Comput. **13**(2), 36–43 (2009)
24. IERC–European Research Cluster on the Internet of Things (2013). http://www.internet-of-things-research.eu/
25. Corredor, I., Martínez, J.F., Familiar, M.S.: Research experiences about internetworking mechanisms to integrate embedded wireless networks into traditional networks. In: Interconnecting Smart Object with the Internet Workshop (European Commission -IETF) (2011)

26. Evrything–Make products smart (2013). http://www.evrythng.com/
27. ThingSpeak (2013). https://www.thingspeak.com/
28. Guinard, D., Trifa, V., Wilde, E.: A resource oriented architecture for the Web of Things. In: 2010 Internet of Things (IOT), pp. 1–8 (2010)
29. Villalonga, C., Bauer, M., López, F., Huang, V., Strohbach, M.: A Resource Model for the Real World Internet, Smart Sensing and Context, vol. 6446, pp. 163–176. Springer, Heidelberg (2010)
30. Zhang, W., Jiang, L., Cai, H.: An ontology-based resource–oriented information supported framework towards RESTful service generation and invocation. In: 2010 Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE), pp. 107–112 (2010)
31. Fielding, R.: Architectural styles and the design of network-based software architectures. Ph.D. Thesis, University of California, Irvine (2000)
32. Katasonov, A., Palviainen, M.: Towards ontology-driven development of applications for smart environments. In: 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 696–701 (2010)
33. Soylu, A., Causmaecker, P.D.: Merging model driven and ontology driven system development approaches pervasive computing perspective. In: 2009 24th International Symposium on Computer and Information Sciences, ISCIS 2009, pp. 730–735 (2009)
34. Tetlow, P., Pan, J., Oberle, D., Wallace, E., Uschold, M., Kendall, E.: Ontology driven architectures and potential uses of the semantic web in systems and software engineering (2006). URL http://www.w3.org/2001/sw/BestPractices/SE/ODA/
35. Management Group (OMG), O.: UML Infraestructure Specification (2011)
36. Gherbi, T., Meslati, D., Borne, I.: MDE between promises and challenges. In: 11th International Conference on Computer Modelling and Simulation, UKSIM '09, pp. 152–155 (2009)
37. Kovatsch, M., Mayer, S., Ostermaier, B.: Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. In: Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012, pp. 751–756 (2012)
38. Kovatsch, M.: CoAP for the web of things: from tiny resource-constrained devices to the web browser. In: The 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '13, p. 1495 (2013)
39. Mayer, S., Basler, G.: Semantic metadata to support device interaction in smart environments. In: The 2013 ACM International Joint Conference on Pervasive and Ubiquitous, Computing, p. 1505 (2013)