Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 347413, 13 pages
http://dx.doi.org/10.1155/2015/347413

Research Article

High Performance Web of Things Architecture for the Smart Grid Domain

David Vernet, Agustín Zaballos, Ramon Martin de Pozuelo, and Víctor Caballero

Engineering Department, Universitat Ramon Llull (URL), La Salle, 08022 Barcelona, Spain

Correspondence should be addressed to Agustín Zaballos; zaballos@salle.url.edu

Received 3 July 2015; Revised 26 November 2015; Accepted 29 November 2015

Academic Editor: Salvatore Distefano

The increasing complexity in the management of Smart Grids is an essential factor in the creation of new technological infrastructures capable of managing the different devices involved in the network. This network has been converted into an example of the Internet of Things. In this regard, the Web of Things enables an improvement in the processing of these data. Besides, the large amount of data in the Smart Grid domain means that a high performance architectural design is able to manage concurrently the entire information processing ability. This paper presents an initial approach for a new architecture and the first results after the system implementation.

1. Introduction

The recent growth of the Internet has fostered the interaction of many heterogeneous technologies under a common environment (i.e., the Internet of Things, IoT). Smart Grids entail a sound example of this situation where several devices from different vendors, running different protocols and policies, are integrated in order to reach a common goal: bringing together energy delivery and smart services. This proposal deploys an IoT-based infrastructure that enables machine-to-machine interactions between small and resource-constrained devices on the Smart Grid domain based on HTTP. It extends the IoT concept by providing a bidirectional human-to-machine interface, inspired by the Web of Things (WoT), which results in a ubiquitous energy control and management system (i.e., uniform access to all devices of the Smart Grid) coined as Web of Energy (WoE) [1].

The recent advances on this domain have led to effective architectures that support this idea from a technical perspective but fail to provide powerful tools to assist this new environment. Hence, the purpose of our technological proposal is to research a novel unified and ubiquitous sensor management interface that uses the advantages featured by the Web of Things to manage the Smart Grid. Therefore, this work opens a new path from the Internet of Things to the WoT and results in a new concept coined as the WoE [1]. Overall, the Web of Energy links all domains and permits a bidirectional communication between electricity domain and the application domain. More concretely, the main objective is to carry out a proof of concept of an open web-based interface that isolates the electricity grid domain from its utility functions. (1) It relies on a distributed storage layer to support the massive amount of data generated by the grid. The database proposed is an open-source document database that provides high performance, high availability, and automatic scaling [2]. (2) It also relies on a Southbound RESTful API that permits an easy and seamless management of the distributed storage layer and the smart objects connected to the system.

2. The Smart Grid as an IoT

Recent advances on Smart Grids have explored the feasibility of considering the power electrical distribution network as a particular case of the IoT [3, 4]. Certainly, this specific domain poses appealing challenges in terms of integration, since several distinct smart devices (also referred to as Intelligent Electronic Devices or IEDs) from different vendors, often using proprietary protocols and running at different layers, must interact to effectively deliver energy and provide a set of enhanced services and features (also referred to as

smart functions) to both consumers and producers (prosumers) such as network self-healing, real-time consumption monitoring, and asset management [5]. Although the latest developments on the IoT field have definitely contributed to the physical connection of such an overwhelming amount of smart devices [6], several issues have arisen when attempting to provide a common management and monitoring interface for the whole Smart Grid [5, 7].

Indeed, integrating the heterogeneous data generated by every device on the Smart Grid (e.g., wired and wireless sensors, smart meters, distributed generators, dispersed loads, synchrophasors, wind turbines, solar panels, and communication network devices) into a single interface has emerged as a hot research topic [4, 8]. So far, some experimental proposals [3] have been presented to face this issue by using the Web of Things (WoT) concept to access a mashup of smart devices and directly retrieve their information using reasonably thin protocols (e.g., HTTP and SOAP) [9]. However, the specific application of these approaches into real-world environments is fairly dubious due to the following reasons: (1) they may open new security breaches [10, 11] (i.e., end-users could gain access to critical equipment), (2) there are no mature electric devices implementing WoT-compliant standards available in the market [5], and (3) industry is averse to include foreign modules (i.e., web servers) on their historically tested and established, but poorly evolved, proprietary systems [12].

Therefore, the authors of this paper explore a new way to overcome these issues through the European project INTelligent Electrical GRId Sensor Communications (INTEGRIS) [5] and Future INtErnet Smart Utility ServiCEs (FINESCE) project [13]. This work provides a management interface for the Smart Grid inspired by the WoT. Continuing the work done in these projects, the aim is to implement an ICT infrastructure, based on the IoT paradigm, to handle the Smart Grid storage and communications requirements [14] to manage the whole Smart Grid and link it with end-users using a WoT-based approach, which results in a new bridge between the IoT and WoT. This proposal, which takes the pioneering new form of the WoT, is targeted at providing a context-aware and uniform web-based novel environment to effectively manage, monitor, and configure the whole Smart Grid. Moreover, conducted developments prove the feasibility and reliability of our approach and encourage practitioners to further research in this direction and to envisage new business models [15, 16].

The open IoT-based infrastructure presented in our Web of Energy proposal will provide new tools to manage energy infrastructures at different levels from IoT-based infrastructure enabled machine-to-machine interactions between small and resource-constrained devices on the Smart Grid domain. Thus, we have extended the IoT concept by providing a bidirectional human-to-machine interface, inspired by the WoT, which results in a ubiquitous energy control and management system coined as Web of Energy [1]. This proposal will combine the web-based visualization and tracking tools with the Internet protocols, which enables a uniform access to all devices of the Smart Grid. In order to provide such an effective and reliable management interface to address the heterogeneous nature of devices residing on the grid, we will continue the deployment of an intelligent subsystem devoted to (1) learn from the real-world events, (2) predict future situations, and (3) assist in the decision-making process.

The tools developed can be provided in an open format available to anyone in the research community and are able to contribute to and enhance the platform building new modules for managing other resources. As a first step, the platform will be demonstrated through the management of Smart Metering resources, based on the formats of DLMS COSEM and IEC 61850, but the whole system will be designed for a much wider scope where every utility, enterprise, and public administration or any organization or single prosumer can build their own sensor application on top of it (Figure 1).

2.1. WoE Presentation and Specified Functions. The concept of Smart Grids has been considered in recent years as an appropriate answer to address new challenges in the energy domain. However, additional resources are still needed in challenges such as the proactive operation of the grid, efficient integration of demand into grid operation, integration of renewable generation, or maximum network reliability to obtain applicable solutions. The WoE will combine energy domains and ICT technologies with the objective of building an interoperable platform for the coordinated planning, operation, and settlement of future distribution and access to networks by integrating demonstrable solutions in real user environments based on Web of Things technologies.

The current control of electricity distribution networks, including the Advanced Distribution Automation (ADA) and Demand Side Management (DSM), is (1) centralized, (2) silo-oriented, (3) fragmented by applications that use specific communication technologies for each purpose but with a lack of integration among them, and (4) generally based on proprietary ICT systems owned by the DSO [18]. So, in practice, the situation is still centralized and fragmented but with existing solutions and trials allowing the integrated management and the distribution of selected granular functions. This contrasts with the distributed and fractal nature of the future Smart Grids, which can only be based on standardization, flexibility, distributed systems, and communication among all the actors [12, 13].

Furthermore, the increased use of renewable and distributed generation means the operation and management of the electric power system must change radically. Increased levels of automation, distributed intelligence, and on-line data mining and management are required to deliver the network control functions, reducing reconfiguration and the restoration times. Reconfiguration of smart grids addresses new challenges during normal operation and also for restoration and management of crisis situations [10, 19]. The connection of end-users (prosumers) to the energy market will facilitate the installation and connection of devices that offer grid services that will help mitigate capital and operational costs of the grid modernization required for energy transition and minimize environmental impact, thus ensuring lower electricity prices for everything involved. New benefits will be generated and shared in a fair way between all actors, from aggregators to industrial end-users and citizens.

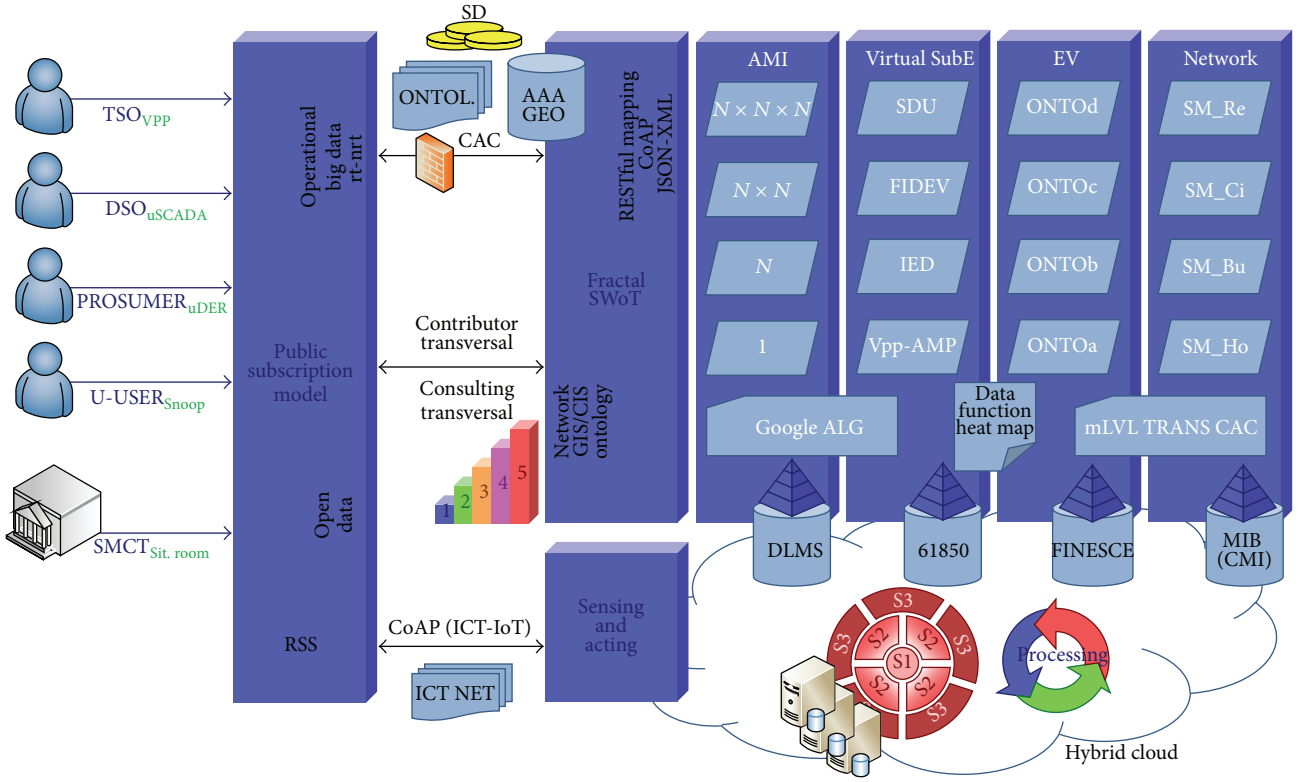

Figure 1: WoE interactions.

A coordinated vision of the grid will provide mechanisms to tackle the challenges mentioned above showcasing them through real and modern applications. WoE will provide synergies at different levels: Smart Grids and other smart networks, individuals, and communities as a first approach to a Software Defining Utility (SDU). The potential benefits of the WoE are framed by (a) innovative communications, acquisition, and processing platform based on the extensive use of the "Real-Time Services" concept providing open and interoperable access, (b) metering integration platform based on multiplatform web technologies, (c) advanced medium/low voltage control centre integrating real-time grid information coming from devices to provide a clear view of the current and the near-future status of the grid thanks to a high performance environment, and (d) energy services market platform. The performance quantification of that WoE concept will be the key to accelerate the implementation of new policies, market rules, and emerging Smart Grid programs [15, 16].

WoE outlines some of the challenges in improving the resiliency of the electrical grid and proposes an approach that makes use of advanced sensor technology (advanced sensors are needed to improve the knowledge of state), analytics, and agile control in a Smart Grid [20, 21]. Furthermore, WoE proposes a Smart Grid supervision infrastructure, which can deliver real-time and high performance notifications on a global scale for transferring measurements from different distributed sensors and take actions over the grid via different communication protocols, informing the different stakeholders (e.g., producers, consumers, aggregators, and system operators) within the adequate time frame [8, 18]. We propose to use a distributed infrastructure based on Web of Things and implement a novel service platform for facilitating distributed control, autohealing, and power grid control.

In addition, WoE technology will tackle the implementation of smart real-time distributed monitoring platforms enabling the data fusion and knowledge extraction for the different faults detection and prevention schemes. Intelligent HTTP based sensors will provide a new source of relevant distribution status information, including loadings, voltage profiles, harmonics, and outage conditions which, combined with equipment condition data, such as power frequency interference signatures, will provide predictive perspectives of potential equipment failures. This platform requires large storage technologies that can hold the massive amounts of information that will be generated by the millions of sensors implemented in the Smart Grids and will make the information available in negligible times to the system or systems demanding it (Figure 2).

Smart Grids need monitoring strategies based on decentralized and uncoupled architectures (service oriented and multiagent), supported by real-time middleware (DDS) capable of dealing with huge amounts of information at different time scales, process events (complex event processing), and discover sequences of them (sequence event discovery) working together with OLAP and data mining solutions. A multiagent conception of the grid is necessary to deal with coordination and optimization requirements for

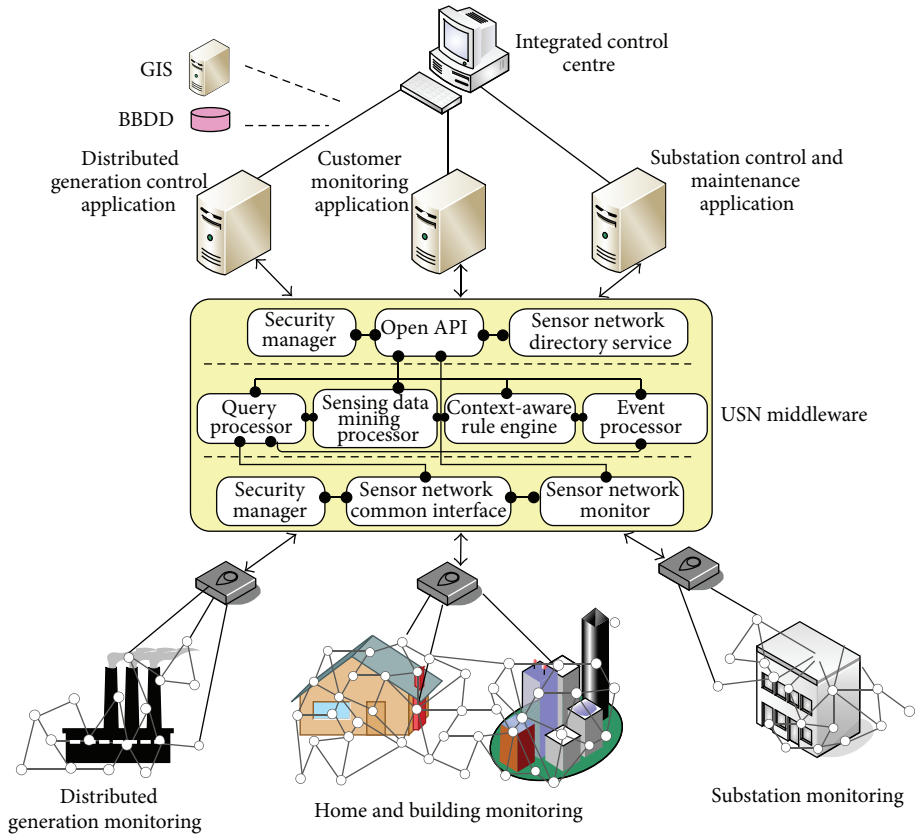

Figure 2: Open API architecture.

an efficient and safe network. Pervasive web monitoring devices with communication capabilities increase security of the network as a whole but imply new cyber-security issues and make optimal restoration and reconfiguration more complicated [22]. Currently, heuristics, metaheuristics, and learning methods are mature and available. Restoration and reconfiguration techniques including autohealing will benefit from these reasoning engines.

3. The Web of Things

Day by day, the number of connected Things is growing exponentially. The latest data shared by Gartner finds that 4.9 Billion of Things will be accessible during 2015 and this figure will increase to 25 Billion in 2020 [23]. The way to access these devices from a single platform is undoubtedly one of the biggest headaches for researchers. In this regard, standardized solutions provided by the rapid evolution of the Internet have laid the foundation of what we call the Web of Things. Therefore, WoT architectures aim to integrate everyday objects with web technologies. Those devices should be able to communicate with each other using existing web standards. Prerequisites for those Things are minimal processing and communication capabilities. WoT researchers try to define and delimit concepts (e.g., what is a Thing? [24]) implied on those envisioned architectures and solve some problems that arise when every Thing may sense or actuate on every Thing.

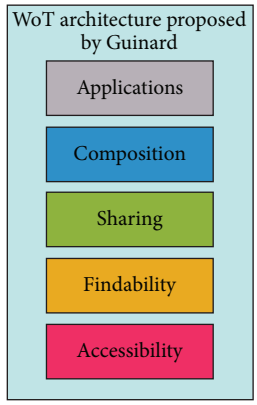

Figure 3: Layered architecture [17].

The architecture presented by Guinard in his thesis [17] proposes a good basis to start other WoT designs. Guinard defines WoT general prerequisites; he also defines what a Thing is and a virtual object is in a WoT architecture and finds solutions to problems like how to discover and find Things and how those Things can connect and push information to a server. In [17], a layered architecture (see Figure 3) that consists of four layers that address four main problems is proposed:

(i) Device Accessibility Layer: how to enable consistent access to all kinds of connected objects?

(ii) Findability Layer: how do we find their services to integrate them into composite applications?

(iii) Sharing Layer: how we preserve privacy?

(iv) Composition Layer: how do we get closer to end-users?

One of the main problems of the WoT architecture is to standardize communication protocols between different Things. Indeed, there is still no clear standard defined for this purpose and there are different options available. In the next section, we present the best placed and supported protocols by researchers and which protocols are also valid/eligible alternatives.

4. WoT Protocols

It would be easy for WoT to use only one protocol, but the heterogeneity of devices composing the WoT makes it unfeasible. Thus, different communication protocols should be considered. The use of each one depends on the final proposed solution.

4.1. Preferred Candidates by the Community

4.1.1. MQTT. As stated in its official webpage [25], MQTT stands for MQ Telemetry Transport. It is a publish/subscribe (pub/sub), extremely simple, and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements while attempting to ensure reliability and some degree of assurance of delivery. These principles also contribute to the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices and to mobile applications where bandwidth and battery power are at a premium. MQTT was introduced by IBM and Eurotech companies.

MQTT is a protocol that uses a pub/sub model, connecting publishers and subscribers via a broker (server). Its headers are small and therefore their overhead is minimum. MQTT can also work over SSL for security reasons, but SSL adds an extra overhead to the communication. As publishers and subscribers connect via a broker, the use of a centralized server leads to a SPF (Single Point of Failure).

4.1.2. CoAP. CoAP [26] was specified and standardized by the CoRE (Constrained RESTful Environments) group in IETF; the Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks, such as those that will form the Web of Things. This protocol shares several similarities with HTTP like its REST architectural style but instead of using TCP it uses UDP to achieve its goals.

As it is a request/response protocol like HTTP, both WoT servers and constrained devices or gateways should act as servers and clients at the same time to ensure bidirectional communication at any time. For example, a constrained device using this protocol may fire an event to the WoT server and the WoT server may request something to the constrained device. Proxies between HTTP and CoAP will achieve interoperability between HTTP and CoAP clients. Translation between CoAP and HTTP is easy and straightforward as equivalences of response codes, options, and methods are present in both protocols. Security is achievable using DTLS and a variety of key management methods.

4.2. Other Candidates

4.2.1. DDS. From [27], "DDS (Data Distributed Service) is an API specification and an interoperability wire-protocol that defines a data-centric publish-subscribe architecture for connecting anonymous information providers with information consumers." DDS follows a decentralized pub/sub model. It differs from MQTT model in the following two key points:

(i) DDS protocol starts to operate on top of the link level layer of the OSI model creating a Common Data Bus where every device can connect in a decentralized manner. This protocol also defines several QoS options.

(ii) As a decentralized protocol, it does not have SPF like the broker in MQTT.

DDS has only implementations for C, C++, and Java and has a higher learning curve compared to MQTT. On the other hand, MQTT clients are implemented for several languages.

4.2.2. XMPP. XMPP [28] (originally named Jabber) is a protocol for person-to-person communication based on XML. Its main use is for chat communication but since the growth of the IoT concept the XMPP Standards Foundation is working on defining extensions (XEPs) for use in the IoT [29]. These extensions aim to specify standards for a wide variety of communication types between IoT devices such as Control, Discovery, Multicast, or pub/sub message types. They use EXI [30] (compressed XML) to reduce the size of messages, as XML is known to produce larger file/message sizes than other text based formats. Even though XEPs for the IoT are not as much as popular as MQTT or CoAP, it is worth keeping track of them as they are growing fast and may be used as a basis to model WoT message formats.

4.2.3. AMQP. AMQP [31] is a message-centric binary wire protocol that uses a centralized broker. AMQP is built on top of the TCP layer (at least, it is assumed to work on top of TCP). Authentication and encryption are made available through SASL and TLS, respectively. As AMQP was created by businesses-to-businesses, it provides transactional modes of operation that allow it to take part in a multiphase commit sequence. The key feature of AMQP is that it was designed for interoperability between vendors. It mandates the behaviour of the messaging provider and client to the extent that implementations from different vendors are interoperable.

Third party implementations of AMQP clients exist for several languages. Although AMQP is a great opponent for

MQTT, the latter seems more suitable to build a proof of concept for the WoT architecture we have been talking about in this section.

5. Some WoT Implementations (Related Work)

This section is going to review some WoT implementations of this architecture. Guinard joined the EVRYTHNG platform and engine [32]. This engine allows Things (they call them "thngs") to be connected to this platform through a RESTful API. They describe two types of thngs:

(i) Unconnected/tagged: they are encoded in 1D/2D bar code or NFC/RFID tag and users can interact with them by scanning the tag.

(ii) Connected: those thngs can interact with the RESTful API of the EVRYTHING engine; they can be sensed and/or actuated.

This engine offers the creation of applications that represent remote client applications (like those used in social networks like Twitter or Facebook). Thanks to its THNG-Push technology (currently in beta), the engine provides a publish/subscribe MQTT M2M broker where WebSockets wrapping MQTT are used to allow communication with browsers. They are also working on adding CoAP support to this technology. Node.js and JavaScript libraries are available to facilitate the use of their API.

In [33], the authors propose and implement holistic web architecture for the Internet of Things. They point out key features and capabilities of holistic architecture and they use a layered model with an abstraction layer for communication among devices.

Node-RED [34] is a visual tool for wiring hardware devices, APIs, and services. From the Node-RED front page, "Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range nodes in the palette. Flows can be then deployed to the runtime in a single-click." Also in beta, this tool aims to provide users with a visual manner connecting things. Node-RED is built on Node.js and customized functions between nodes can be created within the editor using JavaScript. There is an EVRYTHNG Node-RED integration library to add some functionality of the EVRYTHING platform to this tool.

Octoblu [35] is an open-source cloud platform (public, private, or hybrid) built to connect people, devices, and systems through a great variety of protocols like MQTT, CoAP, HTTP(S), and WebSockets using a RESTful API. It also offers a very powerful visual tool for connecting things (nodes). This tool also allows developers to program their own nodes as JavaScript functions. Node.js and JavaScript libraries are also available to facilitate the use of their API.

Neura [36] and TempoIQ [37] are platforms for collecting sensor data through a RESTful API. While Neura is more person-oriented, TempoIQ is a general-purpose data collector. TempoIQ (former TempoDB) also offers tools for monitoring and analyzing this sensor data.

Finally, in [38], the concept of storage registration is introduced for the WoT. In this storage approach a web client announces its interest of storing some sensor data to the server. The server will store the data until the web client requests removal or an expiration time is reached. This prevents the server from reaching its storage limit.

6. Proposed Architecture

The purpose of this section is to present the architecture and announce the key parameters used to link the layers. From our real-world experiences collected during the INTEGRIS and FINESCE [13] projects, we have found that dividing the Smart Grid into these logical layers poses some critical difficulties arising from the fact that typically IEDs are closed devices that do not allow implementing custom developments (e.g., security or information-exchange protocols) as novel experimental devices do. Therefore, we proposed a new device coined as I-Dev [14, 18, 22] which behaves as a frontier between these two layers and implements (1) a communications subsystem that allows heterogeneous network coexistence, (2) a security subsystem that provides a reliable and secure low layer communications infrastructure, (3) a distributed storage subsystem that smartly stores all data generated by IEDs, and (4) a cognitive subsystem that is aware of all events arising from any subsystem of the network.

In Figure 4, an architecture is proposed in order to allow Things to communicate with each other. The proposal explanation will be divided into the following sections: specifications, used protocols, and software design.

6.1. Specifications. In this proposal, a Thing can be understood as every device that has minimum communication, processing, and storage capabilities so it is able to be sensed or actuated and can communicate with another device to send or receive data. The proposed architecture has those goals to satisfy:

(i) As standard Web protocols such as HTTP or WebSockets use resources that may not be available on some constrained Things, this architecture will connect them using a variety of protocols (but ideally one) like CoAP or MQTT that are more suitable for constrained devices.

(ii) Every Thing must be using an open standard protocol and must be understandable by the architecture to connect to the WoT. Gateways serving as a proxy for those Things that use proprietary or nonunderstandable protocols should be used.

(iii) Things that cannot connect to the WoT infrastructure directly should connect to a gateway instead and let that gateway do the connection for them.

(iv) This architecture will allow developers to interact with Things without knowing the protocol they are really using to communicate; hence, there is a need to provide developers with a communication abstraction layer.

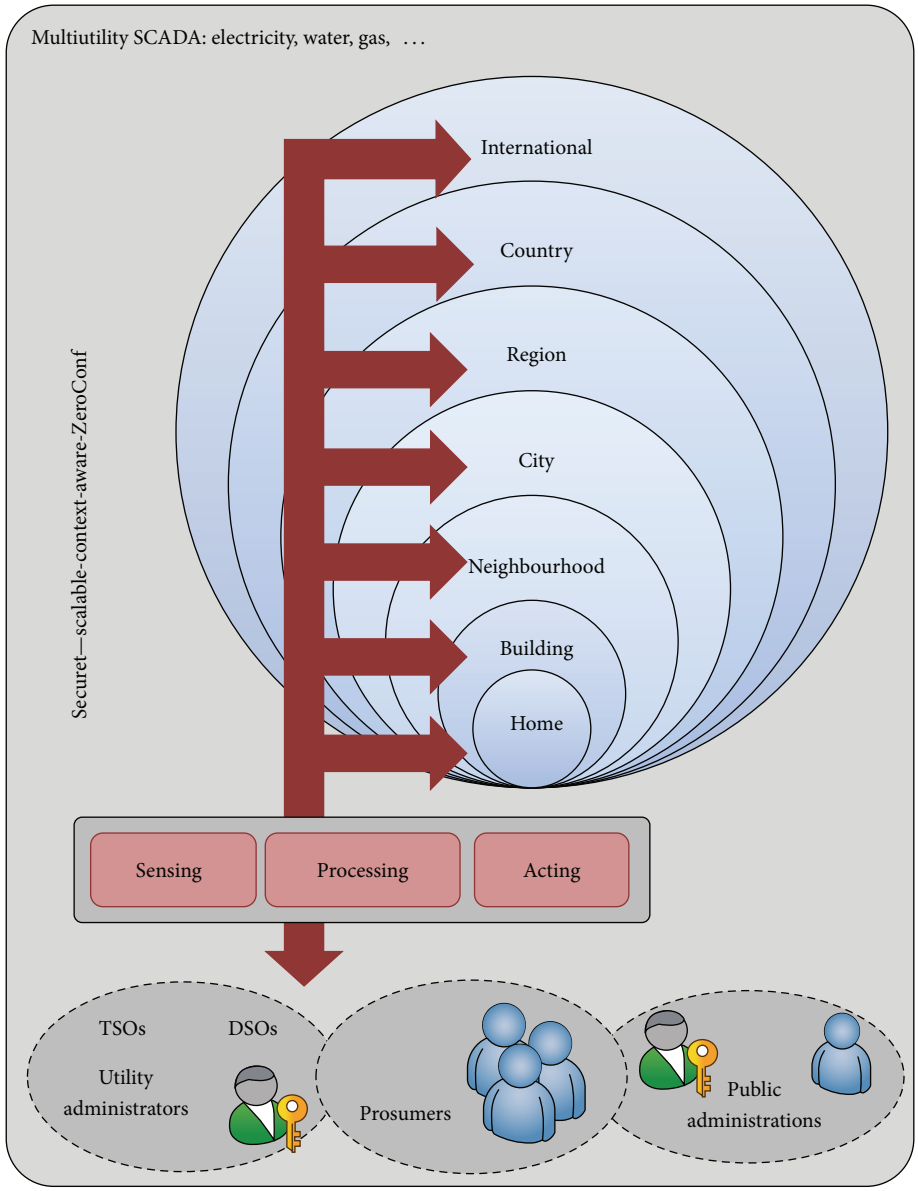

Figure 4: Functional architecture.

(v) It must allow on-demand deployment due to the fact that the number of Things will increase and therefore more resources will be needed to connect them.

(vi) This architecture should be able to balance the load on its servers. Load balancers will prevent or minimize bottlenecks.

(vii) Things will become virtual objects for the architecture and those virtual objects could be aggregated and linked to form other virtual objects.

(viii) Eventually, every virtual object will be accessible by RESTful URI.

(ix) The architecture must have authorization of Things to interact with them to preserve owner's privacy.

(x) Third-party applications must have the architecture authorization to interact with Things.

(xi) Every Thing must have an owner.

(xii) Things can be queried and discovered.

In Figure 5, we can see how Things are connected to servers directly or via a gateway/aggregator being part of the WoT.

6.2. Protocols. While the use of one protocol like HTTP for all the Web of Things would be desirable, it is obvious that it is not achievable due to the heterogeneity of devices. So, a WoT architecture would need interoperability with a variety of protocols to interconnect devices.

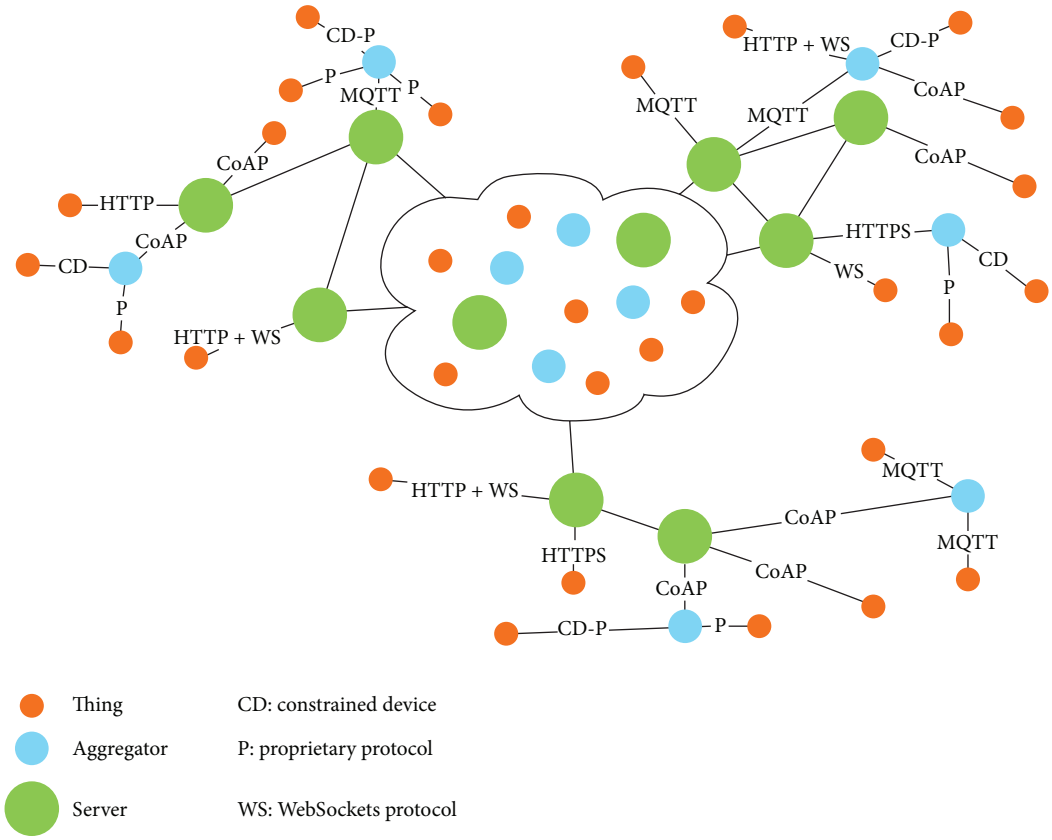

Figure 5: The proposed WoT architecture.

6.2.1. HTTP and WS. As the Web uses HTTP to communicate, it is mandatory for the architecture to support this protocol. It will also provide a RESTful communication using URIs to identify resources and methods to actuate or sense on them.

With IPv4 still in use and incapable of addressing all Internet connected devices, Internet connection to personal computers, for example, is available through port translation mechanisms. This translation makes them incapable of receiving data asynchronously from a server; hence, the use of some sort of mechanism to allow them to receive data in an asynchronous manner is required. In addition, as proposed in [2], WebSockets [12] protocol can overcome this constraint by providing a mechanism for browser-based applications that need two-way communication with servers.

6.2.2. MQTT and CoAP. As described in the previous section, MQTT and CoAP seem to be very suitable protocols for the WoT world. We propose to use these protocols but the architecture should be capable of understanding more protocols (e.g., DDS).

6.3. Layered and Modular Design. To be able to seamlessly intercommunicate these protocols and help developers implement and provide different implementations of functional units (e.g., how a protocol is handled, adding a new protocol to the framework, and how VO is accessed), the architecture relies on a layered and modular design. It is composed of five main layers (Figure 6); they are the following:

(i) WoT Protocol Abstraction Layer: the purpose of this layer is to provide an abstraction mechanism for developers to interact with Things.

(ii) REST to VO/Query: the goal of this layer is to map dynamically generated URIs to virtual objects (VOs). It is an interface to gather information or actuate on VOs, that is, Things. It is also an endpoint for querying Things like the temperature on a specific location.

(iii) Auth: this layer is responsible for requesting and granting access between Things. As mentioned in Section 4, every Thing must have an owner; therefore, WoT servers must request access to Things and VOs must request access to other VOs.

(iv) Virtual Object Space: this is the space for virtual objects. Things will become virtual objects in the virtual world. Those VOs could be then sensed, actuated, and aggregated.

(v) Proxy layer: this abstraction layer will serve as a proxy for servers to communicate. A protocol for this communication must be defined for the architecture.

Those layers will be backed up by the database and the reasoner [39] modules. As the WoT architecture will generate

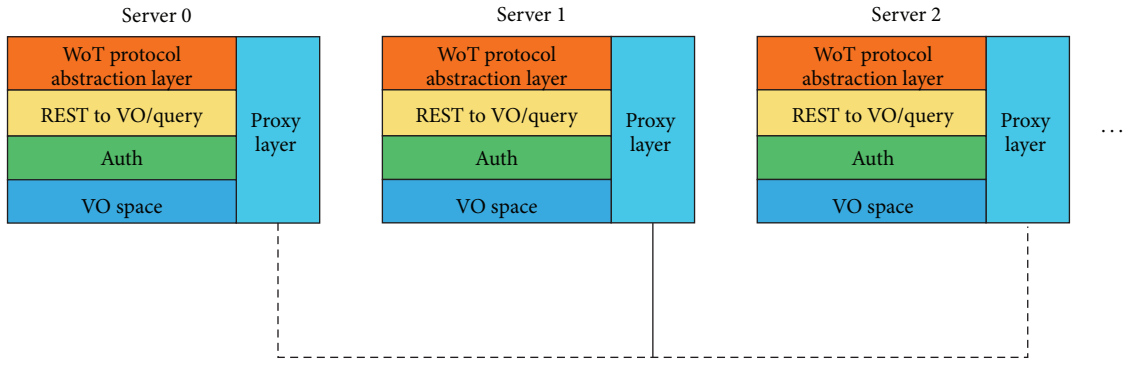

Figure 6: Layered and modular design.

URIs to address Things, a distributed database will be suitable for routing request based on their URI. Given a query, servers can first search in this database if this resource has been discovered beforehand. If this were the case, no discovery protocols would be needed; otherwise, a discovery protocol would start a search to find the resource and once discovered the resource URI would be stored in the distributed database. Servers may also have local storage to implement the storage registration mechanism.

The WoT architecture presented in this paper is based on the architecture proposed by Guinard. This section is focused on the architecture but we will highlight main similarities between two architectures.

As Guinard stated in his thesis, the upper layers presented in Figure 7 do not hide lower layers and instead they are development layers where users with different technical knowledge can develop applications on top of them.

On the other hand, the definition of each of the layers of the proposed architecture and its correspondence with Guinard's proposal would be as follows.

6.3.1. Virtual Object Space. The layer that shares more similarities in Guinard's architecture is the Composition Layer, but in his approach this layer is a Physical Mashup where web services and enabled smart devices can create composite applications. In the proposed system and as in [24], this layer serves the purpose of digitally enabling smart devices for their use by the architecture and by WoT users. Then, applications built on the upper layer of the architecture (e.g., Physical Mashups) could use these digitally enabled smart devices.

6.3.2. Auth. This layer shares the same purpose as the Sharing Layer exposed in Guinard's thesis. The goal is to preserve the privacy of each Thing, allowing Things to have owners that share the capabilities of their Things. A great approach to this authentication and authorization layer would be the same as the one proposed in [17].

6.3.3. REST to VO/Query. The goal of this layer is to dynamically generate meaningful, RESTful URIs for VOs as they are queried for the first time. Once the device has a meaningful base URI, its capabilities can be exposed through expanding its URI and the common HTTP verbs. Querying Things will involve complex semantic processing and a process such as the one described in [32].

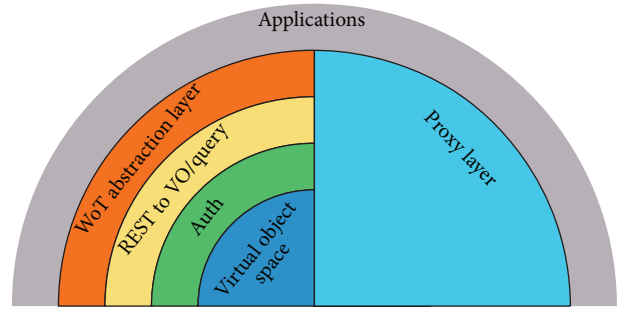

Figure 7: Applications can be developed over the upper layers.

6.3.4. WoT Protocol Abstraction Layer and Proxy Layer. As several protocols can join the WoT architecture, there is a need to unify those protocols at the entrance of the proposed architecture (WoT Protocol Abstraction Layer), translating those protocols to an internal language. This will allow the development of the inner layers of the architecture regardless of the protocol Things are using to connect to the architecture. This layer would accomplish the functions of the Accessibility Layer proposed by Guinard. In Figure 8, the correspondence between the layers defined by Guinard and the layers of the proposal is presented.

As the inner architecture should be agnostic from the outside protocols (those used by Things and the architecture to exchange information), there is a need to develop an internal format and mechanism (proxy layer) to pass messages between different nodes of the architecture. The next section shows a proof of concept of this idea. Firstly, the used dataset for testing is described and, secondly, the discovery algorithm is depicted. Finally, two implementation approaches are presented and the results obtained are discussed.

7. Proof of Concept

7.1. Dataset. The data used for performing the tests of this proof of concept come from public data from the National Statistics Institute of Spain (http://www.ine.es) concerning

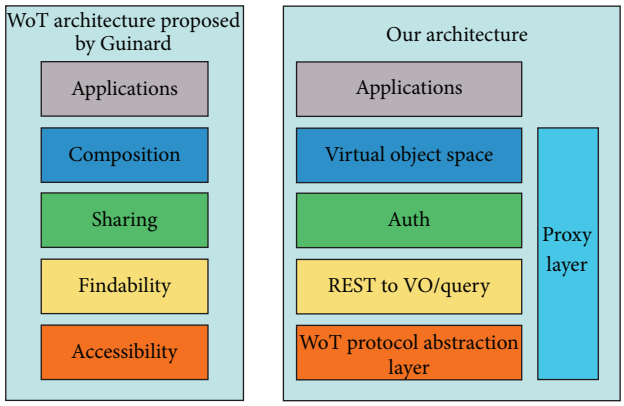

Figure 8: Comparison between the proposed architecture and Guinard's one.

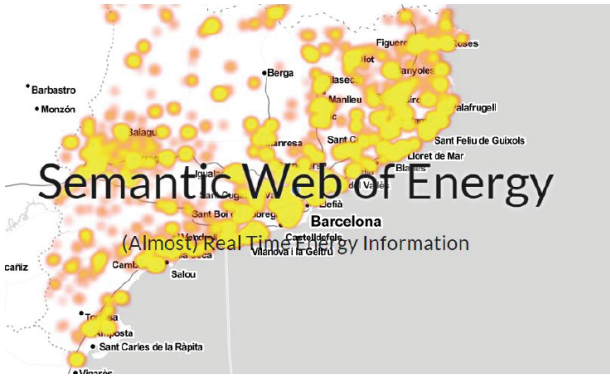

Figure 9: Snapshot of the representation of the devices response.

cities/towns and inhabitants of Spain's municipality. Specifically, all the inhabitants of Catalonia region have been accounted (about 7.5 million), taking into account the following two assumptions:

(i) In each house live an average of three people.

(ii) Only 1% of households have some functional and Internet-controllable device.

Each house has been assigned to a city, and then a device is randomly assigned to each house. Different types of sensors are chosen, including temperature sensors and electricity meters.

For each city, it has been possible to obtain geolocation data (longitude and latitude) and it has been geographically represented on the map. In Figure 9, a snapshot (from the created application, called Semantic Web of Energy) can be observed. The control of these devices is displayed, indicating in color the answer to a request for consultation (Discovery of Thing).

7.2. Algorithm. Based on the assumptions outlined above, a first version of the architecture presented in this paper is implemented. In this sense, we have implemented the first "Discovery of Things" functionality. It is simulated by the example of how to access a temperature sensor using the steps described in the following algorithm:

(1) User A sends a request R1 to its local dispatcher node D1 (located in Madrid).

`P1 = {"action": "GET", "what": "temperature", "loc": "Carrer de Sants"}`

(2) D1 processes de address ("Carrer de Sants"), searches for the Barcelona IP address and send a response to user A.

`R1 = {"request": {"what": "ws-conn", "to": "@IP-BarcelonaNode"}}`

(3) User A sends a request P2 to the node with the IP address @IP-Barcelona.

`P2 = {"request": {"what": "ws-conn"}}`

(4) Barcelona's D2 dispatcher rececives P2 and sends a response R2.

`R2 = {"uri": {"method": "GET", "uri": "/ws/id/1"}}`

(5) User A connects to @IP-Barcelona/ws/id/1 via WebSockets (WS1 connection) and resends P1 through this connection. This connection will be managed by N1 node (server).

(6) N1 processes the requests and searches for an address to forward the request and receive the temperature of Carrer de Sants. N1 has the private address of an aggregator/gateway (A1) that can provide the result of the query. N1 sends a CoAP request P2 to A1.

`P2 = {"action": "GET", "what": "temperature"}`

(7) A1 processes the request and sends a response R3 to N1. `R3 = {"uri": {"method": "GET", "uri": "/temperature"}, "data": {"value": 15, "unit":"Celsius"}}`.

A1 also saves an URI map to its local database: `saveUriMap: "(GET)/temperature" -> (function to get data)`

(8) N1 receives R3 response and saves to its local database: `saveUriMap: "/carrer-de-sants/" -> (GET)@IP-A1/temperature`.

N1 also saves a query map to the distributed database: `saveQueryMap: "Carrer de Sants, temperature" -> "(GET)@IP-N1/carrer-de-sants/temperature"`.

N1 sends a R4 response to user A.

`R3 = {"uri": {"method": "GET", "uri": "/carrer-de-sants/temperature"}, "data": {"value": 15, "unit": "celsius"}}`.

Moreover, if N1 has users that need the same data, sends R4 to these users too.

(9) User A receives the requested data and a URI to identify the resource and made future requests to it.

If the same user A wants to request the same data again, he will send a URI request ((GET)@IP-N1/carrer-desants/temperature) directly to N1 and as query and URI maps are now present the response will return faster as less processing time will be needed.

If user B sends the same request P1 to D1, D1 will respond saying that user B must request a WebSocket connection to N1 and send a request to this URI "(GET)@IP-N1/carrer-desants/temperature."

As shown, the framework is capable of handling the discovery of the new object and further processing and storage in the system for future reference.

7.3. Implementation. In this section two implementation approaches are compared.

7.3.1. First Approach. As a lot of services found in Web 2.0 are implemented using the PHP language (http://trends.builtwith.com/framework) and in order to ease Web of Things applications at every layer for experienced PHP developers, we have developed a prototype implementing the algorithm presented in this section using the PHP language. Although we have succeeded in developing basic functionalities using PHP, several problems have arisen during the implementation.

PHP was born to serve CGIs, for example, to serve HTTP requests with dynamic content. The main workflow where PHP has been used consists in three steps: load, execute, and die. For this reason, few efforts have been made to solve problems such as memory leaks or the fact that executing one PHP statement requires more low-level instructions than the actually needed ones.

In recent years, PHP has undergone some changes in its usages and performance:

(i) With the arrival of Node.js [40], some PHP developers started to implement libraries with the objective of allowing PHP users to create servers in this language as in React PHP [41] instead of relying on web servers like Apache HTTP.

(ii) Compiled PHP frameworks such as the Phalcon Framework [42] with a high performance boost on execution time have motivated the release of PHP 7, solving memory leaks and decreasing the number of low-level instructions needed to execute a PHP statement.

In order to build the prototype, we have used the Phalcon Framework and React PHP to boost performance. The Phalcon Framework presents high speed in performing operations and React PHP presents a novel manner for building PHP applications using the reactor pattern and asynchronous programming. However, the immaturity of React PHP and the lack of asynchronous libraries in PHP present an obstacle for developers to use even basic technologies such as MongoDB.

In this way, PHP has succeeded in building and fast-prototyping Web applications thanks to its low learning curve and its dynamic typing. However, its use at low-level/core layers in a WoT/IoT architecture where program correctness is crucial facilitates the appearance of execution time errors.

Table 1: Benchmarking PHP versus Scala languages.

Feature	PHP	Scala
Natural workflow	Load, execute, die	Always running on a JVM
Speed	1	28 times faster
Backed by research	No	Yes
Libraries	PHP, C extensions	JVM compliant libraries
Distributed libraries	As C extensions	Akka and other JVM libraries
Async. libraries/constructs	ReactPHP	Akka and Futures
Typing	Dynamic	Static

7.3.2. Second Approach. After evaluating other solutions in the market, the Scala [43] language has been selected for reimplementation of the prototype. Although this language usage is not wider than PHP, Scala is experiencing an adoption growth (http://www.indeed.com/jobtrends?q=scala&l=).

Moreover, this language presents key characteristics that make it suitable for building a future WoT prototype architecture. Scala has also been chosen due to its research community. In fact, Scala was born at the EPFL (École Polytechnique Fédérale de Lausanne) thanks to Martin Odersky [43]. Therefore, continuous investigation is being made in order to optimize speed and provide better APIs.

7.3.3. Comparison Table. A comparison between PHP and Scala is presented in Table 1. Data presented in this comparison table have been extracted from the authors' experimentation. Speed has been extracted from benchmarks done in [44].

8. Conclusions and Further Lines

A new framework approach and a proof of concept have been presented in this paper. It has been shown that the proposed architecture is feasible and that the implementation of successive parts can be made using this design.

Although the results exposed are promising, we have realized that PHP lacks libraries and implementation for the most relevant WoT protocols. If there is a valid implementation, it only covers basic features of the protocol.

Moreover, the reactor approach was performed to prototype the architecture using ReactPHP and it was found that there are no libraries to connect the most commonly used databases in distributed systems like Redis or MongoDB or the WoT protocols.

For the reasons exposed, it seems that PHP is not mature enough for the purpose, that is, to develop holistic architecture for the Web of Things. Reimplementation of the architecture using the Scala language has been made, speeding up its performance and opening up the possibility

to take advantage of robust libraries and frameworks built on top of JVM compliant languages.

Heterogeneity, parallelization, and distribution as explained in [45] are also key characteristics of a WoT architecture. More work has to be done to fully achieve these characteristics. The Actor Model [46] seems to be well suited to build an architecture with such characteristics as asynchronous messaging, location transparency, distribution, and concurrency as its core principles.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work received funding from the European Union's 7th Framework Program under FI.ICT-2011 Grant no. 604677-FINESCE (Future INtErnet Smart Utility ServiCEs).

References

[1] J. Navarro, A. Sancho, A. Zaballos, V. Jiménez, D. Vernet, and E. Armendáriz-Iñigo, "The management system of INTEGRIS—extending the smart grid to the web of energy," in *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER '14)*, pp. 329–336, Barcelona, Spain, 2014.

[2] MongoDB Database, https://www.mongodb.org/.

[3] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the internet of things to the web of things: resource-oriented architecture and best practices," in *Architecting the Internet of Things*, pp. 97–129, Springer, Berlin, Germany, 2011.

[4] A. Zaballos, A. Vallejo, and J. M. Selga, "Heterogeneous communication architecture for the smart grid," *IEEE Network*, vol. 25, no. 5, pp. 30–37, 2011.

[5] INTEGRIS, INTEGRIS FP7 Project INTelligent Electrical Grid Sensor communications, ICT-Energy-2009 call (number 247938), http://fp7integris.eu.

[6] C. Bo, C. Xin, Z. Zhongyi, Z. Chengwen, and C. Junliang, "Web of things-based remote monitoring system for coal mine safety using wireless sensor network," *International Journal of Distributed Sensor Networks*, vol. 2014, Article ID 323127, 14 pages, 2014.

[7] S. Aman, Y. Simmhan, and V. K. Prasanna, "Energy management systems: state of the art and emerging trends," *IEEE Communications Magazine*, vol. 51, no. 1, pp. 114–119, 2013.

[8] A. Zaballos, D. Vernet, and J. M. Selga, "A genetic QoS-aware routing protocol for the smart electricity networks," *International Journal of Distributed Sensor Networks*, vol. 2013, Article ID 135056, 12 pages, 2013.

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

[10] D. Zeng, S. Guo, and Z. Cheng, "The web of things: a survey (invited paper)," *Journal of Communications*, vol. 6, no. 6, pp. 424–438, 2011.

[11] E. Bou-Harb, C. Fachkha, M. Pourzandi, M. Debbabi, and C. Assi, "Communication security for smart grid distribution networks," *IEEE Communications Magazine*, vol. 51, no. 1, pp. 42–49, 2013.

[12] V. C. Gungor, D. Sahin, T. Kocak et al., "A survey on smart grid potential applications and communication requirements," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 28–42, 2013.

[13] FINESCE, European Union's 7th Framework Program under the FI.ICT-2011 Grant number 604677, http://www.finesce.eu/.

[14] J. Navarro, A. Zaballos, A. Sancho-Asensio, G. Ravera, and J. E. Armendariz-Inigo, "The information system of INTEGRIS: Intelligent electrical grid sensor communications," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1548–1560, 2013.

[15] A. Bari, J. Jiang, W. Saad, and A. Jaekel, "Challenges in the smart grid applications: an overview," *International Journal of Distributed Sensor Networks*, vol. 2014, Article ID 974682, 11 pages, 2014.

[16] J. Rodríguez-Molina, M. Martínez-Núñez, J.-F. Martínez, and W. Pérez-Aguiar, "Business models in the smart grid: challenges, opportunities and proposals for prosumer profitability," *Energies*, vol. 7, no. 9, pp. 6142–6171, 2014.

[17] D. Guinard, *A web of things application architecture [Doctoral dissertation]*, Eidgenössische Technische Hochschule ETH, Zürich, Switzerland, 2011, Nr. 19891.

[18] J. M. Selga, G. Corral, A. Zaballos, and R. Martín de Pozuelo, "Smart grid ICT research lines out of the European project INTEGRIS," *Network Protocols and Algorithms*, vol. 6, no. 2, 2014.

[19] Y. Oualmakran, J. Meléndez, S. Herraiz, M. López-Perea, and E. González, "Survey on knowledge based methods to assist fault restoration in power distribution networks," in *Proceedings of the International Conference on Renewable Energies and Power Quality (ICREPQ '11)*, Las Palmas, Spain, April 2011.

[20] C. Miller, *The Fractal Grid: Achieving Grid Security, Reliability, and Resiliency through Advanced Analytics and Control*, National Rural Electric Cooperative Association, 2013.

[21] C. Miller, M. Martin, D. Pinney, and G. Walker, *Achieving a Resilient and Agile Grid*, National Rural Electric Cooperative Association, Arlington, Va, USA, 2014.

[22] A. Sancho-Asensio, J. Navarro, I. Arrieta-Salinas et al., "Improving data partition schemes in Smart Grids via clustering data streams," *Expert Systems with Applications*, vol. 41, no. 13, pp. 5832–5842, 2014.

[23] Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020, Stamford, Conn, USA, December 2013, http://www.gartner.com/newsroom/id/2636073.

[24] E. Oriwoh and M. Conrad, "'Things' in the internet of things: towards a definition," *International Journal of Internet of Things*, vol. 4, no. 1, pp. 1–5, 2015.

[25] MQTT, Retrieved March 27, 2015, http://mqtt.org/.

[26] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: an application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.

[27] Data Distribution Service Portal, April 2015, http://portals.omg.org/dds/what-is-dds-3/.

[28] The XMPP Standards Foundation (n.d.), June 2015, http://xmpp.org.

[29] Tech pages/IoT Xeps Explained, June 2015, http://wiki.xmpp.org/web/Tech_pages/IoT_XepsExplained.

[30] Efficient XML Interchange Working Group. (n.d.), June 2015, http://www.w3.org/XML/EXI/.

[31] AMQP, April 2015, https://www.amqp.org/.

[32] EVRYTHNG—Every Thing Connected, April 2015, https://evrythng.com.

[33] D. Tracey and C. Sreenan, "A holistic architecture for the internet of things, sensing services and big data," in *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid '13)*, pp. 546–553, Delft, The Netherlands, May 2013.

[34] Node-RED, April 2015, http://nodered.org/.

[35] Octoblu, "We make all APIs, Platforms and Devices talk to each other. Easily," April 2015, http://octoblu.com/.

[36] Neura, April 2015, http://www.theneura.com/.

[37] TempoIQ, April 2015, https://www.tempoiq.com.

[38] G. Bovet and J. Hennebert, "A web-of-things gateway for KNX networks," in *Proceedings of the European Conference on Smart Objects, Systems and Technologies (SmartSysTech '13)*, pp. 1–8, VDE, Erlangen, Germany, June 2013.

[39] B. Christophe, V. Verdot, and V. Toubiana, "Searching the 'web of things,'" in *Proceedings of the 5th IEEE International Conference on Semantic Computing (ICSC '11)*, pp. 308–315, IEEE, Palo Alto, Calif, USA, September 2011.

[40] Node.js, June 2015, https://nodejs.org/.

[41] ReactPHP, June 2015, http://reactphp.org/.

[42] A Full-stack PHP Framework Delivered as a C-extension, Phalcon PHP, June 2015, https://phalconphp.com/.

[43] Scala, "The Scala Programming Language," June 2015, http://www.scala-lang.org.

[44] The Computer Language Benchmarks Game, 2015, http://benchmarksgame.alioth.debian.org.

[45] M. Odersky, P. Altherr, V. Cremet et al., "An overview of the Scala programming language," Tech. Rep. LAMP-REPORT-2004-006, 2004.

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