Internet of Things 1–2 (2018) 99–112

Contents lists available at ScienceDirect

Internet of Things

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

Internet of things reference architectures, security and interoperability: A survey

B. Di Martino*, M. Rak, M. Ficco, A. Esposito, S.A. Maisto, S. Nacchia

Dipartimento di Ingegneria, Università della Campania Luigi Vanvitelli, Aversa (CE), Via Roma 29, Italy

ARTICLE INFO

Article history:
Received 3 August 2018
Accepted 18 August 2018
Available online 30 August 2018

Keywords:
IoT
Reference Architectures
Security
Interoperability

ABSTRACT

The term Internet of Things (IoT) is used as an umbrella that covers several topics, related to the application of technological means to monitor, measure and act upon the environment. As a result, it is difficult to determine a univocal architecture to identify as a reference and several scenarios, involving different sensors, smart devices, networks or gateways, can unfold. The data exchanged within and among IoT frameworks are growing exponentially, and the pervasiveness of such systems brings them to come in possession of very sensitive information: as a consequence, Security and Privacy have become a hot topic on the IoT scenery. Furthermore, due to the great variety of technological solutions which are currently available, interoperability issues are bound to arise, especially when no standard API interface, or communication protocol, has been officially adopted.

This paper provides a review of the most common architectural solutions available today to shape an IoT system, ranging from already standardized architecture to commercial ones. Elements from such architectures have been compared, analysed and mapped one against the other to determine a stable reference for Security and Interoperability analysis. Current solutions in the Security and API Interoperability domains for IoT have been also analysed.

1. Introduction

Intelligent sensors and devices have become part of our every-day life, as we continuously interact more or less knowingly with smart items. *Internet of Things* (IoT) has become a pervasive reality: while smart-phones, with all the sensors they carry, still represent our main means of connection to the Internet, other intelligent items are currently at our disposal and sometimes they are so blended with the environment that we hardly notice them.

In the modern world, where each item and living being is or can be connected to others and share data through smart sensors and devices, the Internet of Everything has become a new buzz-world, and great attention has aroused for the huge variety of possible applications of such technologies. At the same time, concerns regarding privacy and security have also arisen.

The term Internet of Things has been coined by industry researchers, but it has been rapidly exposed to the public view, becoming a mainstream topic. In a few words, IoT represents the ability of intelligence devices to sense, collect (and sometimes even analyse in loco) data from the world around us, eventually share that data across the Internet or a local

* Corresponding author.
E-mail address: beniamino.dimartino@unina.it (B. Di Martino).

https://doi.org/10.1016/j.iot.2018.08.008
2542-6605/

network, and then act according to the received information. Sharing is one of the main feature of IoT devices, as only in this way data can be efficiently processed and utilized for various interesting purposes.

Numerous and outstanding applications have been developed by employing smart devices:

- Self-parking cars, which exploit complex proximity systems, are already being commercialized.
- Wearable devices and phones can be used to track people's movement, register their exercise habits and daily activities. Wearable devices, equipped with special sensors, can detect physical dangers and communicate it to the wearers or to people in the proximity, via the Internet.
- Vehicle tracking is another common application of IoT devices: once equipped with ad-hoc GPS devices or common smartphones, it is possible to determine the position of delivery trucks or public transports, or to detect traffic congestions in specific areas.
- Domotics appliances can exploit combined sensors' data to provide real *Smart Home* experiences. Devices that automatically tune indoor temperature according to people demands and environmental conditions can be already found in many homes. Intelligent light bulbs, which are turned on and off according to the movements of people detected by sensors inside and outside rooms, or activated through voice commands, are becoming quite common.

However, since vendors have different approaches to how devices can interact among them and with the outside world, several scenarios can unfold even when considering same-purpose devices with very similar functionalities and structure. Depending on the vendor's politics, the IoT devices may or may not be allowed to directly connect to each other or to the Internet: there are many cases in which they can only interact with a *Gateway* device, which acts as a proxy for all requests. Furthermore, the back-end services exploited by the devices could be provided by a *Cloud platform* or by a local/private server. This paper will explore some of the possible scenarios, backed up by real-world examples, in Section 2.4. Considering the wide range of subjects touched by the IoT development, it is obvious that many concerns and issues have arisen regarding different topics.

The first and more immediate questions arisen by IoT regard the *Privacy of personal data*. Wearables, smartphones and other Internet connected devices deliver a lot of information about ourselves: physical location, updates about our weight and blood pressure, our relationships with people we are connected to, and more detailed data about ourselves are continuously streamed over wireless networks and potentially around the world. *Security* also represents a prominent topic: means to protect personal information from malicious sniffing are necessary to avoid data leaking, and security measures are enacted to protect IoT systems from external attacks which may compromise their functionality.

Energy consumption is another topic which IoT experts often have to deal with. Connected devices require energy to work and sustain communications. However, supplying power to this new proliferation of IoT devices and their network connections can be expensive and logistically difficult. Portable devices require batteries that need to be periodically recharged, or to be eventually replaced. Even devices optimized for lower power usage need energy to work, and the cost to supply such power to billions of different electrical components is potentially huge.

As it will be further clarified in this paper, another prominent challenge regards the interoperability issues which inevitably arise when considering the interactions among devices produced by different vendors. Depending on the specific communication level (are we considering interactions among devices, gateways or applications depending on them? Or even a combination thereof?) different interoperability issues may arise, which can be addressed and dealt with differently.

The reminder of this paper is organized as follows: Section 2 describes the most common Reference Architectures which have been defined, as of today, in the IoT ecosystem, taking in consideration both standardized and commercial architectures; Section 3 focuses on Security issues which may arise when interacting with an IoT system, on different levels of the reference architectures; Section 4 describes the current solutions to API interoperability issues at application level, emphasising the role of REST APIs; Section 6 concludes the paper.

2. IoT reference architectures

The term IoT is an umbrella that covers different technologies and various application domains. As a consequence there is a great variety of different solutions and the terms adopted vary from one technological solution to the other. We compared the different reference architectures [1] proposed in scientific papers, existing standards and white-papers published by main vendors (Microspft, SAP, Intel, WS2O).

Recently, a few survey papers [1–3] proposed a definition of IoT system and outlined the main research issues. In particular, Borgia [1] proposed an interesting way of representing the IoT technological stack, illustrated in Fig. 1. According to the author's opinion, is very helpful in outlining the interoperability and security issues typical of the IoT environment.

The stack proposed by Borgia is made of six different layers: the bottom three layers (Sensing, Short-range Communication, Gateway access) outline a very common IoT network: (smart) devices interconnected through a short range wireless network (like Bluetooth or Zigbee). The nodes of such networks are typically low-power devices and they communicate with a gateway, which enables their connection with high bandwidth networks (the fourth layer from the bottom in Fig. 1).

[1] A reference architecture, provides a template solution for an architecture for a particular domain. It also provides a common vocabulary with which to discuss implementations, often with the aim to stress commonality.

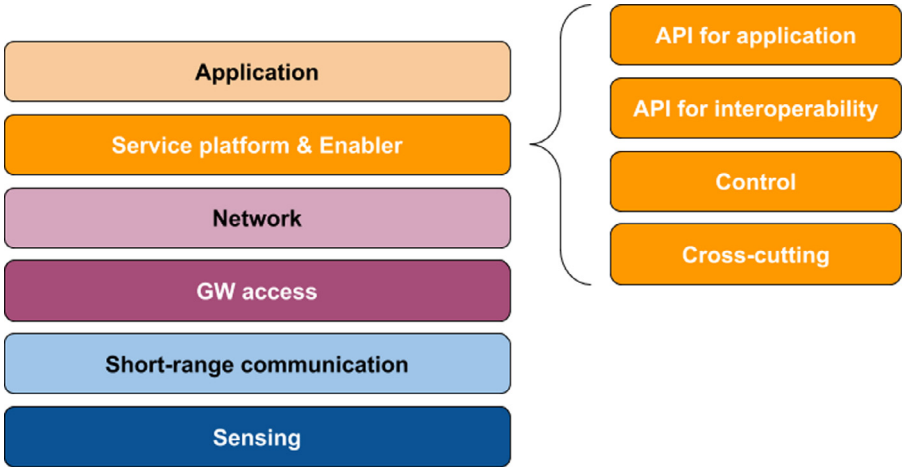

Fig. 1. IBorgia et al. proposed layers.

The fifth layer in the architecture is the IoT Service platform and Enabler: it includes all the software and services devoted to control the IoT system(s), offering API to the applications and granting non-functional requirements like safety, security, availability .... To conclude, the top layer (Application) includes the application domain services, such as services and software devoted to smart home or to e-Health.

Borgia's layers offer an interesting IoT functional view [2], able to catch the two main features of an IoT system: interaction between a local, personal network of low-power devices and the typical high-bandwidth, high computation power systems.

According to such a consideration, we adopted Borgia's layering as a reference to analyze and compare existing standards and the reference architectures publicly proposed by the main vendors.

2.1. IoT standards

As far as IoT is concerned, the applications have been based on fragmented software implementations for specific systems and use cases. The big need for reference architectures in industry has become tangible with the fast-growing number of initiatives working toward standardized architectures. Standard initiatives aim at facilitating interoperability, simplifying development, easing implementation, and identifying possible threats and vulnerability issues. Among the standardization processes in course, we considered ITU-T [4] and ISO [5] as the most interesting.

Fig. 2 summarizes the ITU-T functional view and compares it to the layering proposed by Borgia.

It is worth noticing that ITU-T proposes a four layer architecture with two vertical (cross-cutting) layers. The lower layer includes the functionalities that Borgia distributed among two different layers, while the networks are not differentiated among short-range communications and more common network layers. Application and Service platform layers remain the same in the two proposed architectures.

Similarly, ISO [6] organizes the functional view, reported in Fig. 3, in Domains, assuming three domains at the same level (Operation, Application service, IOT resource and Interchange) on top of the sensing layer. As typical in ISO reference architectures, many cross-cutting (vertical) layers are assumed (not reported in the figure for space motivations).

ISO domains offer a very detailed mapping of the functionalities typical of an IoT system, describing more in details even the typical services and functionalities that should be offered by an IoT service platform. It is worth noticing, however, that even in this case the networks are not differentiated among short-range and high bandwidth, like Borgia proposes. Fig. 3 summarizes the ISO functional view and the mapping against Borgia layers.

In order to simplify the layers analysis, Table 1 summarizes the layer mapping illustrated in the pictures.

2.2. Vendors reference architecture

All the vendors that aim at producing devices and solutions devoted to the IoT market, propose a reference architecture in order to outline how to setup an IoT application relying on the hardware/software components they produces. Figs. 4–7 illustrate the reference architecture proposed respectively by Microsft in [7], by Intel in [8], by SAP in [9], and by WS2o in [10]

Microsoft architecture, illustrated in Fig. 4, proposes a very simple reference architecture made of only three layers: Device connectivity, that includes all the functionalities of the bottom four layers proposed by Borgia, Data processing,

[2] According to ISO a Functional View is a technology-agnostic view of the functions necessary to form an IoT system.

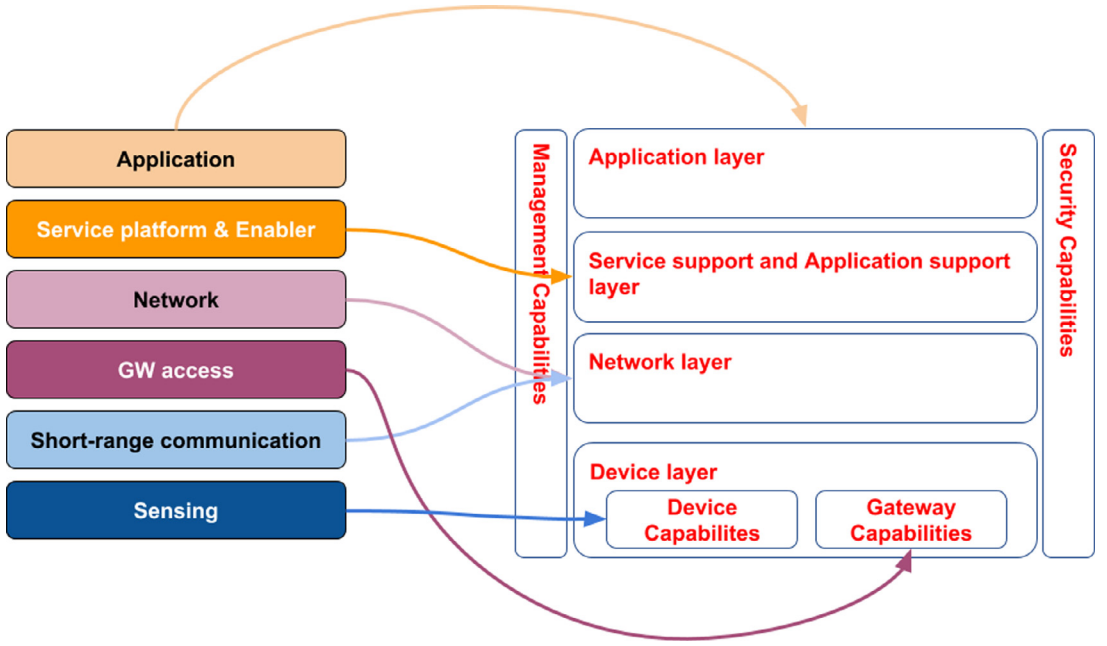

Fig. 2. ITU-T functional view.

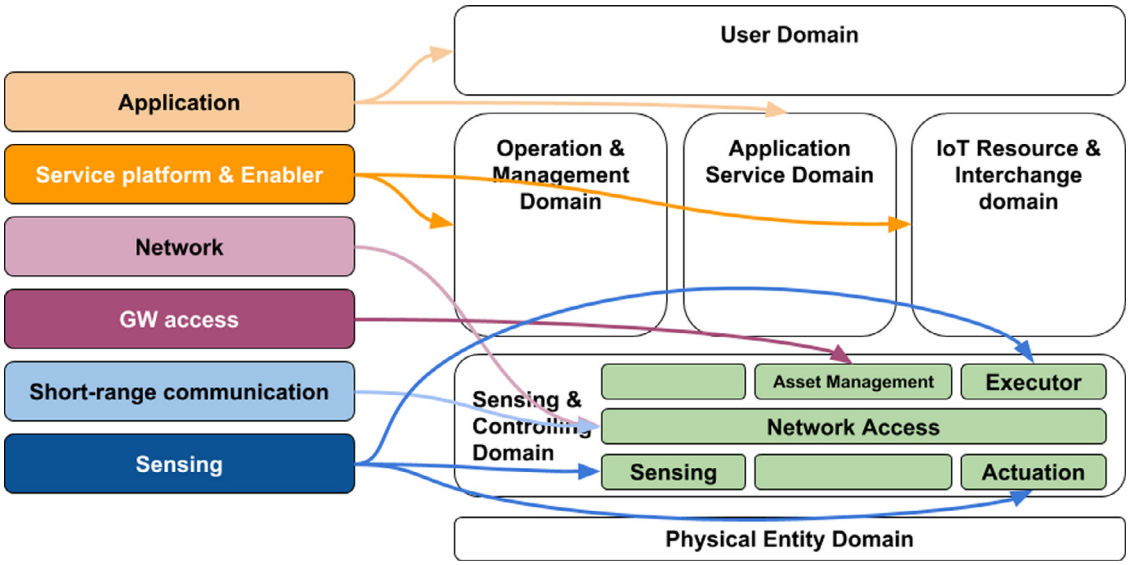

Fig. 3. ISO functional view.

Table 1
Mapping among Borgia layers and existing standards.

Borgia Layers	ISO	ITU-T
Sensing	Sensing, Actuation, Executor	Device capabilities
Short-range communication	Network access	Network layer
GW Access	Asset management	Gateway capabilities
Network	Network access	Network layer
Service platform & enabler	Operation & Management Domain,IoT Resource & Interchange domain	Service support and application support layer, Management capabilities
Application	User Domain,Application service domain	Application layer

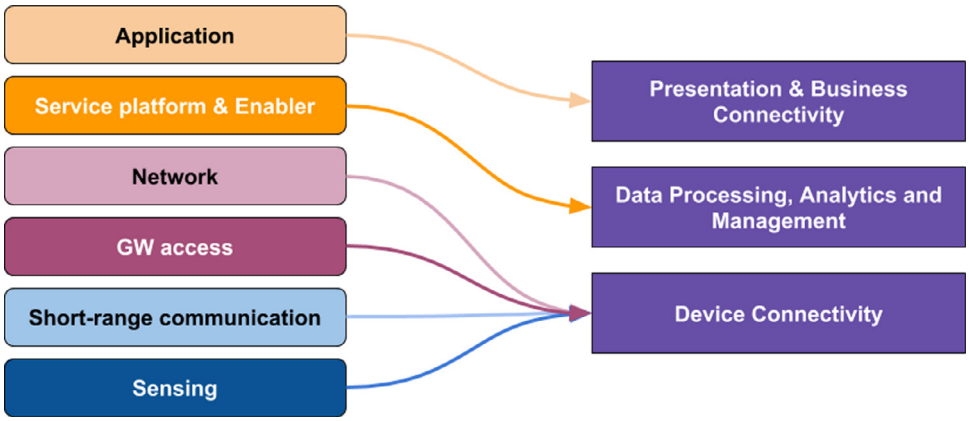

Fig. 4. Microsoft functional View.

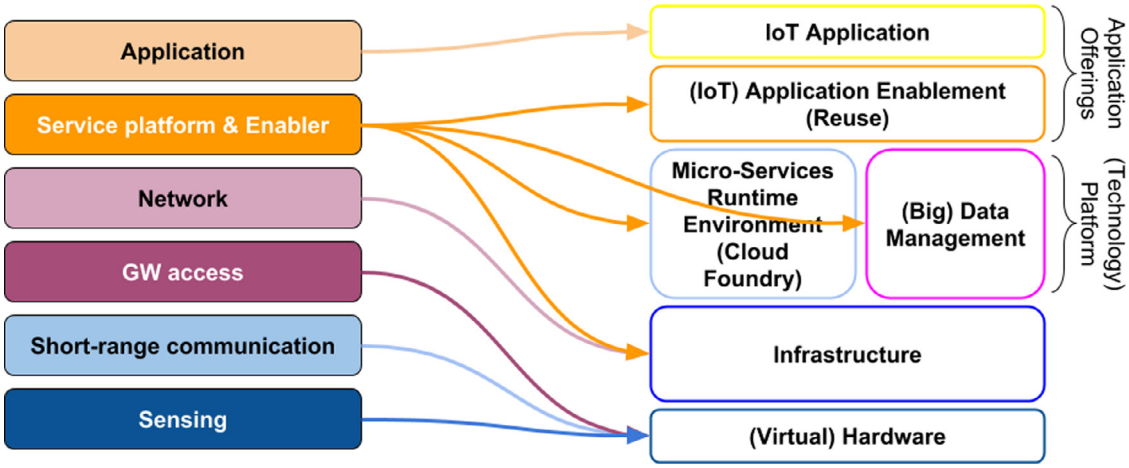

Fig. 5. SAP functional View.

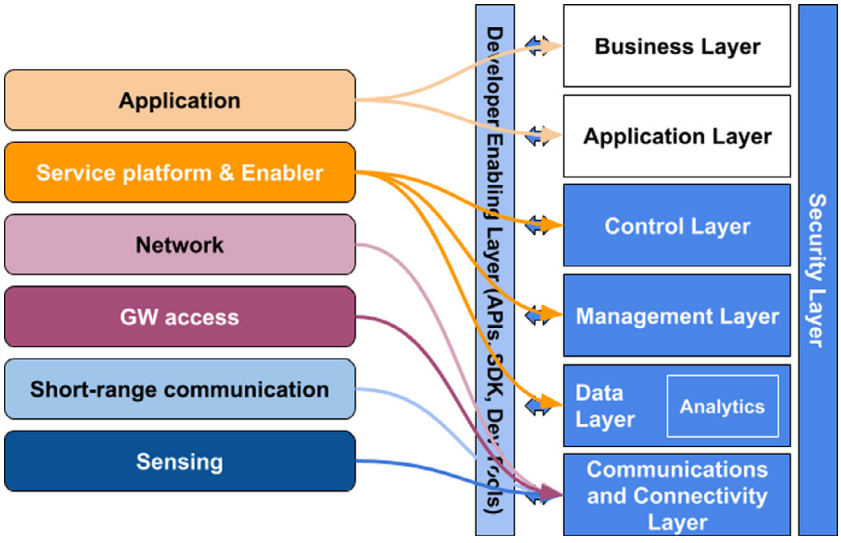

Fig. 6. Intel functional View.

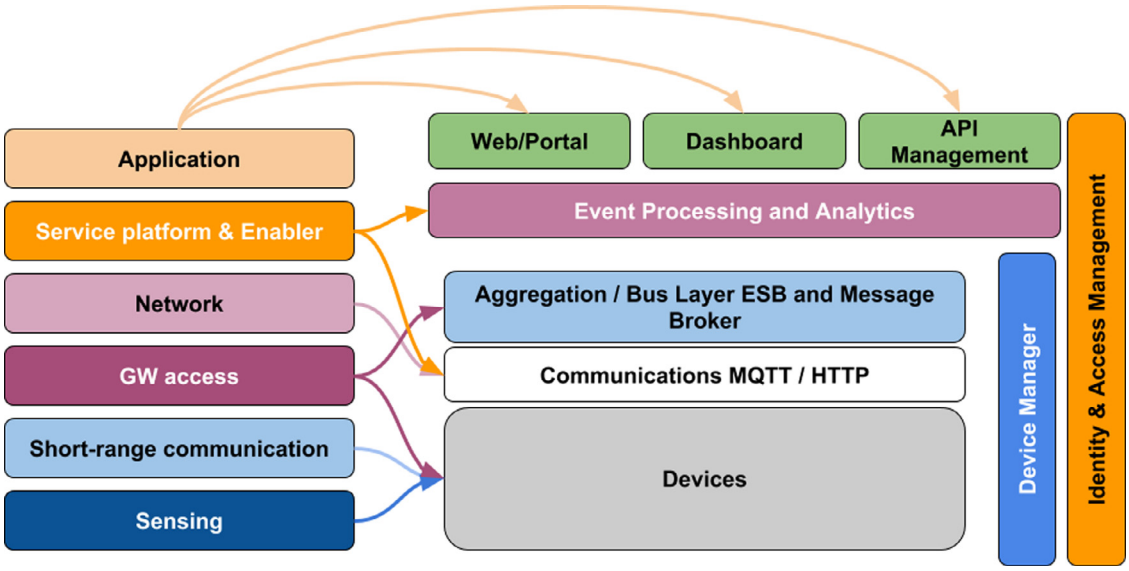

Fig. 7. WS2O functional View.

Table 2
Vendors reference architecture, respect to the Borgia layers.

Borgia Layers	Intel	WS20
Sensing	Communications and connectivity layer	Devices
Short-range communication	Communications and connectivity layer	Devices
GW Access	Communications andconnectivity layer	Devices, Aggregation/Bus Layer ESB and Message Broker
Network	Communications and connectivity layer	Communications MQTT/HTTP
Service platform & enabler	Control layer, Management layer, Data layer	Event Processing and Analytics, Communications MQTT/HTTP
Application	Business layer, Application layer	Web/Portal, Dashboard, API Management

Analytics and Management that correspond to the Service platform layer in Borgia and Presentation and Business connectivity that maps over the Application layer.

It is worth noticing that Microsoft whitepaper [7] offers a lot of details about the Data processing, Analytics and management layer. It contains all the services needed to develop a new IoT solution on top of the Microsoft offering. While the lower layer (Device connectivity) is just relegated to the role of data collection and sharing.

The SAP reference architecture (summarized in 5) proposes a solution very similar to a classical Cloud stack, relegating the IoT nature of the solution to a very generic (virtual) hardware layer that contains every kind of physical device. From SAP point of view, IoT is just another application of the Cloud paradigm, that involves specialized hardware, to be modelled and managed at the higher levels.

To conclude, Table 2 summarizes the vendors mapping in respect to Borgia's Layers. It is worth noticing that all vendors explicitly refer to Cloud services as backed to the IoT system: in practice, all solutions assume that the personal networks, made of devices and interconnected directly or via gateways, are managed and controlled remotely by Cloud services (typically hosted by the vendor). Standardized reference architectures, instead, do not explicitly mention the cloud backed, leaving space to alternatives, like solutions based on a local server, hosting the IoT Application and IoT service platform.

2.3. Conceptual view

The functional views illustrated above, helps in understanding which are the main functionalities invoilved in an IoT system, but the conceptual view illustrated in Fig. 8, even defined in the standard ISO/IEC 30141 [6], describes the main components that implements such functionalities. It is worth noticing that, at best of author's knowledge, there are no other analysis that so clearly identify the main concepts involved in an IoT architecture.

According to the standard definitions, an *IoT Device* is a digital entity which bridges between real-world physical entities and the others digital entities of an IoT system. IoT device interacts with others entities through one or more networks. IoT device exposes one or more endpoints by which interactions are made, as well as it can or cannot uses data stores. *Sensors* and *Actuators* are IoT devices. Sensor is a device that detects and responds to some type of input from the physical environment and outputs digital data that can be transmitted over a network, i.e., a sensor monitors a physical entity. Instead, an Actuator is a device that accepts digital inputs and acts on (changes) one or more properties of a physical entity on the basis of input. These three components are the one that implements the functionalities of the *Perception Layer* in

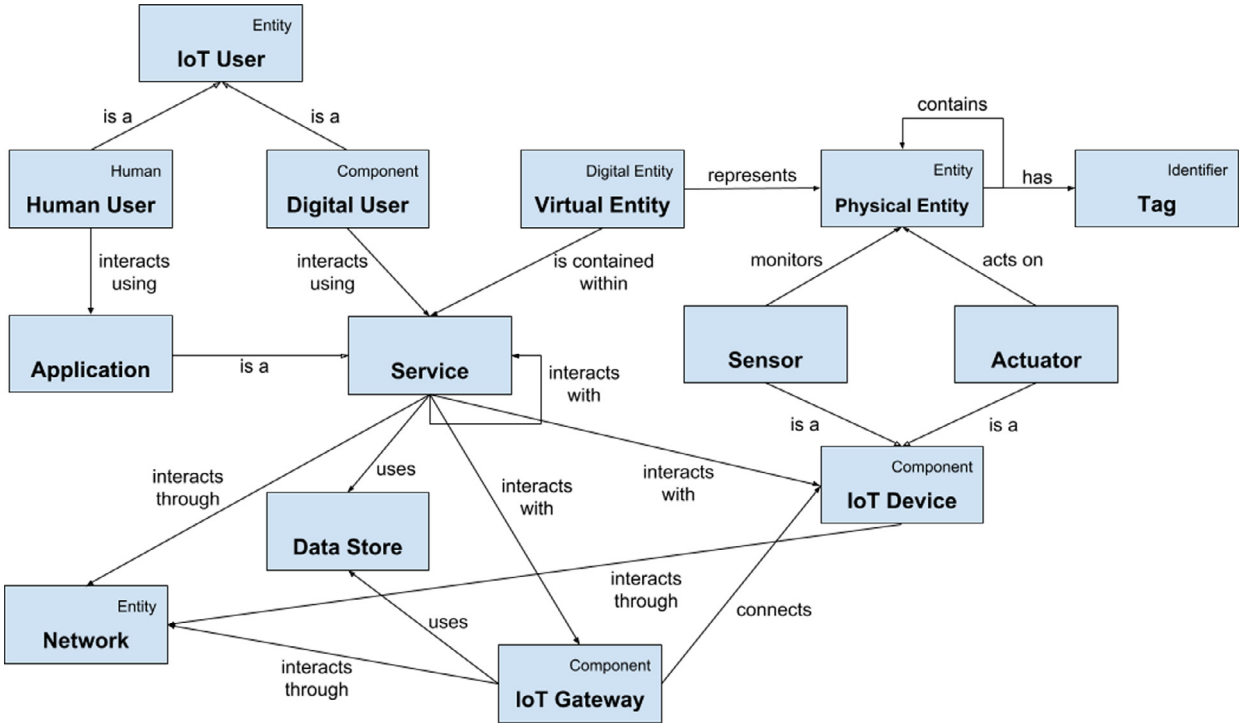

Fig. 8. Overall model for IoT concepts of the conceptual model in ISO/IEC 30141.

our functional view. The Geteway (GW) access level functionalities are implemented by *IoT Gateways*: a digital entity that acts as a means to connect one or more IoT devices to a wide-area network. IoT gateway interacts with IoT device through short-range networks and with Services through the Network layer. A *Service* is a set of distinct capabilities provided by a software component through a defined interface, which may be composed of other services. A service is implemented by one or more components. The services, together with the components that implement them, constitutes the service platform and Enabler layer. Finally, an (IoT) *Application* is a software component that offers a collection of functions with which a user can perform a task. It is worth noticing that an Application is another instance of the Service concept and, in the IoT context, it implements the functionalities typical of the application domain (eHealth, smart home, etc.)

2.4. Example scenarios

The IoT concepts illustrated above, can be adopted in a concrete IoT scenario in many different ways. We propose five different scenarios, divided into two principal application domains: *Smart Home* and *Smart City*.

Smart Home scenarios refer to configurations which are generally limited to single habitations, with no interactions with similar environments, we named them according to the kind of items (of commercial and common use) that adopts them: Google scenario, Xiaomi scenario,and MicroBees scenario. Moreover, in order to outline interoperability aspects, we considered a Google + Xiaomi + BLE Sensor scenario, where different technologies are involved.

Fig. 9a shows a classic scenario in which the user employs devices and services belonging to a single brand. In this particular scenario, each device is an IP capable device, which must have an IP address and is able to communicate directly with the services offered by the remote (Cloud) platform. The devices cannot communicate with each other, as direct connections are forbidden, and the exchange of information and commands happens through the Cloud platform.

The scenario is shown in Fig. 9b: in this case the devices do not have an IP address, but they communicate through a short-range communication protocol (e.g., BLE, ZigBee) with an IoT Gateway. The IoT Gateway is the only device connected to the Internet and it is the only device that can communicate with the services offered by the Cloud platform. As in the Google scanario, the devices cannot communicate with each other directly, except through the IoT Gateway.

The scenario, shown in Fig. 9c, is very similar to the Xiaomi one 9b, but the devices can communicate directly with each other through short-range communication protocols.

In Fig. 9d, the scenario 9a and the scenario 9b work together, and a device that communicates in short-range with an IoT gateway is also included in the ecosystem.

For the *Smart City* scenario, we refer to configurations in which Smart Home environments are able to communicate and interact with each other, we named them CoSSMic scenario, according to the research project that proposes such an architecture. The scenario is shown in Fig. 9e. In this kind of configuration, devices are represented by Smart Meters, which

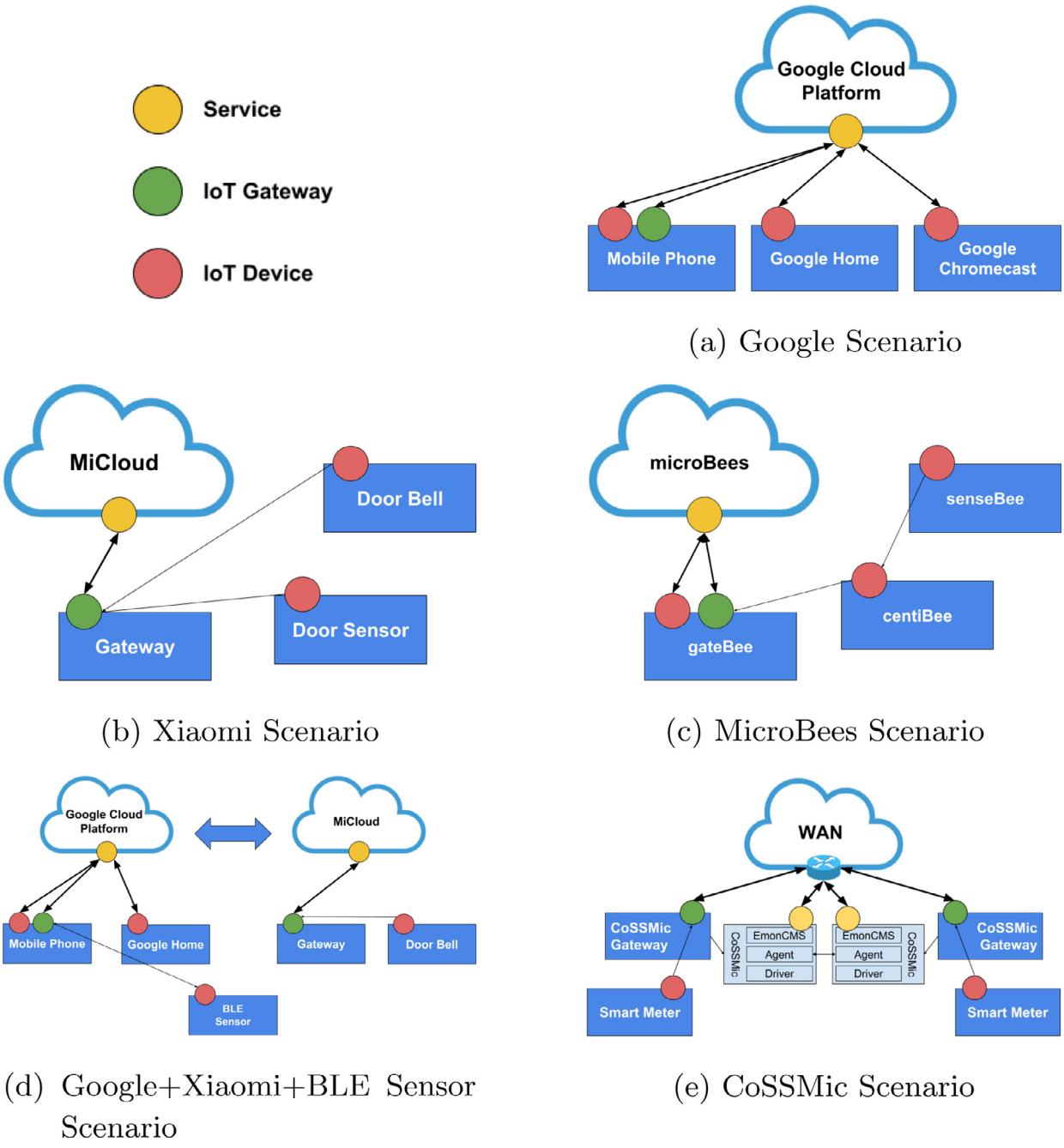

Fig. 9. Examples scenarios.

collect information on the environment. Such meters can only communicate with their reference CoSSMic Gateway, on which the services, provided by the EmonCMS server, run. However, Gateways communicate over a WAN and exchange information regarding the local environment they control, so that a distributed and global management becomes possible.

3. State of art in IoT security

Security is considered one of the most critical subjects related to Internet of things, due to the pervasiveness of the paradigm (cybersecurity starts affecting the concrete world), and its strict connection with the cloud computing paradigm (where security is still considered one of the major inhibiting factors). The reference architectures (Section 2) in some cases consider security as a cross-cut property of an IoT framework, but such aspect is often underestimated.

Accordingly, during the past few years some interesting papers have proposed surveys of security challenges in this context. In the following, we briefly summarize the papers that, from our point of view, best helps in identifying the status of the security research in the Internet of Things context.

Alaba et al. [11] adopts a very simplified reference architecture, reduced to only three layers: Application, Perception and Network. The Perception layer matches with the three lower layers we adopted as a reference in this paper, while the application layers concentrate together the application and IoT Service platform layers. Moreover, it proposes an IoT security taxonomy respect to the *application, architecture*, and *communication* (it is worth noticing that, security taxonomy and reference architecture are not aligned, affecting the overall paper coherence). Respect to the proposed taxonomy the paper proposes a set of typical threats and vulnerabilities of the IoT heterogeneous environment and proposes possible solutions for improving the IoT security architecture.

Zarpelao2017 [12], instead, surveys Intrusion detection techniques in the context of IoT: outlining the possible solutions and the difficulties of the adoption of such strategies in a context made of low-power and little computation capabilities devices. IoT is modelled as a wireless sensor network (WSN) connected to Internet, without the definition of a reference architecture. The paper includes a taxonomy of IDS, including security threats specific for WSN/IoT systems that could be addressed using IDS techniques.

An IETF draft [12] offers an analysis of the security issues in the context of the Internet of Things, based on a All IP model: every device is uniquely defined through an IP address. Accordingly, the paper models IoT as a network, but outlines some specificities typical of the IoT identifying a 3-step life cycle for the devices, that includes: (i) a bootstrapping phase (when a new device/node is inserted into the architecture and there registered), (ii) a maintenance phase that includes the software updates and device rebooting, together with the usual device execution and, finally, (iii) a decommission phase, when the device/node is removed from the network. The paper proposes a list of IoT threats, together with recommendation to address and resolve them, outlining their role respect to the above illustrated life cycle.

Yan det al. [2] investigates the properties of trust, defined as a complicated concept with regards to the confidence, belief, and expectation on the reliability, integrity, security, dependability, ability, and other characters of an entity, measured through the reputation defined as a measure derived from direct or indirect knowledge or experiences on earlier interactions of entities and is used to assess the level of trust put into an entity. The paper, similarly to [11], proposes its own three-layered reference architecture (Application/Network/perception) and mainly proposes a very detailed literature surveys on aspects related to trust for each layer.

Differently from the previous authors, Sicari et al. [13] summarize the security challenges for IoT systems for different security domains: authentication, confidentiality, access control, privacy, mobile security, trust and policy enforcement, summarizing the main research results with respect to each of such domains. Unfortunately, they do not adopt any clear reference architecture for IoT, which is described mostly as WSNs connected through the Internet (similarly to [12]).

The paper written by Roman et al. [14] does not propose a complete reference architecture for IoT, assuming mainly that it is made of interconnected devices coordinated by dedicated applications, but it describes IoT through four alternative scenarios (that are very similar to the ones proposed in our Section 2.4). Security analysis, similarly to [13], outlines IoT security challenges in respect to different security domains: identity and authentication, access control, protocol and network security, privacy, trust and governance and, finally, fault tolerance, outlining for each the promising approaches to solve the related issues. It is worth noticing that, the challenges outlined are not strictly connected to the proposed scenarios and the completeness of the analysis can be hardly granted.

Also PoP et al. [15] does not propose a complete reference architecture for IoT, assuming mainly that it is made of interconnected devices coordinated by dedicated applications, but they present an architecture for supporting the communications among smart devices. Accordingly, the contribution of this work is proposing (i) a gateway-less event-based communication protocol specifically tailored to sensors with constrained resources, (ii) a solution for flexible event-based communications among heterogeneous data sources, and (iii) an approach based on the theory of evidence for data fusion processes that depend on the matching and trust degree of the data to be fused. A fault tolerance approach is also considered in [16].

Finally, The paper [17] explicitly proposes a IoT security road-map to help interested researchers to contribute to this research area. Instead of proposing their own reference architecture, they propose a systematic view of IoT identifying, in the IoT context, the main elements together with their interactions and the main actors together with their relationships. The results of such analysis are two graphs, whose context is, in our humble opinion, similar to the conceptual view of IoT proposed by ISO and summarized in Fig. 8. The paper, then, outlines the security challenges in respect to each element and actor identified.

The above illustrate stat of art outlines that security challenges in IoT are a pretty complex issues, and the missing of a well accepted reference architecture is one of the main issue when building a threat model for IoT system, having the goal of identifying he main issues to address.

4. Analysis of past and current API representations

According to scenario 9d, there are interoperability issues among the services of different providers. Services Orchestration is seen as a possible solution to such issues, especially when Cloud Platforms are strictly involved [18]. There are several efforts towards the definition of an efficient and shareable formalism to describe IoT online services. However, as most of

Table 3
A comparison between WSDL and WADL.

WSDL	*WADL*
Complex design	Simple design
Difficult to read from a human point of view.	Easy to read and implement
Supports all HTTP verbs and several other protocols (e.g SMTP)	Only supports HTTP
Accepts XML parameters only	Parameters types are not limited to XML
W3C recommendation	Not standardised
Authorization mechanisms available	No authorization

IoT services are nowadays callable via RESTful interfaces, the main research line consists in defining new formalisms for the representation of such interfaces, with a focus on sensors and devices. As a consequence, formalisms for API description have been proposed, also to answer the need to a common representation for sensors' interfaces. In this section, we are going to describe the main actors in the API description scenery and to compare them to each other and to the WSDL standard, widely used in the past for web services definition.

4.1. Web Application Description Language (WADL)

The *Web Application Description Language* (WADL) [19] is a machine readable formalism for the description of HTTP based web-services, based on XML. While Sun Microsystems submitted it to the World Wide Web Consortium (W3C) in 2009, there is still no plan to actually standardize it [20]. In respect to WSDL documentation, WADL description is lighter and more straightforward, in order to better adapt to modern web-services interfaces, which generally employ REST. The structure of a WADL document is indeed very simple:

- The core element of the description is represented by the *Resources* tag, which exposes a *base* property to point out the base address of the services.
- The Resources tag contains a collection of *Resource*, each representing the single services, which can be accessed from the base address. In particular, the name of the service is identified by the *path* attribute.
- Within a resource, the methods are defined via the *Method* tag, exposing a *name* attribute (to state the type of HTTP request issued) and an *id* with the exact name of the called service.
- Each Method describes a *Request* and a *Response*. Both of them contain parameters (*param* tag), with at least a name and a type, but which can also expose information on the obligatoriness or cardinality of the parameter. Furthermore, if the parameter can only be chosen from a predefined set of values, the *option* tag can be used to enumerate them.

The hierarchical model used to obtain the WADL description of the web service can be rapidly parsed to obtain the information required to build the service call. Also, the entire document is much smaller and simpler than a WSDL counterpart. Obviously, being simpler than WSDL, WADL is also less flexible and offers less functionalities. Table 3 provides a comparison between the two languages.

4.2. RESTful API Modeling Language (RAML)

The *RESTful API Modeling Language* (RAML) [21] is a vendor-neutral and open specification language built on YAM [22] and JSON for describing RESTful APIs. First proposed in 2014 by the *RAML Workgroup*, it has been taken in consideration by the OpenApi consortium, together with other API specification languages, for standardization. Similarly to other languages for the description or REST and REST-like interfaces, RAML specification tries to be as simple as possible and to offer a lightweight description of web-services. However, it also supports the definition of patterns and data type inheritance, in a fashion very similar to object-oriented languages. The structure of a RAML document is composed of *nodes*, each describing a peculiar element of the API interface:

- *Title* and *Description* respectively contain a string label which identifies the service and a human-friendly description of it. The Description, in particular, should provide directions for the use of the service.
- The *baseURI* and *baseURIParameters* nodes represent fixed elements of every service call. In particular, baseURI defines the base for URIs of all resources. Often used as the base of the URL of each resource. Instead, baseURIParameters defines all the named parameters, which are used in the base URI.
- The *types* node can be used to define new data types (named types), which can be also declared inline (unnamed types) or in external libraries. Both named and unnamed types need to declare a *schema* or *type* node (they are mutually exclusive and their use depends on the RAML version used), which can refer to one of the built-in data types provided by RAML, or to another named type defined elsewhere.
  - *Object types* are a particular built-in type that can declare additional *properties*, so they can be used to define complex parameters.

Table 4
Comparison between WSDL, WADL and RAML.

	Design	*Reading & Implementation*	*Protocols*	*Parameters*	*Standard*	*Authentication*	*Reuse support*
WSDL	Complex	Difficult	HTTP and others	XML	W3C	Yes	Data Types extension
WADL extension	Simple	Easy	HTTP	XML, JSON	No	No	Data Types
RAML	Medium	Requires knowledge of YAML	HTTP	XML, JSON, YAML built-in	No	Yes (Schemas)	Inheritance

- A *Resource* is identified by its relative (to the baseURI node) URI, and it is identified with a slash. It can be defined at root level (top-level resource) or nested within another resource. Its three main components are represented by:
  - A *Method* node containing the list of parameters needed to call the specific service described by the resource (via the mutually exclusive *queryParameters* or *queryString* nodes), the description of the *header* and of the *response* expected from the method call;
  - A *Resource Type* that the current resource inherits;
  - A list of *Traits* that apply to all the methods described by the resource, which can be overridden by the specific method.
- *Resource Types* and *Traits* are at the core of the inheritance schema implemented by RAML, and contribute to the re-use of code and to its consistency and maintenance. A Resource Type specifies a generic resource whose methods and properties can be inherited by actual resources definitions. Traits are instead applied to all the methods exposed by resources, and define characteristics that they share.
- Most REST APIs have one or more mechanisms to secure data access, identify requests, and determine access level and data visibility. The *securitySchemas* node describes the security schemas definitions supported by the API.

RAML tends to be a little more complex than WADL, as it also supports inheritance and code re-use. Also, it can be difficult to read for users who have small or no experience at all with YAML, but are instead more experienced in XML-based documentation. This is not a real problem, as after a very short time using it, users can definitely learn how it is structured and use it seamlessly.

As in the WADL representation, in which responses were defined in external XML documents, here we suppose that external RAML files are available for inclusion (via the *!include* command).

Table 4 provides a three-way comparison among WSDL, WADL and RAML.

4.3. API Blueprint

API Blueprint [23] is a documentation-oriented web API description language, built upon the *Markdown* syntax [24], which is a plain-text syntax for formatting documents that can be immediately translated in HTML pages via the Markdown tool. An API Blueprint document is a plain text Markdown document describing a Web API, structured into logical *sections*. Such section has a specific location within the documentation, can be nested and, while completely optional, if present, they must follow the Blueprint formalism. The language reserves some keywords to identify the section types, so that, they cannot appear as the identifying names of a section. As an instance, HTTP verbs (GET, POST, DELETE, etc.) are keywords in Blueprint. Therefore, a section is composed of:

- A *Keyword* with the *Identifier* of the section (its unique name);
- A section's description, which is any arbitrary Markdown-formatted content following the section definition; It can contain reserved keywords, as it is treated as a comment to the section;
- Content specific for the described section;
- Nested sections.

Also, it is possible to distinguish between two main categories of sections: *Abstract* sections need to be extended as they cannot be used directly; *Section Basics* are instead directly usable to build sections. Among Abstract sections, the language defines:

- A *Named Section* represents the base of all other API Blueprint sections, as it is composed by an identifier, a description and nested sections, which can be alternatively substituted by specific formatted content;
- An *Asset Section* represents the base for all atomic data in Blueprint, as it is described by a pre-formatted code block;
- A *Payload Section* represents the payload transferred as part of an HTTP request or response.

Section Basics define the main building blocks of the API Blueprint documentation, and they are represented by:

- The *Metadata Section* is composed of key-value pairs separated by a semicolon (they provide metadata annotations which are tool specific);
- The *API name & overview section* is the first header in a Blueprint document, as it presents the name and description of the API (it inherits from Named Section);

Table 5
Comparison between WSDL, WADL, RAML and Blueprint.

	Design	*Reading & Implementation*	*Protocols*	*Parameters*	*Standard*	*Authentication*	*Reuse support*
WSDL	Complex	Difficult	HTTP and others	XML	W3C	Yes	Data Types extension
WADL	Simple	Easy	HTTP	XML, JSON	No	No	Data Types extension
RAML	Medium	Requires knowledge of YAML	HTTP	XML, JSON, YAML built-in	No	Yes (Schemas)	Inheritance
Blueprint	Simple	Easy - Requires knowledge of Markdown syntax	HTTP	MSON, JSON, XML	No	No	Named Attributes can be reused in different sections

- The *Resource group* section, identified by the *Group* keyword, represents a group of resources, thus it may include one or more Resource Sections;
- A *Resource Section* represents an API resource, specified by its URI. The formalism allows for four different kinds of Resource section instantiations:
  1. A simple URI template;
  2. An identifier followed by the URI template in square brackets;
  3. An HTTP request method followed by an URI template;
  4. An identifier followed by an HTTP request method and an URI template, in square brackets.

  In the last two cases, the remainder of the Resource section follows the specifics of the Action Section. A Resource section must contain at least one Action Section, an it can contain additional optional sections, such as the Attributes Section.
- The *Attribute Section* describes attributes of a resource, an action or a payload. Named attributes can be referenced by other sections. They are defined via the *Attributes* keyword, followed by an optional MSON (Markdown Syntax for Object Notation) Type. If omitted, the attribute is considered as an object and defines a structured data type containing more attributes.
- The *Action Section* can be introduced by either an HTTP request method, an action name followed by an HTTP request method enclosed in square brackets, or by an action name followed by an HTTP request method and URI template enclosed in square brackets. It is always nested within a Resource Section and it provides the definition of at least one HTTP transaction as performed with the parent resource section. One and only one Parameter section can be defined within an Action, while optional Attributes defined in the action are included as input of the nested Request sections. Multiple Request and Response Sections may be nested in the Action section.
- A *Parameter* Section describes the URI parameters in a Markdown list item. It defines the parameter name, *default* value, *type* and list of possible values the parameter can assume (using the *Members* optional keyword). Each parameter can be *required* or *optional*

The language is very simple to read, even for non experts, and the use of the Markdown syntax surely helps. The specification supports the use of JSON and XML types for HTTP responses and requests, via *Schema Sections*, which describes how JSON and XML data structure should be formatted.

Table 5 provides a comparison among all the formalisms described so far.

5. Swagger (now now as OpenAPI)

Swagger [25] is the former name of both an API specification language and of a framework implementation based on it, which aims at providing a standard representation for APIs, together with a both human and machine readable documentation. Originally developed for Wordnik [26] to support the Wordnik Developer and its underlying API, it was acquired by SmartBear which, in 2015, founded the OpenAPI Initiative, under the sponsorship of the Linux Foundation. SmartBear donated the Swagger specification to the new group, which renamed it as the *OpenAPI Specification*. RAML and API Blueprint are also under consideration by the group.

One of the strong points of the documentation is the ample use of JSON: files describing the RESTful API in accordance with the Swagger specification are represented as JSON objects and conform to the JSON standards. Being YAML a superset of JSON, YAML parsers are able to understand Swagger documents. Also, the adoption of the JSON standard does not limit the type of attributes which can be defined in the API interaction.

The Swagger specification is based on nested objects. The root document, called *Resource Listing* is basically a collection of *Resource Objects*, each providing the *path* to a resource reachable through the API. The Resource Listing also provides information on the Swagger and API versions, additional information (via the *Info Object*), and supports Authorization (*Authorization Object*).

The resources declared via the Resource Listing are described by a correspondent *API Declaration*. An API Declaration provides information about an API exposed on a resource. In particular, it exposes the root URL serving the API (*basePath*)

Table 6
Comparison between WSDL, WADL, RAML, Blueprint and Swagger.

	Design	*Reading & Implementation*	*Protocols*	*Parameters*	*Standard*	*Authentication*	*Reuse support*
WSDL	Complex	Difficult	HTTP and others	XML	W3C	Yes	Data Types extension
WADL	Simple	Easy	HTTP	XML, JSON	No	No	Data Types extension
RAML	Medium	Requires knowledge of YAML	HTTP	XML, JSON, YAML built-in	No	Yes (Schemas)	Inheritance
Blueprint	Simple	Easy - Requires knowledge of Markdown syntax	HTTP	MSON, JSON, XML	No	No	Named Attributes can be reused in different sections
Swagger	Medium	Medium - JSON is not always immediately readable	HTTP	XML,JSON, XML	No	Yes	Objects can be reused in several locations

and the relative path to the resource (*resourcePath*). Authorization schemes can be defined, as for Resource Listings, via Authorizations Objects.

The core of an API Declaration is represented by the *API Object*, which describe one or more possible operations possible on a single path. Each API Object provides the *path* to the operation, its description and list of *Operation Objects*.

An Operation Object describes a single operation on a path. It contains the declaration of the called *method* (an HTTP verb), a unique id to identify the operation (*nickname*), and a list of parameters, expressed via *Parameter Objects*. Response Messages are described via *Response Message Objects* instead, which contains a response code and a message.

Parameter Objects describes a single parameter to be sent in an operation. Each object declares a type, which can be chosen among the values 'path', 'query', 'body', 'header', 'form', and a name which strictly depends on the type and on the path to the operation.

Table 6 extends previous Table 5 by adding Swagger to the comparison.

6. Conclusions

Several Reference Architectures have been proposed to systematize Internet of Things environments. Some of these architectures have been standardized by international committees, while others have been developed by academic researchers and, despite their validity and general applicability, they still have not standardized yet.

In this paper, such architectures have been analyzed and compared, trying to cover standard, commercial and academic proposals. According to the reference architectures, a set of real-case scenarios have been identified and described: such scenarios describe the most common configurations of IoT devices, also in relation to the Service provider backing them up.

According to such scenarios, security and interoperability challenges have been identified and referred to the presented architectures, and current solutions to such challenges have been presented. Of course, both in the case of security and interoperability issues, not all challenges are currently addressed and solved: in many cases, research is still necessary to identify suitable and shareable solutions.

References

[1] E. Borgia, The internet of things vision: key features, applications and open issues, Comput. Commun. 54 (2014) 1–31, doi:10.1016/j.comcom.2014.09.008.
[2] Z. Yan, P. Zhang, A.V. Vasilakos, A survey on trust management for Internet of Things, J. Netw. Comput. Appl. 42 (2014) 120–134, doi:10.1016/j.jnca.2014.01.014.
[3] L.D. Xu, W. He, S. Li, Internet of things in industries: a survey, IEEE Trans. Ind. Inform. 10 (4) (2014) 2233–2243, doi:10.1109/TII.2014.2300753.
[4] Y. Recommendation, 2060 Overview of Internet of Things, ITU-T, Geneva, 2012.
[5] ISO/IEC 30141:2018: Internet of Things (IoT) - Reference Architecture. Topic Maps. International Organization for Standardization, Geneva, Switzerland. https://www.iso.org/standard/65695.html.
[6] ISO/IEC CD 30141:20160910(E): Information technology Internet of Things Reference Architecture (IoT RA).Topic Maps. International Organization for Standardization, Geneva, Switzerland. https://www.w3.org/WoT/IG/wiki/images/9/9a/10N0536_CD_text_of_ISO_IEC_30141.pdf.
[7] Microsoft Docs, Microsoft Azure IoT Reference Architecture Azure, Version 2.0. https://aka.ms/iotrefarchitecture (Retrieved September 2018).
[8] Intel White Papers. The Intel ® IoT Platform. Architecture Specification White Paper - Internet of Things (IoT) (2015) 1–11. https://www.intel.com/content/www/us/en/internet-of-things/white-papers/iot-platform-reference-architecture-paper.html (Retrieved September 2018).
[9] SAP Documentation. Reference Architecture for SAP IoT Solutions. (Retrieved November 2016) https://eaexplorer.hana.ondemand.com/_item.html?id=12210#!/overview.
[10] P. Fremantle, A reference architecture for the internet of things 0 (2015) 21. doi:10.13140/RG.2.2.20158.89922.
[11] F.A. Alaba, M. Othman, I.A.T. Hashem, F. Alotaibi, Internet of things security: a survey, J. Netw. Comput. Appl. 88 (December 2016) (2017) 10–28, doi:10.1016/j.jnca.2017.04.002.
[12] B.B. Zarpelão, R.S. Miani, C.T. Kawakani, S.C. de Alvarenga, A survey of intrusion detection in Internet of Things, J. Netw. Comput. Appl. 84 (January) (2017) 25–37, doi:10.1016/j.jnca.2017.02.009.
[13] S. Sicari, A. Rizzardi, L. Grieco, A. Coen-Porisini, Security, privacy and trust in Internet of Things: the road ahead, Comput. Netw. 76 (2015) 146–164, doi:10.1016/j.comnet.2014.11.008.
[14] R. Roman, J. Zhou, J. Lopez, On the features and challenges of security and privacy in distributed internet of things, Comput. Netw. 57 (10) (2013) 2266–2279, doi:10.1016/j.comnet.2012.12.018.

[15] C. Esposito, A. Castiglione, F. Palmieri, M. Ficco, C. Dobre, G. Iordache, F. Pop, Event-based sensor data exchange and fusion in the internet of things environments, J. Parallel Distribut. Comput. 118 (2) (2018) 328–343.
[16] C. Esposito, M. Ficco, A. Castiglione, F. Palmieri, L. Huimin, Loss-tolerant event communications within industrial internet of things by leveraging on game theoretic intelligence, IEEE Internet Things J. 5 (3) (2018) 1679–1689.
[17] A. Riahi Sfar, E. Natalizio, Y. Challal, Z. Chtourou, A roadmap for security challenges in the Internet of Things, Digit. Commun. Netw. 4 (2) (2018) 118–137, doi:10.1016/j.dcan.2017.04.003.
[18] F. Amato, F. Moscato, F. Xhafa, Enabling iot stream management in multi-cloud environment by orchestration, in: Proceedings of the 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2018, pp. 687–692, doi:10.1109/WAINA.2018.00168.
[19] Marc J. Hadley. 2006. Web Application Description Language (Wadl). Technical Report. Sun Microsystems, Inc., Mountain View, CA, USA. https://dl.acm.org/citation.cfm?id=1698142 (Retrieved September 2018).
[20] Y. Lafon, Team comment on the web application description language submission (2009), http://www.w3.org/Submission/2009/03/Comment (Retrieved August 2011) (2009).
[21] R. Workgroup, Raml-restful api modeling language, 2015, (http://raml.org/). Visited on 10/02/2017.
[22] O. Ben-Kiki, C. Evans, B. Ingerson, Yaml ain't markup language (yaml) version 1.1, yaml. org, Technical Report (2005).
[23] A. Blueprint, Format 1a revision 8., http://github.com/apiaryio/api-blueprint/blob/master/API%20Blueprint%20Specification.md.Accessed (2015) 05–22.
[24] J. Gruber, Markdown: Syntax, URL http://daringfireball.net/projects/markdown/syntax. Retrieved on June 24 (2012).
[25] S. Team, Swagger restful api documentation specification 1.2, Technical Report, 2014.
[26] S. Davidson, Wordnik, Charlest. Advisor 15 (2) (2013) 54–58.