Building the Semantic Web of Things Through a Dynamic Ontology

Francesco Antoniazzi, *Member, IEEE*, and Fabio Viola, *Member, IEEE*

***Abstract*—The Web of Things (WoT) has recently appeared as the latest evolution of the Internet of Things and, as the name suggests, requires that devices interoperate through the Internet using Web protocols and standards. Currently, only a few theoretical approaches have been presented by researchers and industry, to fight the fragmentation of the IoT world through the adoption of semantics. This further evolution is known as Semantic WoT and relies on a WoT implementation crafted on the technologies proposed by the Semantic Web stack. This article presents a working implementation of the WoT declined in its Semantic flavor through the adoption of a shared ontology for describing devices. In addition to that, the ontology includes patterns for dynamic interactions between devices, and therefore we define it as dynamic ontology. A practical example will give a proof of concept and overall evaluation, showing how the dynamic setup proposed can foster interoperability at information level allowing on the one hand smart discovery, enabling on the other hand orchestration and automatic interaction through the semantic information available.**

***Index Terms*—Internet of Things (IoT), linked data, ontology, Semantic Web, Web of Things (WoT).**

I. Introduction

COINED in 1999 by Ashton [1], the Internet of Things (IoT) is characterized by the pervasive presence of smart co-operating devices that fulfill tasks belonging to very different application domains (Asin and Gascon [2] counted more than 50). Everyday objects, indeed, are now increasingly enhanced with computational power and connectivity (e.g., watches, televisions, and cars, just to mention a few of them).

The collected data coming from IoT, together with all available information, contributes to the definition of *IoT context*. The usage of this data to provide new aggregated information and/or new services is the so called context-aware computing [3]. A variety of working implementations have been suggested to this extent, depending on where the calculation is made, giving birth to the fog [4] and cloud computing [5] paradigms. In this area, as well as in the more general IoT, all the involved entities and applications pivot on a shared context whose definition has been clarified by Abowd *et al.* [6] as: any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

However, the IoT is not the last step of this long evolution started at the end of the eighties. In fact, two partially overlapping branches are now emerging from the IoT: 1) the Web of Things (WoT) and 2) the Semantic WoT (SWoT). They both aim to overcome the layered verticality of IoT systems by using the Web protocols to access and use a new kind of Web resource: the *thing*. Although the names are similar, their approaches to the problem of fragmentation are slightly different.

The WoT [7] pushes on the need for Web standards to solve the issues that hinder interoperability among systems because of the high heterogeneity of the technologies involved in IoT applications. It was first introduced by Guinard and Trifa [8] and is now attracting the interest of the W3C[1] and enterprises member of its Working Group (e.g., Siemens [9]). Building blocks of WoT applications are standard protocols like JSON, HTTP, and Websocket and paradigms like REST.

The WoT advises to access things through those protocols, and defines a vocabulary to set a uniform naming to the most important concepts that an IoT environment designer might need to use. The main drawback is that thing resources must be known *a priori*: discovery is possible only through specific local protocols, that may not be implemented in every device.

The SWoT [10], on the other hand, includes in its vision the adoption of the standards coming from the Semantic Web [11], a movement born to redesign the Web as a machine-understandable repository of data. To achieve this scope, a set of standards to identify resources (i.e., Unicode [12] and URI [13]), to encode information (i.e., RDF [14]) and to bind a meaning to every information atom (i.e., RDFS [15] and OWL [16]) were introduced. RDFS and OWL permit the definition of ontologies, formal explicit description of concepts in a domain of discourse, properties of each concept describing various features and attributes of the concept, and restrictions on slots [17].

Manuscript received February 21, 2019; revised May 13, 2019; accepted August 30, 2019. Date of publication September 6, 2019; date of current version December 11, 2019. This work was supported by the SWAMP Project through the European Union's Horizon 2020 Research and Innovation Programme (Call: H2020-EUB-2-2017 - IoT Pilots) under Grant 777112. *(Corresponding author: Francesco Antoniazzi.)*

F. Antoniazzi is with the Advanced Research Center on Electronic Systems and the Department of Computer Science and Engineering, University of Bologna, 40125 Bologna, Italy (e-mail: francesco.antoniazzi@unibo.it).

F. Viola is with the Centro Nazionale per la Ricerca e Sviluppo nelle Tecnologie Informatiche e Telematiche (INFN CNAF), 40127 Bologna, Italy, and also with the Department of Computer Science and Engineering, University of Bologna, 40125 Bologna, Italy (e-mail: fabio.viola@cnaf.infn.it).

Digital Object Identifier 10.1109/JIOT.2019.2939882

[1]https://www.w3.org/WoT/

Leveraging the SWoT consists in setting up an ontology to regulate the relations among the Web resources, and therefore their differentiation based on their reciprocal connections. With this in mind, the SWoT ends up as in the WoT in a Web resource collection: that collection, however, is consistently enriched by the additional semantic content given by any ontology used to describe the things and the context.

Such additional semantic meaning that resources obtain breaks the isolation of existing applications in vertical silos [18]. The information on the environments, regardless to their realizing technology, is uniformly generated and organized and, the most important, any discovery intent can be reached by querying the semantic uniform abstraction (i.e., a triple store), rather than passing through discovery mechanisms available at the lower levels of the ISO-OSI stack.

In this article, we propose an ontology for the SWoT. This ontology, named SWOT, realizes a high-level abstraction of the devices taking part in a smart application and of their capabilities leveraging the concept of thing description proposed for the WoT by Charpenay *et al.* [9].

In addition to that, this article also addresses one of the main limitations that apply to the SWoT: ontologies and semantic-formatted data are considered to be static, while any real context is continuously evolving dynamically. To do so, the ontology presented here offers the tools to build a static description of the *things* along with a set of concepts that regulate the dynamic interaction. We include in the knowledge pattern a prototype of what the actual thing behavior looks like both when an actuation is triggered, or when a sensor is required to communicate its current measurement.

As already said, through the presented ontology we suggest a solution to the problem of discoverability [19] of devices. Along with the main contribution we propose also a framework, named Cocktail, which is a practical realization of both static and dynamic ontological concepts. It is made for the fast and automatic prototyping of software agents, and will allow us to provide a proof of concept of how it is possible to build a SWoT environment and orchestrate it. The ontology, together with its applications and capabilities, will be evaluated.

The SWoT ontology can be employed with any of the available standard SPARQL endpoints. Nonetheless, due to the dynamic nature of IoT applications, and therefore of SWoT applications, the whole study considers and takes advantage of the SPARQL processing event architecture (SEPA) [20], [21] as reference architecture. SEPA aims to enhance triple stores with a publish-subscribe layer on top the SPARQL 1.1 protocol. SEPA clients, then, by using SPARQL 1.1 subscribe[2] and update languages can, respectively, subscribe to and publish semantic data. This means that with SEPA it is possible to easily create a semantic representation of the context and keep it coherent with the physical environment as time passes.

The authors consider that the usage of semantics to enable interactions within devices defines the concept of dynamic ontology as it is intended in the title of this research. In particular, the SWOT ontology includes the concepts devoted to a static representation of devices, as well as their interaction with other things, which is of course characterized by a high mutability. By binding this article to publish/subscribe semantic endpoints like SEPA, we allow the knowledge base to be constantly up to date with the context. The dynamic ontology not only describes the abstract context, but also permits following its real-time evolution.

[2]http://mml.arces.unibo.it/TR/sparql11-subscribe.html

Before going in the details of this article, we propose a summary of the contributions achieved through our approach.

1) Representation of the W3C's Thing Description model (Charpenay *et al.* [9]) through Semantic Web standards (i.e., OWL). The main outcome of this activity is an easy, high-level, and general ontology for the formalization of Web Thing profiles.
2) Such representation, carefully refined after a comparison with the ontology proposed by Serena *et al.* [22], was then extended to support the Semantic Web Thing interaction (see Section IV) in addition to discovery.
3) Concerning the last point, as shown in Sections IV-A and IV-B, the discovery mechanism based on the proposed ontology is flexible and fully customizable (e.g., by further extending the semantic descriptions with other ontologies).
4) Development of an intuitive framework (i.e., Cocktail) providing high-level APIs enabling an even easier approach to the adoption of the ontology.
5) Formalization of a domain-agnostic methodology and a framework supporting the device interaction by means of any standard SPARQL endpoint. In particular, we suggest the adoption of SEPA which provides the ability to develop a responsive system based on its subscription mechanism.

In Section II, an overview of the current state of the art is presented, leading to a motivation for the ontology presented in this article. Then, the following sections focus their attention on the whole ontology. Section III introduces the concept of Semantic Web Thing. Section IV instead presents the property-action-event pattern in the static and dynamic description. Section V describes how to deal with data formatting. Section VI provides evaluation and a proof of concept of the research on a real life simulated scenario. Eventually, in Section VII, the conclusions are drawn.

The ontology will be presented in the following sections through a set of pictures (Figs. 1–4) focusing on specific subsets of concepts. A full view may be appreciated in Appendix A. To enhance readability of the tables, listings, and figures, the SPARQL prefixes used in this article are reported in Appendix C.

II. Related Work

In the past 20 years, several works have introduced and explained the Semantic Web view. Going back to 2001, Berners-Lee *et al.* [11] discussed the driving ideas and concepts of a still prototypical Semantic Web through some practical examples. This article's focus was to highlight in a few examples the situations in which the currently available Web is either insufficient, or insufficiently exploited.

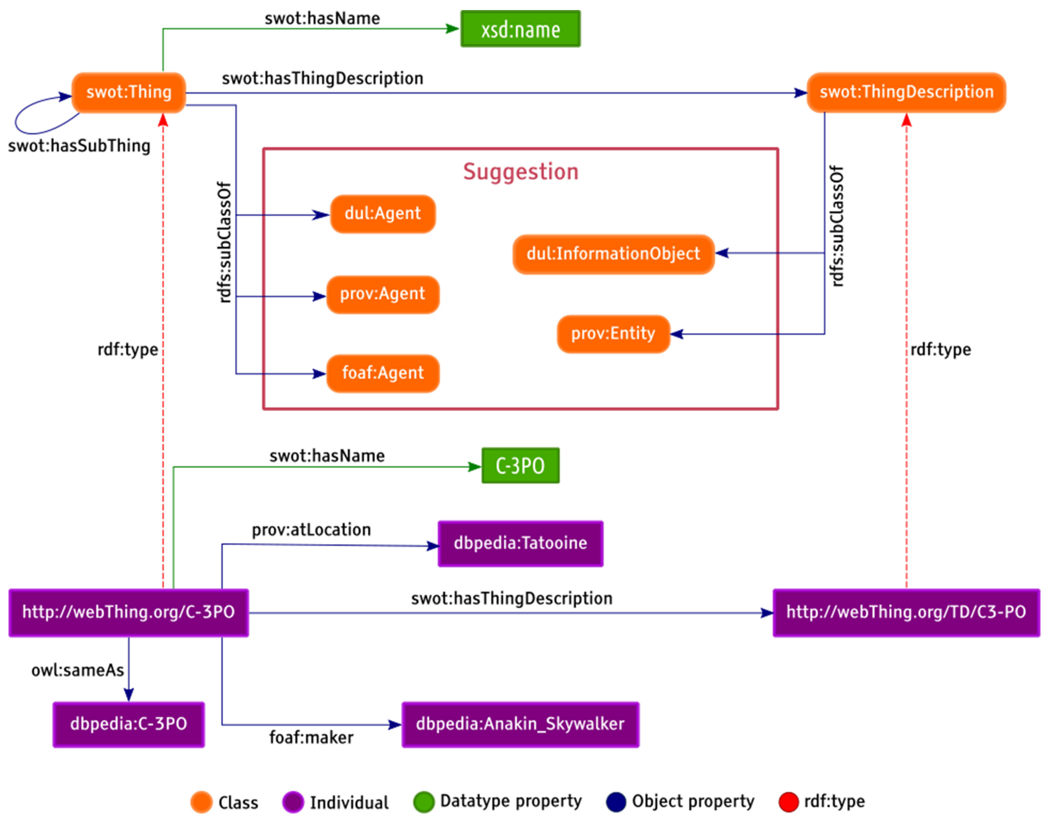

Fig. 1. `swot:Thing` and `swot:ThingDescription` partial ontology and a practical example of instances and some suggestions of extensibility with other ontologies.

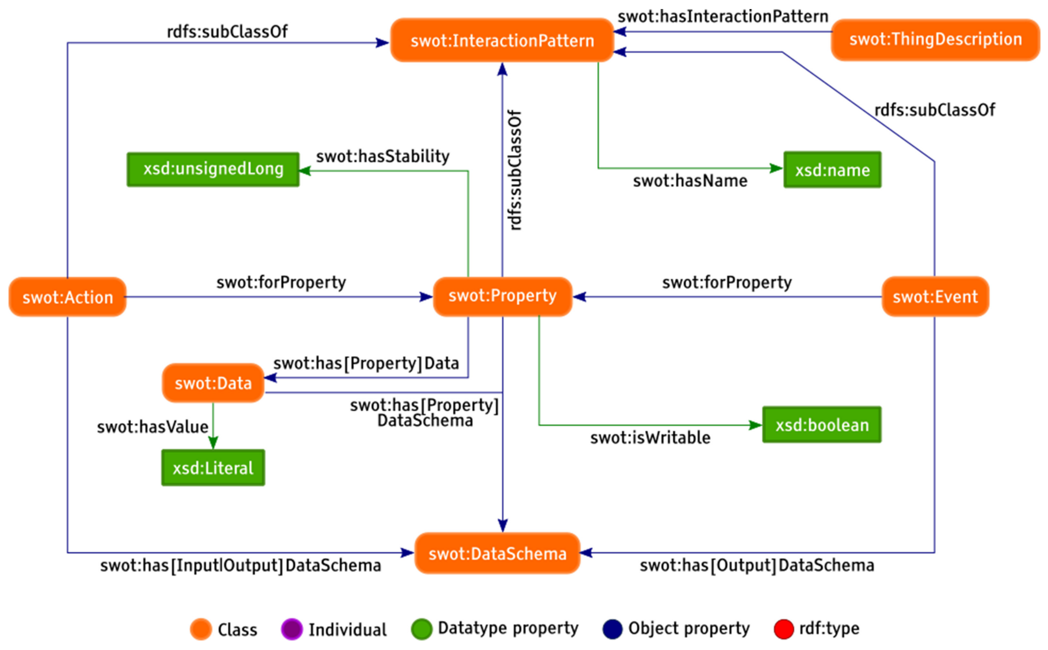

Fig. 2. `swot:InteractionPattern` subset of the ontology. Actions, events, and properties are IPs, receiving inputs, and giving outputs according to a data schema (see Section V).

Following this research stream, Shadbolt *et al.* [23] studied the meaning of the term *ontology* in the Semantic Web. The concept of ontology seems to offer a (at least partial) solution, to the great information disorder that is an inner consequence of the Internet decentralization. Far from the philosophical meaning of the term, i.e., the absolute and unique reality of the being, an ontology is a set of relationships between some well-identified entities, listed in a machine understandable way (namely, the RDF format). The challenges foreseen in Shadbolt's paper, and that effectively we are facing nowadays,

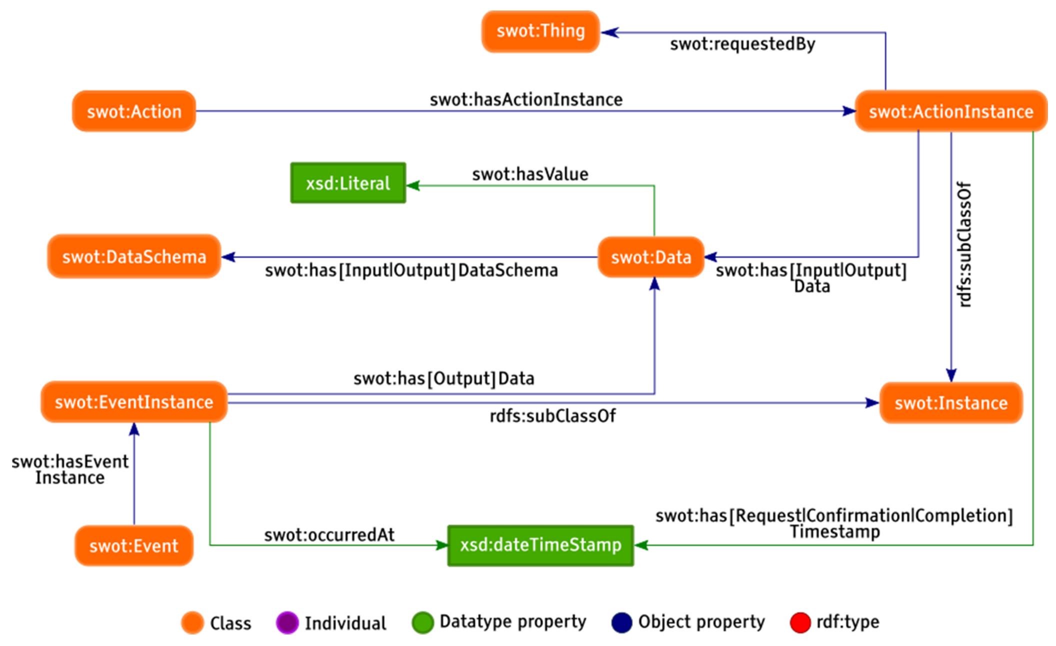

Fig. 3. `swot:Instance` subset of the ontology, i.e., how the subgraph for an action request must be formatted, as well as how an event notification is thrown.

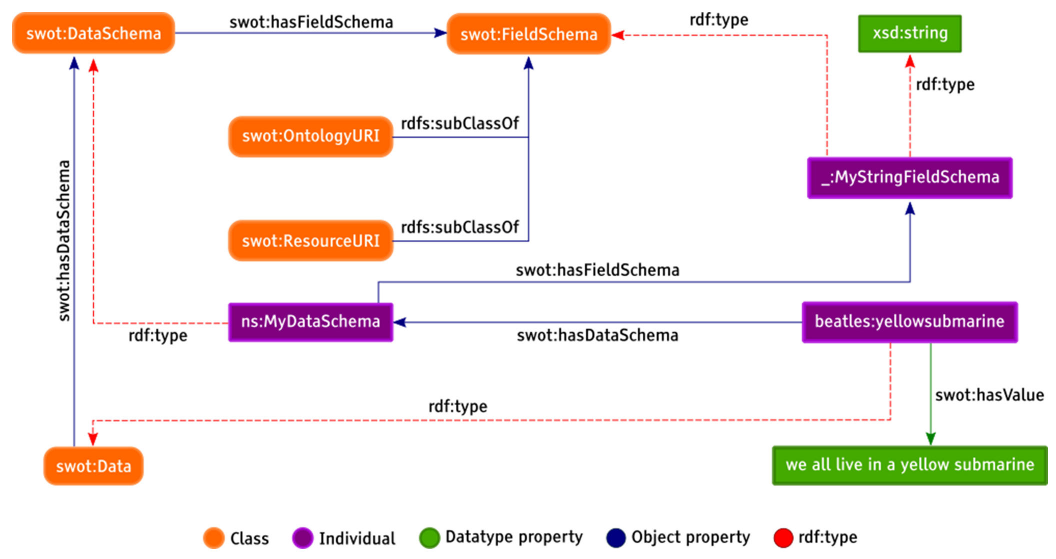

Fig. 4. `swot:DataSchema` and `swot:FieldSchema` ontology subgraph, together with an example of a simple `xsd:string` data schema inclusion.

are the reuse of available ontologies to produce data [24]–[26], the alignment of ontologies exposing the same concepts [27], and the effective exploration and visualization of the data graph [28], [29].

All those concepts apply also to the IoT, whenever an attempt is made to semantically describe its contents. For instance, ontologies modeling the physical-digital interface are, among all, the *sensor, observation, sample, and actuator* (SOSA) and the *semantic sensor network* (SSN) ontologies.[3] Although being largely documented, SSN and SOSA still offer a complex approach to description of hardware, observation of physical entities and actuation, that may be particularly cumbersome if the aim of the work is the formal semantic expression of any IoT service. For this reason, using an ISO-OSI stack metaphor, the ontology presented in this article acts as upper ontology located at application level, while SOSA and SSN are at physical and data-link levels.

In addition to this aspect, IoT presents also another facet, which is the time-related evolution of its context [30]. The *time ontology*,[4] and the *event ontology*[5] have been developed to this extent, in order to categorize both flow of time and asynchronous behaviors in the RDF graph. Their design, however, was made for the static description *a posteriori* of a sequence of events, while the SWOT ontology targets real-time awareness of context evolution.

[3]https://www.w3.org/TR/vocab-ssn/

[4]https://www.w3.org/TR/owl-time/

[5]https://semanticweb.cs.vu.nl/2009/11/sem/

Other works, e.g., OpenIoT [31], IoT-O [32], and IoT-Lite [33], either use one of the previously cited ontologies like SSN, either design a lower level description of devices almost at hardware level. This is something that in this research we want to avoid, to provide to the developer only high level interfaces.

Different works in literature propose IoT architectures enhanced with semantics. The following lines report an overview of these works, starting from those having semantics in a limited set of components and concluding with those oriented at a semantic description of things. Puiu *et al.* [34] presented an IoT framework for smart cities named CityPulse. This framework adopts semantics in two of its components (i.e., namely, data wrapper and data federation). The first provides semantic annotations based on the stream annotation ontology (SAO) and the quality ontology (QO) as well as on the information models developed on top of the above-mentioned SSN ontology, PROV-O, and OWL-S. The second module is instead used to answer users' queries that are translated into RDF stream processing (RSP) requests. Then, the overall role of semantics in this framework is limited to discovery, data analytics, and interpretation of large-scale data. As in our case, semantics has been adopted to foster interoperability among heterogeneous entities. Moreover, CityPulse, is constrained to the domain of smart city applications.

The same domain is addressed by Kamilaris *et al.* [35]. In this article, semantics is the glue among the IoT/WoT elements and is used to annotate sensory data streams. Annotation is achieved through an information model based, once again on SSN and OWL-S and the adoption of ontologies like the above-mentioned SAO, the complex event ontology, and PROV (just to name a few).

Kamilaris *et al.* [36] also proposed Agri-IoT, a semantic framework for IoT-based smart farming applications supporting multiple heterogeneous sensor data streams. The framework provides a complete semantic processing pipeline, offering a common framework for smart farming applications. It reuses a set of components of the CityPulse framework [34] as well as modules from FIWARE, ThingSpeak, and OpenIoT. Devices are handled by the device manager module borrowed from FIWARE IoT Backend that is based on NSGI-LD. In this article, we adopt the Web Thing abstraction to describe devices in terms of properties, events and actions and we applied this model to a SEPA-based ecosystem. SEPA and NGSI-LD are not conflicting, as demonstrated by our research work [37].

All the ontologies mentioned in the previous lines, and many others available for research and usage in the World Wide Web, have the common goal of overcoming a fragmented world, where every solution cannot easily communicate with the one developed in the nearby office [38]. This well known nightmare of IoT researchers is analyzed in [39], for instance, listing the causes of fragmentation of IoT (e.g., the coexistence of resource constrained and rich devices in environments). Many researches suggest the usage of a gateway to solve this problem (e.g., [40] and [41]), while Zachariah *et al.* [42] highlighted the limitations of such kind of approach, though proposing, as for today's state of the art, a rather difficult to realize smartphones-as-a-gateway solution.

Semantic Web was also included in this discussion: for instance, to foster the horizontal communication of vertical silos, Desai *et al.* [18] proposed a semantic approach, studied developing a protocol translation gateway. This idea of enhancing IoT by unification and translation at information level, rather than at lower protocols, is also followed by Gangemi *et al.* [43], where they proposed the IoT application profile (IoT-AP) ontology with the aim of representing and modeling the knowledge in the IoT. In [44], as well, an ontology is suggested and associated with the tasks of discovery and dynamic composition: this article differs from ours, as the ontology there is neither designed with the purpose of context evolution, nor targets the SWoT, but the plain IoT.

The interest of the IoT community in what the Semantic Web has to offer is also demonstrated by ontology repositories for IoT and smart cities [e.g., Ready4SmartCities, OpenSensingCity, linked open vocabularies (LOVs) [45], and LOV4IoT [46]] and their impressive growth [47]. As an example, LOV, standing at the analysis proposed by Gyrard *et al.* [47], stepped from less than one hundred to more than five hundred ontologies in the period between March 2011 and June 2015 (and more than 650 are available as of December 2018). As already said, discovery and orchestration of resources are killer applications of semantics applied to the IoT.

The problem of discovering available resources in a network (i.e., the *discoverability problem* [19]) is well known in research [48] and several solutions have been provided over the years. It can be also addressed through ad-hoc protocols (like the one proposed in [49], focused on privacy requirements), protocol-specific tools (e.g., CoAP-based discovery was proposed by Djamaa *et al.* [50] and Viola *et al.* [51], while XMPP-based solution is proposed in [52]) and gateway-based approaches [53]. Semantics in this scenario has been proposed in several research contributions [18], [54]–[57]. Kamilaris *et al.* [58] presented WOT2SE, a search engine for the WoT based on Web crawlers that scan linked data endpoints. In this article instead, we rely on a central broker, i.e., SEPA, where discovery can be made by means of SPARQL queries/subscriptions either directly or indirectly (e.g., through high-level tools like the WoT store [59]).

On the other hand, orchestration/choreography [60] refers to the centralized/decentralized composition of services to perform complex tasks exploiting multiple elementary components. The creation of a seamless flow of information through IoT devices and services turns out to be a challenging task due to the: 1) heterogeneity of devices; 2) the heterogeneity of data; and 3) the unpredictability of the availability of devices and information. Heterogeneity of shared information can be overcome only through an agreed understanding of its composition, while the latter issue can be addressed through an effective discovery mechanism. It is then clear how semantics may help the development of service composition functionalities in large-scale scenarios. In this sense, it is important to keep track of the provenance of the information and, again, this can be achieved through a well established ontology, like PROV-O [61]. Several approaches to orchestration/choreography have been proposed over the years. Tzortzis and Spyrou [62] presented a semi-automatic approach to service composition. Viola *et al.* [63] proposed an example of orchestration of *virtual things* applied to the Semantic Audio research area. Song *et al.* [64] proposed a middleware based on Semantic Web technologies aimed at the automatic configuration of an heterogeneous network with service composition functionalities. In literature it is also common to find approaches based on large IoT frameworks and architectures supporting service composition like arrowhead [65], OpenIoT [66], and IoT-A [67].

As highlighted by Barnaghi *et al.* [68], the heterogeneity of devices makes interoperability a challenging problem, which prevents generic solutions from being adopted on a global scale. Due to the key role of Semantic Web technologies in fostering interoperability in the IoT, a new research area pivoting on them is born: the SWoT.

Unfortunately, the application of semantic technologies to the IoT is not straightforward due to the nature of IoT requirements (e.g., constrained devices and unreliable connections) [68] and this motivates the birth of this new research area.

One of the first research works mentioning the SWoT is the one by Pfisterer *et al.* [69]. The authors propose a service infrastructure to make information produced by sensors available to all the possible users through the linked open data cloud, and not just to a single application. While we propose a high-level abstraction of sensors and actuators, Pfisterer *et al.* [69] focused on the nature of sensors. In both cases tools for the automatic representation of information are provided: in SPITFIRE, knowledge about sensors is inferred and eventually confirmed by the user, while in Cocktail the developer is required to declare properties, events and actions. As regards the discovery mechanism, Pfisterer *et al.* [69] declared that an important functionality is searching for entities with a certain state at the time of the query. Due to the high dynamicity of IoT scenarios, SPARQL is not applicable and they developed a heuristic-based system. In our architecture, SEPA (through its subscription mechanism) allows using SPARQL to perform this task also in IoT scenarios.

Ruta *et al.* [70] defined the SWoT as the adoption of Semantic Web technologies in IoT application. That said, the purpose of their research is rather different from the work disclosed in this article. In fact, Ruta *et al.* [70] mostly focused on one of the common criticisms to the Semantic Web protocols: their efficiency. The formats adopted in the Semantic Web are generally considered too verbose to allow efficient data storage and management in IoT applications and this motivates their work on efficient compression methods. Despite the different topic, it is interesting to compare the system architectures: the project by Ruta *et al.* [70] is based on layer named *ubiquitous knowledge base* (u-KB), providing access to the information embedded into semantic-enhanced micro-devices. It is a fully decentralized system, in contrast with SEPA, where the information is always available thanks to a central broker hosting data. In both architectures, devices are fully decoupled, but SEPA hosting the knowledge base allows: 1) reducing the number of accesses to devices, important with constrained devices or when the network is not reliable and 2) hosting the whole knowledge base in a powerful node granting faster access and inference.

As mentioned in Section I, the W3C founded a working and an interest group dedicated to the WoT, whose challenges are depicted by Raggett [71]. Among the various contributions proposed by these groups, it is worth mentioning again Charpenay *et al.* [9]. Within their research, the authors describe a vocabulary specifically built for the WoT. Their main objective, with such vocabulary, is the alignment with the pre-existing W3C achievements on IoT semantic reordering. The cited work relies on the identifier, resource, entity (IRE) ontological pattern, which states that Web resources may act as proxies for real world entities. From this article, we borrow the concept of thing description as *semantic resource formally describing a unique WoT Thing that a software agent can interact with*, and the concepts of property, action, and event [the *interaction patterns* (IPs) of Web Things]. For all those borrowed concepts, however, the SWOT ontology creates the semantic background that in W3C approach is limited to the JSON-LD availability for the thing description. In addition to that, we introduce the ontological view of real-time instances for actions and events. The framework proposed in Section VI leverages these concepts to provide a practical implementation of all the tools needed to create a full environment.

Ontologies for the so-called (Semantic) WoT have been proposed also by other authors. For instance, Serena *et al.* [22] proposed an ontology for the discovery of devices in the SWoT. With respect to this article, again, our research goes beyond the pure discovery of devices, enabling the interaction through the semantic broker. The ontology by Serena *et al.* [22] is also used by Noura *et al.* [72] that propose a framework for the goal-oriented description of Web Thing interactions. A framework for semantic interoperability in the WoT is presented also in [73] which combines an extension of the SSN ontology and machine learning techniques. No details are provided regarding the way subscriptions can be defined.

III. Semantic Web Things

The core concept of SWoT ontology is the `swot:Thing` class representing Web Things. Any software, any real-world item connected to the Internet with a semantic representation of its capabilities can be considered an instance of this class. In the next sections, the precise patterns that are used in the ontology to describe the Web Thing capabilities will be discussed.

Such definition of Semantic Web Thing is indeed unrelated to the technology realizing it. We might also argue that even the human body can be considered as a connected Web Thing in some situations: applications in healthcare [74], of course, but also research on wearable IoT for everyday life [75], [76] and music [77] are valid examples.

The collection of Web Things acting and interacting in a semantic context will be referred to as the Semantic Web Thing environment (SWTE). Querying the SWTE will eventually result in an inner context-awareness. In the next sections, in fact, we will see that the evolution of the context is taken into account by the architecture, and therefore the actual physical

```
SELECT * WHERE { ?thing rdf:type swot:Thing }
```

Listing 1. SPARQL subscribe to list all Web Things available in the RDF store.

environment is represented on-the-go in the stored RDF representation. So, in a very simplified example, we may need to be notified of any new device entering in our environment. This can be done with the subscription in Listing 1. As outlined in the Introduction, this research exploits the SEPA subscription mechanism (whose description is out of the scope of this article). As opposed to MQTT protocol,[6] where notifications are topic-based and not specifically focused on RDF knowledge bases, SEPA natively allows SPARQL queries to differentially follow their subgraph over time.

In a slightly more complex situation we may need to be notified of a temperature overcoming a threshold. This will be possible by subscribing to an event once, in Section IV, the *IPs* will be explained.

The whole ontology, both in the static and dynamic description, is designed to support and easy respond to enquiries on the control of the dynamic evolution of the context, providing a SPARQL-based context awareness. Through the Cocktail framework, presented in Section VI, examples on how to code and use the controls and the ontological description of evolving SWTEs will be provided.

In the RDF representation, a Web Thing's URI can be a dereferenceable resource or, in any case, it should be an appropriately formatted address. A standard compliant WoT ecosystem would rely on HTTP(S) addresses over TCP/IP as URIs.

As already proposed by Guinard and Trifa [19], Web Things can be declared to act as proxies for other Web Things. This may be useful in case of constrained devices (see [78] for a complete definition of the term) that are not directly reachable at application level, and/or unable to declare themselves in the SWTE. The proxy Web Thing receives and forwards requests in the right format to the proxied Web Things.

Semantic Web Thing discovery is not limited to the SPARQL example provided in Listing 1. Exactly as in the IoT, the number of possible ways in which things can be described is almost unlimited. Even the same object, in two different environments, can be described in different ways leveraging, for instance, on other ontologies targeting other descriptive aspects. For this reason, the Semantic Web Thing discovery is tightly connected to the semantic feature description of the object: the basic features of a Web Thing are contained in the *thing description*, while an example including other ontologies is given in Section VI.

In the ontology, the `swot:Thing` is bound to the `swot:ThingDescription` through the predicate `swot:hasThingDescription`. While the former, as already said, is not necessarily a Web resource but must be unique for each Web Thing, the second should be. In particular, any HTTP GET to the thing description resource should respond with a full JSON-LD description of the features of the Web Thing (i.e., the IPs: actions, properties, and events, as it is described in the next sections). This is a useful feature, especially for devices that must be available both from inside the SWTE, and from outside (i.e., the World Wide Web).

To give an example on how to use this first ontology subset, consider Fig. 1, where the color code is defined and used in Protégé.[7] In the red box, that is only for suggestion and does not belong to the ontology presented in this article, we added a few straight-forward connections to other ontologies, like PROV-O [61], DUL,[8] and FOAF,[9] proving that SWOT ontology integration with other ontologies is possible as well as its usage with DBpedia resources.

IV. Interaction Patterns: The PAE Paradigm

When an explanation is needed on *what an object is?* people often tend to answer to a different question, which is in fact *what is it made for?* This is in general a reasonable topic change, especially from the engineer's point of view, as the real matter of discussion are the possibilities that can be explored through the usage of the object.

IoT, WoT, and indeed SWoT, comply with this vision: users, both machines and humans, will be discovering the SWTE looking for Web Things because they want to use them in order to achieve something. There is, for this reason, the need of a semantic unified description of the capabilities of objects. Such description has to be both machine and human understandable, as we would like to enable people and AI to choose the right device [79].

Within this section, a full description of Semantic Web Thing interactive framework is provided, as explanation of the ontology. As already discussed, the ontology presented in this article borrows some concepts from other works, and extends them with an original contribution. For instance, Charpenay *et al.* [9], on behalf of the considerable work of W3C interest and working group, introduced the thing description object, while Serena *et al.* [22] defined the concepts of property, action, and event which in this article we call the PAE paradigm. On April 5, 2018, the W3C released the Thing Description Draft,[10] that leverages the two works aforementioned. Our research takes its origins in such draft.

A. Static Interaction

The *static interaction* is the abstract description of a connected device feature: in our context the *feature* is basically the need we have to fulfil when using the device. Properties, actions, and events have been identified by W3C as the best way to represent that concept of feature.

1) Properties address the need of storing, fixing, and defining a device's current state: for example, a smart car's property may be the percentage of gas remaining in its reservoir.

[6]http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[7]https://protege.stanford.edu/

[8]https://lov.linkeddata.es/dataset/lov/vocabs/dul

[9]http://xmlns.com/foaf/spec/

[10]https://www.w3.org/TR/2018/WD-wot-thing-description-20180405/

2) Actions are the active interactions with the world (i.e., the need to produce, sooner or later and in a finite amount of time, an effect on the environment): a smart car's action may be to switch on the radio.
3) Events, which implement the inner asynchronous nature of any agent oriented environment (i.e., the need to be aware of changes in the environment): a smart car's event may notify the driver that the rear seatbelt is detached, or call for help in case of accident.

In this article, these three entities are represented as the classes `swot:Property`, `swot:Action`, and `swot:Event`, which are all subclasses of the *IP* concept `swot:InteractionPattern`. In Fig. 2, the ontology subgraph for the IP is shown: it can be noticed that all IPs have a friendly name, and they all can refer to one or more `wot:DataSchema` to format their data (see Section V), which can be input data for actions, output for actions and events, or property data for properties. While for properties the data is an essential part, and therefore the connection with the DataSchema is compulsory, this is not the case for actions and events. They both may produce some output, and actions may need some input: the choice of including input or output is left to the specific needs of the Web Thing designer.

The property, in particular, is slightly different from actions and events, representing a current state of the device. We then semantically describe it with a friendly name (inherited from the `swot:InteractionPattern`), a stability and writability flag. The stability's `xsd:unsignedLong` value identifies in milliseconds the average time that the property is expected to remain constant. The writability's `xsd:boolean` flag indicates if the property is software-definable or not. If yes, it is possible that: 1) an action exists able to allow such software modification or 2) an external physical action is required to modify the value (e.g., a mechanical toggle position).

Actions and events, on their side, apart from inherited object and data properties offer the dynamic interaction which will be treated in the next section pointing to the `swot:Instance` concept (see Fig. 3). In addition to that, both actions and events can also refer to the `swot:Property` they may have effect on (i.e., through the `swot:forProperty` object property).

B. Dynamic Interaction: Interaction Pattern Instances

The core discussion of this section is how the dynamicity of interaction is achieved within the SWTE, through the usage of the ontology proposed. In fact, the semantic description of the interaction among Web Things plays a key role for the discussion of the contributions presented in this article. The evolution of a WoT environment cannot be observed through the immutable characters of the context. The ability to represent also the interactions among the agents is why we attributed the *dynamic* adjective to the our ontology.

To render a mutable environment like the Web Thing environment over a semantic platform, the latter has to be enriched with a subscription engine. Therefore, to handle such dynamic interaction, we will consider a RDF knowledge base on top of which we add a publish-subscribe architecture like the SEPA, proposed by Roffia *et al.* [20]. SEPA has been designed to be communication-protocol agnostic. In this article, HTTP(S) protocols for query and update, plus WebSocket(S) for subscriptions will be considered. There would be, of course, no difference if other choices were made: for instance, a CoAP oriented SEPA engine called C minor is being developed for time-constrained musical applications in [51].

Once the Semantic Web Thing description has been given, we expect at run time two possible situations: 1) the request for the execution of an action, which we call ActionInstance, and define through the `swot:ActionInstance` class and 2) an event notification, which we call EventInstance, and define in the `swot:EventInstance` class. They both are subclasses of `swot:Instance`.

See Fig. 3 for the subset of ontology related to dynamic evolution of instances.

Let us first consider an event instance. In Table I, the triples that a Semantic Web Thing must insert in the RDF store to notify the occurrence of an event with its output content can be observed. Among these triples there is the `xsd:dateTimeStamp` of occurrence, that is essential to keep a timeline for notifications, and the output of the notification itself. As it will be better explained in the last paragraph of this section, the exchange of inputs and outputs is another of the sources of dynamicity in the WoT that requires the publish-subscribe mechanism.

In Table II, on the other hand, are shown the triples that concern the `swot:ActionInstance`. Action instances are inserted into the RDF store by any entity requiring the execution of an action. Among them, the authorship of the request is shown by the `swot:requestedBy` predicate, and the timestamp of the request.

The two examples available have either an input given, either an output. As already discussed, actions can have both, or none of them. Events, on their side, can have an output or just be empty. What is interesting, here, is the timings with which those pieces of information are inserted into the knowledge base.

Let us consider first actions. A common definition for them, according to W3C WoT working group, is a kind of interaction taking a finite amount of time to reach an end. It is therefore reasonable to consider that the outputs of an action, if available, will be given *after* that amount of time. So, while inputs need to be given immediately together with the action instance to trigger the execution, we definitely run into an asynchronous behavior whenever we expect an output as result of the execution.

Events, instead, are by definition asynchronous. So, when they happen, they should carry all the information they need (i.e., their output if they have one). A different reasoning has to be done concerning properties: they are, in fact, the reference for information specific to the Semantic Web Thing and therefore they exist because of the information itself. So they have no input nor output, but they just supply their property data.

TABLE I
SWOT:EVENTINSTANCE TRIPLES TO BE INSERTED IN THE RDF STORE TO TRIGGER AN EVENT NOTIFICATION TO ALL INTERESTED ENTITIES. NOTICE THAT THE CONCEPT OF DATA SCHEMA IS EXPLAINED IN SECTION V, WHILE THE TRIPLES NEEDED FOR THE DEFINITION OF SWOT:MYSTRINGDATASCHEMA ARE AVAILABLE IN THE HIGHLIGHTED ROWS OF TABLE III

subject	predicate	object
`ns:MyEvent`	`rdf:type`	`swot:InteractionPattern, swot:Event`
`ns:MyEvent`	`swot:hasEventInstance`	`ei:<uuid>`
`ei:<uuid>`	`rdf:type`	`swot:EventInstance`
`ei:<uuid>`	`swot:occurredAt`	`"2018-12-06T17:00:00Z"^^xsd:dateTimeStamp`
`ei:<uuid>`	`swot:hasOutputData`	`ei:<uuid>/outputdata`
`ei:<uuid>/outputdata`	`swot:hasValue`	`"This is an output"^^xsd:string`
`ei:<uuid>/outputdata`	`rdf:type`	`swot:Data`
`ei:<uuid>/outputdata`	`swot:hasDataSchema`	`swot:MyStringDataSchema`

TABLE II
SWOT:ACTIONINSTANCE TRIPLES TO BE INSERTED IN THE RDF STORE TO TRIGGER THE ACTION PERFORMANCE. NOTICE THAT THE CONCEPT OF DATA SCHEMA IS EXPLAINED IN SECTION V, WHILE THE TRIPLES NEEDED FOR THE DEFINITION OF SWOT:MYSTRINGDATASCHEMA ARE AVAILABLE IN THE HIGHLIGHTED ROWS OF TABLE III

subject	predicate	object
`ns:MyAction`	`rdf:type`	`swot:InteractionPattern, swot:Action`
`ns:MyAction`	`swot:hasActionInstance`	`ai:<uuid>`
`ai:<uuid>`	`rdf:type`	`swot:ActionInstance`
`ai:<uuid>`	`swot:hasRequestTimeStamp`	`"2018-12-06T17:00:00Z"^^xsd:dateTimeStamp`
`ai:<uuid>`	`swot:requestedBy`	`ns:AnotherWebThing`
`ai:<uuid>`	`swot:hasInputData`	`ai:<uuid>/inputdata`
`ai:<uuid>/inputdata`	`rdf:type`	`swot:Data`
`ai:<uuid>/inputdata`	`swot:hasValue`	`"This is the input"^^xsd:string`
`ai:<uuid>/inputdata`	`wot:hasDataSchema`	`swot:MyStringDataSchema`

The input or output resource, of course, needs to be filled out with actual information. Within the ontology inputs and outputs belong to the class `swot:Data` when given by user or retrieved from execution. In our vision, however, there is no sensible difference between `swot:Data` and the DUL class `dul:InformationObject` as the piece of information is here collected, citing DUL `rdfs:comment` of `dul:InformationObject`, *independently of how it is concretely realized*.

Eventually, as it can be seen, all `swot:Data` instances refer to a `swot:DataSchema`. This is necessary, as an IP might accept as input different formats, or release its output in different formats: connecting the information with the actual interpretation statement is therefore essential. For a complete tractation of the `swot:DataSchema` concept, see Section V.

V. DATASCHEMA AND FIELDSCHEMA

As it was told in the previous sections, `swot:Thing` is an abstraction for connected objects in a SWoT environment. Similarly to any object in the real world, the Semantic Web Thing instance needs several connections with the virtualized semantic environment. Those connections enable all the interactions that things may have in their own context. Hence, they are very important because they permit a dynamic flow of information from thing to thing, as well as from user to thing and *vice versa*. Actions, then, will be able to receive information on how they are expected to perform their task: the *parameters*. According to such information the performance may change dramatically: just consider the difference between asking a printer to make 2 or 200 copies of the same file!

Similar interfaces, moreover, are needed when actions or events have to produce some kind of output. And eventually, as we already said, also properties may need one, as they store status information.

Formatting parameters is not a negligible problem. Furthermore, when discovering the available Web Things in a SWTE, it may be of great interest to query for actions that require an input formatted in a specific way (e.g., a thermostat where the target temperature is expressed in Celsius degrees in an XML file), or for events that generate outputs with a particular syntax (e.g., a temperature sensor with output expressed in Fahrenheit degrees in a JSON file).

Within the SWOT ontology this problem is addressed by the `swot:DataSchema` and the `swot:FieldSchema` classes. In Fig. 4, the relationships between these two classes and the ones introduced in the previous sections, as well as a simple example are shown.

To better understand the meaning and the usage of data schemas and field schemas, it is useful to distinguish the basic situations that can occur within a Semantic Web Thing data interaction. All the following cases can happen either in inputs and outputs of any of the `swot:InteractionPattern` subtypes, i.e., actions, events, and properties.

1) The data exchanged (or stored, in the case of a property) is a basic data type, that fulfils the definition of any of the types in XML schema,[11] like `xsd:integer` and `xsd:string`.
2) The data exchanged is a complex data type collecting in various ways a cluster of basic data types: this might be (but not limited to) the case of a JSON or an XML file.
3) The data exchanged is a resource: a text file, audio, video, and so on.

[11]https://www.w3.org/TR/xmlschema-2/

TABLE III
EXAMPLES OF BASIC DATATYPES. WHITE LINES REFER TO THE FIRST EXAMPLE. GRAY LINES HAVE TO BE ADDED TO REALIZE THE SECOND EXAMPLE DISCUSSED IN SECTION V-A

subject	predicate	object
`ns:MyAction`	`rdf:type`	`swot:InteractionPattern, swot:Action`
`ns:MyAction`	`swot:hasDataSchema`	`ns:MyIntDataSchema, ns:MyDoubleDataSchema`
`ns:MyAction`	`swot:hasInputDataSchema`	`ns:MyIntDataSchema`
`ns:MyAction`	`swot:hasOutputDataSchema`	`ns:MyDoubleDataSchema`
`ns:MyIntDataSchema`	`rdf:type`	`swot:DataSchema`
`ns:MyDoubleDataSchema`	`rdf:type`	`swot:DataSchema`
`ns:MyIntDataSchema`	`swot:hasFieldSchema`	`_:MyIntFieldSchemaBN`
`_:MyIntFieldSchemaBN`	`rdf:type`	`swot:FieldSchema, xsd:integer`
`ns:MyDoubleDataSchema`	`swot:hasFieldSchema`	`_:MyDoubleFieldSchemaBN`
`_:MyDoubleFieldSchemaBN`	`rdf:type`	`swot:FieldSchema, xsd:double`
`ns:MyAction`	`swot:hasDataSchema`	`ns:MyStringDataSchema`
`ns:MyAction`	`swot:hasInputDataSchema`	`ns:MyStringDataSchema`
`ns:MyStringDataSchema`	`rdf:type`	`swot:DataSchema`
`ns:MyStringDataSchema`	`swot:hasFieldSchema`	`_:MyStringFieldSchemaBN`
`_:MyStringFieldSchemaBN`	`rdf:type`	`swot:FieldSchema, xsd:string`

TABLE IV
COMPLEX DATATYPE TRIPLE DESCRIPTION EXAMPLE. NOTICE THAT `NS:MYSTRINGDATASCHEMA` DEFINITION IS NOT INCLUDED IN THE TABLE, AS IT IS ALREADY AVAILABLE IN TABLE III, GRAY-COLORED LINES

subject	predicate	object
`ns:MyAction`	`rdf:type`	`swot:InteractionPattern, swot:Action`
`ns:MyAction`	`swot:hasDataSchema`	`ns:MyStringDataSchema, ns:MyComplexTypeDataSchema`
`ns:MyAction`	`swot:hasInputDataSchema`	`ns:MyStringDataSchema`
`ns:MyAction`	`swot:hasOutputDataSchema`	`ns:MyComplexTypeDataSchema`
`ns:MyComplexTypeDataSchema`	`rdf:type`	`swot:DataSchema`
`ns:MyComplexTypeDataSchema`	`swot:hasFieldSchema`	`ns:MyJSONFieldSchema`
`ns:MyJSONFieldSchema`	`rdf:type`	`swot:FieldSchema, xsd:Literal`

4) The data exchanged is a Semantic graph formatted according to a specific ontology.

The following sections will demonstrate their usage giving practical examples.

A. Basic and Complex Datatype

In this section, the complex and basic datatypes are examined. Tables III and IV are provided to exemplify the meaning and the usage of `swot:DataSchema` and `swot:FieldSchema` classes.

In particular, Table III (first example) reports the triples needed by an action requiring as input information a unique `xsd:integer`, and outputting the square root of that number as `xsd:double`.

In a smarter Web Thing, the same action might be able to read also an `xsd:string`, and parse it independently toward the integer. In that case, there would be no difference with the previous situation except that, as in Table III (second example), the gray-colored entries are included.

The complex datatype is slightly more challenging and is represented in Table IV. The table contains the triples necessary for an action that requires as input an `xsd:string`, α, and outputs a JSON object having an entry for every distinct character of α, and value the number of times such character appears in α.

The two examples provided come in help to explain the concepts of data schema and field schema.

First of all, it is important to observe that an instance of `swot:DataSchema` should not be considered thing-specific. Therefore, we expect that numerous actions, events, and properties (no matter the Web Thing they belong to) share the same data schema, like is done for `ns:MyStringDataSchema`. This is an essential point, to guarantee interoperability as well as to reduce the amount of data in the knowledge base. Besides, it has to be highlighted that any `swot:DataSchema` should neither be bound to a role of fixed and immutable input or output: action *A* may need data schema *D* as input, while event *E* as output. The data schema must be a Web resource to be easily identified. As a matter of fact, it should be reachable from the Web, and should reply to requests with a JSON-LD describing the data format.

Second, having a closer look to the tables, we can outline the usage of the `swot:FieldSchema` entity. As it can be seen, the field schema closely depends on the data schema. It is a semantic resource that acts similarly to a collection point for data formats. Field schemas can be provided in the SWTE as blank nodes and as resources, depending on the needs of the IP: in the basic data case, the field schema is a blank node typed as an `xsd` resource. There should be no need for further format description and interpretation support, as `xsd` refers to a well-known standard.

Inversely, in the complex data case, the field schema is a full resource typed as a generic `xsd:Literal`. The field schema resource URI, now, should be a reachable resource on the Web (i.e., a blank node here is not acceptable), containing all the needed information to interpret the literal entity. That is, in the case of Table IV, we intend the resource to answer to an `HTTP GET` with a JSON Schema according to Listing 2.

Multiple field schemas can be connected to a data schema, signifying that the IP is able to use (or expecting) data formatted in more than one way.

```
Request:
GET MyJSONFieldSchema
Host: ns

Result:
200 OK
Content type: application/json+schema

{
  "definitions": {},
  "$schema": "http://json-schema.org/schema#",
  "$id": "ns:MyJSONFieldSchema",
  "type": "object",
  "title": "Char␣counter␣Action␣output",
  "patternProperties": {
    "^[a-z]$": {"type": "integer"}
  },
  "additionalProperties": false
}
```

Listing 2. JSON schema expected in response to an `HTTP GET` to `ns:MyJSONFieldSchema` as in Table IV referring to the complex datatype example in Section V-A.

Once this answer reaches the client, there is a global understanding of how the data should be formatted and/or interpreted. Notice that a DataSchema may point to more than one FieldSchema, meaning that all of them will be given (if it is an output), or all of them are required (if it is an input).

B. Web Resource Datatype

It is not rare for a device to need a file to perform an action: especially (but not limited to that case) if the device is virtual. While a generic and maybe small textual file can be treated as in the previous section as an `xsd:Literal`, a more complex situation is here considered of actions, events, and properties dealing with generic resources located on the Web.

The most important point, when the Semantic Web Thing is parameterized with Web resources, is to define the semantic description in order to allow a correct usage of the given item. This task belongs to the data schema and the field schema. As a first example, let us consider Table V, containing the triples necessary for an action that is capable of playing an audio file by using a generic audio player (which will be referred to as `play`). In such case, the Web Thing's inner logic would be parameterized so that the parameter URI is interpreted as a link to music file. Once the resource URI is received, its usage is fully dependent on the device purpose. In general, we consider two possibilities: 1) the resource is downloaded and used and 2) the resource is not directly downloadable (i.e., a database access point, a streaming resource). Cocktail framework (see Section VI) can implement such audio player in both ways, either if the logic is equivalent to

```
$ download audioFile.mp3 from music_link
$ play audioFile.mp3
```

or to

```
$ play music_link
```

A second example of this same kind might be a database access action. In this case, we consider the action to expect as input an `xsd:string` containing the SQL query, and a `swot:ResourceURI`, Web address of the database. Consider Table VI for the triples. Given those inputs, the software logic would probably be something similar to the command

```
Request:
POST "SHOW␣DATABASES"
Host: <my_DB_resource_URI>
```

Of course, there is no difference in the case of an output resource: an upload is to be expected toward the resource address, or the creation of the server itself, responding to queries on that resource. This may also be a powerful solution, to be combined with a REST architecture.

C. Semantic Resource Datatype

Eventually, we refer to the possible occurrence of a `swot:InteractionPattern` designed to produce or to consume a semantic graph. It is important to say that there is not such a big difference with the previous case exposing an action querying a database. In fact, as it is shown in Table VII, the difference is that the field schema is now given by a complete resource instead of a blank node, and of course it is not a `swot:ResourceURI` but a specialized `swot:OntologyURI`. With this data/field schema construction, the user can download or explore the ontology given by the field schema, and use the patterns there described either to format a triple graph, if it is an input graph, or to query the triple graph, if it is an output. The gray lines in Table VII include the triples to be added in order to perform an action request.

It might be useful, in other scenarios, to join the usage of the `swot:ResourceURI` with the `swot:OntologyURI`, for more complex situations. For instance, let us consider the same input DataSchema of Table VII, and an action whose task is to perform a SPARQL query like.

```
SELECT ?v1 ... ?vN
WHERE { <some very complex query pattern> }
```

Let us consider, also in this case, that the FieldSchema is `foaf`. Then, the actual `swot:Data` parameter expected here is the Web location of a graph resource which we know is formatted according to `foaf`, so that the query will be able to be performed successfully. To make things more complex, let us add a line to the query:

```
SELECT ?v1 ... ?vN
FROM <graph_resource>
WHERE { <some very complex query pattern> }
```

If we include a `swot:ResourceURI` as second FieldSchema as we did for the case in Section V-B, it will be possible to give as a parameter also the `graph_resource` variable.

TABLE V
WEB RESOURCE DATATYPE TRIPLE DESCRIPTION EXAMPLE

subject	predicate	object
ns:MyAction	rdf:type	swot:InteractionPattern, swot:Action
ns:MyAction	swot:hasDataSchema	ns:WebAudioDataSchema
ns:MyAction	swot:hasInputDataSchema	ns:WebAudioDataSchema
ns:WebAudioDataSchema	rdf:type	swot:DataSchema
ns:WebAudioDataSchema	swot:hasFieldSchema	_:MyAudioResourceBN
_:MyAudioResourceBN	rdf:type	swot:FieldSchema, swot:ResourceURI

TABLE VI
WEB RESOURCE DATATYPE TRIPLE DESCRIPTION FOR A DATABASE QUERY swot:Action CLIENT. NOTICE THAT ns:MyStringDataSchema IS NOT INCLUDED, AS IT IS ALREADY AVAILABLE IN TABLE III

subject	predicate	object
ns:MyAction	rdf:type	swot:InteractionPattern, swot:Action
ns:MyAction	swot:hasDataSchema	ns:DBAccessDataSchema, ns:MyStingDataSchema
ns:MyAction	swot:hasInputDataSchema	ns:DBAccessDataSchema, ns:MyStingDataSchema
ns:DBAccessDataSchema	rdf:type	swot:DataSchema
ns:DBAccessDataSchema	swot:hasFieldSchema	_:MyDBResourceBN
_:MyDBResourceBN	rdf:type	swot:FieldSchema, swot:ResourceURI

TABLE VII
GRAPH RESOURCE DATATYPE TRIPLE DESCRIPTION

subject	predicate	object
ns:MyAction	rdf:type	swot:InteractionPattern, swot:Action
ns:MyAction	swot:hasDataSchema	ns:GraphAccessDataSchema
ns:MyAction	swot:hasInputDataSchema	ns:GraphAccessDataSchema
ns:GraphAccessDataSchema	rdf:type	swot:DataSchema
ns:GraphAccessDataSchema	swot:hasFieldSchema	foaf:
foaf:	rdf:type	swot:FieldSchema, swot:OntologyURI
ns:MyAction	swot:hasActionInstance	ai:<uuid>
ai:<uuid>	rdf:type	swot:ActionInstance
ai:<uuid>	swot:hasRequestTimeStamp	"2018-12-06T17:00:00Z"^^xsd:dateTimeStamp
ai:<uuid>	swot:requestedBy	ns:AnotherWebThing2
ai:<uuid>	swot:hasInputData	ns:foafgraphResource
ns:foafgraphResource	rdf:type	swot:Data
ns:foafgraphResource	swot:hasInputDataSchema	ns:GraphAccessDataSchema

VI. CASE STUDIES AND EVALUATION

This section contains the discussion over the SWOT ontology and its conceptualization into the Cocktail framework. To do so, the first section will describe Cocktail and show how the ontology can be easily employed to build interoperable applications. Afterwards, an overall evaluation of the SWOT ontology will be provided, according to a set of literature metrics.

A. Cocktail Framework

Once the SWoT ontology is given and both its static and dynamic parts have been addressed, we have the ingredients to setup a SWoT environment. As already stated in the previous sections, in our implementation the SEPA has been adopted to dispatch events and notifications to control the dynamic evolution of the SWTE. On the other hand, the Web Things will use instances of the static subset of the SWOT ontology to declare themselves and to discover their context. In a broader view, starting from here, the SEPA may act as a Semantic Cloud engine for a generalized SWoT over the SWOT ontology.

To do so, the SEPA implementation available on GitHub[12] will be used together with the baseline APIs developed for it.[13] On top of them the SWoT SEPA APIs are built as a complete framework named *Cocktail*. The Cocktail framework is also freely available on GitHub[14] with its documentation and the explanation of the reasons behind its name.

Cocktail contains high level functions and classes to:

1) declare the things, assign them a friendly name, an URI, and a thing description resource;
2) append to the thing description resource all the IPs needed, i.e., actions, events, and properties, with their friendly names, URIs, data as described in Section IV;
3) define, if needed, new data schemas;
4) query the SWTE for things, IPs, and basic discovery mechanisms;
5) request the execution of an action, post its output and wait for it if necessary, together with all the needed timestamps;
6) throw, and wait for event notifications;
7) delete those instances.

All these functions, indeed, share a common point. They perform specific requests to SEPA: either SPARQL updates, or SPARQL queries, or SPARQL subscriptions. Cocktail uses SPARQL updates to spawn new things, actions, events, properties and data schemas and their internal relationships; in

[12]https://github.com/arces-wot/SEPA

[13]https://github.com/arces-wot/SEPA-python3-APIs (branch `dev-0.9.5`)

[14]https://github.com/fr4ncidir/SemanticWoT

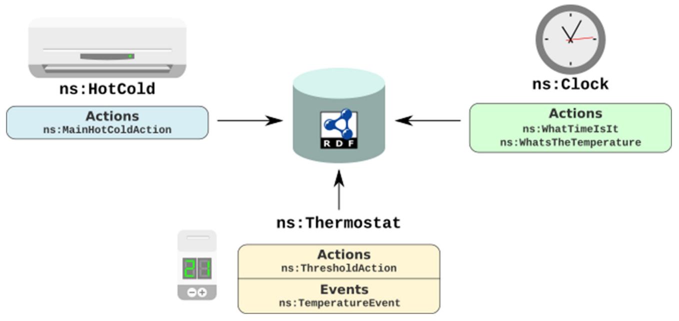

Fig. 5. Example of SWTE.

```
SELECT ?thing ?action WHERE {
  ?thing rdf:type swot:Thing, sosa:Actuator;
    sosa:actsOnProperty ns:Temperature;
    swot:hasThingDescription/swot:hasAction
       ?action .
  ?action swot:hasInputDataSchema <ψ_uri>
}
```

Listing 3. Smart discovery for Web Things with an action acting on temperature and requiring a ψ-formatted input.

addition to that, they are needed also to inject in the graph new action or event instances and output data. Those triples, once inserted in the knowledge base, will be captured by the SEPA subscriptions engine, that will trigger an action execution, or notify that an event has occurred or that an output is available. Eventually clients, humans, or Web Things, will be allowed to perform SPARQL queries looking for all kinds of information in the graph: e.g., we expect standard requests like *"list all Web Things in the SWTE"* along with more complex ones, like *"what IPs give as output mp3 files?"* or *"what Web Things have at least an action which is described through the pizza ontology?"* or also *"how can I format my data so that a specific action can use it as input?"*

It is therefore clear that Cocktail is composed by two facets: 1) the SPARQL code, that interacts with triples in the knowledge base and 2) the thing business logic, that takes care of performing the main tasks of the device, and the communication with SEPA. In our implementation, targeting a proof of concept rather than a full realization of the platform, Python3 has been adopted to address the thing business logic. Indeed, equivalent APIs will be developed for other languages in the future, to be used also in more constrained devices, like the Arduino family and so on.

It is worth noticing that the SPARQL code remains the same in all those implementations. As already said, it is available in Cocktail repository, to be used within our Python3 setup or to be called directly from others services.

B. Cocktail: In-Use Analysis

Cocktail's collection of SPARQL updates, queries, and subscriptions on the top of SEPA proves that a SWoT implementation is achievable in an overall limited amount of lines of code.

Nevertheless, an evaluation is required, both of the framework's usability and of the ontology itself. Notice that an

```
SELECT ?thing ?event WHERE {
  ?thing  rdf:type swot:Thing, sosa:Sensor;
    sosa:observes ns:Temperature;
    swot:hasThingDescription/swot:hasEvent
       ?event.
  ?event swot:hasOutputDataSchema <λ_uri>
}
```

Listing 4. Smart discovery for Web Things that are `ns:Temperature` sensors triggering events with λ-formatted output.

evaluation of the SEPA and its processing units is not within this article's scope.

Let us start by analyzing the usage of the framework. To do so, the following small SWTE composed by three Semantic Web Things (depicted in Fig. 5) and a few additional triples has been developed.

1) `ns:Thermostat` is a smart thermostat also declared as a `sosa:Sensor` observing the special resource `ns:Temperature` (two additional triples). This smart Web Thing has an action called *ThresholdAction*, and an event called *TemperatureEvent*. By calling the *ThresholdAction*, the user can define T_{low} and T_{high}, and therefore setup his/her desired temperature interval. The thermostat performs also a smart discovery: it looks for Web Things acting on `ns:Temperature`, and providing an action that requires some input formatted with a specific data schema ψ. The SPARQL code used for discovery is available in Listing 3. If there are available actions of this kind, and the temperature $t \notin [T_{\text{low}}, T_{\text{high}}]$, the thermostat triggers *ThresholdAction*. Every 5 s, also, the thermostat throws a *TemperatureEvent* with the current temperature value (data schema λ).
2) `ns:HotCold` is a smart air conditioner and heater. It is also typed as `sosa:Actuator` acting on resource `ns:Temperature` (two additional triples). It has an action that can be triggered with input formatted according to dataschema ψ. The device performs a different smart discovery, available in Listing 4, searching for `ns:Temperature` sensors and subscribing to their temperature-change event formatted as λ dataschema.
3) `ns:Clock` is a smart clock with an action that, when requested, replies with the current timestamp (data schema ξ), and an action that replies with the current temperature (data schema λ). Also, `ns:Clock` is a `sosa:Sensor` observing `ns:Temperature` (two additional triples). It is worth saying that this is a dummy device that proves the effectiveness of smart discoveries. Listing 3 discovery will not match with `ns:Clock`, because it is not a `sosa:Actuator`, and Listing 4 because there is no corresponding event.
4) Other triples are related to the resource `ns:Temperature` and to the data schemas. In order to allow the more complex discovery methods previously cited, two `rdf:type` references are added to the temperature resource using the SOSA ontology, `sosa:ActuatableProperty`

and `sosa:ObservableProperty` (two additional triples).

As stated in Section V-A, all the dataschemas ψ, λ, and ξ are considered to be already available in the SWTE. Given that, if we imagine that temperature is $t = 3$ °C and $T_{\text{low}} = 18$ °C, what happens in this environment is that:

1) if the thermostat is declared first it will only notify the temperature event until the smart discovery is becomes effective. Once `ns:HotCold` is available, the thermostat is notified that an action formatted as requested is now present in the SWTE. The action is therefore immediately triggered, and will continue until a temperature event is notified so that $T_{\text{low}} \leq t \leq T_{\text{high}}$;
2) if `ns:HotCold` is declared first, it will just start waiting normally for its action to be triggered. No temperature event from sensors is available, as `ns:Clock` generates temperature through an action. Once, finally, the thermostat is available the action is triggered upon its request and the temperature starts rising, until it reaches a level between the thresholds.

As it can be seen, Cocktail on its own is enough to build an environment in which Semantic Web Things are totally independent, and basic environmental discoveries are available. *Cocktail allowed to concentrate on the basic mechanisms of the SWTE.*

Cocktail alone, however, would have been insufficient to help the thermostat decide the best action to trigger. To make the decision, if the action in `ns:HotCold` or in `ns:Clock`, we need to introduce the 8 SOSA-related triples. Those additional triples allowed us to identify the Web Thing needed, the action to be requested and the events to which we have to subscribe.

This result, in particular, proves that our ontology is easily extensible, and that even a small addition to it can lead to interesting autonomous behaviors, by expanding context awareness.

The code realizing the SWTE is available in Cocktail's GitHub repository, and shows that the realization of the software modules is rather simple and based on a schema that can be summarized as follows.

1) Identification and description of IPs.
2) Posting to the SWTE the triples.
3) Definition of actions' and events' behavior.
4) Device loop: business logic, and events' throwing.
5) *Optional* small Web server dedicated to JSON-LD thing description.

C. Ontology Evaluation

The task of evaluating an ontology is rather complex due to the fact that it has to be performed as a balance between *expressiveness* and *effective usage*. Although the ontology may address various abstraction levels, the target applications must be taken into account in order to distinguish pros and cons. Philosophical ontologies will be mostly evaluated by their expressiveness, while the engineered ones will need also a contact with real applications.

In this article, furthermore, the target is an even different concept, which is in fact a quite new situation in the panorama: the ontology has been defined as *dynamic* to highlight the fact that it is built up of a descriptive and static part, and of a context-evolutionary part. While proceeding with an empirical evaluation of the work presented in this article this factor, that has strong relevance in the realization of the whole application, will be taken into account.

Fernández *et al.* [80] defined a set of 12 metrics to measure the quality of an ontology. In this article, those metrics are partially adopted and partially rearranged to be reasonably applied to the work. In fact, the authors believe that in this specific case, due to the small dimension of this SWOT ontology, not every metric among the ones suggested in origin would be completely appropriate. Keeping this in mind, the evaluation is hereby reported.

1) *Number of Classes:* The number of classes in the SWOT ontology. The overall value, for the ontology, is 14: among these, ten are used for the static SWTE description, and the remaining four have a dynamic role.
2) *Number of Properties:* Number of (datatype or object) properties in the ontology. The datatype properties are overall nine, four of which allow the static description, and five the dynamic one. The object properties, instead, are nine static, seven dynamic, and four belonging to both sets, 20 in total. Eventually, 20 more (inverse) properties can be added. Being redundant elements, they will not be considered from now on.
3) *Number of Individuals:* Number of individuals defined in the presented ontology. At present, the SWOT ontology does not include any individuals: however, as it has been already said, `swot:DataSchema` and the `swot:FieldSchema` instances are a kind of entity that may be considered similar to the concept of individual, since they should be available in the SWTE before the setup of things, and in general they should not be removed.

Those first three points have a considerable impact on the overall composition of a Cocktail-based SWTE because their values affect the complexity of the needed SPARQL enquiries. Being the ontology composition almost equally bipartite, with 27 entries belonging to the static description and 20 belonging to the dynamic evolution, in fact the SPARQL obtained for the complete Cocktail setup appears to be unexpectedly simple.

Another metric that can help understanding the SWOT ontology capabilities is the number and the sequence of SPARQL interactions (updates, queries, and subscriptions) that are required to have a running Cocktail-based SWTE. The metric, similarly, outlines an evaluation of the computational resources necessary for a working SWTE setup, identifying the minimal requirements for a running Web Thing. To give an example, let us consider the same example of Section VI-B, considering Web Things *as is*, out of the general application logic.

1) `ns:Thermostat` requires six SPARQL updates to globally post the Web Thing (1), its action (1) and event plus its notifications (2). In addition to that, there is also the update for the additional background (1) and, eventually, the external action request, that is also a SPARQL update. Concerning subscriptions, one permits to be notified of external requests toward the thermostat

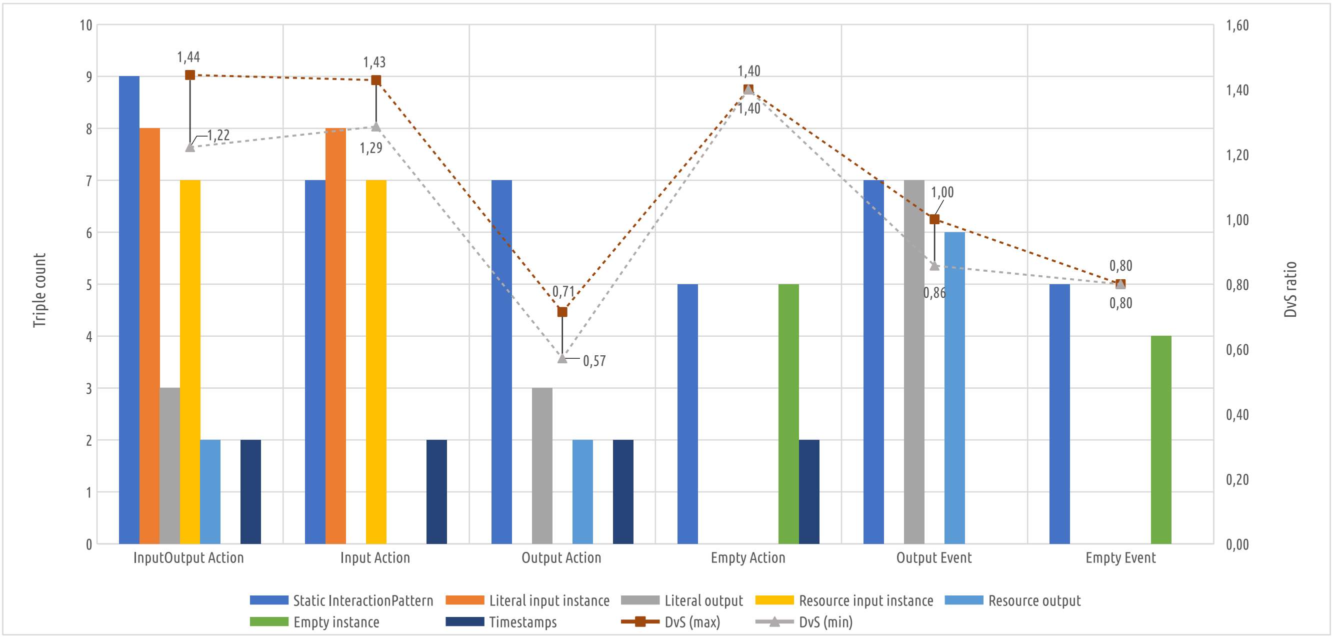

Fig. 6. Number of triples for every IP, and their *DvS* ratio value maximum and minimum.

action, and one is required for the smart discovery mechanism, as already described in the previous section.

2) `ns:HotCold` similarly requires five updates and three subscriptions.

3) `ns:Clock` requires four updates and two subscriptions.

In the end, this three-Web Thing SWTE requires 15 updates (some of which are performed into a loop according to the application logic), and to keep seven subscriptions opened. Not to mention, moreover, the deletion of resources, once they are no more needed. Most the times (namely, the ones related to the dynamic part of the ontology), the deletion is embedded in the same SPARQL update that performs triple addition. So, it has been already counted in the previous discussion. The deletion of static resources, on the contrary, is more rare, and we do not count it, as it has to be made on explicit request by the system maintainer.

Indeed, this is an interesting result, even though no graph level security and privacy mechanisms have yet been implemented. For now, by extrapolation from this example, one can notice that the SWOT ontology requires from the device the capability of dealing with U updates, linearly increasing with the number of IPs; and with S subscriptions, whose minimum number increases also linearly on IPs, and whose actual number depends on the application logic involved. In this performance evaluation, eventually, it is important to mention that a great impact is related to subscriptions. They imply, with the SEPA compliant Cocktail implementation, the capability of the device's hardware to keep a WebSocket opened over a long period of time, which is not always possible because of restricted computational or energetic constraints. A complementary solution would be the usage of queries instead, resulting in devices explicit request of the contents of the knowledge base at specific instants. This implementation is not available in Cocktail, but is possible and is scheduled for a future work.

1) *Maximum and Minimum Web Thing Triple Count T:* How many triples are needed to setup a Semantic Web Thing? As it has been already said, this calculation depends on the IPs that the Web Thing has to implement. However, by parsing the SPARQL updates we get a total of

$$T = 4 + 9A_{io} + 7(A_i + A_o + E_o) + 5(A_e + E_e) + 13P_v + 12P_e + f + C \quad (1)$$

where four triples are dedicated to `swot:Thing` and `swot:ThingDescription`, nine to a number A_{io} of input–output actions, seven to input, output actions and output events, five to empty actions and events, and 13 (or 12) to properties. To be precise, P_e is the number of properties that have data formatted as `swot:ResourceURI` or `swot:OntologyURI`, and P_v the ones that have data as a literal). A constant f is related to `swot:forProperty` connections, and C includes connections to third parties ontologies. Notice that f cannot be greater that the number of actions and events times the number of properties. Given this, concerning the static description of Web Things, it is possible to outline that four triples is the absolute minimum reachable for a special Web Thing without IPs, and that properties are the pattern that requires the greatest number of triples, due to the fact that they store also data.

2) *Triple Count for Interactions and Dynamic Versus Static (DvS) Ratio:* How many triples are inserted when a new action request is made, or when a new event notification is triggered? Similar to 1, it is possible to obtain the number of triples required for the dynamic control of event and action instances. Refer to Fig. 6 to observe in each situation what are the requirements.

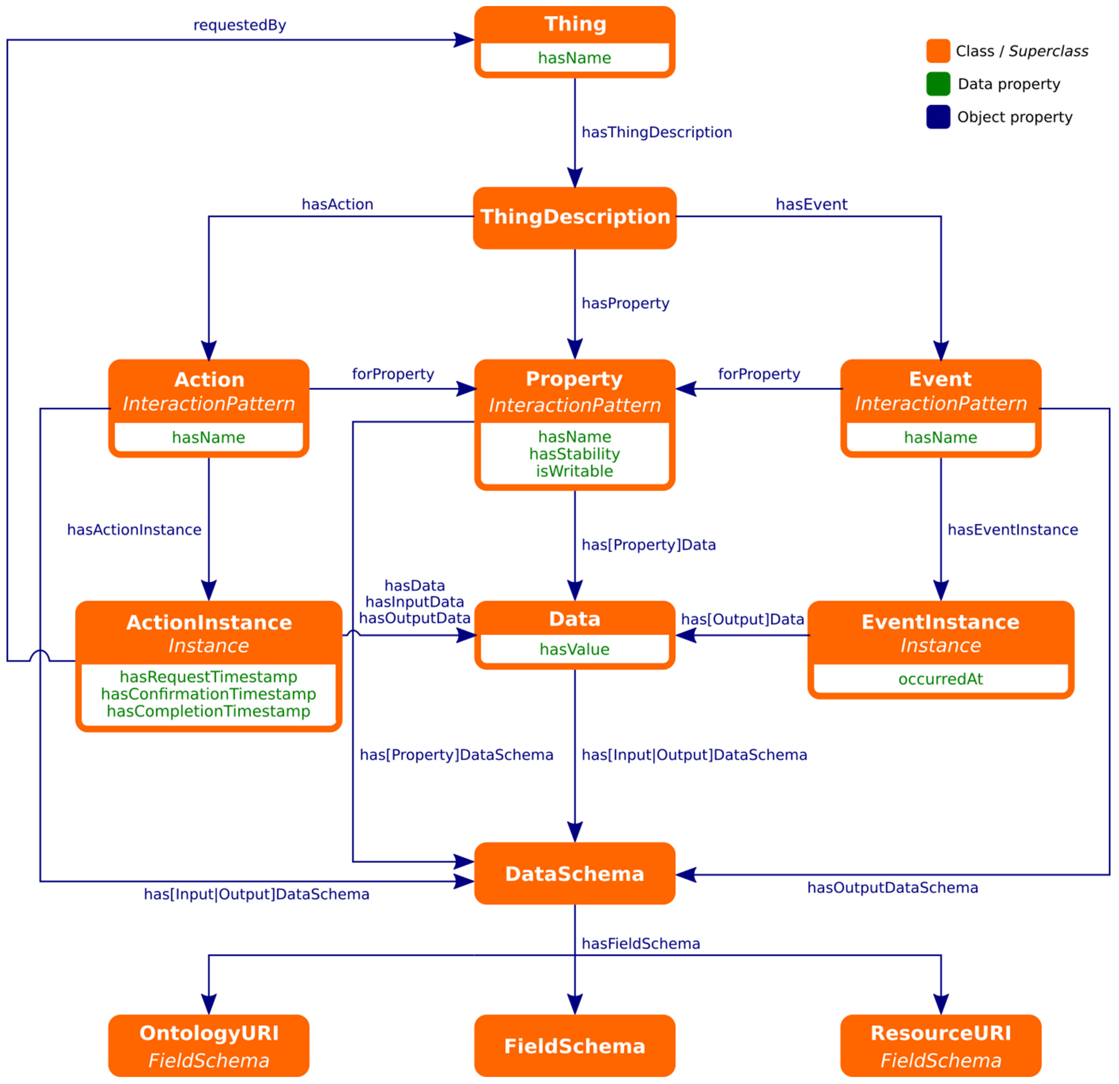

Fig. 7. Full view of the SWoT ontology. `swot:` prefix is omitted from the items shown.

Fig. 6 also introduces the *DvS* ratio (i.e., the triple count for static description over the triple count for dynamic interaction). *DvS* depends on the kind of IP only and is bound to the Web Thing functionality within its context. It is therefore a metric that is obtainable only once its description according to Section IV has been defined by the programmer. *DvS* ratio can be applied to a real Web Thing with all its setup: the greater the ratio, the higher the Web Thing requirements in term of real-time interactions.

3) *Data Format Influence on Triple Dimension:* As explained in Section V, the format of data exchange is of great importance. The complexity of the description of any SWoT device, however, does not depend on that of exchanged data, nor on its dimension or type. The possible alternatives have been fully described and exemplified in Tables I and II, which show that the number of triples exchanged is the same.

In order to make a more complete evaluation of SWOT ontology, some additional considerations can be made thanks to the usage of online tools like PerfectO[15] and OOPS![16] By performing a scan of SWOT ontology through the tools listed in those Websites, we were able to make relevant enhancements to SWOT ontology.

OOPS! tool, however, provides only a formal check of the `.owl` file, while a global view is also needed. This kind of evaluation is possible by examining this article through a set of criteria globally accepted by the research community. As an example, the guidelines for submission to the well-known ISWC Conference[17] are very helpful. Most of the suggested points were largely covered in the previous sections and/or in Appendix B. See Tables VIII and IX. What has to be noticed globally is that SWOT ontology addresses a relevant topic of current research that targets a fusion of Semantic Web and IoT and provides tool for a working implementation. Moreover, it is documented and freely available on GitHub, GNU GPL

[15]http://perfectsemanticweb.appspot.com/?p=ontologyValidation
[16]http://oops.linkeddata.es/
[17]http://iswc2018.semanticweb.org/call-for-resources-track-papers/#

TABLE VIII
MIRO Report of the SWOT Ontology—Part I

A. The basics	
A.1 Ontology name MUST	Semantic Web of Things Ontology (SWoT), version 0.1
A.2 Ontology owner MUST	Francesco Antoniazzi
A.3 Ontology license MUST	GNU General Public License v3.0
A.4 Ontology URL MUST	https://github.com/fr4ncidir/SemanticWoT/blob/master/swot.owl
A.5 Ontology repository MUST	https://github.com/fr4ncidir/SemanticWoT
A.6 Methodological framework MUST	The ontology is clearly divided into static and dynamic description. The former was developed taking into account previously available works on the Web Thing description made by W3C. An additional part was included, related to data formatting and parametrization. The dynamic part was also studied to be coherent and effective. A proof of concept was given by writing the SPARQL Updates and Queries/Subscriptions needed by the Cocktail framework.
B. Motivation	
B.1 Need MUST	The aim of the ontology is to permit the development of Semantic Web of Things applications focusing equally on discovery and accessibility of devices . In fact, as regards this last point, the ontology allows reading properties of Web Things as well as subscribing to their events or invoking their actions.
B.2 Competition MUST	Web of Things ontology [22]
B.3 Target audience MUST	Developers of Semantic Web of Things applications.
C. Scope, requirements, development community	
C.1 Scope and coverage MUST	The ontology defines all the concepts belonging to the Semantic Web of Things domain. The aim of the ontology is to provide a mean for a semantic-enriched interaction with Web Things. This allows developers to exploit semantics for more than just discovering devices. The ontology provides the definitions needed to model the Thing Description as well as the interaction patterns provided by every device.
C.2 Development community MUST	Advanced Research Center on Electronic Systems (ARCES) of the University of Bologna
C.3 Communication MUST	https://github.com/fr4ncidir/SemanticWoT/issues
D. Knowledge acquisition	
D.1 Knowledge acquisition method MUST	Analysis of the literature about Web of Things and experiments carried out at the ARCES department of the University of Bologna in the context of the HABITAT Italian research project and at the Centre for Digital Music (C4DM) of the Queen Mary University of London (QMUL) in the AudioCommons European project.
D.2 Source knowledge location SHOULD	–
D.3 Content Selection SHOULD	The main entities to be represented ontology have been selected according to the literature about Web of Things. In fact, the concepts of Web Thing, Property, Event and Action play a crucial role in the ontology. Moreover, to make the ontology suitable to control devices, some classes have been added to map the input and output data.
E. Ontology content	
E.1 Knowledge representation language MUST	OWL 2 generated by Protégé v5.5.0beta; however, the ontology is at this stage only descriptive, and it uses a reduced subset of OWL 2 capabilities, being the Description Logic ALCRIF(D).
E.2 Development environment OPTIONAL	Protégé v5.5.0beta
E.3 Ontology metrics SHOULD	Number of classes: 14; number of object properties: 20; number of data properties: 9; 0 individuals. Application metrics: Web Thing triple count (Equation 1), DvS contents ratio, Data format impact;
E.4 Incorporation of other ontologies MUST	The ontology is stand-alone. Examples are given on how DUL[8], PROV-O [61] and SOSA[3] can be included. See Appendix C.
E.5 Entity naming convention MUST	Entities follows the CamelCase notation. Both datatype and object properties are named as verb senses with mixedCase notation.
E.6 Identifier generation policy MUST	The SWoT ontology does not Identifiers of the instances must be generated by the application
E.7 Identity metadata policy MUST	All entities have an `rdfs:comment` natural language explanation.
E.8 Upper ontology MUST	No upper ontology is used in this work, to keep SWoT ontology as close as possible to real applications. A suggestion is given on how to include references to DUL[8] in Fig. 1.
E.9 Ontology relationships MUST	20 object properties (plus 20 inverse properties); 9 datatype properties.
E.10 Axiom pattern MUST	The ontology is not yet to be used with reasoners, but rahter oriented at a clear and operactional definition of the concepts in the Semantic Web of Things. 316 axioms included (of which 175 logical axioms, 7 `SubClassOf`, 4 `DisjointClass`, 23 `SubObjectPropertyOf`, 20 `InverseProperty`, 5 `DisjointObjectProperty`, 4 `FunctionalObjectProperty`, 2 `IrreflexiveObjectProperty`, 4 `SubDataPropertyOf`, 8 `FunctionalDataProperty`, 77 `AnnotationAssertion`)
E.11 Deferencable URI OPTIONAL	Some of the entities of the ontology have been conceived to be reachable from the Web. For instance, thing descriptions, data schemas and field schemas. This is a best practice to be followed, to expand interoperability towards future uses.
F. Managing change	
F.1 Sustainability plan MUST	The SWoT Ontology will be adopted in wide research projects (as already done with Habitat and AudioCommons). Feedbacks collected during these activities will guide the future development of the ontology.
F.2 Entity deprecation strategy MUST	No class will be deleted from the ontology. Deprecated classes will be labelled as obsolete with a proper annotation property.
F.3 Versioning policy MUST	The SWOT Ontology adopts sequence-based identifiers for its versions with a major number and a minor number, separated by a dot. A novel release featuring only small changes will cause a switch of the minor number, while relevant and/or structural changes affects also the major number.

licensed, but, as a work in progress, still not submitted to community registries like LOV.

VII. Conclusion

In this article, a complete setup for the new panorama of a more interoperable IoT has been presented, analyzed, and implemented. This contribution is framed in the area known as WoT, inflected here in its SWoT flavor. The rationale behind the work here presented is to exploit the power of semantics in a research field that is evolving very quickly, but still is constrained by devices and systems unable to share all the knowledge they collect in a really interoperable way. The

TABLE IX
MIRO REPORT OF THE SWOT ONTOLOGY—PART II

G. Quality assurance	
G.1 Testing MUST	The tests for SWoT ontology are closely bound to the ones for the Cocktail framework. A first successful test is the realization itself of all Cocktail's SPARQL enquiries. Secondly, the `unittests` available in the repository, allowing to check their consistency by direct usage.
G.2 Evaluation MUST	Some of the metrics reported in [80] have been used to evaluate the SWOT Ontology. In particular the number of classes, properties and individuals have been measured. Moreover, the maximum and minimum Web Thing triple count, the triple dimension of dynamic interactions and the data format influence on triple dimension have been defined to deal with the dynamic aspect of the ontology.
G.3 Examples of use MUST	An example of the ontology is reported in [63]. Other examples are available on the GitHub repository: https://github.com/fr4ncidir/SemanticWoT/tree/master/SWTE_example
G.4 Institutional endorsement OPTIONAL	None.
G.5 Evidence of use MUST	Evidences of use are provided by [63] and [74].

TABLE X
EXPANDED SPARQL PREFIXES

Prefix	**URI**
`rdf:`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
`rdfs:`	`http://www.w3.org/2000/01/rdf-schema#`
`owl:`	`http://www.w3.org/2002/07/owl#`
`swot:`	`http://wot.arces.unibo.it/ontology/web_of_things#`
`xsd:`	`http://www.w3.org/2001/XMLSchema#`
`dul:`	`http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#`
`prov:`	`http://www.w3.org/ns/prov#`
`sosa:`	`http://www.w3.org/ns/sosa/`
`foaf:`	`http://xmlns.com/foaf/0.1/`
`dbpedia:`	`http://dbpedia.org/resource/`
`ns:`	*whatever personal valid namespace*
`ei:`	`ns:MyEvent/instance/`
`ai:`	`ns:MyAction/instance/`

resulting verticality is no more acceptable as the IoT targets the big data, and therefore aims at providing a uniform access to resources, in one hand, and to their information, in the other.

More in detail, this article proposed the SWoT ontology that leverages previous work by Charpenay *et al.* [9], Serena *et al.* [22], and the W3C and pushes toward the ability to control and orchestrate Web Things, more than just discovering them. While attempting to provide a solution to the above-mentioned issues of interoperability, it still needs to be extensively tested in new more and more complex systems, to determine with a higher accuracy its performance and application limits: this is, in fact, the future direction that spans from this article.

APPENDIX A
SWoT ONTOLOGY VIEW

Fig. 7 proposes a full glimpse of SWOT ontology. The prefix (i.e., `swot:`) has been omitted in the image to maximize reading clarity.

APPENDIX B
SWoT ONTOLOGY DOCUMENTATION ACCORDING TO MIRO GUIDELINES

This section reports the documenting information of the SWoT ontology written according to the MIRO guidelines [81]. See Tables VIII and IX.

APPENDIX C
SPARQL PREFIXES TABLE

In Table X, the prefixes used in this article and their expanded identifiers, as a reference for better interpretation of contents are listed.

ACKNOWLEDGMENT

The authors would like to thank L. Turchet (University of Trento) and L. Roffia (University of Bologna) for their valuable hints and support. The work presented in this article has being developed in collaboration with the *Advanced Research Center on Electronic Systems Ercole De Castro*, University of Bologna, Bologna, Italy, and the INFN Research Center *Centro per la ricerca e lo sviluppo nelle tecnologie informatiche e telematiche*, Bologna, Italy.

REFERENCES

[1] K. Ashton, "That 'Internet of Things' thing," *RFID J.*, vol. 22, no. 7, pp. 97–114, 2009.

[2] A. Asin and D. Gascon, "50 sensor applications for a smarter world," Libelium Comunicaciones Distribuidas, Zaragoza, Spain, Rep., 2012.

[3] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 414–454, 1st Quart., 2014.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. ACM 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[5] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[6] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proc. Int. Symp. Handheld Ubiquitous Comput.*, 1999, pp. 304–307.

[7] J. Heuer, J. Hund, and O. Pfaff, "Toward the Web of Things: Applying Web technologies to the physical world," *Computer*, vol. 48, no. 5, pp. 34–42, May 2015.

[8] D. Guinard and V. Trifa, "Towards the Web of Things: Web mashups for embedded devices," in *Proc. Workshop Mashups Enterprise Mashups Lightweight Composition Web (MEM)*, vol. 15. Madrid, Spain, 2009.

[9] V. Charpenay, S. Käbisch, and H. Kosch, "Introducing thing descriptions and interactions: An ontology for the Web of Things," in *Proc. SR SWIT ISWC*, 2016, pp. 55–66.

[10] F. Scioscia and M. Ruta, "Building a semantic Web of Things: Issues and perspectives in information compression," in *Proc. IEEE Int. Conf. Semantic Comput. (ICSC)*, 2009, pp. 589–594.

[11] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic Web," *Sci. Amer.*, vol. 284, no. 5, pp. 34–43, 2001.

[12] J. Klensin and M. Padlipsky, "Unicode format for network interchange," IETF, Fremont, CA, USA, Rep. RFC 5198, 2008.

[13] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifier (URI): Generic syntax," RFC 3986, 2005. [Online]. Available: https://www.ietf.org/rfc/rfc3986.txt

[14] O. Lassila and R. R. Swick, "W3C resource description framework (RDF) model and syntax specification," 1998.

[15] D. Brickley, R. V. Guha, and B. McBride, "RDF schema 1.1," W3C Recommendation, Feb. 2014.

[16] D. L. McGuinness *et al.*, "Owl Web ontology language overview," W3C Recommendation, Oct. 2004.

[17] N. Noy and D. Mcguinness, *Ontology Development 101: A Guide to Creating Your First Ontology*, Knowl. Syst. Lab., 2001.

[18] P. Desai, A. P. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for IoT interoperability," in *Proc. IEEE Int. Conf. Mobile Services (MS)*, 2015, pp. 313–319.

[19] D. Guinard and V. Trifa, *Building the Web of Things: With Examples in Node. JS and Raspberry Pi*. Shelter Island, NY, USA: Manning, 2016.

[20] L. Roffia *et al.*, "A semantic publish-subscribe architecture for the Internet of Things," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1274–1296, Dec. 2016.

[21] L. Roffia, P. Azzoni, C. Aguzzi, F. Viola, F. Antoniazzi, and T. S. Cinotti, "Dynamic linked data: A SPARQL event processing architecture," *Future Internet*, vol. 10, no. 4, p. 36, 2018.

[22] F. Serena, M. Poveda-Villalón, and R. García-Castro, "Semantic discovery in the Web of Things," in *Proc. Int. Conf. Web Eng.*, 2017, pp. 19–31.

[23] N. Shadbolt, T. Berners-Lee, and W. Hall, "The semantic Web revisited," *IEEE Intell. Syst.*, vol. 21, no. 3, pp. 96–101, Jan./Feb. 2006.

[24] M. Uschold, M. Healy, K. Williamson, P. Clark, and S. Woods, "Ontology reuse and application," in *Formal Ontology in Information Systems*, vol. 179. Amsterdam, The Netherlands: IOS Press, 1998, p. 192.

[25] E. P. Bontas, M. Mochól, and R. Tolksdorf, "Case studies on ontology reuse," in *Proc. Int. Conf. Knowl. Manag. (IKNOW)*, vol. 74, 2005, p. 345.

[26] J. Z. Pan, L. Serafini, and Y. Zhao, "Semantic import: An approach for partial ontology reuse," in *Proc. 1st Int. Conf. Modular Ontol.*, vol. 232, 2006, pp. 71–84.

[27] S. Karim, K. Latif, and A. M. Tjoa, "Providing universal accessibility using connecting ontologies: A holistic approach," in *Proc. Int. Conf. Univ. Access Human–Comput. Interact.*, 2007, pp. 637–646.

[28] F. Antoniazzi and F. Viola, "RDF graph visualization tools: A survey," in *Proc. 23rd Conf. Open Innov. Assoc. (FRUCT)*, Helsinki, Finland, 2018, pp. 27–38. [Online]. Available: http://dl.acm.org/citation.cfm?id=3299905.3299909

[29] N. Bikakis and T. Sellis, "Exploration and visualization in the Web of big linked data: A survey of the state of the art," *arXiv preprint arXiv:1601.08059*, 2016.

[30] M. Rinne, E. Blomqvist, R. Keskisärkkä, and E. Nuutila, "Event processing in RDF," in *Proc. WOP*, 2013, pp. 52–64.

[31] J. Soldatos *et al.*, "OpenIoT: Open source Internet-of-Things in the cloud," in *Interoperability and Open-Source Solutions for the Internet of Things*. Cham, Switzerland: Springer, 2015, pp. 13–25.

[32] N. Seydoux, K. Drira, N. Hernandez, and T. Monteil, "IoT-O, a core-domain IoT ontology to represent connected devices networks," in *Proc. Eur. Knowl. Acquisition Workshop*, 2016, pp. 561–576.

[33] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor, "IoT-Lite: A lightweight semantic model for the Internet of Things and its use with dynamic semantics," *Pers. Ubiquitous Comput.*, vol. 21, no. 3, pp. 475–487, 2017.

[34] D. Puiu *et al.*, "CityPulse: Large scale data analytics framework for smart cities," *IEEE Access*, vol. 4, pp. 1086–1108, 2016.

[35] A. Kamilaris, A. Pitsillides, F. X. Prenafeta-Bold, and M. I. Ali, "A Web of Things based eco-system for urban computing-towards smarter cities," in *Proc. IEEE 24th Int. Conf. Telecommun. (ICT)*, 2017, pp. 1–7.

[36] A. Kamilaris, F. Gao, F. X. Prenafeta-Boldú, and M. I. Ali, "Agri-IoT: A semantic framework for Internet of Things-enabled smart farming applications," in *Proc. IEEE 3rd World Forum Internet Things (WF-IoT)*, 2016, pp. 442–447.

[37] F. Viola, F. Antoniazzi, C. Aguzzi, C. Kamienski, and L. Roffia, "Mapping the NGSI-LD context model on top of a SPARQL event processing architecture: Implementation guidelines," in *Proc. 24th Conf. Open Innov. Assoc. (FRUCT)*, 2019, pp. 493–501.

[38] R. Tommasini, P. Bonte, E. D. Valle, E. Mannens, F. De Turck, and F. Ongenae, "Towards ontology-based event processing," in *OWL: Experiences and Directions–Reasoner Evaluation*. Cham, Switzerland: Springer, 2016, pp. 115–127.

[39] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, "Toward better horizontal integration among IoT services," *IEEE Commun. Mag.*, vol. 53, no. 9, pp. 72–79, Sep. 2015.

[40] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "IoT gateway: BridgingWireless sensor networks into Internet of Things," in *Proc. IEEE/IFIP 8th Int. Conf. Embedded Ubiquitous Comput. (EUC)*, 2010, pp. 347–352.

[41] S. K. Datta, C. Bonnet, and N. Nikaein, "An IoT gateway centric architecture to provide novel M2M services," in *Proc. IEEE World Forum Internet Things (WF-IoT)*, 2014, pp. 514–519.

[42] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The Internet of Things has a gateway problem," in *Proc. ACM 16th Int. Workshop Mobile Comput. Syst. Appl.*, 2015, pp. 27–32.

[43] A. Gangemi, R. Lillo, G. Lodi, A. G. Nuzzolese, and V. Presutti, "A pattern-based ontology for the Internet of Things," in *Proc. WOP@ ISWC*, 2017.

[44] W. Wang, S. De, R. Töenjes, E. S. Reetz, and K. Moessner, "A comprehensive ontology for knowledge representation in the Internet of Things," in *Proc. IEEE 11th Int. Conf. Trust Security Privacy Comput. Commun. (TrustCom)*, 2012, pp. 1793–1798.

[45] P.-Y. Vandenbussche, G. Atemezing, M. Poveda-Villalón, and B. Vatant, "Linked open vocabularies (LOV): A gateway to reusable semantic vocabularies on the Web," *Semantic Web*, vol. 8, no. 3, pp. 437–452, 2017.

[46] A. Gyrard, C. Bonnet, K. Boudaoud, and M. Serrano, "LOV4IoT: A second life for ontology-based domain knowledge to build semantic Web of Things applications," in *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud (FiCloud)*, 2016, pp. 254–261.

[47] A. Gyrard, A. Zimmermann, and A. Sheth, "Building IoT-based applications for smart cities: How can ontology catalogs help?" *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3978–3990, Oct. 2018.

[48] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 1, pp. 70–95, Feb. 2016.

[49] D. J. Wu, A. Taly, A. Shankar, and D. Boneh, "Privacy, discovery, and authentication for the Internet of Things," in *Proc. Eur. Symp. Res. Comput. Security*, 2016, pp. 301–319.

[50] B. Djamaa, M. A. Kouda, A. Yachir, and T. Kenaza, "FetchIoT: Efficient resource fetching for the Internet of Things," in *Proc. IEEE Feder. Conf. Comput. Sci. Inf. Syst. (FedCSIS)*, 2018, pp. 637–643.

[51] F. Viola, L. Turchet, F. Antoniazzi, and G. Fazekas, "C minor: A semantic publish/subscribe broker for the Internet of musical things," in *Proc. IEEE 23rd Conf. Open Innov. Assoc. (FRUCT)*, 2018, pp. 405–415.

[52] P. Waher and R. Klauck, "Internet of Things-discovery," 2018.

[53] S. Cirani *et al.*, "A scalable and self-configuring architecture for service discovery in the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 5, pp. 508–521, Oct. 2014.

[54] S. Mayer and D. Guinard, "An extensible discovery service for smart things," in *Proc. ACM 2nd Int. Workshop Web Things*, 2011, p. 7.

[55] S. B. Fredj, M. Boussard, D. Kofman, and L. Noirie, "Efficient semantic-based IoT service discovery mechanism for dynamic environments," in *Proc. IEEE 25th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, 2014, pp. 2088–2092.

[56] M. Ganzha, M. Paprzycki, W. Pawlowski, P. Szmeja, and K. Wasielewska, "Semantic technologies for the IoT—An inter-IoT perspective," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, 2016, pp. 271–276.

[57] F. Gao, M. I. Ali, and A. Mileo, "Semantic discovery and integration of urban data streams," in *Proc. S4SC*, vol. 7, 2014, pp. 15–30.

[58] A. Kamilaris, S. Yumusak, and M. I. Ali, "WOTS2E: A search engine for a semantic Web of Things," in *Proc. IEEE 3rd World Forum Internet Things (WF-IoT)*, 2016, pp. 436–441.

[59] L. Sciullo, C. Aguzzi, M. Di Felice, and T. S. Cinotti, "WoT store: Enabling things and applications discovery for the W3C Web of Things," in *Proc. 16th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2019, pp. 1–8.

[60] K. Dar, A. Taherkordi, R. Rouvoy, and F. Eliassen, “Adaptable service composition for very-large-scale Internet of Things systems,” in *Proc. ACM 8th Middleware Doctoral Symp.*, 2011, p. 2.

[61] T. Lebo *et al.*, “PROV-O: The PROV ontology,” *W3C Recommend.*, vol. 30, 2013.

[62] G. Tzortzis and E. Spyrou, “A semi-automatic approach for semantic IoT service composition,” in *Proc. Workshop Artif. Intell. Internet Things Conjunction SETN*, 2016, pp. 1–6.

[63] F. Viola, A. Stolfi, A. Milo, M. Ceriani, M. Barthet, and G. Fazekas, “Playsound.space: Enhancing a live music performance tool with semantic recommendations,” in *Proc. 1st Int. Workshop Semantic Appl. Audio Music*, 2018, pp. 46–53.

[64] Z. Song, A. A. Cárdenas, and R. Masuoka, “Semantic middleware for the Internet of Things,” in *Proc. IEEE Internet Things (IOT)*, 2010, pp. 1–8.

[65] J. Delsing, *IoT Automation: Arrowhead Framework*. London, U.K.: CRC Press, 2017.

[66] P. P. Jayaraman, D. Palmer, A. Zaslavsky, A. Salehi, and D. Georgakopoulos, “Addressing information processing needs of digital agriculture with OpenIoT platform,” in *Interoperability and Open-Source Solutions for the Internet of Things*. Cham, Switzerland: Springer, 2015, pp. 137–152.

[67] A. Bassi, M. Bauer, M. Fiedler, and R. V. Kranenburg, *Enabling Things to Talk*. Heidelberg, Germany: Springer, 2013.

[68] P. Barnaghi, W. Wang, C. Henson, and K. Taylor, “Semantics for the Internet of Things: Early progress and back to the future,” *Int. J. Semantic Web Inf. Syst.*, vol. 8, no. 1, pp. 1–21, 2012.

[69] D. Pfisterer *et al.*, “SPITFIRE: Toward a semantic Web of Things,” *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 40–48, Nov. 2011.

[70] M. Ruta, F. Scioscia, and E. Di Sciascio, “Enabling the semantic Web of Things: Framework and architecture,” in *Proc. IEEE 6th Int. Conf. Semantic Comput.*, 2012, pp. 345–347.

[71] D. Raggett, “The Web of Things: Challenges and opportunities,” *Computer*, vol. 48, no. 5, pp. 26–32, May 2015.

[72] M. Noura, S. Heil, and M. Gaedke, “GROWTH: Goal-oriented end user development for Web of Things devices,” in *Proc. Int. Conf. Web Eng.*, 2018, pp. 358–365.

[73] Z. Wu, Y. Xu, Y. Yang, C. Zhang, X. Zhu, and Y. Ji, “Towards a semantic Web of Things: A hybrid semantic annotation, extraction, and reasoning framework for cyber-physical system,” *Sensors*, vol. 17, no. 2, p. 403, 2017.

[74] F. Antoniazzi, G. Paolini, L. Roffia, D. Masotti, A. Costanzo, and T. S. Cinotti, “A Web of Things approach for indoor position monitoring of elderly and impaired people,” in *Proc. IEEE 21st Conf. Open Innov. Assoc. (FRUCT)*, 2017, pp. 51–56.

[75] M. Swan, “Sensor Mania! The Internet of Things, wearable computing, objective metrics, and the quantified self 2.0,” *J. Sensor Actuator Netw.*, vol. 1, no. 3, pp. 217–253, 2012.

[76] P. Gope and T. Hwang, “BSN-care: A secure IoT-based modern healthcare system using body sensor network,” *IEEE Sensors J.*, vol. 16, no. 5, pp. 1368–1376, Mar. 2016.

[77] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, “Internet of musical things: Vision and challenges,” *IEEE Access*, vol. 6, pp. 61994–62017, 2018.

[78] C. Bormann, M. Ersue, and A. Keranen, “Terminology for constrained-node networks,” IETF, Fremont, CA, USA, Rep. RFC 7228, 2014.

[79] P. Hitzler and F. Van Harmelen, “A reasonable semantic Web,” *Semantic Web*, vol. 1, nos. 1–2, pp. 39–44, 2010.

[80] M. Fernández, C. Overbeeke, M. Sabou, and E. Motta, “What makes a good ontology? A case-study in fine-grained knowledge reuse,” in *The Semantic Web*, A. Gómez-Pérez, Y. Yu, and Y. Ding, Eds. Berlin, Germany: Springer, 2009, pp. 61–75.

[81] N. Matentzoglu, J. Malone, C. Mungall, and R. Stevens, “MIRO: Guidelines for minimum information for the reporting of an ontology,” *J. Biomed. Semantics*, vol. 9, no. 1, p. 6, Jan. 2018. [Online]. Available: https://doi.org/10.1186/s13326-017-0172-7

Francesco Antoniazzi (M’17) was born in Conegliano, Italy, in 1991. He received the B.Sc. degree in electronics and telecommunication engineering and the M.Sc. degree in electronics engineering from the University of Bologna, Bologna, Italy, in 2013 and 2016, respectively, where he is currently pursuing the Ph.D. degree in computer science and engineering.

He is a Research Fellow with the University of Bologna. He was also a Research Fellow with the CNAF Section, Istituto Nazionale di Fisica Nucleare, Rome, Italy. His current research interest includes future Web and Internet of Things technologies.

Fabio Viola (M’16) was born in Lecce, Italy, in 1986. He received the degree in information engineering from the University of Salento, Lecce, in 2011, and the master’s (*summa cum laude*) degree and the Ph.D. degree in computer science and engineering from the University of Bologna, Bologna, Italy, in 2014 and 2019, respectively.

From 2014 to January 2019, he was a Research Fellow with the Advanced Research Center on Electronic Systems, University of Bologna. He is currently with the Centro Nazionale per la Ricerca e Sviluppo nelle Tecnologie Informatiche e Telematiche (INFN CNAF), Bologna. His current research interest includes semantic technologies for the Web of Things.