ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN:
Computer Science and Engineering

Ciclo XXXII

Settore Concorsuale: 09/H1
Settore Scientifico Disciplinare: ING-INF/05

SEMANTICS DRIVEN AGENT PROGRAMMING

Presentata da: **Francesco Antoniazzi**

Coordinatore Dottorato
Prof. Davide Sangiorgi

Supervisore
Prof. Tullio Salmon Cinotti

Co-Supervisore
Prof. Luciano Bononi

Esame finale anno 2020

This work is licensed under a Creative Commons "Attribution-NonCommercial-NoDerivatives 4.0 International" license.

✉ francesco.antoniazzi@unibo.it
✉ francesco.antoniazzi@outlook.com

Think green before printing this Thesis

Contents

Abstract in English

In the last two decades the Information Technology changed substantially the life of people all around the World. Just a few years ago, for instance, paper support was needed to exchange all kind of data, while now electronics is indeed the main instrument of communication. This mutation was originally due mostly to the efficiency, while now it is, hopefully, also due to an increased attention towards environmental issues.

Information and data have proved over the time the importance of their role, contributing to a plethora of applications that allow the physical world to interact with mankind by the means of services dispatched pervasively and freely accessible. The Internet is the kernel of such complex setup that is called Internet of Things (IoT).

The IoT inherited from the Internet a chaotic interface. Protocols, conventions, mechanisms are different from an application to the other, and it is difficult and expensive to discover and make applications compatible with one another. From this consideration two exceptional ideas were born, namely the Semantic Web and the Web of Things (WoT). The latter would unify the IoT on an application level shared view, enabling standard discovery mechanisms and definitions. The former, on the other hand, intents to provide the tools to formalize the knowledge contents of the World Wide Web in a simultaneously human and machine understandable way.

This Thesis aims to explore both these two concepts and merge them into the Semantic Web of Things using the best of each. Therefore we hereby propose, describe, evaluate and use two ontologies: the Internet of Musical Things ontology, aiming to outline a semantic description of IoT; and a Semantic WoT ontology, aiming to push further the state of the art of IoT unification and standardization through a dynamic semantic approach.

Abstract in Italiano

Negli ultimi due decenni le nuove Tecnologie dell'Informazione hanno cambiato radicalmente la vita delle persone in tutto il mondo. Soltanto qualche anno fa, per esempio, lo scambio di informazione era necessariamente effettuato sotto forma cartacea in quasi ogni ambito. Oggi, invece, il mezzo elettronico viene privilegiato sempre più per questioni di efficienza nonché, recentemente, si spera anche per motivi legati alla sostenibilità ambientale.

L'informazione ha dato prova, nel corso del tempo, della sua importanza. Ha contribuito a rendere possibili numerosissime applicazioni in grado di far interagire l'umanità con il mondo fisico attraverso un'astrazione composta da servizi facilmente accessibili e distribuiti ovunque. Internet è il cuore di questo grande sistema chiamato Internet of Things (IoT).

L'IoT ha in comune con Internet la sua interfaccia caotica e la mancanza di ordine. I protocolli, le convenzioni, i meccanismi cambiano da una applicazione all'altra, rendendo difficile e costoso scoprire e creare sistemi compatibili. Da queste considerazioni ormai accettate dalla comunità traggono origine due concetti eccezionali: il Semantic Web e il Web of Things (WoT). Quest'ultimo ha come fine quello di unificare l'IoT ad un livello applicativo condiviso rendendo disponibili definizioni e meccanismi standard per la scoperta dei dispositivi. Il primo, invece, fornisce degli strumenti per formalizzare la conoscenza distribuita nel World Wide Web in modo che sia contemporaneamente fruibile all'uomo e alle macchine.

Questa Tesi si accinge ad esplorare i due concetti appena descritti, ed a riunirli usando il meglio di entrambi nel Semantic Web Of Things. Per fare ciò si proporranno, descriveranno, valuteranno ed useranno due ontologie: l'ontologia dell'Internet of Musical Things, che servirà per mostrare una definizione semantica dell'IoT; e l'ontologia del Semantic WoT, il cui scopo è di spingere oltre lo Stato dell'Arte nell'unificazione dell'IoT e nella sua standardizzazione attraverso un approccio semantico e dinamico.

Abstract en Français

Les deux dernières décennies ont vu les nouvelles Technologies de l'Information changer de manière radicale la vie des gens partout dans le monde. Il n'y a que quelques années, par exemple, des supports en papier étaient nécessaires pour l'échange des données, alors qu'à présent l'instrument principal est l'électronique. Ce changement était dû à l' origine à l'efficacité de la communication. Maintenant, on l'espère, la raison est aussi liée à la tutelle de l'environnement.

Il a été largement démontré que l'information joue un rôle essentiel : innombrables applications ont été développées pour connecter le monde physique et l'humanité à travers des services distribués partout et librement accessibles. Internet est au centre de toute cette infrastructure, qui n'est autre que l'Internet des Objets (IoT).

L'IoT et Internet ont en commun leur organisation chaotique. Les protocoles, les conventions, les fonctionnements internes peuvent être très différents d'une application à l'autre : il est souvent difficile et coûteux de découvrir et créer des applications compatibles avec le reste des systèmes qui sont à disposition. Ce concept est à l'origine de deux idées exceptionnelles : le Web Sémantique, et le Web des Objets (WoT). Celui-ci a pour but d'unifier l'IoT à un niveau applicatif commun avec des mécanismes de découverte et un vocabulaire standard. Le premier, d'autre part, propose les instruments pour mettre de l'ordre dans la connaissance du World Wide Web, de façon à la rendre à la fois compréhensible à l'être humain, et aux machines.

Cette Thèse explore donc les deux idées à peine présentées, et additionne leurs meilleures qualités pour obtenir le Web des Objets Sémantiques. Ainsi sont proposées, décrites, évaluées et utilisées deux ontologies : celle de l'Internet des Objets Musicaux, pour produire une description sémantique de l'IoT ; et celle du WoT Sémantique, qui voudrait avancer l'état de l'art de la recherche sur l'unification de l'IoT de manière sémantique et dynamique.

Introduction

Semantic Driven Agent Programming idea joins together concepts that are relatively well-known in literature. Agent programming, as it will be discussed, dates back to the early 90's [1], while the Semantic Web was introduced by Tim Berners Lee in 2001 [2]. The innovation proposed by this Thesis, therefore, is not given by those concepts taken on their own, but is rather the outcome of their integration into several ideas and projects, which are going to be explored in the following Chapters.

The effort made to define such collaboration targets the third pole of interest of the Thesis, namely the Internet of Things (IoT) [3]. As a matter of fact, the IoT represents one of the most outstanding global creativity sources of the last century. With the advent of affordable computing, pervasive electronics and the Internet, technology started to follow a quick and exponential growth trend that today incredibly changed our society [4]. Moreover, technology became part of the life of almost everyone as a service provider. Alongside with this revolution, the growth trend involved also an easier access to the tools needed to create new things and to put into practice new ideas, giving birth to *maker* movement [5, 6]. Anyone, with some know-how in electronics and informatics, can try to implement his/her own projects, share it to the community and, eventually, also make money with it [7, 8].

All this, with a continuous feedback process to and from the industry and the academy, resulted in a worldwide productivity that still benefits almost every technological field.

Over the time, however, it became clear that this race to innovation was extremely chaotic [9]. In fact, projects were designed in a vertical way from the beginning to the end, and the problem of joining any two of them was often the same as restarting both of them from scratch in a single one performing both tasks [10].

The present work tries to fight with this verticality enabling the IoT to use the tools of the Semantic Web. Ontologies, in the semantic interpretation of the term, represent a great opportunity to provide a common discussion pattern among vertical projects [11]. In particular, two ontologies will be provided here in the following Chapters.

The former, in Section 2.2, is the Internet of Musical Things (IoMusT) [**iomust**] ontology. The discussion over this ontology will allow us to introduce a full set of concepts and ideas related to an horizontal approach to IoT. Besides, the musical IoT application represents an innovative application in the whole research field.

The latter, in Chapter 3, is the Semantic Web of Things (SWOT) ontology [**swot**].

Such ontology is located at a higher level compared to the previous one. As a matter of fact, in this case we target a newer approach to IoT that treats the pervasive electronics resources as web resources and renames IoT as Web of Things (WoT) [12, 13]. The innovation proposed here is the addition of semantics to further prevent the formation of fragmented entities.

Applying the Semantic Web technologies to the IoT and the WoT by the means of an ontology will give a predefined shared way to set up projects at information and organization level. For instance, in IoT systems there is often a lack of coherence about the definition of the *thing*, which is a core concept that guarantees interoperability. A standardized approach for concept interpretation would definitely help in building or integrating software and data architecture. In their interesting survey, Liu et al. [14] listed a number of different possible definitions for the *thing*. The two most relevant among them, for this Thesis, are W3C's and IEEE's hereby reported for reader's convenience.

IEEE [The thing] *is any physical object relevant from a user or application perspective.*

W3C [The things] *can be virtual representations of a physical or abstract entities. They can be connected or not connected. Each thing can have one or more virtual representations. Things can have histories, and have identities, rich descriptions, services, access control and data handling policies. They have URIs.*

Both those definitions make clear reference to the fact that the *things* are not static elements in an application, but instead are interactive factors of change. They are either active or passive in their environment, and consequently contribute to modify the application context. For this reason, as the term *thing* is rather confusing and yet not far from a philosophical concept, in this Thesis we will refer to it with a different word, i.e. *agent*, unless otherwise stated. Such lexical change is derived from the comparison between the two definitions above and the aforementioned paper by Shoham [1]. Some other works, like Mzahm et al. [15] and Savaglio et al. [16] also go in that same direction. In particular, we refer to the agent as an entity that has access to context information, which the Semantic Web is able to realize through its concept interconnection expressivity.

We will discuss how those agents moving and interacting in the semantic architecture can be programmed and, even better, how they can provide us as much services as possible, with the lesser effort. Globally, as a result, this work contains a set of propositions that target an integration of the Semantic Web in the IoT towards new working, plug-and-play and fully integrated solutions.

Consequently, the outcome of this study is a complete description of why and how we should put some order in the IoT as it is known today, as well as decide a shared approach to perform this task. It is worth noticing, moreover, that a common way to think about and use the IoT like the one suggested here may enable new opportunities also in other related fields of ICT, like the one connected to the Big Data revolution [17].

The Thesis is organized as follows. In Chapter 1 the research background of the whole work will be explained over its complete state-of-the-art. For instance, the IoT evolution is there summarized, along with a view on the protocols and the challenges that originated from it. Then the Semantic Web will be introduced and, eventually, the hardware-software architectures that have been developed to use it overcoming its limitations. A research on how to visualize semantic graphs is also included, which we performed with didactic purposes.

Chapter 2, then, will provide a semantic layered view of the IoT by means of the ideas outlined in Chapter 1. The Internet of Musical Things ontology is there fully explained, leveraging general concepts that may be useful to grant horizontal interoperability among various IoT systems within different environments and backgrounds.

Chapter 3, instead, includes the basic concepts needed to start talking about semantic IoT, and extends the discussion to the WoT. It contains, therefore, the description of the Semantic Web of Things Ontology: it compares the approach with the pre-existent works made by W3C, and following this it proceeds with a semantic description of *semantic* Web Things *dynamic semantic* interaction.

Following the theoretical views, we provide in Chapter 4 a rather complete evaluation of the ontology. To do so, we show the functionality of a Python framework to use the classes and the relationships previously introduced on a simple example, yet rich in interesting insights. Then, the ontology itself is considered as a whole for evaluation, leading us to ideas for future works to achieve additional results.

Eventually, we make a synthesis and conclude the Thesis.

The work discussed in this Thesis was carried in collaboration with:

- Industrial Research Center on ICT (CIRI ICT) - University of Bologna;
- Advanced Research Center on Electronic Systems *Ercole de Castro* (ARCES) - University of Bologna;
- National Institute of Nuclear Physics (INFN) - Section CNAF;
- Centre for Digital Music (C4DM) - Queen Mary University of London;

Additionally, the following research projects represent the background in which all the PhD and this Thesis work has been performed:

HABITAT: Home Assistance Based on the Internet of Things for the AuTonomy, a POR-FESR 2014-2020 project related to home-caring Internet of Things;

`http://www.habitatproject.info/`

OPEN-NEXT: Real-time and open-source software for embedded platforms of next generation, a POR-FESR 2014-2020 project aiming to develop a platform for industrial real-time applications working with different devices and technologies;

`http://www.open-next.it/en/homepage-en/`

AUDIOCOMMONS, an Horizon 2020 project (research and innovation grant 688382) aiming at *bringing Creative Commons audio content to the creative industries*;

https://www.audiocommons.org/

SWAMP: IoT based methods and approaches for smart water management in precision irrigation domain, a project for smart agriculture development Horizon 2020-EUB-2017, n. 777112, funded by the European Commission under: H2020-EU.2.1.1. - INDUSTRIAL LEADERSHIP - Leadership in enabling and industrial technologies - Information and Communication Technologies (ICT);

http://swamp-project.org/

Chapter 1

Background Research

As discussed in the Introduction Section, this Chapter provides a complete related work overview of the main relevant topics addressed in the Thesis. In particular, Section 1.1 contains a large reference on the IoT-related technologies that represented a source of inspiration or a basis for the Chapters that will follow. In Section 1.2, instead, the Semantic Web vision and its capabilities are outlined. Some applications will be described to exemplify.

Then, in Section 1.3, the base of IoT semantic approach for interoperability developed at ARCES is explored. Almost all the work exposed in this Thesis lays on such approach, that enables dynamic responsiveness in semantic contexts.

1.1 Internet of Things

Electronics is everywhere: it is *pervasive* [18] both in the spatial and conceptual meaning of the term. That is, sensors are dispatched in almost every environment, and measure almost every physical entity available contributing with their data to countless applications. Actuators, in a similar way, trigger changes within the environment, so that users can benefit from the effects.

The IoT revolution [4, 19] deeply impacted many sectors of everyday life: nowadays research and industry are proceeding towards the smartification of reality, including smart homes [20], smart cities [21, 22], smart health-care [23, 24, 25], smart agriculture [26, 27] and so forth.

Listing all possible architectures employed in IoT applications is a task that is almost impossible, due to the huge variety of the topics that over the time have been concerned by the smartification process. As a reference, anyway, here follows a description at high level of some of them, mentioning in particular the ones that are relevant for the next Chapters.

IoT, as we said, is a term that was coined by Kevin Ashton in 1999 [3]. The vision, indeed, has been since then modified to be coherent with the current available technologies. Just consider that back at that time Internet access was possible, but not as common as it is today, available in almost every home. This is an interesting indicator of how different might have been the perspective.

Devices, before being connected to the Internet, were directly communicating with one another and therefore we had short distance and small environments. To this extent, a rich collection of protocols and standards were developed like the USB standard, dating back to 1996, and Bluetooth, whose first device was being sold in late 1999[1]. The connectivity through the Internet, due to the limited bandwidth available for data transfer, was not yet ready to support online sensors and actuators and the exchange of their data.

During the following years silicon production processes and studies on computer architectures considerably evolved, revealing the bases of modern IoT. First of all, it was possible to have CPUs with more calculation power in less space; secondly, broadband connection to the Internet became a reality, together with new mobile communication systems (3, 4, and now 5G). In addition, various wireless protocols like Wi-Fi, ZigBee, 6LoWPAN, LoRa became easily available [28]. In some cases, e.g., LoRa and 6LoWPAN, they were designed with the explicit goal of enabling IoT technologies. As Mulligan states in [29], *"The concept was born from the idea that the Internet Protocol could and should be applied to even the smallest of devices"*.

Researches like [30, 31] provide a complete discussion on the IoT background, covering the various design layers from calculation units to communication protocols and global architecture.

Such availability of new computation and communication instruments revealed itself to be a powerful trigger for the spreading of IoT, which was then interpreted as the solution to many complex important problems. Moreover, the variety of approaches exponentially increased along with the number of questions that were answered through IoT techniques. So, as a result, defining and realizing an IoT project eventually produced others ideas, in a positive and creative feedback loop similar to the ones in Figg. 1.1 (a) and (b), that are a synthesis of the one exposed by Jacobson et al. [32]. As it can be seen there the *problem* definition derives from the *evaluation* of a previous project; a *technical analysis* follows, where the state of the art is analyzed, and choices are made: which is the best hardware platform, which are the most effective communication protocols for this application? How to, in general, *design* the IoT solution?

Although depicting a rather general approach to engineering problems, Fig. 1.1 fits in a very special way the IoT, hiding its main drawback. Every step in these two flow charts is developed independently from one project to the other. This, eventually, produces applications and systems that work on their own, that are connected to their network, but still are unable to interoperate. Such broken communication may happen at any level of ISO-OSI stack, that is the centre of IoT interoperability, as it is reported by Banerjee et al. [33] and by Rayes et al. [34]:

Physical and DataLink Layer: two IoT systems will not be able to interact properly if it is not possible to share the physical communication mean and if there is no

[1] `https://www.bluetooth.com/about-us/our-history/`

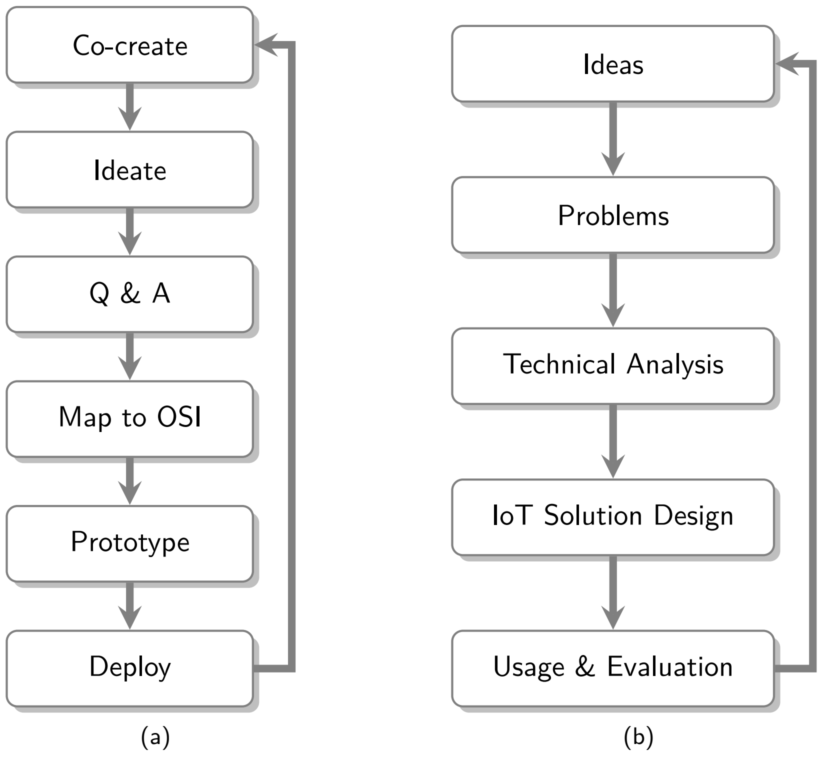


Figure 1.1 – Typical problem solving workflow for IoT projects, as exposed by Jacobson et al. [32] (1.1a), and in a higher level synthesis (1.1b).

match in the how they perform direct contact. This includes the usage of legacy and/or constrained networks or devices requiring specific setups [35]. Example: system A communicates with USB protocol, system B with Bluetooth.

When layers 1 and 2 are not matching, device and service *local discovery* is impossible, signifying that devices and systems cannot even be aware that other devices and systems are dispatched in the same environment.

Network and Transport Layer: given that two IoT systems share the same Physical and DataLink layers, their remote connection is possible only if routing is also possible from one to the other, and if they agree on how to perform information exchange. For instance, a system exploiting UDP vs a system using TCP and targeting a very different audience.

Layer 3 and 4 match is necessary for *remote discovery* and complex data share mechanisms [36].

Upper layers: are necessary when the setup gets more complex than mere communication of raw data. IoT systems exploiting upper layers share higher level information, which has to be interpreted in the very same way: from the character sequence choice to the content scheme used to format the data.

A relevant example, here, may be the interaction of two different entities when one produces JSON-formatted data, and the other expects XML. Or, similarly, when they both interact through JSON, but the former uses the tag `book` to identify an instance of literary creation, and the latter as the action of making a reservation for a flight.

A sequence of choices is made for every IoT system within this stack. This builds up the concept of *vertical silos*, which means that once that two developed applications are up and running, either the design choices are the same, either at some point information will get stuck and collaboration will not be possible. Studies on how to overcome this fragmented vertical reality were performed since the very beginning of IoT era, and resulted in an extremely rich literature [37, 38]. Among the results, research provided over the years also

- New protocols like AMQP, MQTT, CoAP [39, 40]. In such context, devices are supposed to be able to use application layer protocols (i.e., they have enough memory and computational power to implement all the stack), and use a topic-based logic to exchange information. That is, interoperability is defined as an agreement on a topic taxonomy, and data is exchanged through a middleware that is able to implement communication on top of topic channels.

- Translators from a protocol to the other, like presented in [41, 42], or from one hub to the other [43]. This means, in particular, that the communication between entities is mediated by a third entity designed to act as a gateway [44, 45]. This approach, in some cases, generated heavy critics because of gateways development complexity, and privacy issues [46];

- Information level interoperability, which may be considered as a mixture of the two previous points, in addition with semantic techniques [47, 48, 49, 50]. This approach will be largely explored in this Thesis: it requires applications and systems to organize their data according to standardized schemas (i.e., ontologies).

Globally, anyway, what is clear is that there must be a level shared by all IoT projects and systems to make environments communicate to each other and to assure that, for the future, service update will not imply a complete and expensive redesign.

It is worth saying also that the concepts of Cloud Computing before [51], and Fog Computing later [52], were in the end pursuing that same idea of achieving full connection of information. In fact, Big Data [53] has been largely feeding from both of them but still, among its known drawbacks [54, 55], the disorder and incoherence of information are probably the mostly well known.

The three points previously listed besides show that information interoperability is the key for breaking the silos, even more than protocol aspects (whose lifecycle brings them to evolve continuously following technological trends). As a matter of fact, even the new facet of IoT, i.e. the Web of Things (WoT) [12, 56], whilst calling for the uniform usage of the protocols of the web in the IoT, must face the problem of interpretation of resources.

In this Thesis, to achieve this shared access to information we exploit the capabilities of the Semantic Web to describe data generated by machines in a machine understandable way [2, 11]. The next Section will provide some general examples of it, while Section 1.3 will explain some tools necessary for an application of the Semantic Web to the Internet of Things world: the SPARQL Event Processing Architecture.

1.2 The Semantic Web

The possibility to freely interlink any piece of information and any service with all the others is to be considered over the time one of the reasons of the great success of the Internet. However, the idea that Web resources could be better organized is also quite old. Internet chaotic approach was heavily criticized as the Web reached its status of information and service provider.

How to discover resources, how to perform requests, if no agreement is made on how contents are placed?

A first architectural answer to these questions was given in the research that introduced the REpresentational State Transfer (REST) pattern [57] conducted in 2000 by Roy Fielding. The suggested approach provides a set of rules on how to organize systems whose information and services are web-based. Nowadays this is *de facto* a standard, and contributed heavily to the development of actual standards like HTTP 1.1 [58] and URI [59]. Moreover, the IoT and the WoT were also influenced, and several architectures were proposed in litterature [60, 61, 12].

Nevertheless, RESTful architecture and principles (and its constrained version, called CoRE[2]) do not provide a logical taxonomy to describe the reciprocal relationship between resources, excepted the request of a tree setup. Such logical meta-information is demanded to the standards introduced in the Semantic Web [2, 11], namely OWL[3], RDF[4], RDFS[5] and SPARQL Language[6]. Briefly, this can be summarized as it is explained in Fig. 1.2: the Semantic Web allows to organize resources, given as URIs, blank nodes, literals, in the form of triples *subject-predicate-object*. The subject can be a URI or a blank node; the predicate must be a URI; and the object can be a URI or a blank node or a literal.

This network of triples, that can be extremely complex, results in a resource graph, which is also known as Knowledge Graph or Knowledge Base (KB). To make an example of Knowledge Graph it is worth citing DBpedia[7], which is *a crowd-sourced community effort to extract structured content from the information created in various Wikimedia projects*, and is therefore open to all. It can be explored by using the SPARQL language, which we will largely use in the remaining of the Thesis.

[2] https://datatracker.ietf.org/wg/core/about/
[3] https://www.w3.org/TR/owl2-overview/
[4] https://www.w3.org/TR/rdf11-mt/
[5] https://www.w3.org/TR/rdf-schema/
[6] https://www.w3.org/TR/sparql11-overview/
[7] https://wiki.dbpedia.org

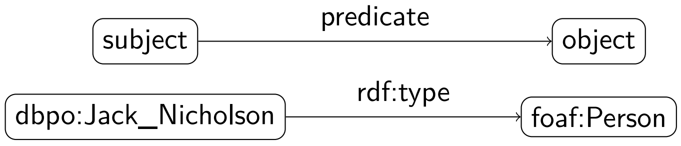


Figure 1.2 – An example of semantic triple storing the information that the resource URI related to Jack Nicholson makes reference to an entity of type Person, as it is defined in [`foaf`] ontology.

To get an idea of how powerful is the setup provided by DBpedia, let us consider the following request:

Design a software that, by querying the Internet, will list an "actor genealogy".

Starting from two actors, an "actor genealogy" is a sequence of triples `(actor1, actor2, film)` that allows in the minimum number n of steps to connect the two actors through their films. For instance, trying to connect Jack Nicholson to Matt Damon should give, as possible result, a single triple connecting them through the well known 2006 film *The Departed*. Instead, trying to connect Jack Nicholson and Julia Roberts results in two triples with George Clooney respectively in *Batman_1989_series* and *Ocean's Eleven*.

To address a problem like this, it is possible to avoid using DBpedia. Various services, on the web, provide APIs to list films, actors and their respective awards. However, here we will show that the semantic approach is definitely less expensive and more simple. For a non semantic approach, since we are interested in the shortest path from one actor to the other, a breadth first search is probably the easiest solution. Therefore, we expect to perform a sequence of requests to the web service, and list the films in which an actor starred, the other actors that were involved, iterating until there is a match for every actor in the sequence.

Instead, a semantic approach would simply search for patterns increasing in depth by 1 at each loop, as shown in the pattern in Listing 1.1. Consequently, by using the semantic KB contained in DBpedia we will perform exactly n requests for a n-step match, and we will not have to implement a breadth first search over a tree result.

Listing 1.1 – SPARQL pattern example to solve the actor genealogy problem. Consider Table 4.4 for expanded prefixes.

```
# Direct match
SELECT ?film1 WHERE {
  ?film1 dbpo:starring <actor_in>, <actor_out>.
}

# One step match
SELECT ?film1 ?actor_1 ?film2 WHERE {
  ?film1 dbpo:starring <actor_in>, ?actor_1.
  ?film2 dbpo:starring ?actor_1, <actor_out>.
}
```

This example gives just a simple overview of how the Semantic Web can help. In general, however, we have that the graph in the KB is formatted according to specific ontologies, that represent a pattern for triple organization [62]. In the *List of Ontologies* Section it is possible to have a quick look on the ontologies that will be used in this Thesis.

The ontology pattern is often considered as a bottleneck for the usage of Semantic Web. Ontologies, in many cases, are quite difficult to be understood and used: this creates a steep learning curve of the concepts and relationships expressed that, especially in a learning environment, can hinder the realization of applications. To address these obstacles, in the next Section some visualization techniques for the Semantic Web will be studied.

1.2.1 Visualizing the Semantic Web

This survey Section is based on a research work made at ARCES, aiming to create a better teaching-learning support for the students of the course *Interoperability of Embedded Systems* held at the School of Engineering, University of Bologna. The following paragraphs are therefore inspired from the works[8] that were issued as a result of the research.

Accessing and understanding the content of a database is hardly ever a negligible task for programmers. When data is stored in a relational database, the views are obtained by transforming into a table the output of a query written in one of the various flavors of the SQL language. Smart-written queries on equally smart-built databases can efficiently perform a lot of calculations over data, as well as outline special and complex relationships even between apparently distant entries. Therefore the *know-your-data* principle, typical of Data Mining and Big Data theory, is in fact a more general and solid base from which to start any data-related implementation, though implying sometimes great study effort from the developer in the initial phase of software creation [63]. It is, as a matter of fact, common knowledge that frequently programmers have to spend more time in organizing and reformatting their data more than in the actual programming logic.

The appearance of the Semantic Web in the panorama of information technology gave ways more than a simple new tool to explore the Web, but a new interpretation of the resources available on the Internet. Through the SPARQL language and the Resource Description Framework, the Internet network is considered as a whole a special database whose resources are interconnected in a labeled directed graph. The main idea of the Semantic Web, therefore, is to exploit the concepts of URI to bind resources through triple-based statements (i.e., the already mentioned subject-predicate-object triple). Any connection is in that fashion not a simple reference as hypertext linking is, but contains in addition the information given by the inner content of the resources. Then, according to W3C recommendations (see footnotes of Section 1.2), the Semantic Web in the end

[8] © 2018 IEEE. Reprinted, with permission, from Antoniazzi, F., & Viola, F. (2018, November). RDF Graph Visualization Tools: a Survey. In *2018 23rd Conference of Open Innovations Association (FRUCT)* (pp. 25-36)

takes the form of a graph, where both the contents and the statements contribute to the overall meaning.

A few considerations are needed, however, when we start discussing about the possibility to store information in a semantic graph. In fact, some critical points are present, and have to be highlighted. First of all, as it is depicted in [64], without regulations, the semantic graph is doomed to chaos, i.e. to an unpredictable information taxonomy. The solution to this issue is the ontological description of knowledge, that consists in the formal definition of all the classes, relationships and statements that can be present in the graph. Once the programmers agree on the ontology, there is no uncertainty on how the data is organized. All the information needed to query the graph is stored in an OWL file, standing for Web Ontology Language. OWL is, according to W3C, a *semantic markup language for publishing and sharing ontologies*, and has been widely used to define all sort of ontologies and vocabularies (which are smaller ontologies): an interesting repository, in this field of study, is the Linked Open Vocabularies website[9] and, for the next Sections of this Thesis, the List of Ontologies.

A second critical point is connected to the dimension of the graph, which can be considerable not only when we are discussing about web-located knowledge bases like DBpedia [65], or the Internet itself, but also in smaller applications exploiting RDF and SPARQL. To make an example, let us perform the query of Listing 1.2 to DBpedia.

Listing 1.2 – SPARQL query to all classes parent of the resource `dbpedia:The_Lord_of_the_Rings`

```
SELECT (count(?o) as ?count)
WHERE {
  dbpedia:The_Lord_of_the_Rings
        rdf:type ?o
}
```

Query execution:

Last visited: 15/10/2019

Listing 1.3 – SPARQL query to every link (except `rdf:type`) with `dpbedia:The_Lord_of_the_Rings` as origin, and the destination's class

```
SELECT ?p (count(?t) as ?count)
WHERE {
  dbpedia:The_Lord_of_the_Rings ?p ?o.
  ?o rdf:type ?t
  FILTER (?p != rdf:type)
}
```

Query execution:

Last visited: 15/10/2019

The output of the query, which is a simple request to count all the classes that "The Lord of the Rings" resource belongs to, is equal to 25. Clearly, far from a naive and optimistic expectation of a few outputs similar to `:Book`, `:Novel` and so on. Listing the actual values consequently results in a 25-rows table that outlines the evidence of the hidden complexity in the results analysis, even in simple situations. The complexity

[9] `https://lov.linkeddata.es/dataset/lov`

grows considerably if we proceed querying on the following level (Listing 1.3).

The query available in Listing 1.3 with the variable `?count` outputs the number of links of type `?p` outgoing from the resource `dbpedia:The_Lord_of_the_Rings`, except from `rdf:type` links, that can be viewed by performing the query in Listing 1.2.

The direct outcome after running the queries in the Listings 1.2 and 1.3 is given by the possibility to observe the available variability of results and to look for their meanings. The output of those simple SPARQL queries highlights, for instance, that the "Lord of the Rings" resource is individual of at least 25 classes which we expect to have a specific meaning and a description of their own. Going further with the latter query, moreover, the number of elements connected to the resource is even more increasing, both as connectivity spread, and in diversity of ontological classes involved. In general the description of all those resources can be as pragmatic as an algorithm, or philosophical, or mathematical. However, it is clear that without the Semantic Web it would be hardly achievable to obtain such a multi-layered description of a resource, apart from using natural language. In fact, when it comes to exploit the tools of Semantic Web, a frequent feeling is that it is not possible to reuse previously available data, because it would imply to understand completely all the resources, all the classes, and all the ontologies that are standing behind. According to [66], expressiveness, in this situation, is a bottleneck for Semantic Web.

This is where visualization tools for the semantic graph come to help: they provide a step by step approach to the knowledge base that, together with filtering techniques, and the possibility to see the contents, are useful to go through the relevant concepts.

Possible visualizations

As we said previously, the most frequent way to produce a view of a database is the tabular representation. This is a possible solution also for queries made in SPARQL language to RDF triple stores like Blazegraph, Fuseki and Virtuoso. The view of a SPARQL `SELECT` is a direct consequence of the number of variables concerned by the inquire: i.e., the number of variables in the `SELECT` clause is the same as the number of columns contained in the results. To be more precise, to obtain the column number either (i) it is necessary to count the variables queued after the `SELECT` keyword, like in listings 1.2 and 1.3; or (ii), variables have to be obtained from the `WHERE` clause, as in the `SELECT * WHERE {...}` case.

The drawbacks with table views are, unfortunately, already quite visible when the number of rows reaches as little as few dozen entries. Aside from the fact that there is not a group view of the overall query result, the table often is required to contain more than one row for the same conceptual entity. This happens for instance when a resource is connected to another via more than one predicate, or when it is connected to different objects, through the same predicate.

In such situations, the result table can contain not only plenty of lines with the same meaning disturbing the overall understanding of the query output, but also, as already said, a high number of columns. Moreover, some entries in the table can also be empty, as an effect of `OPTIONAL` statements in the query. Detecting particular cases, in sparse and large tables, becomes a time-consuming and error-prone task in those situations. On the other hand, a few workarounds are available in `SPARQL` language to crunch into a single line the occurrence of multiple table lines for a single concept, but they usually imply slowing down the performances, and have the effect to concatenate the values into strings. That is, we lose the possibility to check if they are represented as IRIs, literals, or blank nodes.

The multi-table approach is a graph visualization technique that tries to address the problem of having limited control over the complete data table. Let's consider a query selecting all triples in the RDF store: `SELECT * WHERE {?a ?b ?c}`, and let's suppose that in the store only 5 distinct resources might correspond to the `?b` variable. With this background a full-table approach would return an $n = 3$ column table, where n is the number of variables. Instead a multi-table approach would outcome with 5 smaller tables, one for each one of the `?b` resources, each of them built up of $n - 1 = 2$ columns. An interesting work about the complexity of translation from SPARQL to other languages, included table view, was provided by Chebotko et al. in [67]: among all the contributions, this paper perfectly shows the complexity of a multi-table approach.

Finally, last but not least, the RDF knowledge base representation can be performed through a labeled graph visualization. Although the RDF concept is defined for directed graphs, in most of the cases the label is sufficient to get at view time the direction of the connection. This allows the usage of algorithms for undirected graphs. Nevertheless, the drawbacks of this approach are also related to the knowledge base dimension, as the understanding of contents is tightly bound to the possibility of identify paths and node types easily and effectively.

Graph drawing algorithms

There is a complex relationship between the domain of the semantic application, the tool that is being used, and the algorithm that is implemented to visualize the graph. To make an example, let's consider a knowledge base in which information about some people is stored. If the application working on the knowledge base is not interested in literal terms, the sight of the graph would be effectively simplified and clearified by just removing all the links towards literal terms, e.g. names, surnames and birth dates.

In other RDF triple stores more than one unique ontology may have been used to define resources, exploiting for instance simultaneously the [**foaf**] ontology and the Dublin-Core ontology [**dc**]. If an application is interested only in the [**foaf**]-related connections, and in a small part of the DC's, there would be no use in trying to represent everything.

A full description of all the algorithms available for graph drawing is out of the scope of this Thesis. In this Section, nevertheless, a brief overview of a few works available in literature is given, before proceeding in the next paragraph to the analysis of the tools.

A complete theoretical overview of the main algorithm logic available to draw graphs is given by Kobourov in [68]. Spring algorithms and their variations for instance are explained: they usually aim to reproduce an aesthetically pleasant view, even if their best performance is obtained in most of the cases when the graph has less than 40 vertices. However, as it has been said, the semantic graph is definitely a large graph, or very large, and for this reason it demands particular approaches that imply multiple scale algorithms. Nodes organization is not necessarily done on a plane: possible alternatives are to dispose them on a sphere or other geometrical objects. In [69] more than one plane is used, which can be a technique also to represent the evolution of data over time. How to show in an effective way dynamic evolution of contents in a graph is also the topic of survey [70] by Beck et al.

Listing 1.4 – SPARQL `CONSTRUCT` that identifies in DBpedia the Fantasy-genre books written between 1900 and 2018 having more than 200 pages.

```
CONSTRUCT {
  ?book rdf:type dbpo:Book;
    dbpo:literaryGenre :Fantasy_novel;
    dbpo:author ?author;
    dbpo:releaseDate ?date;
    rdfs:comment ?comment;
    rdfs:label ?label;
    dbpo:numberOfPages ?num;
    foaf:isPrimaryTopicOf ?topic.
  ?author rdf:type foaf:Person
}
WHERE {
  ?book rdf:type dbpo:Book;
    dbpo:literaryGenre :Fantasy_novel;
    dbpo:author ?author.
  ?author rdf:type foaf:Person.
  ?book rdfs:comment ?comment;
    rdfs:label ?label;
    dbpo:numberOfPages ?num;
    foaf:isPrimaryTopicOf ?topic.
  FILTER langMatches(lang(?label), "EN")
  FILTER langMatches(lang(?comment), "EN")
  FILTER (isIRI(?author))
  FILTER (xsd:integer(?num) > 200)
  OPTIONAL {
    ?book dbpo:releaseDate ?date .
    FILTER (?date > 1900)
    FILTER (?date < 2018)
    FILTER (isLiteral(?date))
    FILTER (datatype(?date) = xsd:integer)
  }
}
```

Query execution:

Last visited: 15/10/2019

Graph visualization tools

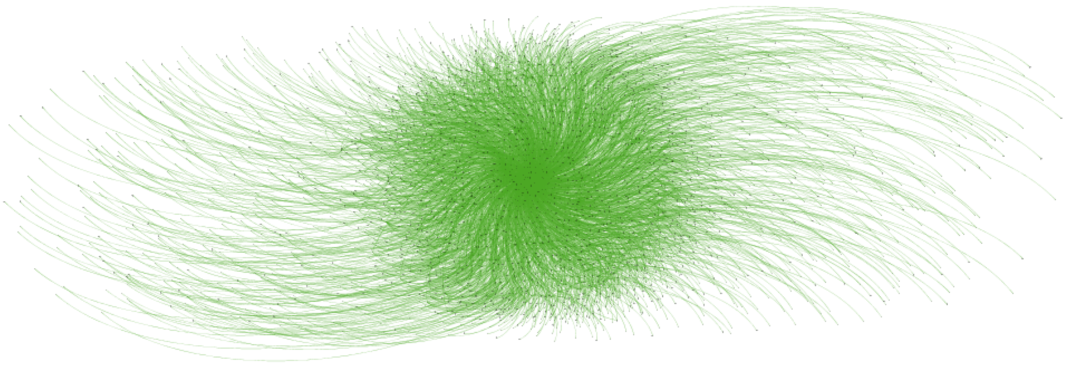
Hereby a detailed analysis of the main tools for the visualization of RDF knowledge bases and ontologies is proposed. We focus on the tools providing a graph visualization of RDF statements. The tools presented in this Section are reported in alphabetical order.

CytoScape is a tool for network data integration, analysis and visualization. Support to Semantic Web technologies is provided by a set of extensions hosted on CytoScape's App Store, such as General SPARQL, SemScape and Vital AI Graph Visualization [71]. General SPARQL allows to navigate semantic web KBs through an extensible set of predefined queries. The plugin is pre-configured to retrieve and visualize data from public endpoints (e.g., Reactome, Uniprot, HGNC, NCBI Taxonomy, Chembl). SemScape supports the interaction with remote SPARQL endpoints by means of SPARQL queries. In this way, CytoScape can be employed to visualize the results of a query. Vital AI Graph Visualization is not limited to semantic databases, but provides access also to SQL and NoSQL databases as well as Apache Hadoop instances. To the best of authors' knowledge, this tool only allows the visualization of data compatible with the BioPAX format.

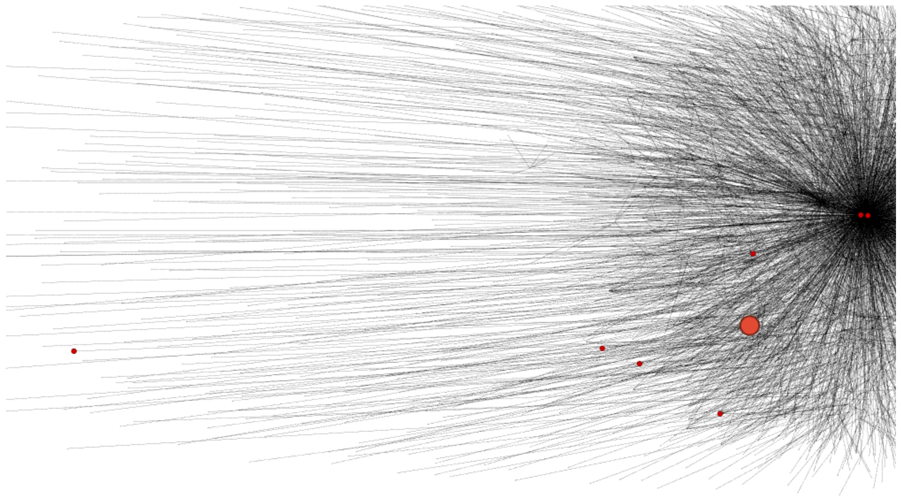
Fenfire was a tool for the visualization and editing of RDF graphs aimed at an interactive exploration of the graph [72]. Authors face the problem of scalability by limiting the exploration of the graph to one thing at a time. The visualization in facts, diplays only one central node and its surroundings. The central node, at the beginning of the exploration is selected exploiting the `foaf:primaryTopic` property (if present), otherwise is selected by the user. The nodes surrounding the central one (named *focus*) are placed on the plane according to a simple strategy: on the left, all the nodes being subjects of the statements linking to the focus. On the right, those being objects of the statements. Development of Fenfire stopped in 2008.

Gephi is a very powerful tool designed to represent not only semantic graphs, but every kind of graph or network [73]. Support to RDF graphs is provided by two external plugins, VirtuosoImporter and SemanticWebImport (this one developed by INRIA). Gephi is able to retrieve data from SPARQL endpoints (through REST calls) as well as to load RDF files. Gephi supports filtering the KB through SPARQL queries. The look of the graph visualized by Gephi is fully customizable, in terms of colors and layouts; moreover the tool supports grouping similar nodes and this helps achieving better results when dealing with very complex graphs. As regard exporting the graph, Gephi is the tool that supports the highest number of file formats for exporting the graph. Among these, it is worth mentioning `csv`, `pdf` and `svg`.

In Figure 1.3a we can see a view of the graph that Gephi is able to retrieve from DBpedia by using the SPARQL `CONSTRUCT` available in Listing 1.4. The tool performs the representation very quickly, and implements various possible algorithms to build the graph. Unfortunately, as it can be seen, it is quite difficult to get the overall idea of the composition. Although there is the possibility to add the labels of nodes and



(a) The Figure is the output of Gephi's `CONSTRUCT` in Listing 1.4 to DBpedia. According to its logger, the triples represented in this graph are 6529.



(b) With Gephi some nodes can be highlighted, to help the user to go through the knowledge base. When the number of edges and nodes is high, however, it's not easy to outline the information. The nodes in red are related to L. Alexander's novel "The Black Cauldron".

Figure 1.3 – Gephi [73] output example.

edges, the output is not reader-friendly, and the research in it is a rather impossible task. A practical example can be observed also in Figure 1.3b, where we highlighted the nodes related to the novel "The Black Cauldron" by L. Alexander. Eventually, a number of statistical functions can be applied to the network, like the *Network Diameter*, the *Density* and the *Average Path Lenght*: the only problem is that they have, as for the Authors' knowledge, very limited use when applied to a Semantic Graph.

Glow is a visualization plugin for the ontology editor Protégé [74]. Force-directed, Node-link tree and Inverted radial tree are the three layout algorithms provided by

GLOW. The items are arranged automatically with every layout, and cannot be moved. The tool is able to represent a set of ontologies and optionally their individuals. To the best of authors' knowledge, this tool is not developed anymore. No information about the license could be found.

IsaViz is a 2.5D tool for the visualization of RDF graphs originally developed by E. Pietriga (INRIA) in collaboration with Xerox Research Centre Europe [75]. IsaViz, as the name suggests, is based on GraphViz [76] and allows importing and exporting from/to RDF/XML, Notation 3 and N-Triple files. The result of the visualization can be also exported as a `png` or `svg` file. In the *Graph view* it is possible to select resources and access a textual list of properties (this view is named *Property Browser*). A third view is named *Radar* and presents an overview of the graph, since the graph view may contain only a portion of it. Finally, it is worth mentioning the search tool provided by IsaViz, whose results are highlighted one by one in the graph view. Unfortunately, the last development version of this tool dates back to 2007.

Jambalaya is a Protégé plugin for the visualization of ontologies [77]. Jambalaya is characterized by the integration of the SHriMP (Simple Hierarchical Multi-Perspective) [78] visualization technique, designed to improve the user experience in browsing, exploring, modelling and interacting with complex information spaces. This technique, originally born to help programmers understanding software, was applied to Protégé to build a powerful visualization of classes and relationships. The tool proposes a nested graph view and the nested interchangeable views. Nesting is used to represent the subclass relationships among classes as well as the link between classes and their instances (different colors allow to distinguish between classes and instances). Jambalaya also provides an easy way to search for items in the ontology. Despite being an interesting tool developed with support from the National Center for Biomedical Ontology (NCBO), Jambalaya is not developed anymore.

LOD Live is a web-based tool for the incremental navigation of Linked Data available on a selected SPARQL Endpoint (e.g., DBpedia) [79]. Endpoints can be configured through a JSON map of their parameters, similarly to what happens in Tarsier [69]. The purpose of this tool is to demonstrate that the powerful Semantic Web standards are also easy to understand; the aim is to foster the spread of Big Data. Every resource drawn by LOD Live is surrounded by a set of symbols representing different kinds of relationship (e.g., direct relations, group of direct relations, inverse relations and group of inverse relations). The incremental navigation, joined to the ability of the tool to group properties allows to draw a very clean graph. No support for statistics or advanced filtering (e.g., based on SPARQL) is provided. To the best of our knowledge, directly exporting the graph is not possible. In Figure 1.4 it is shown how LOD Live performs a similar task as the one in Figure 1.3b: exploring data is easier, but there is no way to perform requests like the one in Listing 1.4.

Ontograf is one of the visualization tools provided by the famous ontology editor Protégé [80]. The tool allows to build a custom visualization of the ontologies loaded in

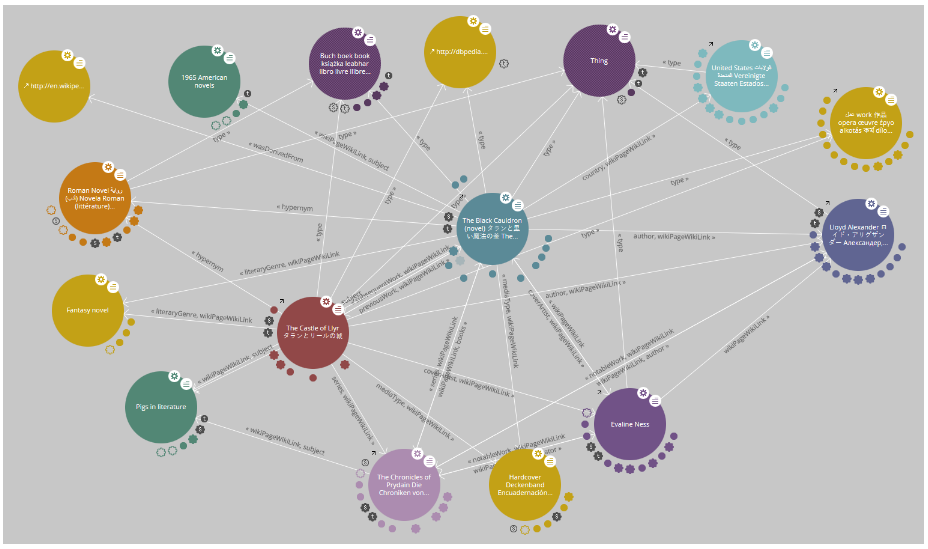


Figure 1.4 – To use LOD Live [79] a resource must be fixed. Then, the knowledge related to the resource can be expanded as shown. Like in Figure 1.3b, the example here is based also on L. Alexander's novel "The Black Cauldron".

Protégé by iteratively enabling or disabling the desired classes. Ontograf proposes a grid layout (with classes sorted in alphabetical order), a spring layout and a (vertical or horizontal) tree layout. Individuals of a class can be visualized in its tooltip, but this is uncomfortable when dealing with a high number of assertional statements. Ontograf allows to export the visualized graph as a png, jpeg, gif or dot file. This tool exploits the layout library provided by Jambalaya. Fig. 1.5 shows a graph created with OntoGraf using the DBpedia ontology. Classes `work` and `written work` were initially selected. Then, a double click on the latter allowed to expand it and visualize all the subclasses (solid blue line), and all the classes linked to it by means of an object property (dashed lines). The last version of Ontograf dates back to April 2010, but is still included in the last stable version of Protégé[10] (the 5.5.0, as of August 2019). The tool is useful to select and visualize (a small number of) classes from the ontologies loaded in Protégé and the existing relationships.

OntoSphere is one of the two tools (the other is Tarsier [69]) that proposes a three-dimensional visualization of the graph [81]. The rationale behind OntoSphere is that exploiting a 3D space it is possible to better arrange items. Moreover, the 3D visualization is quite natural for humans and the exploration can then be more intuitive. Colors allow to easily convey information about the different nature of represented items. OntoSphere is aimed at representing both terminological and assertional statements. Four scene types are proposed to fulfill different requirements. The `RootFocus` scene shows all the concepts and their relationships on a sphere. The `TreeFocus` scene draws the tree

[10] `https://protege.stanford.edu/`

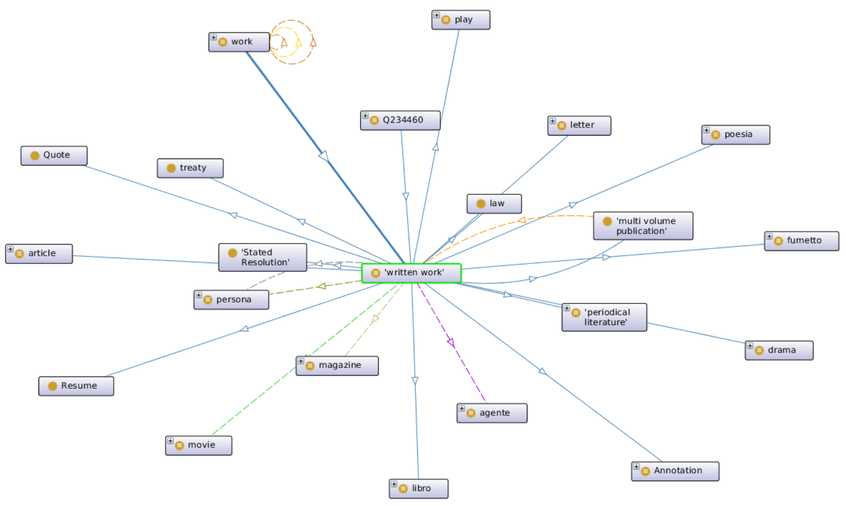


Figure 1.5 – A portion of the DBpedia ontology visualized in Ontograf [80].

originating from a concept, while the ConceptFocus scene proposes a view containing all the items linked to a concept. The tool is aimed at domain experts dealing with the development and review of ontologies, as well as novice users that wants to understand the represented data and the links among concepts. OntoSphere is a standalone applications, but can also be run inside Protégé and Eclipse. The last version on the source code repository is dated 2008, so the development stopped ten years ago.

OWLViz is a plugin for Protégé that enables the incremental visualization of the classes in the class hierarchy [82]. As the name suggests, this tool, like IsaViz, is based on the famous library GraphViz developed by the AT&T and allows exporting the visualized graph as png, jpeg and svg. Through OWLViz is easy to visualize classes and is-a relationships. OWLViz is not developed anymore, but is still included in the last version of Protégé (August 2019).

Paged Graph Visualization (PGV) is a Java software for the visualization of RDF graphs [83]. It is based on [84], a high performance RDF storage. With PGV, the exploration starts from a point of interest and then incrementally includes more data. Such point of interest can be selected interactively from a list or using a complex SPARQL query. Then, it is drawn in the center of the graph using the color green, and its direct neighbors are shown as blue rectangles placed around it. Literals on the other hand, are represented with the white color. The user is able to explore nodes by double-clicking on them: explored nodes are then displayed in green, while edges connecting explored nodes are depicted in red. Deligiannidis et al. [83] declare that the tool's strength relies in helping the user willing to explore data without knowing the exact information and graph patterns he is looking for, while in other situation a standard visualizer could be more appropriate. This tool seems to be not developed anymore.

RelFinder is a web tool developed using Adobe Flex and can be tried using the web

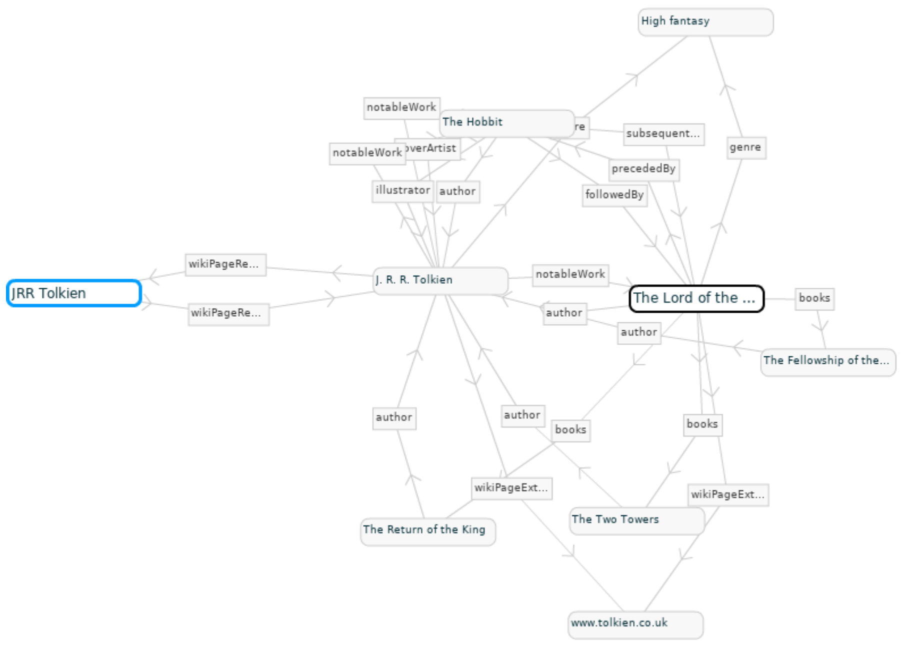
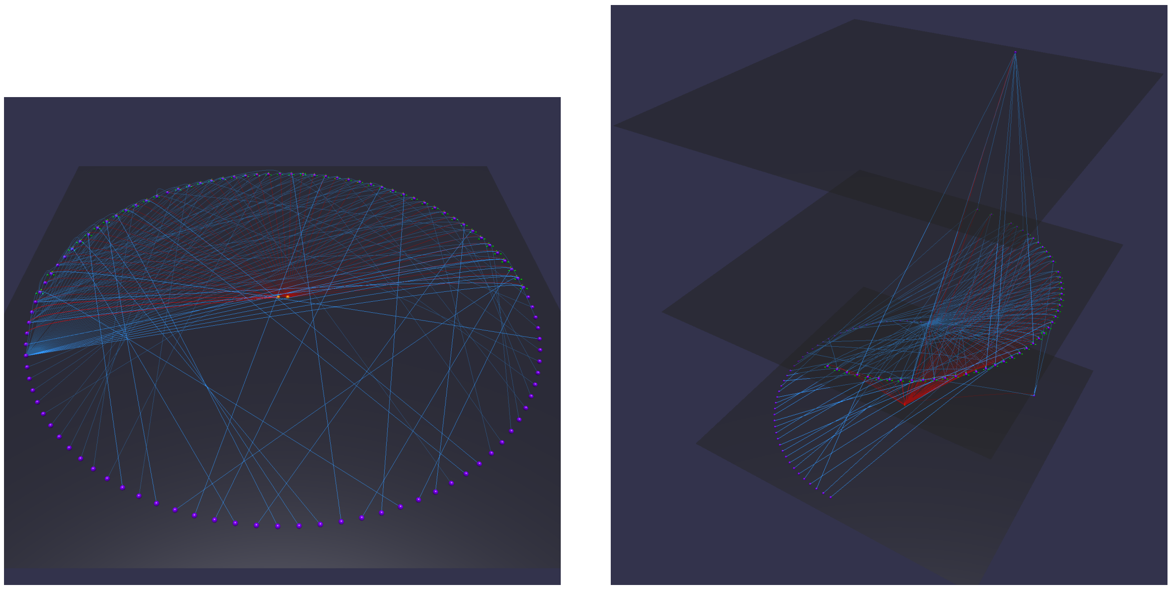


Figure 1.6 – RelFinder view of the paths from "JRR Tolkien" to "The Lord of the Rings".

instance linked in the homepage of the project (configured to access DBpedia) [85]. RelFinder differs from the other tools proposed in this survey, since it is aimed at visualizing all the paths connecting two resources. So, its purpose is to answer a very specific question, rather than providing a tool for the free exploration of the knowledge base. The tool supports filtering to increase or reduce the number of relationships shown simultaneously. It also implements a smart drawing algorithm to reduce overlapping and the user is allowed to move and pin items. To the best of authors' knowledge, this tool is not actively developed but the online instance is still available for tests on the DBpedia endpoint. Fig. 1.6 reports an example of this application where all the paths between two DBpedia resources, i.e, "JRR Tolkien" and "The Lord of the Rings", are shown.

Tarsier is a tool developed by ARCES research group. It is a software for the interactive exploration of an RDF graph in a three-dimensional space, aimed at the visualization of small and medium-sized knowledge bases. The main contribution of the tool is the introduction of the metaphor of semantic planes that group RDF terms sharing a common concept. The purpose of the tool is threefold: 1) Tarsier can be used as a support for didactic (e.g., to help newcomers to deal with Semantic Web technologies); 2) It is useful to figure out the nature of a new KB for developers (i.e., activity known as "sensemaking" [86]) ; 3) It allows debugging of semantic knowledge bases.

Tarsier retrieves data from SPARQL endpoints. The initial knowledge base can be determined through a SPARQL Contruct query: this pre-filtering stage allows to efficiently interact also with very large knowledge bases (e.g., DBpedia, that contains more than 6.6M entities). Tarsier proposes a classification of all the RDF terms among


(a) Tarsier showing the graph of all the fantasy books published from 1900 to 2018 and their authors. This subgraph is retrieved from DBpedia.

(b) Tarsier showing two semantic planes over the main knowledge base: one showing books, the other (the topmost) showing the author Marion Zimmer Bradley.

Figure 1.7 – Tarsier [69] user interface.

classes, resources, blank nodes, literals, object and datatype properties. This grouping is exploited by Tarsier's web interface to provide a set of controls for advanced filtering: through them, the user is allowed to toggle visibility of items or to move them across semantic planes.

An example of Tarsier is shown in Figg. 1.7a and 1.7b. Tarsier was set up to retrieve data from DBpedia, and in particular to extract all the fantasy books published between 1900 and 2018 and their authors. While Fig. 1.7a shows the unfiltered knowledge base, in Fig. 1.7b is shown one of the peculiarities of Tarsier: the semantic planes. Two semantic planes were created over the main knowledge base to extract respectively books and one of the authors, i.e., Marion Zimmer Bradley. In this way, it is easy to notice how this instance of the class `foaf:Person` is linked with the graph.

TGVizTab is yet another visualization plugin for the ontology editor Protégé [87]. It is designed to be lightweight and support both T-Boxes and A-Boxes visualization, and it relies on TouchGraph, an open source Java environment aimed at creating and navigating network graphs in an interactive way. The tool supports exporting the graph in an XML file, to be loaded in other TouchGraph applications. The graph is drawn using the spring layout: similar nodes are drawn close to each other. TGVizTab, like other tools (e.g., Fenfire), asks the user to select a focal node among classes and instances to generate the graph. Then, the user is able to further modify the graph by right-clicking on the represented nodes: in this way the so-called *Node Menu* is shown, containing four options (i.e., expand, collapse, hide, view). Then TGVizTab allows to incrementally build the desired visualization.

VOWL (Visual OWL) is available as a web-based tool (WebVOWL [88, 89]), a plugin for Protégé (ProtégéVOWL [90]), a tool able to directly interact with Linked Data endpoints (LD-VOWL [91]), and as a visual query language tool (QueryVOWL [92]). Here we will refer to the web based version, WebVOWL. As the name suggests, software in the VOWL toolkit are designed to graphically represent ontologies. They propose a force-directed graph layout. The basic representation rules adpoted by VOWL consists in:

- Classes are depicted using circles where the color depends on the type: light blue for OWL classes, purple for RDFS classes, dark blue for those imported by other ontologies, gray for deprecated classes.
- OWL object and datatype properties are represented with black solid lines with, respectively, light blue and green labels, while RDFS properties have purple labels.
- Relationships `subClassOf` are depicted with a dashed line.

The graph drawn by VOWL can be exported as an `svg` image or as a `json` file. A click on a node or edge allows visualizing the associated metadata and statistics. Statistics also report the number of individuals of the selected class, but unfortunately this is the only information about individual that is possible to obtain using VOWL. As regards filtering, VOWL provides a basic support to filters that allows to show/hide object/datatype properties, solitary classes, class disjointness and set operators.

VOWL is actively developed and an online instance is available. As the tool is designed for ontologies, importing the output of the `CONSTRUCT` in Listing 1.4 results in representing only the two `rdf:type` relationships. The other tools are still being developed and at the moment do not allow to perform a customized request to DBpedia.

Overall considerations on graph visualization

Table 1.1 summarizes the main features of the analyzed software. Columns of the table are:

- **Software** – reports the name of the software;
- **T-Boxes** – this column tells if the tool supports the visualization of terminological statements;
- **A-Boxes** – this column shows if the tool supports the visualization of assertional statements (and can then be used to explore a knowledge base, rather than just ontologies);
- **Statistics** – a boolean field showing if the tool provides or not statistics on the visualized data;

- **Filtering** – filtering allows to show/hide elements in the visualization according to a set of user-defined criteria. Filtering can be implemented in very different ways (e.g., SPARQL queries, or UI controls to select classes, just to name a few). This column indicates whether the related tool provides at least one filtering mechanism.
- **Editing** – This Thesis surveys visualization tools for semantic data, but some of them also offer editing functionalities. This column states whether or not the related tool supports the manipulation of the ontology/knowledge base;
- **Standalone** – Many of the surveyed tools were born as plugins for the ontology editor Protégé. Other can be run as standalone software. This column tells if the related software is embedded in other tools or is a standalone application.
- **Plugin** – Not all the presented tools were born to visualize semantic knowledge bases. Then, some of them need additional plugins to achieve this task.
- **Domain** – This column contains the specific domain (if any) where the related application can be applied.
- **Reference** – This column reports the reference number of the paper(s) describing the tool.

Plus, additionally, other information about the entities that started the development of the tool, the license and the current status of the project.

1.2.2 AudioCommons Project

We will show in this Section an example of usage of the Semantic Web to foster information interoperability in an European Project called AudioCommons[11] (AC).

According to the project's website, its goal is to *bring Creative Commons audio content to the creative industries*. Therefore we will hereby discuss the development of a tool (the AC Mediator) that promotes a synergy between web technologies and musical audio production and sharing services (i.e., the industry). While here such approach will support us in providing a common platform-methodology for audio sharing, enjoying and researching, in Section 2.2 the Semantic Web will be our starting point to implement from scratch a new concept of IoT connected to music.

Let us consider the following four online tools:

Jamendo[12]	Freesound[14]
Europeana[13]	Internet Archive[15]

[11]The author was involved in the project in the context of a collaboration with the Centre for Digital Music at the Queen Mary University of London. Se also the Introduction for further references.

[12] `https://www.jamendo.com/`

[13] `https://www.europeana.eu/portal/en`

[14] `https://freesound.org/`

[15] `https://archive.org/index.php`

With reference to AudioCommons, they are all interesting sources of audio (and a lot more!) material mostly stored with Creative Commons licenses[16]. The Internet Archive, for instance, *is a non-profit library of millions of free books, movies, software, music, websites, and more.* Freesound, instead, targets only audio media into a collaborative environment. Jamendo aims to *bring together a worldwide community of independent music, creating experience and value around it.* Lastly, Europeana *provides access to over 50 million digitized items - books, music, artworks and more.*

By interacting with those services, consequently, end users are not only authorized, but also encouraged to search for contents and use them into their local projects. The Creative Commons license agreements regulate the rights of the users and the authors [93].

How to explore the databases? First of all, contents can be accessed in a very simple way from a regular web browser. Queries can be issued and their results can be examined and downloaded in a user-friendly interface requiring a manual approach. APIs are also available to perform this task.

While this is indeed a good result, it actually appears that a complete search implies a manual exploration of the query outputs coming from the four services separately. According to [93], this required manual approach is one of the issues hindering the spread of freely accessible medias. In particular, it is possible to outline two main bottlenecks: (i) *incomplete and wrong metadata*, as there is often a lack of proper annotation within medias; (ii) *incoherence among content providers data representation*, e.g., it is not possible to access Jamendo's database in the same way in which we access the Internet Archive. The former is due, for instance, to the author's bad description of his own work (e.g., by choosing the wrong musical genre, by adding in the title unnecessary information, ...) [94]. The latter, instead, refers to the differences into specific implementation choices for every content provider, which are included into the APIs, authentication policies and output formats. As we said in Section 1.1, typically we have that query results may refer to the same type of content, but still are incompatible because they are presented in radically different ways: namely, the file format or tagging vocabulary inconsistencies.

One of the main achievements of AudioCommons is the work on this specific vocabulary matching topic. The main question, therefore, would be: how to grant through a single query the access to the four services from a unique common endpoint, and still be able to compare results coherently? All this, clearly, looking to the future and creating a setup easily extendable with new features.

The first task to be addressed was the common interpretation of the results given by the content providers. This is a typical situation in which having a Semantic description of the information exchanged would have provided a direct solution. In this case, therefore, there is the need of aligning the vocabularies used by the four providers into a unique one which would then represent our real access point. This is the reason behind the AudioCommons Ontology [`aco`], developed by Ceriani et al. [95].

A complete and full description of the AC ontology is out of the scope of this Thesis. We focus, however, on the methodology to exploit it. Given that we have a shared

[16] `https://creativecommons.org/`

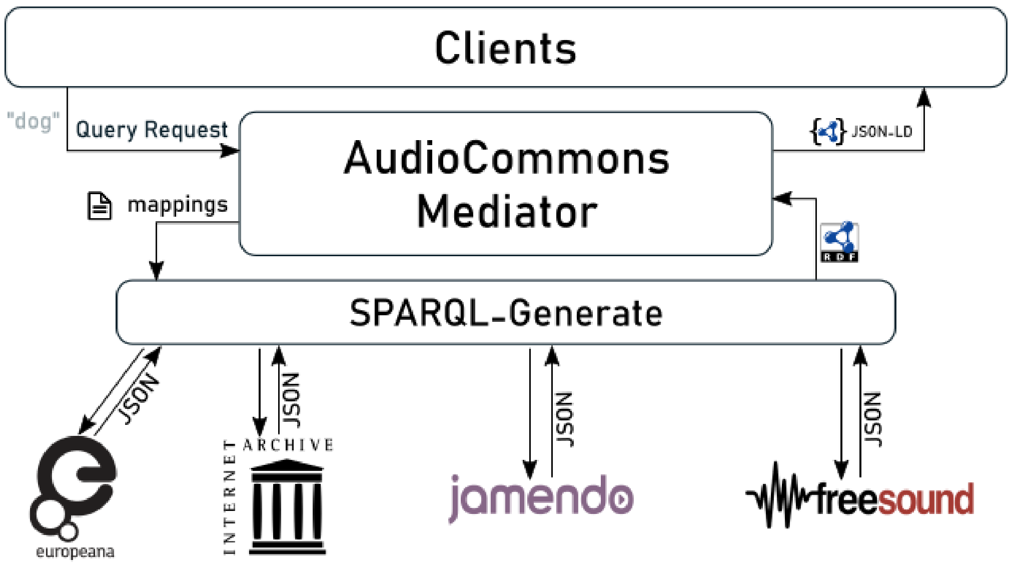


Figure 1.8 – Schema of the AudioCommons Mediator working logic.

view on how to represent information following the aforementioned AC ontology, we need to transform the information coming from the media providers in a format that is compliant to it. To do so, consider Fig. 1.8: the proposed schema outlines the actual working procedure of the core tool, the AC Mediator. Let us consider a client querying for audio contents related to *dogs*: as we said, it would be possible to perform manually a query to each of the contents providers, and try to obtain comparable results through complex parsing algorithms.

Fig. 1.8, instead, takes a different direction: the *dog* request is given to a service called `SPARQL-Generate` [96, 97] together with a set of *mappings*. Each mapping is specific for a content provider, and carries the needed information to correctly contact the provider and to interpret and parse the expected output.

In our setup, available on Github[17], the mapping has to translate the outputs from Europeana, the Internet Archive, Freesound and Jamendo, which are given in JSON format, into a shared semantic RDF graph. This is the *generate* phase. This procedure, of course, implies that

1. The mapping contains all the needed information to correctly contact the content provider;

2. The `SPARQL-Generate` request is made having prior knowledge of the AC Ontology, essential to correctly format the RDF graph;

3. The `SPARQL-Generate` request is made having prior knowledge of the schema used by the media provider to build its JSON output;

[17] `https://github.com/AudioCommons/semanticMediator`

4. Adding a new provider to the mediator means simply adding a new mapping, resulting in a setup that is extremely flexible and extensible.

The RDF graph eventually obtained from the `SPARQL-Generate` after querying the four providers is the actual result of the process. It can be stored in a knowledge base, like Blazegraph or Virtuoso, for further usage. Alternatively, it can be simply returned back to the original client.

A further development of this unification procedure is connected to particular queries that need a long time to be performed, like the one suggested by Xambó et al. [98]. In the next Section 1.3 we will introduce an additional tool that is one of the core instruments of this Thesis, i.e. the SEPA. By using the SEPA, as it will be shown, it will be possible to address the long-query problem with a fully semantic setup.

1.3 SPARQL Event Processing Architecture

This Section will introduce one of the core tools that allowed the realization of the work outlined in this Thesis, i.e., the SPARQL Event Processing Architecture (SEPA). As a complete discussion on the internal specific mechanism of SEPA is out of the scope of this report, we will make some references to the published research [99], and present a higher level explanation.

Some light examples will also be shown, although the real test bench application can be found in Section 3.2, where the SEPA will represent the core engine supporting of our vision of Semantic Web of Things agents.

1.3.1 Origins

The SPARQL Event Processing Architecture is the last step of a long path made of ideas and projects in which ARCES research center has been involved since approximately 2010. At the time the idea was expressed in the Smart-M3 platform [100], which had the intent of implementing a middleware architecture to support information sharing among devices and software. The challenge of data interoperability was addressed through a semantic approach: applications would agree on an ontological description of data, and use a shared semantic endpoint to store their information.

An additional requirement, in this middleware setup, is dynamicity. This is due to the fact that dealing with devices and software is not a static task, but rather is something that is subject to continuous change. As a matter of fact, while the Smart-M3 architecture started to be applied also to IoT, this aspect became even more relevant.

Addressing dynamic data exchange, in this context, created over the time the concept of Publish-Subscribe: according to [101], a few types of Publish-Subscribe can be outlined. Well known and frequently used protocols, like AMQP and MQTT belong to the *topic based* methodology for which, in a broad view, the *topic* is a tag that identifies a communication channel. A *content based* approach, instead, will simultaneously define a tag for the channel and request some further conditions on the data exchanged (e.g., if the channel is tagged `Temperature`, we call for `Temperature>25` degrees). The *type* *based*, eventually, tag the channel with the format of the data that is passing.

The Smart-M3 architecture, in such panorama, is positioned as we said including part of each of the three aforementioned ideas plus a semantic approach. The middleware was implemented in various flavors, keeping nevertheless the reference to its function as a *Semantic Information Broker*. It is possible to cite some of the implementations, namely RedSib [102], OSGi SIB [103], CuteSIB [104], PySIB [105]: they were largely exploited and evaluated [106, 107], implement approximately the same paradigm and the same (non standard) protocols to perform semantic data insertion/deletion and subscription.

SEPA starts from that point on. With the appearance of the concepts of Web of Things and Linked Data, the semantic middleware needed to be updated in order to be compliant to web standards. In the next Section this complete refactoring is explained.

1.3.2 Architecture

SEPA's working logic is rather simple. It is a semantic middleware for interoperability that has to

1. Store and retrieve semantic data upon request;
2. Add and delete semantic data;
3. Control the evolution of data dynamically and notify changes;

The clients, in such context, will require their data to be safely kept: to do so, SEPA relies on external tools to store semantic information, like Blazegraph[18], Fuseki[19] or Virtuoso[20]. These are RDF stores, that can be queried and updated by the means of SPARQL 1.1 Query and Update Language.

The event monitoring service is added by our architecture on top of the semantic storage. To better understand how such subscription engine works, consider Fig. 1.9. In the figure we outline the various possible behaviors that we can expect from a client (i.e., the line on top of the schema). The box at the bottom, instead, represents the SPARQL endpoint, or equivalently an RDF store instance.

Queries, as it is shown, pass almost in a transparent way through SEPA, with the only constraint that query and update requests cannot be executed out of order. Updates, on the other hand, are conceptually bound to the subscription engine, as their side effect is to generate notifications. They are both executed through HTTP POST.

Once an update is received, the actual triples to be added, and the ones to be removed, are matched to the list of subscriptions that are active. If the subscription SPARQL pattern corresponds to some of the triples in either way, then the notification is triggered, through Websocket. For a complete and precise description of every internal mechanism, please refer to [99].

Let us make a simple SEPA usage example, by calling back the *actor genealogy*

[18] `https://www.blazegraph.com/`
[19] `https://jena.apache.org/documentation/fuseki2/`
[20] `https://virtuoso.openlinksw.com/`

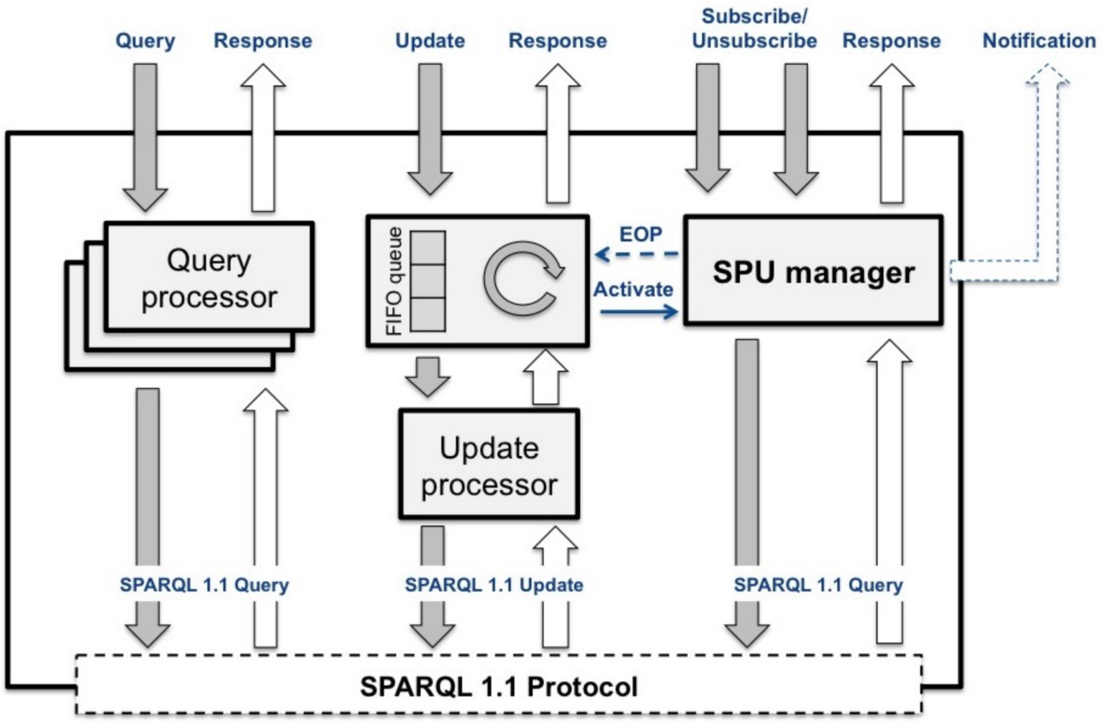


Figure 1.9 – SEPA internal setup, from [99].

introduced in Section 1.2. Should we implement the algorithm proposed (available in Github[21]), we could have a query approach like the following:

```
$ python3 hollywoodgenealogy.py "Jack␣Nicholson" "Julia␣Roberts"
(0) :Jack_Nicholson starred with :George_Clooney in
:Batman_(1989_film_series)
(1) :George_Clooney starred with :Julia_Roberts in :Ocean's_Eleven
```

Indeed, to stay up to date with this information, it is necessary to repeat the search, at least each time a new film starring either Jack Nicholson or Julia Roberts is released. SEPA subscription engine would avoid this query polling procedure: by making the subscriptions in Listings 1.5 and 1.6, we can be notified if a film in which both the actors have a role is updated to DBpedia. Furthermore, as it can be seen, the SPARQL that identifies the subscription is the same that is posted for the simple query.

Listing 1.5 – Subscription to new direct connections

```
SELECT ?film1 WHERE {
 ?film1 dbpo:starring dbpedia:Julia_Roberts,dbpedia:Jack_Nicholson.
}
```

[21] https://github.com/fr4ncidir/HollywoodGenealogy

Listing 1.6 – Subscription to new one-step connections

```
SELECT ?film1 ?actor_1 ?film2 ?actor_1 WHERE {
  ?film1 dbpo:starring ?actor_1 , dbpedia:Julia_Roberts.
  ?film2 dbpo:starring ?actor_1 , dbpedia:Jack_Nicholson.
}
```

A slightly more complex example was already mentioned in the previous Section while discussing the European project AudioCommons. The AC Mediator, in this case, was designed to dispatch uniformed versions of query results coming from various services. By introducing their research Xambó et al. [98] suggested a further expansion of the AC Mediator including in the queries some characters that may require audio analysis and elaboration. This activity, clearly, can be time-consuming and may cause scalability problems.

A SEPA powered solution to this challenge can be implemented as follows. We already said that query results are transformed into RDF graphs and then inserted into an RDF knowledge base. Let us consider a SEPA engine, in this case, with an underlying Blazegraph triple store. With reference to Fig. 1.8, we can modify the schema to include *long queries* as it is shown in Fig. 1.10. Clients, here, make the request to the AC Mediator, that in this case is aware of what are the services that need the SEPA approach. If the incoming query request is one of them, the Mediator will (i) generate an URI to identify within the RDF store a subgraph in which results will be available in the end; (ii) use such URI to immediately reply to the client. With this information, the client is able to subscribe to the results of the query, and to be notified as soon as they are available (see Listing 1.7) without blocking.

Listing 1.7 – Subscription example for the SEPA-enhanced AC Mediator.

```
SELECT *
FROM <http://this_is_the_generated_uri>
WHERE {?a ?b ?c}
```

1.3.3 Future

As it was detailed, SEPA introduces a relevant dynamism to the semantic information. The IoT, in particular, will be the target of this in the prosecution of this Thesis, as by monitoring events we can follow the context evolution, and through semantics we can overcome the vertical silos fragmentation at information level.

However, SEPA presents various research opportunities for the future.

First of all, there is the need to study and enhance SEPAs performances. SEPA, in fact, suffers of quick degradation of performances as the number and complexity of subscriptions grows. Not to mention that the number of triples contained in the knowledge base has also a considerable impact on the subscription triggering engine.

Secondly, following a technological trend of cloud distribution of services, a possible future direction is to work on a distributed SEPA, addressing a set of coherence and

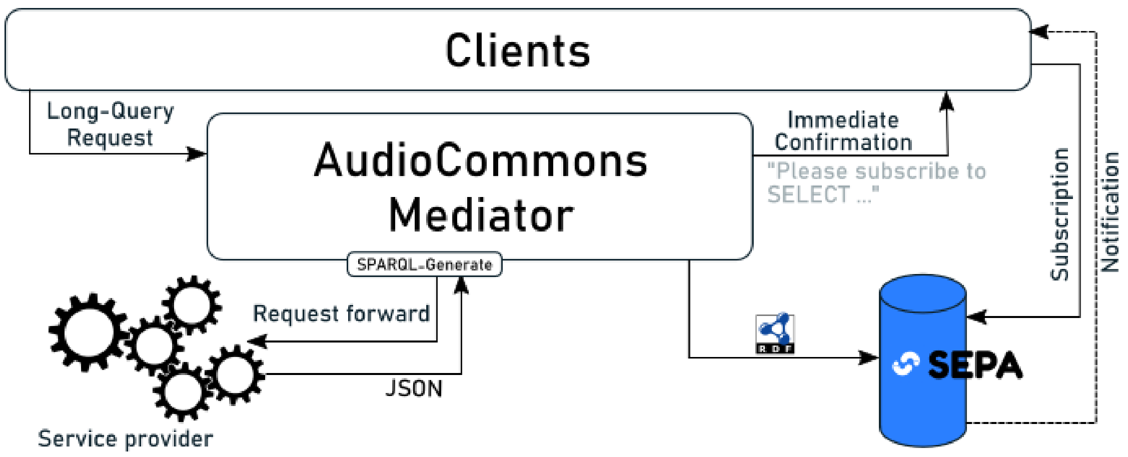


Figure 1.10 – Schema of the AC Mediator working logic for long query services.

reliability issues that, to the best of the author's knowledge, have not yet been studied over linked data endpoints.

Eventually, RDF knowledge bases can be exploited with reasoning techniques. Their effect, in particular, is the modification of the knowledge base according to some rules. The SEPA architecture has not yet been tested with reasoning, and therefore it could be interesting to see how SEPA and rule engines behave when used together in an application.

Table 1.1 – Summary of the features of the tools for the visualization of semantic knowledge bases. Legend: ✓= yes, ✗= no, != partial, ☑= multiple options available, ?= unknown, −= Not applicable

Software	T-Boxes	A-Boxes	Statistics	Filtering	Editing	Standalone	Plugin	Domain	Developed	License	Active	Reference
CytoScape	✓	✓	✓	✓	✓	✓	General SPARQL, SemScape, Vital AI	Biology	CytoScape Consortium	GPL	✓	[71]
Fenfire	✓	✓	✗	✗	✓	✓	−	General	University of Jywäskylä and Digital Enterprise Research Institute of the National University of Galway	GPL	✗	[72]
Gephi	✓	✓	✓	✓	✗	✓	SemanticWebImport, VirtuosoImporter	General	Gephi Consortium	GPL	✓	[73]
Glow	✓	✓	✗	✓	✗	✗	−	General	Erasmus University Rotterdam	?	✗	[74]
IsaViz	✓	✓	✗	✓	✓	✓	−	General	INRIA in collaboration with Xerox Research Centre Europe	GPL	✗	[75]
Jambalaya	✓	✓	✗	✓	✗	✗	−	General	Chisel Lab (University of Victoria)	Individual	✗	[77]
LOD Live	✓	✓	✓	✗	✗	✓	−	General	lodlive.it	MIT	✓	[79]
Ontograf	✓	!	✗	✗	✗	✗	−	General	Stanford Center for Biomedical Informatics Research	LGPL	✗	[80]
OntoSphere	✓	✓	✗	✗	✗	☑	−	General	Politecnico di Torino	LGPL	✗	[81]
OWLViz	✓	✓	✗	✗	✗	✗	−	General	University of Manchester	LGPL	✗	[82]
PGV	✓	✓	✗	✗	✗	✓	−	General	LSDIS Lab and Computer Science (University of Georgia), Kno.e.sis Center (Wright State University)	?	✗	[83]
RelFinder	✗	✓	✓	✓	✗	✓	−	General	Visualization and Interactive Systems (University of Stuttgart), Agile Knowledge Engineering and Semantic Web (University of Leipzig), Interactive Systems and Interaction Design (University of Duisburg-Essen)	GPL	✗	[85]
Tarsier	✓	✓	✗	✓	✗	✓	−	General	Advanced Research Center on Electronic Systems (University of Bologna)	GPL	✓	[69]
TGVizTab	✓	✓	✗	✗	✗	✗	−	General	IAM Group (University of Southampton)	GPL	✗	[87]
VOWL	✓	!	✓	✓	✗	☑	−	General	Visualization and Interactive Systems (University of Stuttgart), Alexandru Ioan Cuza University	MIT	✓	[89], [88], [90]

Chapter 2

Semantic Internet of Things

Within this Chapter a few facets of the Internet of Things will be discussed, highlighting in particular its integration with and within semantic technologies. Over the Sections some actual implemented ideas will explain which are the benefits of adding the semantic layer, both with a SEPA dynamic background, and without.

In addition to this, in Section 2.2 we will completely present the first major contribution of this Thesis, i.e., the Internet of Musical Things ontology. Its main goal is to open the way to collaborative musical production, with connected musical devices. Although it may appear that we address a limited subset of the whole IoT, Section 2.2 will give proof that the methodology is on the contrary general and open for further enhancements.

2.1 Semantic Interoperability

Chapter 1 gave a broad view of the IoT panorama. The addition of semantics, as we said, could represent a technique to overcome vertical fragmentation. As a matter of fact, interoperability was one of the leading ideas since the first appearances of the Semantic Web concept [2].

But what does *interoperability* actually means? As we said in Section 1.1, from a technical point of view the term interoperability implies that there should be, among the considered systems, the same ISO-OSI stack. Consequently, interoperability should allow systems to be aware of the existence of other systems regardless of how they internally work. Furthermore, a successful exchange of requests should also be possible, including failure robustness.

Discovery mechanisms, for this reason, represent the bigger difference among interoperable and non interoperable systems: they allow agent-awareness and, if implemented at higher level, also successful communication feedback. Literature is rich, in this sense, and over the time provided a plethora of examples, middlewares and projects, some of which have been already cited. They, however, often operate in a closed environment, and mostly assume that parallel projects, whenever interested in a collaboration, should

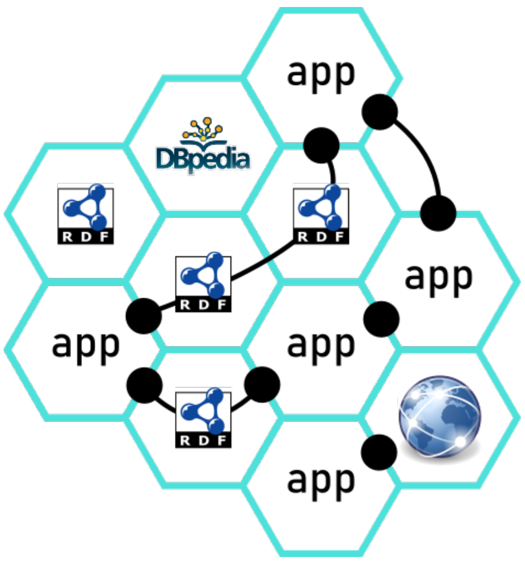


Figure 2.1 – Conceptual vision of the Semantic IoT. Local applications can directly communicate with each others, or with the Internet. Alternatively, they can share data through RDF formatting according to one or more ontologies.

be reprogrammed to be compliant with their view.

The Semantic Web overcomes this required parallelism at information level as shown in Fig. 2.1. As it can be seen, the Semantic Internet of Things keeps its specific implementations, assumptions and realizations (the "`app`" boxes). However, while the incompatibilities in previous IoT were either not treated, or case by case solved with complex elaboration, we here transform it in semantic subgraphs that can coexist.

Consequently, interoperability is in this Section granted among collaborating systems relying on a continuous process of ontological inclusion. Such process was also targeted by previous research (e.g., [108, 109]). The Internet of Musical Things Ontology (see next Section) in this context provides a methodology to create such inclusion. The choice of a musical background, furthermore, is not limiting the suggested view: the core of the ontology, in other environments, is the common IoT universe (i.e. the `iot` namespace). The goal of Section 2.2 is to setup an ontology by joining together other preexisting ontologies and taxonomies. Besides the innovative result materializing the IoMust in the Semantic Web panorama, one of the related outcomes is the interoperability achieved with IoMusT and any application made on top of the provided ontology components.

In Chapter 3, in a different way, we will provide a broader view (but still semantically compatible with the present one), which includes also the web resources, unifying the Web and the IoT in the WoT over a semantic platform exploiting also SEPA.

2.2 Internet of Musical Things

This Section takes inspiration from a recently submitted research paper: © Turchet, L., Antoniazzi, F., Viola, F., Giunchiglia, F., & Fazekas, G. (2019). The Internet of Musical Things Ontology. Submitted to *Journal of Web Semantics.*.

The Internet of Musical Things (IoMusT) is an emerging research area consisting of the extension of the Internet of Things paradigm to the musical domain. This field is positioned at the confluence of music technology, the Internet of Things, human-computer interaction, and artificial intelligence, and relates to the networks of computing devices embedded in physical objects (Musical Things) dedicated to the production and/or reception of musical content. Considering the computer science perspective, Turchet and colleagues [110] defined a Musical Thing as

> *a computing device capable of sensing, acquiring, processing, or actuating, and exchanging data serving a musical purpose*

and defined the IoMusT as

> *the ensemble of interfaces, protocols and representations of music-related information that enable services and applications serving a musical purpose based on interactions between humans and Musical Things or between Musical Things themselves, in physical and/or digital realms. Music-related information refers to data sensed and processed by a Musical Thing, and/or exchanged with a human or with another Musical Thing*

Various kinds of Musical Things can be envisioned, which may be categorized according to the musical purpose they serve (e.g., to control, generate, or track responses to musical content). Examples of existing Musical Things are the “smart musical instruments”, a new family of musical instruments encompassing sensors, actuators, wireless connectivity, and on-board processing [111]. These musical devices are able to directly exchange musically-relevant information with one another as well as communicate with a diverse network of external devices, such as smartphones, wearables, virtual reality headsets, or stage equipment. Instances of smart musical instruments include the Smart Cajón reported in [112] and the Sensus Smart Guitar developed by MIND Music Labs [113]. Another example of Musical Things are “musical haptic wearables” [114], a novel class of wearable devices embedding haptic stimulation, tracking of gestures and physiological parameters and wireless connectivity features. On the one hand, such devices were conceived to enhance communication between performers as well as between performers and audience members by leveraging the sense of touch in both co-located and remote settings. On the other hand, they were devised to enrich musical experiences of audiences of music performances by integrating haptic stimulations, as well as provide new capabilities for creative participation thanks to embedded sensor interfaces.

Musical Things are connected by an infrastructure that enables multidirectional communication, both locally and remotely, between different stakeholders such as composers, performers, audience members, audio producers, live sound engineers, as well as music students and music teachers. The ecosystems that will form around Internet of Musical Things technologies are envisioned to support novel forms of interactions between such stakeholders by means of novel musical applications and services. This has the potential to revolutionize the way music is composed, performed, experienced, learned, and recorded.

To accomplish the IoMusT vision, the Musical Things within an ecosystem need to dialog through a common language. A central unsolved issue within the IoMusT paradigm is how facilitating interoperability among heterogeneous Musical Things, which may serve radically different purposes (e.g., real-time analysis of musical content, generation and delivery of haptic, visual, or olfactory sensory layers additional to the musical content, delivery of content-recommendation services for music students). To date, interoperability across musical devices has mostly relied on protocols for the exchange of musical messages such as Musical Instrument Digital Interface (MIDI) or Open Sound Control (OSC) [115] and tools based on it (e.g., libmapper [116]).

However, the existing musical protocols are not adequate to support interoperability across the wide heterogeneity of Musical Things, as they are typically not flexible, lack high resolution, not equipped with inference mechanisms, and do not support the integration with the Web. Semantic technologies, such as semantic web [2] and knowledge representation [117], possess these features. For this reason, they have been recently envisioned as a solution to enable interoperability across heterogeneous Musical Things [110]. Existing ontologies devised for the musical domain to date, such as the Music Ontology [**music**], the Studio Ontology [**studio**] or the Audio Features Ontology [**afo**], are insufficient to represent the wide knowledge base that the variety of the possible Musical Things entail. An ontology specific to the IoMusT scenario is currently missing. As a consequence, the use of semantic technologies in Internet of Musical Things contexts is limited to scenarios involving homogeneous Musical Things serving similar musical purposes or ad-hoc interactions designed for a specific, fixed scenario.

The first effort towards the application of semantic technologies to the IoMusT context is reported in [118]. The authors proposed a semantically-enriched Internet of Musical Things architecture relying on a semantic audio server and edge computing techniques. Specifically, a SPARQL Event Processing Architecture [99] was employed as an interoperability enabler allowing multiple Musical Things to cooperate, relying on a music-related ontology. A limitation of the developed architecture was the involvement of an ontology restricted to the representation of simple musical features, which prevented Musical Things dedicated to purposes other than music generation to join the ecosystem formed around the architecture.

Semantic technologies based on an ontology for the IoMusT can assist in managing, querying, and combining information characterizing an IoMusT ecosystem, including data about the music produced, the involved stakeholders, the utilized Musical Things and their application and services. This has the potential to spur the exploration of novel artistic avenues, such as performance and composition, for instance based on emergent properties of an IoMusT ecosystem [119].

In this Section the "Internet of Musical Things Ontology" [**iomust**] is exposed. The full design and evaluation process of the IoMusT Ontology is hereby given from the beginning to the current version, i.e., 1.0.0. The description of the IoMusT Ontology follows the MIRO (minimum information for the reporting of an ontology) guidelines [120]. For reference, the paper reports the MIRO designations (e.g., E.9 for Ontology relationships), where the specific information item is provided. The ontology name (A.1) and its need (B.1) have been already introduced. The ontology is available on the web (see [**iomust**]) (A.4) with license GPL3 (A.3).

2.2.1 Methodology, audience, and scope

This section describes the methodology adopted for the design and development of the IoMusT Ontology, as well as the audience of the ontology and its scope.

Methodology for ontology development

The ontology is developed and maintained by the authors and other members of the emerging IoMusT research community, which is currently composed by leading research institutes in sound music computing and Internet of Things (A.2 and C.2).

The design and development of the IoMusT Ontology was mostly inspired by the *Methontology* methodological framework [121] (A.6). Such a framework is composed by six phases: i) the specification, i.e., the identification of the audience, scope, scenarios of use, and requirements; ii) the conceptualization of an informal model; iii) the formalization of the ontology namespaces, classes and properties; and iv) the integration of existing ontologies in a description and its reproduction in an OWL2 file [122]; v) the implementation of the ontology with an appropriate serialization language; vi) the maintenance of the ontology once implemented.

Moreover, the aforementioned framework identifies three tasks that are accomplished during the whole life of the ontology, which are orthogonal to the five phases: i) knowledge acquisition through research of related ontologies and models as well as gathering data from potential users), to inform multiple phases of the design process, mainly conceptualization and integration; ii) documentation of the process phases (internal) and the ontology specification (public); iii) the evaluation of the ontology before its release.

Other works, like Uschold [123] and more recently by De Nicola et al. [124] suggest different methodologies for ontology engineering. These papers however include techniques that aim to formalize the setup from scratch of new ontologies. This is not the case in the current research, where the goal is to provide a new contribution based as much as possible on the integration of pre-existing contents.

Scope and audience

The role of the IoMusT Ontology is to offer a common data model enabling interoperability among heterogeneous Musical Things, which allows both people and virtual agents to seamlessly generate, explore, access, or transform music-related content produced within an IoMusT ecosystem. Therefore, the scope of the ontology (C.1) is represented by all ecosystems forming around existing or future IoMusT technologies.

The target audience of the ontology (B.3) is represented by all actors and stakeholders that are involved in such ecosystems, including performers, audience members, composers, studio producers, live sound engineers, and choreographers.

2.2.2 Related ontologies and data models

Before defining an ontology specific to the IoMusT domain we conducted a review of existing ontologies. The IoMusT vision is intrinsically multisensory and highly interdisciplinary [110]. This section describes ontologies and data models (B.2) that are related to such a vision. They have been gathered through the research of literature and online resources (D.1 and D.2) and evaluated as part of the design process (D.3).

Ontologies for the audio domain

Several ontologies have been proposed in recent years for the audio and music domains, in recognition of the complexity and broad ranging applications of such ontologies, and the fact that much of the information exchanged on the Web today is multimedia, of which music is a very substantial component, rather than text. The scope of such ontologies are wide ranging, starting from very focused areas of music production such as audio effects [125], to larger binding ontologies that target the description and retrieval of audio resources on the web in general [95].

The Music Ontology (MO) [**`music`**] [126, 127] is a general purpose high-level ontology for the music domain that models the music value-chain from production to consumption. Therefore its focus is on editorial metadata, e.g. artist name and title associated with audio recordings, as well as the representation of major steps in the production of recorded music, from composition, through performance and recording, to release.

The Music Ontology does not deal with the nuances of technical workflows in music production. This is the area covered by the Studio Ontology [**`studio`**] (SO) [128].

The Audio Features Ontology [**`afo`**] (AFO) addresses another audio domain that requires detailed conceptualisations. Audio Features are descriptors that represent specific characteristics of sound signals. These may relate to measurable properties of the signal, such as bandwidth or spectral centroid, perceptual qualities like pitch and loudness, and musical characteristics such as notes, musical key and chords.

The Musical Instruments Ontology [129] is highly relevant to the domain of IoMusT. It provides an ontological model for encoding well known instrument classification systems, e.g. for grouping instruments into categories such as *Idiophones* or *Aerophones*, based on their sound production or excitation mechanism.

The Audio Commons Ontology (ACO) [**`aco`**] [95] is an example of a higher level domain ontology that binds several audio related ontologies together. It was designed to facilitate the integration of audio content repositories on the Web as well as content consumption by software agents.

Ontologies for sensors, actuators and connectivity

Among the ontologies designed for the IoT, two of the most diffuse are SSN (Semantic Sensor Network) and SOSA (Sensor, Observation, Sample, and Actuator) [**ssn, sosa**]. Both SSN and SOSA adopt a complex approach to description of hardware, observation of physical entities and actuation. SSN [130] covers the majority of the SensorML standard[1] and has been designed to describe sensors and observations, as well as the deployment in which sensors are employed. SOSA [131] adopts a lightweight approach to describe sensors, actuators and the acts of observation and actuation. SOSA acts as a replacement of the Sensor-Stimulus-Observation (SSO) design pattern provided by SSN, that provides greater expressivity [132].

At a higher level of abstraction, things in IoT can be represented according to the Web Thing model[2] proposed by the W3C. In this sense, devices are provided with the so-called thing description, a detailed profile reporting properties, events and actions exposed through its interface. A first attempt to semantically represent this model has been provided by Charpenay et al. in [133], later on envisioned by Serena et al. in [134] for a discovery framework. The Web of Things ontology discussed by Charpenay et al. and Serena et al. has been employed by Viola et al. [135] to build a semantic Web of Things enviroment for recommendations in the audio domain. Eventually, Antoniazzi et al. [136] provided a semantic version of the Web of Things.

2.2.3 Specification

The acquired knowledge was then analzed to identify a set of requirements that the ontology should satisfy [137]. The literature review led to a total of 15 scenarios (5 scenarios from [110], 5 from [111], and 5 defined by the authors or derived from recent experiments with users described in the literature). For each scenario we derived a set of requirements, and then applied a thematic analysis [138] to reduced them. The resulting requirements are represented below as a list of example questions that the ontology should be able to support answering [139], and a list of formal requirements.

Competency questions

The following sample questions are meant to be asked with respect to an IoMusT ecosystem:

1. Which type of Musical Things are used by the local and remote performers during the live concert?
2. How many Musical Things used by the audience provide haptic feedback?
3. What smart instruments are controlling the smartphones used by the audience?
4. What is the mood of the music at a given time during the live performance?

[1] https://www.opengeospatial.org/standards/sensorml
[2] https://www.w3.org/Submission/wot-model/

5. How many audience members are actively participating to the music creation process thanks to their Musical Things?
6. Which kind of stage equipment is used at a given time during the concert?
7. Which gestural and biometric parameters are tracked from the audience during the live performance?
8. How many and which kind of networks are used during a performance?
9. Which pedagogical applications are available for a smart violin?
10. With which music content repository a smart ukulele can interact?
11. Which services are available for a smart guitar and what are their purposes?
12. What type of sensors and actuators compose a smart musical instrument or a musical haptic wearable?

Formal requirements

The IoMusT Ontology should be able to:

1. represent the concept of Musical Things, including:

   (a) its type (e.g., musical instrument, wearable device, stage equipment);

   (b) its characteristics including the number and type of inputs (e.g., sensors tracking movements or biometric parameters) and outputs (e.g., auditory, visual, haptic, olfactory);

   (c) the type of person for which it is conceived (e.g., performer, audience member, live sound engineer, producer);

   (d) its function (e.g., a smart instrument used to produce musical content, a musical haptic wearable aiming at enriching the listeners' musical experience, an interface used by audience members for participatory purposes, a device used to infer the mood of audience members based on sensed quantities);

   (e) its geographical position;

   (f) the type of data that it generates (e.g., audio signal, text message);

2. represent the concept of connectivity, including:

   (a) the type of network involved (e.g., local network, remote network, Wi-Fi-based, millimeter waves-based);

   (b) the attributes of the network (e.g., bandwidth, speed, synchronization mechanisms);

   (c) the time taken by the network to deliver/receive a message to/from a certain Musical Thing;

3. represent the concept of application and service, including:

   (a) its purpose (e.g., for music learning, performance, composition, studio production)

   (b) its level of interactivity (e.g., interactive, non-interactive)

    (c) its type (e.g., social network, online music content repository)

    (d) its user (e.g., composer, performer, studio producer, educationalist, student, audience member)

4. describe attributes of the music (produced live) at a given time, including:

    (a) low-level features (e.g., the density of notes);

    (b) high-level features (e.g., the mood)

5. describe attributes of the ecosystems, including:

    (a) the number and type of Musical Things present in the network at a given time and a given space;

    (b) which Musical Things are interacting;

    (c) the number and type of applications and services available within the ecosystem;

    (d) the number and type of networks used at a given time.

2.2.4 Ontology description

The Internet of Musical Things ontology (the IoMusT Ontology) has been developed incrementally. As a matter of fact, the task of developing ontologies is in general complex, and needs an approach that involves continuous refinement and check of concepts and relationships. This can be done in several ways, and may be performed iteratively as long as the expected match of the ontology with the real subject is achieved.

Not surprisingly, the first step is to split the domain of interest in smaller parts if possible. For each of those smaller parts, secondly, iterations are needed to ensure that all relevant concepts are included. Sometimes this is done by surveying a pool of experts and/or future users of the ontology, to get their feedback. Clearly, this check helps designers to avoid wrong naming on resources, as well as to detect and correct contradictory assertions.

Then, the smaller parts have to be joined together to form the ontology. Again, the expressiveness of the complete work has to be checked, and in this paper this is provided by requirement analysis and evaluation. The question to be answered, here, is: *is my ontology capable to describe my context? If so, is the description made with the precision needed?* This process also may be performed iteratively.

IoMusT Ontology is not an exception. On the contrary, it is very important to notice that the formalization of a vocabulary for Internet of Musical Things needs this feedback process to ensure a coherent representation of music-related entities with general-purpose contents.

A bottom-up process was deployed for our case. In particular, jumping from wider to narrower concepts, the first idea to be discussed is indeed the connection that stands between the global interpretation of Internet of Things and how to decline it into the Internet of Musical Things. See also Fig. 2.4. Clearly, the former is larger than the latter, which should represent a specialization and rely on it. The usage of the IoMusT Ontology, as a consequence, should allow a transparent view of any Musical Thing context as an IoT system. There would be no point in ignoring this core aspect, because the core idea of ontology engineering is to provide a shared and interoperable way to collaborate between different fields of knowledge. Any design choice opposed to this view would have as a direct result the creation of another vertical silos within the IoT chaos [140].

In order to replicate in the ontology this necessary duality, this work will suggest the adoption of two new namespaces:

1. `iot`, that will be used to connect concepts that belong to the broader view of generic devices;

2. `iomust`, which is an extension of `iot` defined as `iot:musical`. Within this namespace are organized the concepts of music-related IoT;

For the sake of clarity the prefixes are kept in a contracted form. To see their expanded version, please see the List of Ontologies.

`iot` namespace

The first concept to be defined is the *Thing* as it is intended in the acronym IoT. Liu et al. [14] survey and comment the spectrum of definitions that have been suggested in literature over the time. Among the surveyed entries, the one proposed by the IEEE is coherent with our requirement of generality: the thing

is any physical object relevant from a user or application perspective

meaning that we consider things as items exploiting, or being exploited by, other items. Therefore, from now on, the class `iot:Thing` has to be considered according to this definition. Notice that also regular everyday life objects may be `iot:Thing`s, like chairs, pillows, a scarf, a painting and, indeed, a musical instrument.

Clearly, this is a generic class that needs to be further specialized in subclasses. Again, a lot of help can come from the listings in [14], as `iot:Thing` is definitely a huge container. For instance, things can be wearable objects: so, the class `iot:WearableThing` can be defined to represent this category. Similarly, devices can also be *smart*, so we call for `iot:SmartThing` class: smart things, e.g., a smartphone, a smart TV, include special technological features or artifacts that provide them with relevant added value over the basic version of the same object.

Eventually, things can be connected to a communication network: they are, in this case, instances of the class `iot:ConnectedThing`. Notice that the aforementioned Studio Ontology [**studio**] contains a rich environment of properties and classes related to connectivity (e.g., the *Connectivity* and the *Device* sub-ontologies).

It would neither be reasonable, nor useful, to list here a hundred of possible subclasses. For this reason, in the present paper only a few will be defined, as the discussion requires them. It is important to notice that there is complete freedom to include new classes whenever needed, as this is precisely the kind of incremental approach for ontology engineering which was above aforementioned.

Musical things in the `iomust` namespace

IoMusT Ontology, as already said, aims to develop the `iot` namespace in its musical flavour. To do so, the reference to a vocabulary connected to music is essential. Our reference in this work was introduced in Section 2.2.2: it is the Music Ontology [**music**], which will be mentioned as the `music` namespace. An important contribution of this namespace in IoMusT Ontology is its supporting role in creating the archetype of Musical Thing, i.e. the class `iomust:MusicalThing`. In the present work, our definition for this class is the following: the Musical Thing is *a thing used to produce or enjoy music, with reference to its context.* As a consequence, IoMusT Ontology will consider that a smart loudspeaker, a CD by David Bowie belong to that class, as well as a smart violin located in a concert hall. The same smart violin, however, if stored for exposition in a museum, is no more an `iomust:MusicalThing` because it loses its musical production interest.

The class `iomust:MusicalThing` is indeed less generic than its superclass `iot:Thing`, because it provides a light form of contextualization. Yet, however, we need more precise solutions to be even less abstract. All the items identified in the example above (the smart loudspeaker, the CD, the smart violin) would point to `iot:Thing` through the `rdf:type` predicate. Then, to include an explicit reference to music, and introduce the Internet of Musical Things namespace, the following rule applies:

Rule 1 *If an `iot:Thing` instance is also connected through `rdf:type` to a class belonging to the Music Ontology, then it is also an instance of `iomust:MusicalThing`.*

A typical application of Rule 1 is the aforementioned smart violin: consider Expression 2.1 as an example, where a simple triple representation is given of the implication expressed. Notice that Rule 1 is not intended to be strictly reversible: during a concert, lights and smoke machines may be intended as Musical Things because of their essential contribution to the listening experience, and yet may not be included in one of the `music` namespace categories.

$$\texttt{ns:SmartViolin a iot:Thing, music:Instrument} \Rightarrow \texttt{ns:SmartViolin a iomust:MusicalThing} \quad (2.1)$$

The Generic Musical Thing definition is not enough to build the complete IoMusT. Here it follows a sequence of new classes to be introduced in the `iomust` environment, descending from the Musical Thing. Each of the classes here corresponds to a rule similar to Rule 1 in the OWL.

`iomust:SmartMusicalThing` is a Musical Thing that is also an `iot:SmartThing`;

`iomust:SmartInstrument` is a Musical Thing that is also a `music:Instrument`;

`iomust:WearableMusicalThing` is a Musical Thing that is also an `iot:WearableThing`;

`iomust:StageEquipment` is a collection of Musical Things serving as equipment. The definition of collection can be extracted from external ontologies designed *ad hoc* for this, like [**collection**].

Table 2.1 – Example of usage for `iot` and `iomust` namespaces. We here show how objects part of an Internet of (Musical) Things environment can be considered instances of the classes introduced in this research. Extended prefixes are available in the List of Ontologies.

	`ns:Bob`	`ns:Wardrobe`	`ns:SmartCar`	`ns:Violin`	`ns:SmartViolin`	`ns:TShirt`	`ns:StageLight`	`ns:VR_HeadSet`	`ns:HeartBeatSensor` for music experiment
`foaf:Person`	✓								
`iot:Thing`		✓	✓	✓	✓	✓	✓	✓	✓
`iot:SmartThing`			✓		✓		✓	✓	
`iot:ConnectedThing`					✓		✓	✓	✓
`iot:WearableThing`						✓		✓	✓
`music:Instrument`				✓	✓				
`iomust:MusicalThing`				✓	✓		✓		✓
`iomust:SmartMusicalThing`					✓		✓		
`iomust:SmartInstrument`					✓				
`iomust:StageEquipment` item							✓		
`iomust:WearableMusicalThing`									✓

Table 2.1 contains some practical examples of usage for the `iot` and `iomust` namespace entities.

`iot` & `iomust` sensing, actuating and interacting

So far the discussion on the IoMusT Ontology was conducted as a set of broad definitions for the baseline concepts. Here, instead, space is given to how the integration of other ontologies enables our vision of the IoMusT from a lower level standpoint.

First of all it is necessary to describe the smart devices more in detail, and include additional information related to the electronic devices embedded in the `iot:Thing` (e.g., micro-controllers, sensors, actuators). The `iot:SmartThing` was previously introduced to this effect, though without any other specificity. Consequently, to provide greater precision on the actual available sensing and actuating units, other information is needed. Taking into consideration Table 2.1 as an example, we have to provide a way to semantically distinguish between two instances of `iot:SmartThing`, like the smart violin, and the virtual reality headset, based on their setup. To achieve such goal, this work suggests the inclusion of an ontology already existing and well known in the panorama, namely, SOSA [**sosa**]. The choice of SOSA has three main advantages that greatly benefit IoMusT Ontology: (i) SOSA is *de facto* a light version of SSN, and therefore the IoMusT Ontology can be furtherly extended towards SSN integration very easily; (ii) SOSA is very simple, which is always a relevant factor when studying, building and integrating ontologies; (iii) SSN and SOSA, eventually, are a relatively recent W3C recommendation (the last draft dates back to 2017), which means that they are globally accepted as a reference.

The realization of this ontological alignment is made by including as a plug-in the concept of `sosa:Platform` in the IoMusT Ontology subgraph for the `iot:Thing` and its aforementioned subclasses. According to SOSA documentation, the `sosa:Platform` is an *entity that hosts other entities, particularly Sensors, Actuators, Samplers, and other Platforms*, that is precisely the facet missing until now in the `iot` namespace. In Fig. 2.2 a few examples are provided to show how the connection can be made. As it can be seen, the smart guitar instance `ns:SmartGuitar` has also as `rdf:type` the `sosa:Platform` class. This additional type allows us to include references to the sensors and actuators on board, as well as the entity they measure. Further details on sensing and measurement description, extensively discussed in previous researches like [141, 142] and surveyed in [143], are out of the scope of this paper. For the future, anyway, the possibility to integrate new ontologies still exists: for the ones exploiting SOSA and SSN, such process should be trivial.

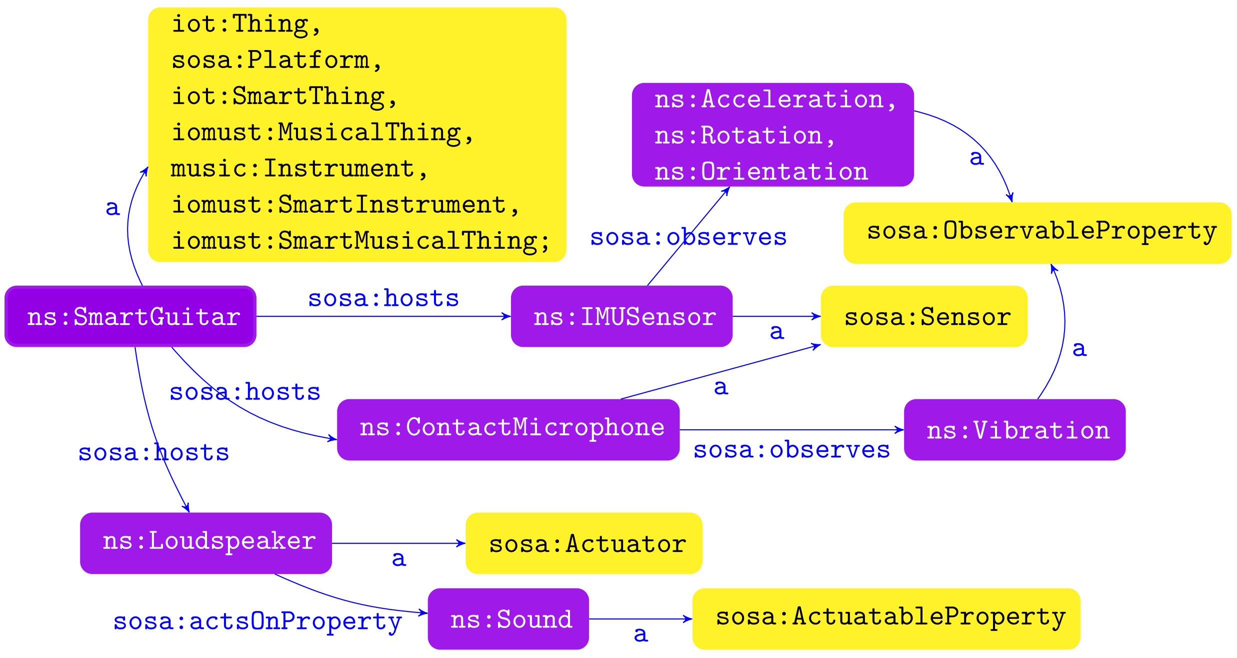


Figure 2.2 – SOSA integration with the IoMusT Ontology. Extended prefixes are available in the List of Ontologies. The color scheme is the same used in Protégé [144].
Legend: Classes **Individuals** **Object Properties**.

Sensing and actuating are in general part of a greater intent of interactive IoT system design. Data collection, then, provides the tools to create a feedback to control actuation and, eventually, to show smart behavior. Interaction is an unavoidable part of this process and, consequently, it should also be represented in the ontology alongside with sensors and actuators. Once such semantic prototype is given, it is possible to distinguish the active resources from the environmental passive ones and an interaction is finally possible. Besides, if the semantic view is shared among various systems horizontally, a strong and effective interoperability is automatically achieved.

The study of entities interacting within their environment is a well established field in literature, leveraging the concepts of *agent* (e.g., [145, 146, 147] and many others) and *semantic agent* (see, for instance, [148, 149]).

Within the IoMusT Ontology, the agent is referred to as any entity, human, object or virtual, that is capable of triggering any kind of dynamic evolution in an environment populated by instances of `iot:Thing` class. Both `iot` and `iomust` namespaces do not include directly such content, as their focus is the device, regardless the interaction aspect. For this reason, and for the discussion above, the IoMusT Ontology needs to rely on external ontologies to properly provide a definition of agent. Similarly to what has been suggested in the previous paragraphs with SOSA, we suggest here to exploit well-known ontologies, namely [**`foaf`**] and [**`prov`**].

The former, once connected to the IoMusT Ontology, defines the `foaf:Agent` as *person, group, software or physical artifact*, and *things that do stuff.* The idea of agent suggested in the previous paragraph is clearly derived from FOAF, although its real utility, in our research, is its capability of including the human being class `foaf:Person` and relationships in the semantic environment. Agents, intended as physical and virtual devices, are described through the latter, PROV-O, where the agent is *something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity* [150]. This idea, in particular, includes also entities running software, which belong to `prov:SoftwareAgent`. Listing 2.1 shows an example of using FOAF and PROV-O, and introduces in the `iot` namespace the ownership property `iot:owns`.

Listing 2.1 – FOAF & PROV-O integration with the IoMusT Ontology. Extended prefixes are available in Table 4.4.

```
ns:cristina      a              foaf:Agent, foaf:Person,
                                prov:Agent, prov:Person;
                 foaf:name      'Cristina';
                 iot:owns       ns:SmartGuitar.
```

Ownership and actual usage do not necessarily coincide: it may happen, for instance, that people use a tool belonging to someone else. Besides, ownership does not imply any sort of activity with the device. A setup for activities, part of the IoMusT Ontology, is available in Fig. 2.3.

As it can be seen, Fig. 2.3 contains a rather complex subgraph. First of all, the application introduces the resource URI `ns:bob` as a music performer by exploiting the Music Ontology. FOAF ontology then provides the `foaf:knows` relationship with other people semantically represented.

After that, by using the `iot` namespace, we start setting up a semantic network to identify the ongoing process involving things and users. In this case the user “Bob” is the subject for the predicate `iot:isInvolvedIn`, that targets a new resource URI with type `iot:Application`. This application class can be explained as the semantic endpoint

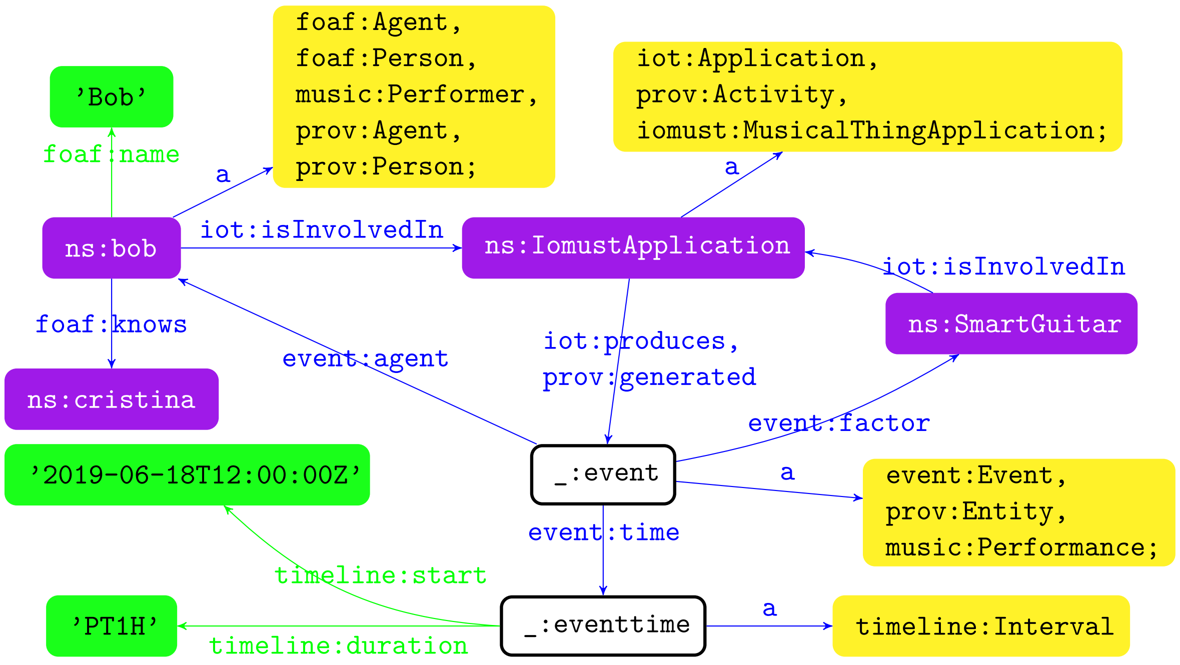


Figure 2.3 – Activities in the IoMusT Ontology. Undefined resources can be found in previous Figures and Listings. The color scheme is the same used in Protégé [144].
Legend: Classes **Individuals** **Object Properties** **Literals, Data Properties** **Blank Nodes**.

tagging together all elements, items and agents involved in an activity. A similar description is given by PROV-O documentation for the `prov:Activity` class. Notice that also the device `ns:SmartGuitar` points to the same instance of `iot:Application` accordingly. In addition to this, in order to create the musical background for the IoMusT Ontology, a subclass of the `iot:Application` is suggested for specific IoMusT usage, as reported in Rule 2.

Rule 2 *If an `iot:Application` instance is also connected through `iot:isInvolvedIn` to an instance of a class belonging to the Music Ontology, or to the iomust namespace, then it is also an instance of `iomust:MusicalThingApplication`.*

The application, indeed, is not only a matter of involving the participation of people and objects in an activity. The goal of the IoMusT Ontology is also to represent the application following its sequence of steps over time. Fig. 2.3 highlights how this is possible through the usage of the predicate `iot:produces`. The logic supporting this predicate refers to the application as timed sequence of events, where the *event* is semantically represented by the Event [**event**] ontology over the `event` namespace. As it is reported, the event is spawned as a blank node (it may appear on the go), and fully benefits of the predicates available: in a few triples we get full informations on the acting agents (e.g., `ns:bob`), the tools used (e.g., `ns:SmartGuitar`), and the timings by further addition of the Timeline [**timeline**] ontology. Moreover, being the event a source of information, we declare it also as a `prov:Entity`, alongside with any other information that may be interesting for the user (e.g., the event is a `music:Performance`). Summarizing, Listing 2.1 and Fig. 2.3 together refer that `ns:bob` performed some music playing `ns:cristina`'s smart guitar in a performance that lasted 1 hour.

Location of devices

Another relevant problem is location of entities in IoT and IoMusT environments. Such piece of information is extremely useful, for example in making spatial statistics on collected data. In order to provide the ontological tools to locate devices, a few considerations follow.

Currently, PROV-O ontology already has an object property devoted to location, namely `prov:atLocation`. The triple pattern, in such case, is represented in Listing 2.2 (Example 1) and, as it can be seen, requires the location to be a semantic resource URI. For the example, a DBpedia resource was chosen. To address also situations in which more precision is required, a data property with range `xsd:string` has been added to the `iot` namespace, `iot:atLocation`, that is used in Example 2.

Listing 2.2 – Location triples alternatives. Extended prefixes are available in Table 4.4.

```
ns:SmartPiano    a        iot:Thing, iot:SmartThing,
                          iomust:MusicalThing, music:Instrument,
                          iomust:SmartInstrument, sosa:Platform,
                          iomust:SmartMusicalThing, prov:Entity;

[Example 1]
                 prov:atLocation dbpedia:London.

[Example 2]
                 iot:atLocation  "51°30’49.3’’N 0°05’59.9’’W",
                                 "GW72+F2,␣London",
                                 "Paternoster␣Row,␣London,␣UK".
```

2.2.5 Implementation and maintenance

The ontology development is accomplished in an online public git repository hosted on GitHub[3] (A.5). The issue tracking system offered by GitHub, will be used as communication channel for maintenance and future development of the ontology (C.3).

The IoMusT vision is structured around several subdomains and related fields, from interfaces for musical expression to the connectivity infrastructure [110]. The creation of an ontology encompassing all the possible facets of the IoMusT domain in all their complexity would be a very significant task that is beyond the scope of this work. For this reason, the IoMusT Ontology is an implementation-driven ontology that is evaluated

[3] `https://github.com/fr4ncidir/IoMusT`

and evolves during its use while developing applications. This means that the ontology will be growing depending on the appearance of new components around which IoMusT ecosystems are structured, such as novel Musical Things, connectivity infrastructures, or innovative application and services (F.1). On the technical level, the last version of the ontology will always be accessible at the IoMusT Ontology URI, while past versions will accessible using an URI scheme including the version ID (F.3). For backward compatibility's sake, all the defined concepts will remain in the ontology and keep their current meaning. In case at some point the ontology maintainers decide that a concept is "not to be used any more", it will be annotated as deprecated (F.2).

In its current version, the IoMusT Ontology describes the IoMusT in general terms. As a matter of fact, the work presented in this paper targets a system engineering view enriched with musical content. Consequently, the intent of this research is to provide tools for a global description and easy integration of a new and promising field of IoT. Such premises, as it appears in Section 2.2.4, result in a description schema that overviews the IoT in its musical flavour and its higher level features, but does not give in the examples a taxonomy for the specific devices (i.e., there is no attempt at all to define any form of *Guitar ontology, Violin ontology*, and so forth).

Indeed, looking towards the future, it is clear that any musical instrument-specific ontology together with the IoMusT Ontology would represent a set of shared and consistent axioms able to provide a complete semantic approach to internet-connected instruments. Extremely precise discovery over contexts described with a music-professional view may be enabled in this way.

Looking to Fig. 2.4, moreover, the forthcoming path is quite easily understandable. First of all, the inclusion of new lower level vocabularies-taxonomies-ontologies to describe as clearly and easily as possible the IoT. Secondly, the enhancement of `iomust` namespace leveraging both the core `iot` and the new music related ontologies that may appear in the panorama. Eventually, a continuous feedback by developers trying to make innovative and groundbreaking connections between distant fields. *Is the IoMusT Ontology easy to use when it comes to coding? Was it possible to develop your project of connecting the IoMusT Ontology and the new Automotive ontology together?* Implementation and maintenance, in this situation, overlap almost completely.

2.2.6 Evaluation

The IoMusT Ontology was assessed by using formal methods as well as checking its fitness for our domain and purposes.

Metrics and Formal Validation

Evaluating an ontology is always a matter of identifying the best trade-off between its expressiveness and the performance of applications based on its concepts (i.e., the effective usage). The former is the prevailing aspect in philosophical ontologies, while the latter is of course the most important when dealing with engineered ones.

Fernández et al. [151] defined twelve metrics to measure the quality of an ontology

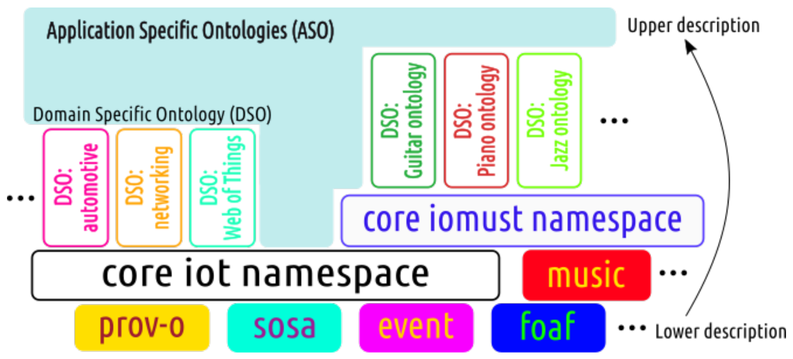


Figure 2.4 – The IoMusT Ontology is built up incrementally leveraging lower level concepts. It provides the base for other *Domain Specific Ontologies* (DSO) and other Application Specific IoT ontologies (ASO).

that we hereby report. In the current paper, not all the metrics have been applied, and some of them required slight modifications to fit the scenario. The reason for this is that, of course, ontology engineering is often a matter of personal interpretation of the designer. Similarly to coding, where evaluation of different implementations and algorithms is made on complexity and performances, the metrics considered relevant for this paper are those belonging to the class of "Knowledge coverage and popularity measures". On the other hand, as IoMusT Ontology is built up as a compound of sub-vocabularies, global metrics are considered less relevant, an will not be included here.

- **Number of classes**: it consists of the number of classes in the analysed ontology.
- **Number of properties**: this value represent the number of datatype and object properties in a given ontology.
- **Number of individuals**: it is the number of individuals in the ontology.
- **Direct popularity**: this metric represents the number of ontologies importing the given ontology. Being a novel ontology, the popularity is of course equal to zero.
- **Inverse popularity**: the number of well established ontologies, classes and properties imported within the given ontology. It is a way to measure of interoperability with other works vs the novelty introduced, and is calculated on the most basic possible usage (i.e., the one provided in the OWL of the ontology).

Values for this metric are reported in Table 2.2.

Based on our previous experience on developing ontologies, metrics belonging to the "structural ontology measures", have been replaced by an alternative set of metrics:

- **Minimum Musical Thing triple count**: the minimum number of triples needed to describe a Musical Thing. According to the previous examples available in

Table 2.2 – Evaluation of the IoMusT Ontology according to the "Knowledge coverage and popularity measures" proposed by Fernandez et al. [151] as well as by University of Rostock in their Ontometrics Wiki[4].

Metric	**Value**	**Metric**	**Value**
Number of classes	21	Inverse popularity:	
Number of properties	15	- *Ontology imports*	7
- *Datatype properties*	4	- *Classes*	29%
- *Object properties*	11	- *Properties*	7%
Number of individuals	0	Schema metrics:	
Direct popularity	0	- *Inheritance richness*	0.57
		- *Relationship richness*	0.6

Listings 2.1, 2.2 and Figg. 2.2, 2.3, a very simple Musical Thing can be described with less than 20 triples.

- **Maximum Musical Thing triple count**: this is the maximum number of triples that can be used to describe a Musical Thing. In our case this value is unlimited, depending on the complexity of the devices.

Classes and properties have been provided with a textual description (`rdfs:comment`) in English (E.7). The ontology editor Protégé [144] and the Visual Notation for OWL Ontologies tool (VOWL) [152] have been used to check the correctness of the ontology. The logical consistency has been checked by running (through Protégé) three reasoners, HermiT (version 1.3.8.413) [153], Pellet (version 2.2.0) [154], and FaCT++ (version 1.6.5) [155] and no inconsistencies have been found.

The evaluation of the ontology went on through the OntOlogy Pitfall Scanner! (OOPS!) online service [156]. This service performs a set of checks to detect common pitfalls in ontology design (based on the existing literature). No major pitfalls have been detected in the IoMusT Ontology. Minor pitfalls have been identified due to: 1) the absence of labels defined through `rdfs:label`; 2) the absence of an inverse relationship; 3) the presence of URIs containing file extensions. As regards the first point, it is ascribable to a design choice: since the ontology (in our opinion) is already easy to read, the adoption of labels would be redundant. The last two points instead, depends on two of the imported ontologies (i.e., the Event and Timeline ontologies).

Evaluation for Requirements and Answer to Competency Questions

Metrics calculation is a good solution to obtain comparable evaluation of ontologies. However, not surprisingly numerical solutions do not take into account the actual topics treated. To address this facet, it is necessary to dive into the ontology, ask questions and evaluate the answers.

[4] `https://ontometrics.informatik.uni-rostock.de/wiki/index.php/Schema_Metrics`

We hereby suggest three sets of questions, which will be applied to the IoMusT Ontology:

1. The academic community developed over the time some suggestions for ontology engineering. In particular one of the major Conferences for Semantic Web research, namely ISWC, defined in its website[5] a pool of guidelines.

2. Miro evaluation [120], that provides an organized list of standardized questions. The report[6] of their application to the IoMusT Ontology is available in the ontology's Github repository.

3. Section's 2.2.3 competency questions.

Let us start with ISWC guideline analysis, which are also included partially in Miro report. Concerning the *Impact* section, we can definitely say that the IoMusT Ontology fulfills the requests. The answers to the questions were largely discussed over the previous paragraphs of this work, although it is worth repeating that the IoMusT has a dual value, contributing to both the IoT and Music domains. *Reusability*, then, is answered by the explanations given in Section 2.2.4, and is maximized by plugging into the IoMusT Ontology well established ontologies like SOSA, FOAF and PROV-O. Eventually, *Design & Technical Quality* and *Availability* are appropriately fulfilled by the concepts provided in Section 2.2.5.

Among all evaluations, anyway, the check for competency questions and requirements satisfaction is the most important, because it justifies the whole work. In particular, the 12 competency questions in Section 2.2.3 are almost completely successfully handled. With the exception of question 4 and 10, the IoMusT Ontology provides all the tools to perform semantic discoveries as complex as needed. So, the ontology provides all the tools necessary to format SPARQL queries that would answer the questions. Question 4, by its side, refers to an aspect that should be treated with the AS ontologies of Fig. 2.4. Instead competency question 10 may be addressed by a complex discovery including also the concepts of the AudioCommons ontology [**aco**].

Concerning Formal Requirements (Section 2.2.3) the discussion is similar, as some points can be obtained by direct usage of IoMusT ontology as we described it, and some others need the inclusion of additional resources. For example, consider question 5: it is fully achievable by performing SPARQL discoveries as described in the previous paragraph.

Competency question 4, on the contrary, refers to live attributes for music, which were not directly targeted here, as they are connected to music and the specific application, and not to devices. Questions 1 and 3 can be achieved by exploiting IoMusT ontology along with specific concepts in the AudioCommons Ontology, Studio Ontology, and Music Ontology. Question 2, then, refers to concepts available in the Studio/Connectivity ontologies.

[5] `http://iswc2018.semanticweb.org/call-for-resources-track-papers/#`

[6] `https://github.com/fr4ncidir/IoMusT/blob/master/MIRO%20report.md`

Table 2.3 – MIRO Report [120] of the IoMusT Ontology – Part I of III

A. The basics	
A.1 Ontology name MUST	Internet of Musical Things Ontology (IMTO), 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/IoMusT/blob/master/iomust.owl`
A.5 Ontology repository MUST	`https://github.com/fr4ncidir/IoMusT`
A.6 Methodological framework MUST	The ontology defines the needed concepts to create a Musical Things IoT environment by introducing the namespaces `iot` and `iomust`, and connecting them to well known ontologies like [`sosa`] and [`prov`].
B. Motivation	
B.1 Need MUST	The Internet of Musical Things is an unexplored field of IoT that at the moment lacks of semantic representation.
B.2 Competition MUST	At the moment, only with ontologies in IoT panorama. So far, no ontologies are available joining Music and IoT.
B.3 Target audience MUST	Developers of IoT applications applied to music.
C. Scope, requirements, development community	
C.1 Scope and coverage MUST	The ontology covers the concepts necessary to create a Musical Things IoT environment. The two namespaces identified in addition are extended, by plugging in references to other well known ontologies. The result is a complete vocabulary available to develop interoperable applications within and without the musical and artistic domain.
C.2 Development community MUST	Advanced Research Center on Electronic Systems (ARCES) of the University of Bologna. Centre for Digital Music (C4DM), Queen Mary University of London.
C.3 Communication MUST	`https://github.com/fr4ncidir/IoMusT/issues`
D. Knowledge acquisition	
D.1 Knowledge acquisition method MUST	Analysis of the available literature on Semantic Web, ontologies and IoT. In particular, how to represent music and musical instruments, devices and their components. Competency questions on the relevant domain.
D.2 Source knowledge location SHOULD	Competency Questions
D.3 Content Selection SHOULD	Things, Musical Things, Smart Things, Wearable Things: devices for IoT, applied to the collaborative production of musical content.

Table 2.4 – MIRO Report [120] of the IoMusT Ontology – Part II of III

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: 21; number of object properties: 11; number of data properties: 4; 0 individuals.
E.4 Incorporation of other ontologies MUST	[`sosa, prov, music, event, timeline, foaf`]
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	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	See point E.4.
E.9 Ontology relationships MUST	11 object properties; 4 datatype properties.
E.10 Axiom pattern MUST	158 axioms included (of which 68 logical axioms, 40 declaration axioms, 12 `SubClassOf`, 6 `EquivalentClass`, 1 `DisjointClass`, 6 hidden GCI, 5 `InverseObjectProperty`, 2 `FunctionalObjectProperty`, 1 Inverse Functional, 4 Asymmetric Object Properties, 4 Irreflexive, 11 ObjectPropertyDomain and Range, 3 Functional DataProperty, 4 DP domain and range, 50 annotation assertions).
E.11 Deferencable URI OPTIONAL	It is possible to use deferencable URIs, but no assumption on this is made in the ontology.
F. Managing change	
F.1 Sustainability plan MUST	Some research projects are being prepared to leverage the ontology.
F.2 Entity deprecation strategy MUST	Deprecated classes will be labelled as obsolete with a proper annotation property.
F.3 Versioning policy MUST	The IoMusT 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.

Table 2.5 – MIRO Report [120] of the IoMusT Ontology – Part III of III

G. Quality assurance	
G.1 Testing MUST	Tests have been made by checking competency questions and formal requirements in the presentation paper.
G.2 Evaluation MUST	Metrics, and discussions over IoMust ontology evaluation have been discussed in the presentation paper.
G.3 Examples of use MUST	At the moment, only theoretical examples of usage in the presentation paper.
G.4 Institutional endorsement OPTIONAL	None.
G.5 Evidence of use MUST	The ontology is still new, but we plan to use it in forthcoming projects.

IoMust ontology in

Chapter 3

Semantic Web of Things

Previously the discussions outlined that various approaches are considered in literature to define Things [14]. With reference to the views suggested in Section 2.2 we designed the Internet of Musical Things ontology relying on IEEE's interpretation of thing definition. Such definition, by focusing on *application perspective*, implies a specific description of the various layers building the application.

For this reason we created the IoMusT ontology leveraging a layered setup (see Fig. 2.4) including sensing and actuating, location, agents as well as generic activities. However, while the single musical application was described (see Fig. 2.3), a global view of systems interacting with other systems was missing.

Such global and inclusive interpretation of IoT is the subject of this Chapter, where the transition to Semantic Web of Things will be explored, and where we suggest some ideas that are currently scheduled for the future, and some of the topics of the next Chapter.

The contents of this Chapter are inspired from the main research carried by the author of this Thesis and his colleagues over the PhD duration, and recently published[1].

3.1 W3C Web of Things vision

We already mentioned important research on the WoT [12, 13, 56]. The most complete idea and implementation guidelines of Web of Things, to the best of our knowledge, have been given over the last years by a working group created by the W3C[2].

In particular, the working group realized a set of drafts that are (as for September 2019) now Candidate Recommendations. We will here go through two of them, namely the *W3C WoT Architecture*[3] and the *W3C WoT Thing Description*[4], that over their

[1] © 2019 IEEE Reprinted, with permission, from Antoniazzi, F. & Viola, F. (2019) Building the semantic web of things through a dynamic ontology. IEEE Internet of Things Journal, early access

[2] `https://www.w3.org/WoT/WG/`

[3] `https://www.w3.org/TR/2019/CR-wot-architecture-20190516/`

[4] `https://www.w3.org/TR/2019/CR-wot-thing-description-20190516/`

evolution represented a starting point for our work.

Let us consider, first of all, the W3C WoT Architecture. The draft is organized to provide a Web of Things vision of the future connected environments: therefore a complete report of possible use cases is given. All of them, except the thing-to-thing paradigm have in common the fact that the data retrieved by sensing units is consumed far beyond the limits of the local environment. That is, we imply that there is an information step similar to the one included in a fog computing [52] environment: data is locally collected as well as locally elaborated to get some higher level information to be sent into the Internet to consumers. The usages are unnumbered, and are already part of everyone's everyday life.

The implementation patterns, in this context, are also analyzed: the thing-to-thing paradigm is observed side by side with gateway mediated approaches. Gateway solutions in some cases have also access to the Web, providing the possibility to interact from outside the system, and in some others situations are limited to an internal function of communication enablers among different technologies. Another idea included is the *digital twin*, i.e. the possibility to interact with a virtual representation of the device and not necessarily with the physical item directly. The twin concept is at the basis of the Thing Description as we intend in this Thesis (see Section 3.2.2), where a semantic responsive twin is provided through the capabilities of SEPA (see Section 1.3).

Going further, a set of requirements are given for the Web of Things based on the aforementioned technologies. For instance it is stated that the usage of the standard protocols of the Web, and the realization of virtual environments accessible through a RESTful approach is an essential core feature of the WoT. Interoperability and scalability are an additional must, although they are subject to case-by-case development. In this work, we will suggest a semantic interoperability.

Devices, indeed, must also be discoverable. At application level this implies that we should have the possibility to describe devices, and that a uniform method should be provided. Things, moreover, exchange data with each other, with humans, with the cloud, in various formats. The description for discoverability should include the formatting information formalization, as there is no way to achieve interoperability if shared information meaning is not common among the actors.

W3C changed and adjusted the Thing Description logic several times since the creation of the Working Group. Consequently in this Thesis there may be some differences with the actual candidate recommendations available (September 2019). In next Section, in any case, we will clearly specify which is the draft that has to be considered as a reference.

When considering the Thing Description explained by W3C versus the semantic solution outlined in this Thesis it is worth considering what follows. The Thing Description, as it is visible in the provided drafts, is given as a JSON-LD file containing the information formalized in a taxonomy identifying thing capabilities in a shared fashion. The goal of such taxonomy is not to grant a semantic vision of the Web of Things, but rather to assure as much as possible machine understanding of descriptions. Differently, by the means of SEPA's publish subscribe, we hereby not only give access to a common understanding of descriptions, but also the possibility to interact through that same platform, i.e., in an environment designed to be interoperable.

3.2 A Dynamic Ontology for the Semantic Web of Things

As it is reported in the previous Sections, the Web of Things 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 Web of Things and relies on a WoT implementation crafted on the technologies proposed by the Semantic Web stack. This Section presents a working implementation of the Web of Things declined in its Semantic flavour 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.

This ontology, named SWOT [**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.[133].

In addition to that, this Thesis 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.

Moreover, through the presented ontology we suggest a solution to the problem of discoverability [157] of devices. Along with the main contribution we propose also a framework, named Cocktail, which is a practical realization of both the static and dynamic ontological concepts (see Sections 3.2.3). 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) [158, 99] 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[5] 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

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

physical environment as time passes.

We 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 our work 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.

Before going in the details of the presented work, we propose a summary of the contributions achieved through our approach:

1. Representation of the W3C's Thing Description model (Charpenay et al. [133]) 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. [134], was then extended to support the Semantic Web Thing Interaction (see Section 3.2.3) in addition to discovery;

3. Concerning the last point, as shown in Sections 3.2.2 and 4.2.2, 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 see Section 4.2) 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.

Head to Fig. 3.5 for a full view of the SWOT ontology.

3.2.1 Related Work

In the past twenty years, several works have introduced and explained the Semantic Web view. Going back to 2001, Tim Berners-Lee et al. discussed in [2] the driving ideas and concepts of a still prototypical Semantic Web through some practical examples. The paper's focus was to highlight in a few examples the situations in which the currently available Web is either insufficient, or insufficiently exploited.

Following this research stream, Shadbolt et al. in [11] 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, are the reuse of available ontologies to produce data [159, 160, 161], the alignment of ontologies exposing the same concepts [162], and the effective exploration and visualization of the data graph [163, 164].

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* and the *Semantic Sensor Network* [**sosa, ssn**] ontologies. 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 Thesis 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 [165]. The *Time ontology*[6], and the *Event ontology* [**event**] have been developed to this extent, in order to categorize both the 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.

Other works, e.g. OpenIoT [166], IoT-O [167], IoT-Lite [168], 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. [169] 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. in [170]. In this work, semantics is the glue among the IoT/WoT elements and is used to annotate sensory data streams.

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

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. also proposed Agri-IoT [171] 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 re-uses a set of components of the CityPulse framework [169] 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 our work 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 [172].

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 [173]. This well known nightmare of IoT researchers is analyzed in [10], for instance, listing the causes of fragmentation of IoT (e.g., the coexistance of resource constrained and rich devices in environments). Many researches suggest the usage of a gateway to solve this problem (e.g., [174, 175]), while in [46] Zachariah et al. highlight 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. propose a semantic approach, studied developing a protocol translation gateway [47]. This idea of enhancing IoT by unification and translation at information level, rather than at lower protocols, is also followed by Gangemi et al. in [176], where they propose the IoT Application Profile (IoT-AP) ontology with the aim of representing and modelling the knowledge in the Internet of Things. In [177], as well, an ontology is suggested and associated with the tasks of discovery and dynamic composition: this work differs from ours, as the ontology there is neither designed with the purpose of context evolution, nor targets the Semantic WoT, 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, LOV (Linked Open Vocabularies) [178] and LOV4IoT [179]) and their impressive growth [180]. As an example, LOV, standing at the analysis proposed by Gyrard et al. [180], 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* [157]) is well known in research [181] and several solutions have been provided over the years. It can be also addressed through ad-hoc protocols (like the one proposed in [182], focused on privacy requirements), protocol-specific tools (e.g., CoAP-based discovery was proposed by Djamaa et al. in [183] and Viola et al. in [184], while XMPP-based solution is proposed in [185]) and gateway-based approaches [186]. Semantics in this scenario has been proposed in several research contributions [187, 188, 47, 189, 190]. Kamilaris et al. in [191] presented WOT2SE, a search engine for the Web of Things based on web crawlers that scan Linked Data endpoints. In our work 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 [192]).

On the other hand, orchestration/choreography [193] 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; 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**]. Several approaches to orchestration/choreography have been proposed over the years. Tzortzis et al. [194] present a semi-automatic approach to service composition. Viola et al. [135] propose an example of orchestration of *virtual things* applied to the Semantic Audio research area. Song et al. [195] propose a middleware based on Semantic Web techonologies 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 [196], OpenIoT [197] and IoT-A [198].

As highlighted by Barnaghi et al. [199] 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 Internet of Things, a new research area pivoting on them is born: the Semantic Web of Things (SWoT).

Unfortunately, the application of semantic technologies to the IoT is not straigthforward due to the nature of IoT requirements (e.g., constrained devices, unreliable connections) [199] and this motivates the birth of this new research area.

One of the first research works mentioning the Semantic Web of Things is the one by Pfisterer et al. in [200]. 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. focus 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. declare 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. in [201] define the Semantic Web of Things as the adoption of Semantic Web technologies in Internet of Things application. That said, the purpose of their research is rather different from the work disclosed in the present Thesis. In fact, Ruta et al. 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. 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 the architectures, devices are fully decoupled, but SEPA hosting the knowledge base allows: 1) reducing the number of accesses to devices, imporant with constrained devices or when the network is not reliable; 2) hosting the whole knowledge base in a powerful node granting faster access and inference.

As mentioned in the Introduction, the W3C founded a Working and an Interest Group dedicated to the Web of Things, whose challenges are depicted by Ragget [56]. Among the various contributions proposed by these groups, it is worth mentioning again Charpenay et al. [133]. 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 IRE (Identifier, Resource, Entity) ontological pattern, which states that Web resources may act as proxies for real world entities. From this work, 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* 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 4.2 leverages these concepts to provide a practical implementation of all the tools needed to create a full environment.

Ontologies for the so-called (Semantic) Web of Things have been proposed also by other authors. For instance, Serena et al. in [134] proposed an ontology for the discovery of devices in the Semantic Web of Things. With respect to this paper, again, our research goes beyond the pure discovery of devices, enabling the interaction through the semantic broker. The ontology by Serena et al. is also used by Noura et al. [202] that propose a framework for the goal-oriented description of web thing interactions. A framework for semantic interoperability in the Web of Things is presented also in [203] which combines an extension of the SSN ontology and machine learning techniques. No detail are provided regarding the way subscriptions can be defined.

3.2.2 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 [204], of course, but also research on wearable IoT for everyday life [205], [206] and music [207] 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 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 3.1. As outlined in the Introduction, this research exploits the SEPA subscription mechanism (whose description is out of the scope of this Thesis). As opposed to MQTT protocol[7], 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.

Listing 3.1 – SPARQL subscribe to list all Web Things available in the RDF store.

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

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 3.2.3, the *Interaction Patterns* 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 4.2, 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 Web of Things ecosystem would rely on HTTP(S) addresses over TCP/IP as URIs.

As already proposed by Guinard and Trifa [157], Web Things can be declared to act

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

as proxies for other Web Things. This may be useful in case of constrained devices (see [35] 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 3.1. 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 4.2.2.

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 interaction patterns: 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. 3.1, where the colour code is defined and used in Protégé [208]. In the red box, that is only for suggestion and does not belong to the ontology presented in this work, we added a few straight-forward connections to other ontologies, like [**`prov, dul, foaf`**], proving that SWOT ontology integration with other ontologies is possible as well as its usage with DBpedia resources.

3.2.3 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 [209].

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 work borrows some concepts from other works, and extends them with an original contribution. For instance, Charpenay et al. in [133], on behalf of the considerable work of W3C Interest and Working group, introduced the thing description object, while

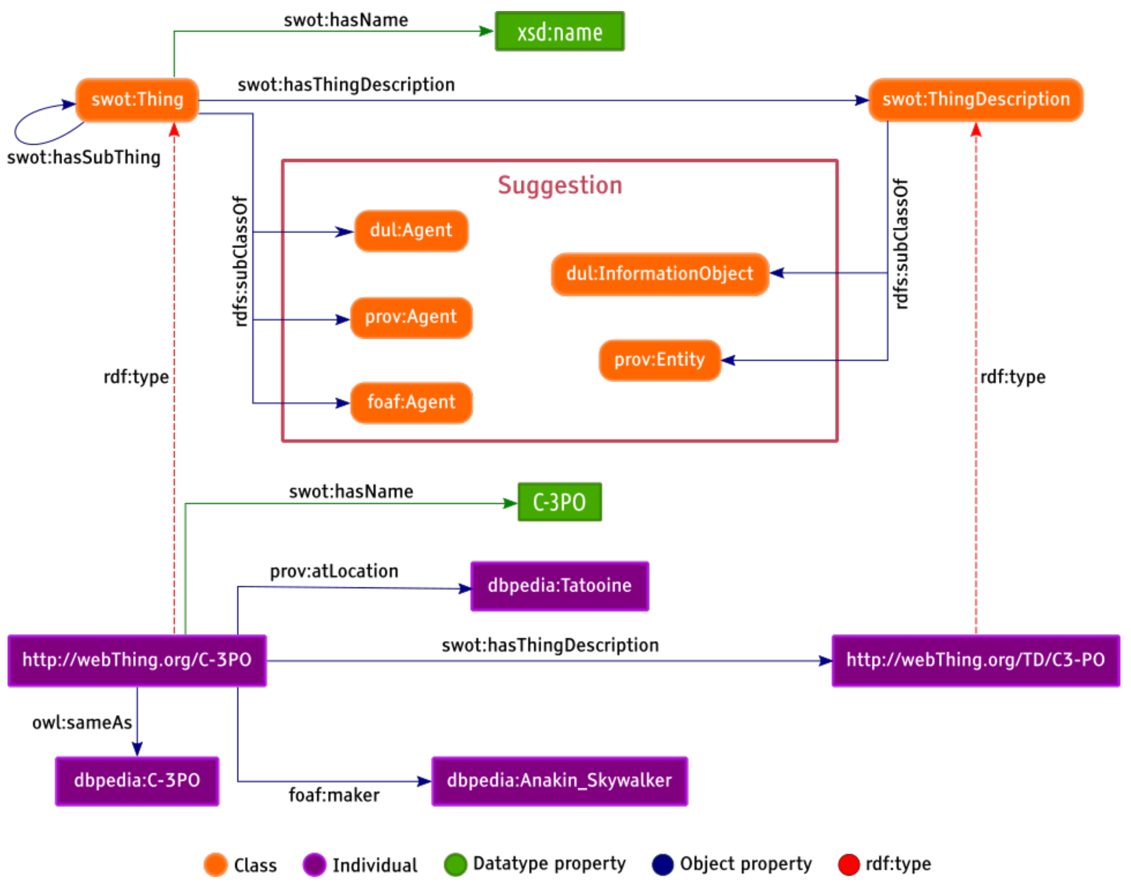


Figure 3.1 – `swot:Thing` and `swot:ThingDescription` partial ontology and a practical example of instances and some suggestions of extensibility with other ontologies.

Serena et al.[134] defined the concepts of Property, Action and Event which in this Thesis we call the PAE paradigm. On the 5th April 2018 the W3C released the Thing Description Draft[8], that leverages the two works aforementioned. Our research takes its origins in such draft.

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:

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

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;

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 the present Thesis these three entities are represented as the classes `swot:Property`, `swot:Action`, `swot:Event`, which are all subclasses of the *interaction pattern* concept `swot:InteractionPattern`. In Fig. 3.2 the ontology subgraph for the interaction pattern (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 3.2.4), 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 (i) an Action exists able to allow such software modification, or (ii) 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 Subsection pointing to the `swot:Instance` concept (see Fig. 3.3). 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).

Dynamic interaction: interaction pattern instances

The core discussion of this Subsection 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 here. 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

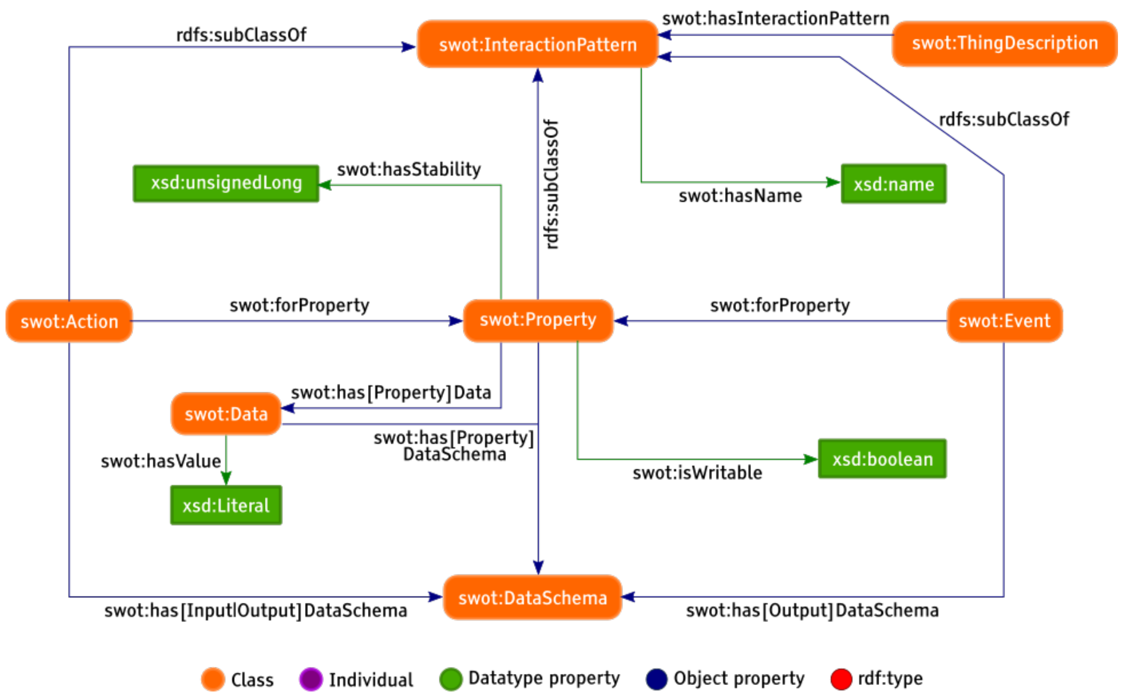


Figure 3.2 – `swot:InteractionPattern` subset of the ontology. Actions, events and properties are interaction patterns, receiving inputs and giving outputs according to a data schema (see Section 3.2.4)

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. [158]. SEPA has been designed to be communication-protocol agnostic. In this Thesis 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 [184].

Once the Semantic Web Thing Description has been given, we expect at run time two possible situations: (i) the request for the execution of an Action, which we call ActionInstance, and define through the `swot:ActionInstance` class; (ii) an Event notification, which we call EventInstance, and define in the `swot:EventInstance` class. They both are subclasses of `swot:Instance`.

See Fig. 3.3 for the subset of ontology related to dynamic evolution of instances.

Let us first consider an Event instance. In Table 3.1 the triples that a Semantic

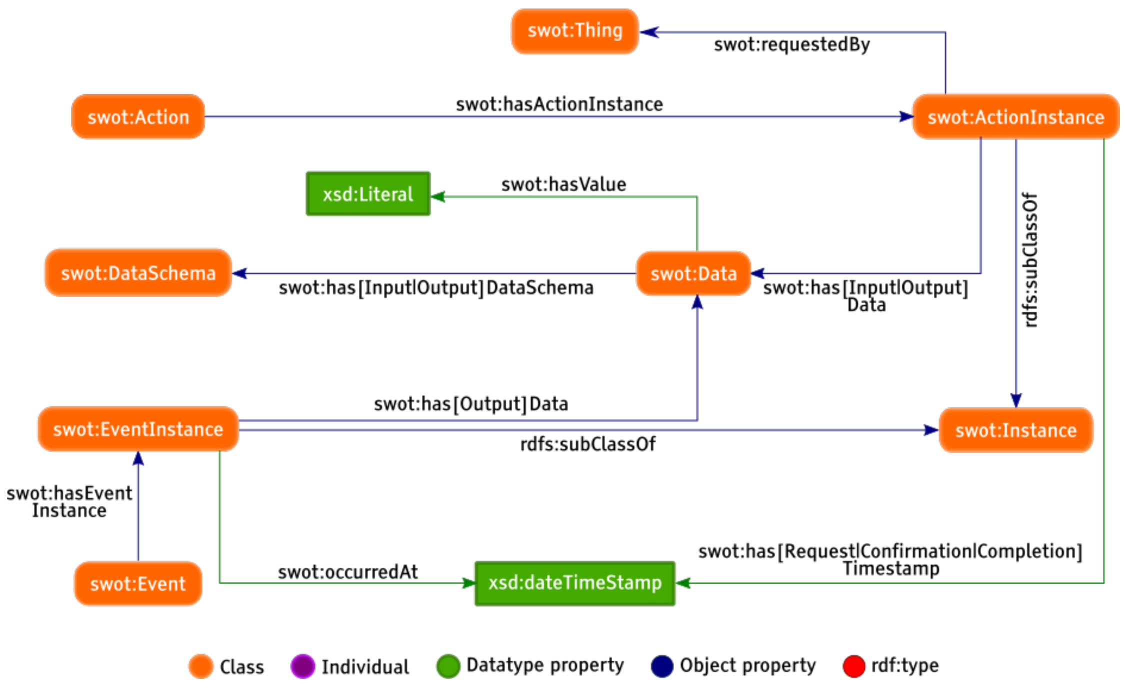


Figure 3.3 – `swot:Instance` subset of the ontology, i.e., how the subgraph for an action request must be formatted, as well as how an Event notification is thrown.

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 Web of Things that requires the publish-subscribe mechanism.

In Table 3.2, on the other hand, are shown the triples that concern the instance type `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's 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

Table 3.1 – `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 3.2.4, while the triples needed for the definition of `swot:MyStringDataSchema` are available in the highlighted rows of Table 3.3.

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 3.2 – `swot:ActionInstance` triples to be inserted in the RDF store to trigger the action performace. Notice that the concept of data schema is explained in Section 3.2.4, while the triples needed for the definition of `swot:MyStringDataSchema` are available in the highlighted rows of Table 3.3.

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`

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.

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 InteractionPattern 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 3.2.4.

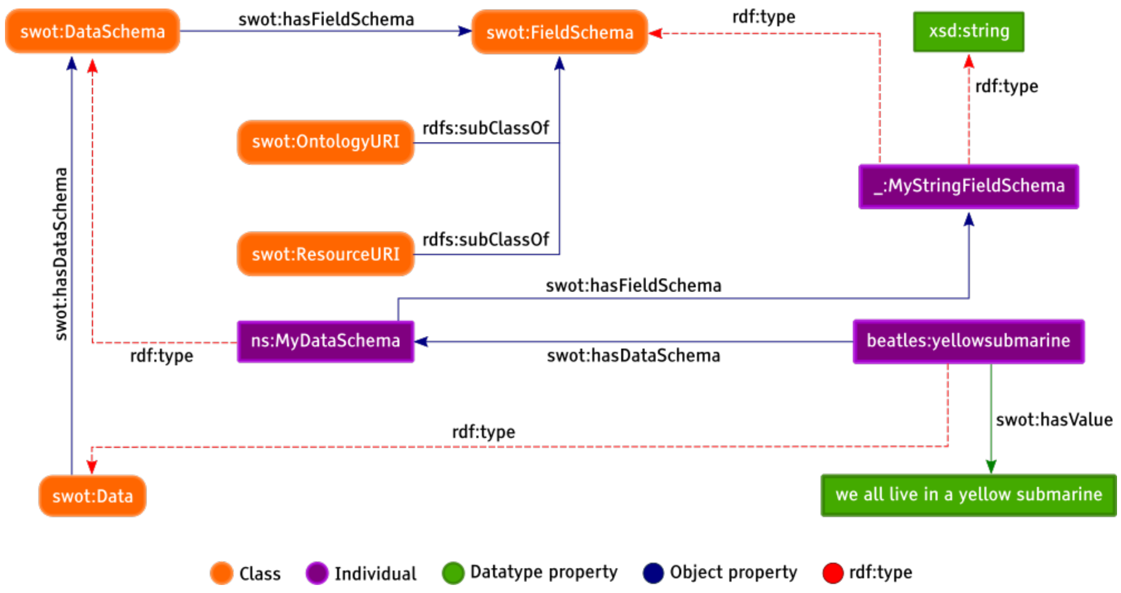


Figure 3.4 – `swot:DataSchema` and `swot:FieldSchema` ontology subgraph, together with an example of a simple `xsd:string` datas chema inclusion.

3.2.4 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 enviroment. 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. 3.4 are shown the relationships between these two classes and the ones introduced in the previous sections, as well as a simple example.

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[9], like `xsd:integer`, `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;
4. The data exchanged is a Semantic graph formatted according to a specific ontology.

The following Subsections will demonstrate their usage giving practical examples.

Basic and Complex datatype

In this Section the complex and basic datatypes are examined. Tables 3.3 and 3.4 are provided to exemplify the meaning and the usage of `swot:DataSchema` and `swot:Field-Schema` classes.

In particular, Table 3.3 (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 towards the integer. In that case, there would be no difference with the previous situation except that, as in Table 3.3 (second example), the grey-coloured entries are included.

In a smarter Web Thing, the same action might be able to read also an `xsd:string`, and parse it independently towards the integer. In that case, there would be no difference with the previous situation except that, as in Table 3.3 (second example), the grey-coloured entries are included.

The complex datatype is slightly more challenging and is represented in Table 3.4. The Table contains the triples necessary for an action that requires an `xsd:string` as input, 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.

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

Table 3.3 – Examples of basic datatypes. White lines refer to the first example. Grey lines have to be added to realize the second example discussed in Section 3.2.4 (Basic and Complex datatype).

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 3.4 – Complex datatype triple description example. Notice that `ns:MyStringDataSchema` definition is not included in the Table, as it is already available in Table 3.3, grey-coloured 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

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.

Secondly, having a closer look to the Tables, we can outline the usage of the entity `swot:FieldSchema`. 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 interaction pattern: 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 3.4, we intend the resource to answer to an `HTTP GET` with a JSON Schema according to Listing 3.2.

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

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 ouput), or all of them are required (if it is an input).

Listing 3.2 – JSON Schema expected in response to an `HTTP GET` to `ns:MyJSONFieldSchema` as in Table 3.4 referring to the complex datatype example in Section 3.2.4.

```
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
}
```

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 Subsection as an `xsd:Literal`, a more complex situation is here considered of actions, events and properties dealing with generic resources located

Table 3.5 – 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

on the Web.

The most important point, when the semantic Web Thing is parametrized 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 3.5, 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: (i) the resource is downloaded and used (ii) the resource is not directly downloadable (i.e., a database access point, a streaming resource). Cocktail framework (see Section 4.2) 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 3.6 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 towards 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.

Table 3.6 – 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 3.3.

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

Table 3.7 – 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

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 3.7, 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 grey lines in Table 3.7 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 3.7, and an action whose task is to perform a SPARQL query like:

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

Let's 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 3.2.4 (Web Resource datatype), it will be possible to give as a parameter also the `graph_resource` variable.

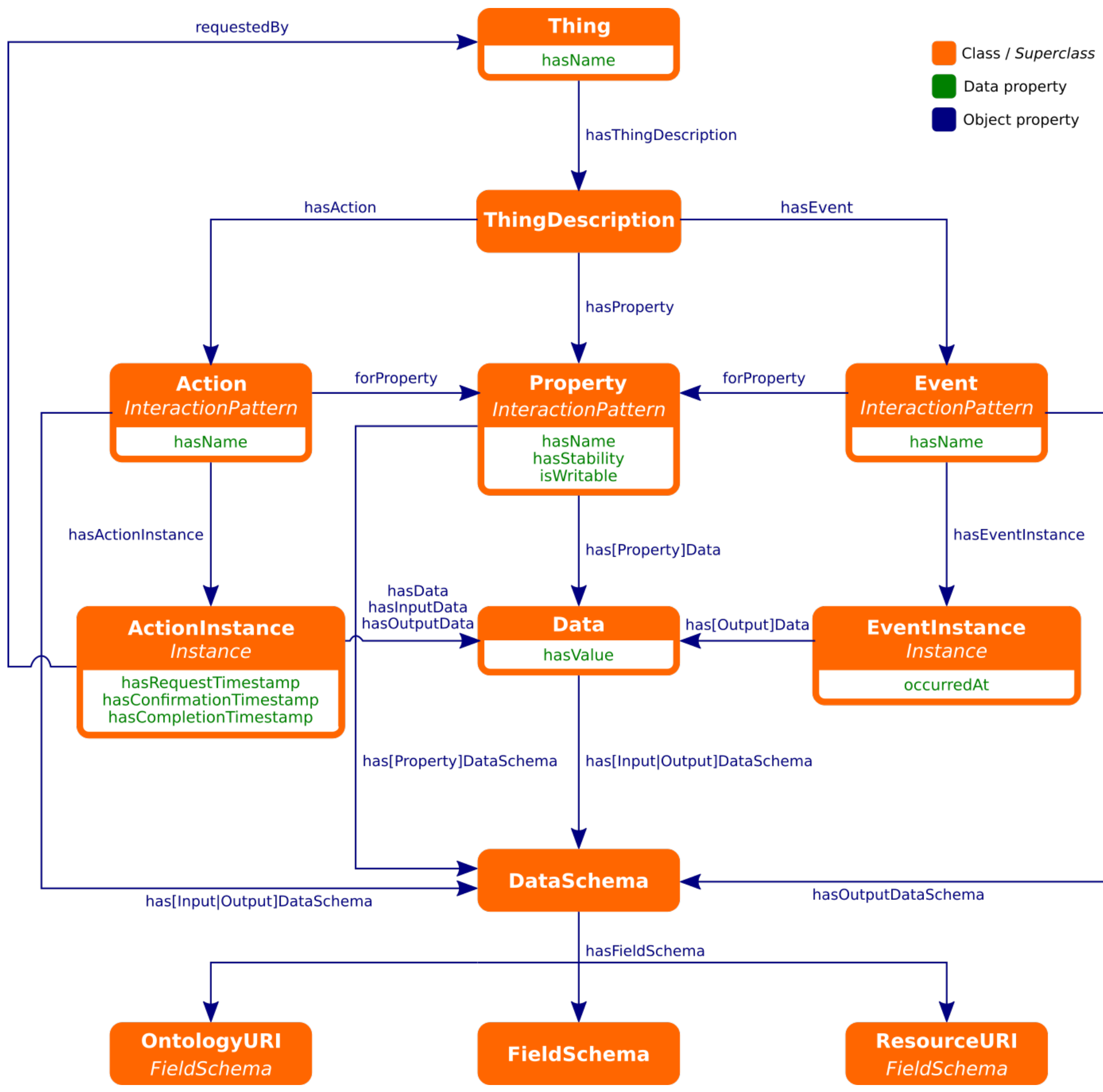


Figure 3.5 – Full view of the SWoT ontology. `swot:` prefix is omitted from the items shown.

Chapter 4

Semantic Driven Agent Programming

This Chapter reports the current status of a research that is still ongoing. Starting from the Semantic Web of Things Ontology presented in the previous Chapter, we hereby proceed studying how it would be possible to apply this semantic approach to address real problems.

To do so, Section 4.1 provides some references to the origins of this research and to other relevant works on the subject. Section 4.2, then, develops a practical implementation for the ontological work available in Section 3.2. The *Cocktail* framework is presented, described in detail and realized. Eventually, in Section 4.3 various examples inspired by real world needs are given. Once analyzed, we show our possible solutions with increasing complexity, by exploiting Cocktail and the Semantic Web of Things ontology alongside with other relevant methodologies.

The Chapter takes inspiration from the works published and submitted during the PhD[1].

4.1 Related Works

Agent-oriented programming idea was originally introduced by Yoav Shoham in 1993 [1]. Within his exceptional work an extended view is proposed over object-oriented programming: agent-oriented programming interprets systems as a continuous collaboration and competition of computational entities. Such entities, i.e. the agents, are modeled as the result of complex mutual influences of *beliefs*, *decisions*, *capabilities* and *obligations*. Along with this work, similar theories can also be listed, like the Belief-Desire-Intention (BDI) by Georgeff et al. [210].

The belief, first of all, is related with the knowledge of the agent's relevant sur-

[1] © 2019 IEEE Reprinted, with permission, from Antoniazzi, F. & Viola, F. (2019) Building the semantic web of things through a dynamic ontology. IEEE Internet of Things Journal, early access

rounding environment and entities. In literature this facet was later on intended as the context, and was declined in works focusing on the so called context-aware computing [211, 212]. Abowd et al. suggested in [213] a definition of context that is still up-to-date:

> Context is 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.

From this definition on, a plethora of works started to explore the subject. Just to name a few, Kofod-Petersen et al. [214] used activity theory to connect contexts to task identification; Sezer et al. [215] surveyed the relationships between context awareness, Internet of Things and Big Data. Baldauf et al. [216], among many others [217, 218, 219, 220], outlined the relevance of Semantic Web in this discussion introducing ontologies to realize applications.

Decisions and capabilities, secondly, define the agent's potential influence over the context. The set of capabilities, on one hand, should contain description of what agents can do, regardless their local status. This is the reason why, recently, the concept of capability was declined in works on device discovery mechanisms both in academy (e.g., [221, 222]) and in industry (e.g. [223], and the already cited work within W3C WoT Working and Interest groups).

Decisions, on the other hand, would outline the connection between the current context and the most relevant and applicable capability available, to maximize compliance with obligations. To achieve these results, indeed, current research leverages various Artificial Intelligence (AI) techniques that span from classic AI to machine learning [224, 225], as well as, in our case, to the usage of semantic information [226, 227].

Lastly, obligations: they are the constraints that agents must respect. Usually, the first idea of "obligation" presented to students is given by I. Asimov's Three Laws of Robotics included in the well-known novel *I, Robot*. Although they have been source of prolific discussion (e.g., [228]) over the last decades because of their visionary relevance, it is clear that with the advent of IoT systems the concept of obligation was slightly redirected on a different level. Obligations can be related to constrained device environments [35], minimize power, maximize security and privacy, and so on [229], or related to the application business logic.

Both the works by Shoham and Georgeff were made before the pervasive electronics and connected IoT real revolution. Over the time, the exponential increase in the number of devices reached a bottleneck in system integration and in capability of interoperate [10, 230]. The context awareness, as discussed, would move to a context of contexts, to create systems of systems into an "Agents of Things" environment [231]. Consider also, in this field, the survey by Kotseruba et al. [232].

The Semantic Web revealed itself to be part of this discussion according to the view summarized in the previous Sections of this Thesis, and in particular our dynamic approach and exploitation of graph knowledge bases. With reference to the BDI model, the SWOT ontology will hereby be our tool to allow intelligent behavior of agents, through SEPA mediated semantics.

4.2 SWOT agents framework and Evaluation

This Section contains the discussion over the SWOT ontology and its conceptualization into the Cocktail framework. To do so, the first Subsection 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.

4.2.1 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 Semantic Web of Things 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 Semantic Web of Things over the SWOT ontology.

To do so, the SEPA implementation available on Github[2] will be used together with the baseline APIs developed for it[3]. On top of them the Semantic WoT SEPA APIs are built as a complete framework named *Cocktail*. The Cocktail framework is also freely available on Github[4] 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 interaction patterns needed, i.e., actions, events, properties, with their friendly names, URIs, data as described in Section 3.2.3;

3. Define, if needed, new data schemas;

4. Query the SWTE for things, interaction patterns, 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.

[2] `https://github.com/arces-wot/SEPA`
[3] `https://github.com/arces-wot/SEPA-python3-APIs` (branch `dev-0.9.5`)
[4] `https://github.com/fr4ncidir/SemanticWoT`

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 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 interaction patterns 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: the SPARQL code, that interacts with triples in the knowledge base; and 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.

4.2.2 Cocktail: in-use analysis

Cocktail's collection of SPARQL updates, queries and subscriptions on top of SEPA proves that a Semantic WoT 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 evaluation of the SEPA and its processing units is not within this paper's scope.

Let's start by analyzing the usage of the framework. To do so, the following small SWTE composed by 3 semantic Web Things (depicted in Fig. 4.1) and a few additional triples has been developed:

`ns:Thermostat` is a smart thermostat also declared as a `sosa:Sensor` observing the special resource `ns:Temperature` (2 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 4.1. If there are available actions of this kind, and the temperature $t \notin [T_{low}, T_{high}]$, the thermostat triggers *ThresholdAction*. Every 5 seconds, also, the thermostat throws a *TemperatureEvent* with the current temperature value (data schema λ).

`ns:HotCold` is a smart air conditioner and heater. It is also typed as `sosa:Actuator` acting on resource `ns:Temperature` (2 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 4.2, searching for `ns:Temperature` sensors and subscribing to their temperature-change event formatted as λ data schema.

`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` (2 additional triples). It's worth saying that this is a dummy device that proves the effectiveness of smart discoveries: Listing 4.1 discovery will not match with `ns:Clock`, because it is not a `sosa:Actuator`, and Listing 4.2 because there is no corresponding Event.

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` (2 additional triples).

Listing 4.1 – Smart discovery for Web Things with an Action acting on temperature and requiring a ψ-formatted input

```
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 4.2 – Smart discovery for Web Things that are `ns:Temperature` sensors triggering Events with λ-formatted output

```
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>
}
```

As stated in Section 3.2.4, all the dataschemas ψ, λ, ξ are considered to be already available in the SWTE. Given that, if we imagine that temperature is $t = 3°C$ and $T_{low} = 18°C$, what happens in this environment is that:

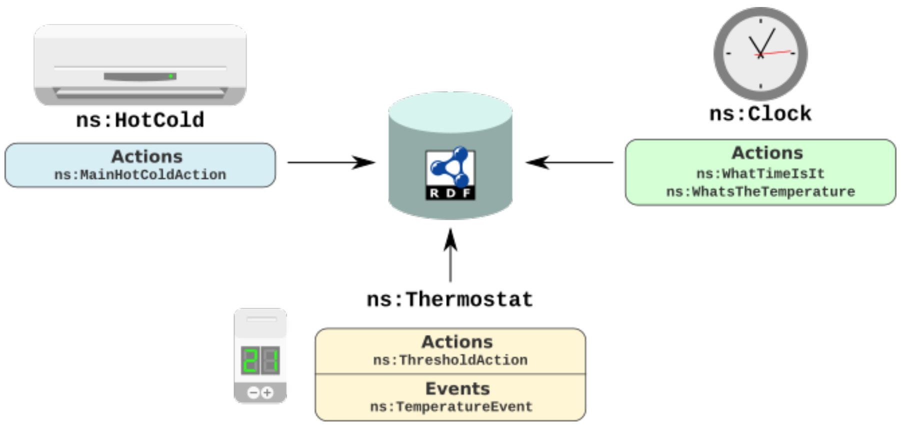


Figure 4.1 – An example of Semantic Web Thing Environment

- 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 $T_{low} \leq t \leq T_{high}$;
- 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 behaviours, 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 following:

1. Identification and description of interaction patterns;
2. Posting to the SWTE the triples;

3. Definition of actions' and events' behaviour;

4. Device loop: business logic, and events' throwing;

5. *Optional* small Web server dedicated to JSON-LD thing description.

4.2.3 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 work, 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 paper this factor, that has strong relevance in the realization of the whole application, will be taken into account.

Fernández et al. [151] defined a set of 12 metrics to measure the quality of an ontology. In the current paper those metrics are partially adopted and partially re-arranged 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:

- **Number of classes**: the number of classes in the SWOT ontology. The overall value, for the ontology, is 14: among these, 10 are used for the static SWTE description, and the remaining 4 have a dynamic role;

- **Number of properties**: number of (datatype or object) properties in the ontology. The datatype properties are overall 9, 4 of which allow the static description, and 5 the dynamic one. The object properties, instead, are 9 static, 7 dynamic and 4 belonging to both the sets, 20 in total. Eventually, 20 more (inverse) properties can be added. Being redundant elements, they will not be considered from now on.

- **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's consider the same example of Subsection 4.2.2, considering Web Things *as is*, out of the general application logic:

`ns:Thermostat` requires 6 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 towards the thermostat Action, and one is required for the smart discovery mechanism, as already described in the previous Subsection.

`ns:HotCold` similarly requires 5 updates and 3 subscriptions.

`ns:Clock` requires 4 updates and 2 subscriptions.

In the end, this 3-Web Thing SWTE requires 15 updates (some of which are performed into a loop according to the application logic), and to keep 7 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 interaction patterns; and with S subscriptions, whose minimum number increases also linearly on interaction patterns, 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.

- **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 interaction patterns that the Web Thing has to implement. However, by parsing the SPARQL updates we get a total of

  $$T = 4 + 9A_{io} + 7(A_i + A_o + E_o) + 5(A_e + E_e) + 13P_v + 12P_e + f + C \qquad (4.1)$$

  where 4 triples are dedicated to `swot:Thing` and `swot:ThingDescription`, 9 to a number A_{io} of input-output Actions, 7 to input, output Actions and output Events, 5 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 4 triples is the absolute minimum reachable for a special Web Thing without Interaction Patterns, and that Properties are the pattern that requires the greatest number of triples, due to the fact that they store also data.

- **Triple count for interactions and DvS ratio**: how many triples are inserted when a new action request is made, or when a new event notification is triggered? Similarly to Equation 4.1, it is possible to obtain the number of triples required for the dynamic control of event and action instances. Refer to Fig. 4.2 to observe in each situation what are the requirements.

  Fig. 4.2 also introduces the *dynamic vs static* (DvS) ratio (i.e., the triple count for static description over the triple count for dynamic interaction). DvS depends on the kind of interaction pattern 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 3.2.3 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.

- **Data format influence on triple dimension**; as explained in Section 3.2.4 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 3.1 and 3.2, 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[5] and OOPS![6].

[5] `http://perfectsemanticweb.appspot.com/?p=ontologyValidation`
[6] `http://oops.linkeddata.es/`

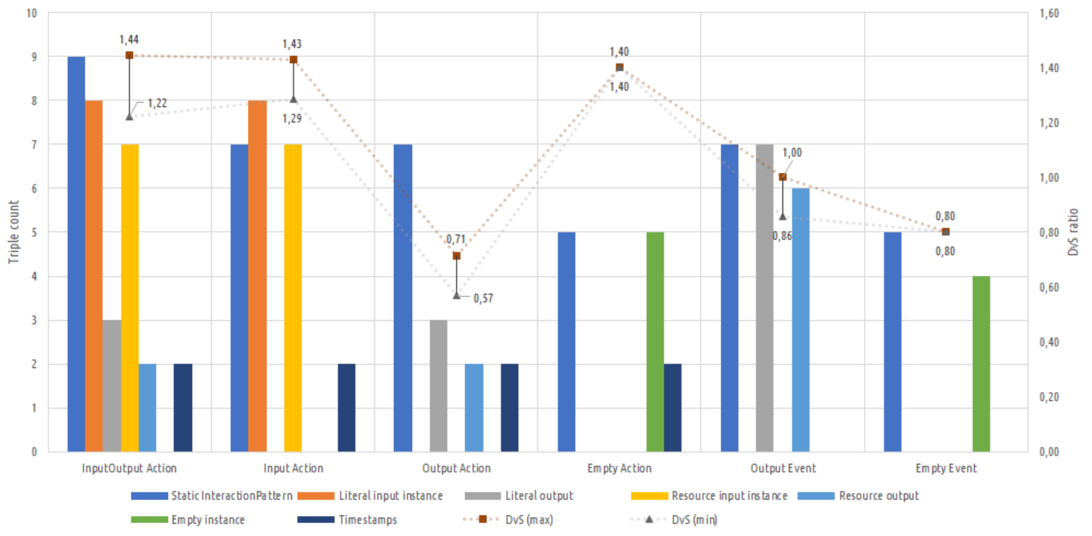


Figure 4.2 – Number of triples for every interaction pattern, and their DvS ratio value maximum and minimum.

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 our work through a set of criteria globally accepted by the research community. As an example, the guidelines for submission to the well-known ISWC Conference[7] are very helpful. Most of the suggested points were largely covered in the previous Sections and/or in Tables 4.1, 4.2, 4.3. 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 licensed, but, as a work in progress, still not submitted to community registries like LOV.

4.3 Next Steps

This Section will address the possibilities that Cocktail offers when the Semantic WoT Environment (see SWTE definition in Section 3.2.2) created is intended to work in collaboration with some AI mechanisms. A general idea has been already given in Section 4.2.2, although we will here discuss in more detail the opportunities and the setups.

[7] http://iswc2018.semanticweb.org/call-for-resources-track-papers/#

In addition to what has been introduced in the previous paragraphs, we will also refer to the ideas expressed in some research papers that have been published [204, 233, 234] over the time at (or with the collaboration of) ARCES lab.

While reading this Section please consider that some of the suggestions given are either pending research (as of October 2019), either planned for future studies and implementations. Although not yet complete, they deserve to be mentioned anyway because they participated as a global frame to the creation of this Thesis work.

4.3.1 Cocktail example

Let us now consider the setup described to explain Cocktail available in Fig. 4.1 and fully discussed in Section 4.2.2. As we said, by the means provided by SEPA our small application was using the SPARQL subscriptions to dynamically discover the environment and to call the right `swot:Action` at the right time, as well as to stop it when there was no more need. In order to autonomously respond to context variations, in particular, the heating unit needed to be informed of the availability of a thermostat capable of formatting its output in a convenient manner. *Vice versa*, the thermostat needed to be aware of heating units close by (see Listings 4.1 and 4.2).

In our case, additionally, the thermostat had to be aware of the requirements coming from the user in terms of expected temperature in the room. Even though that assumption might appear reasonable here, it is worth mentioning that in general this is not the case. It may not always be possible to have such a *Master WebThing*, which we define here as a device containing all the preferences of the user regarding the politics of context control within his/her own domain.

An IoT environment built up on a middleware setup is probably the easiest way to create the Master WebThing abstraction: all information passes through the central unit, and therefore can be forwarded to the context master for AI elaboration. Clearly, in this description, Cocktail framework and SEPA together offer this capability providing to the Master WebThing the same abstraction as an ordinary WebThing, including a discovery mechanism that is as flexible as SPARQL, as dynamic as publish-subscribe.

The main limit of the example is the way in which the user preference expression $T_{low} \leq t \leq T_{high}$ is given. As it is now, we have that the preference is hardcoded in the thermostat business logic. Even if Cocktail allows quite easily to add a new Action that modifies the two thresholds upon user request, this solution implies that every other device implementing this rule should be visited and updated in the same way.

A possible alternative, that is currently a preliminary study work still pending implementation is shown in Fig. 4.3 and refers to the Master WebThing concept. First of all, we consider here that the added `ns:Temperature` resource might be in fact a separate RDF subgraph within SEPA. The temperature graph is required to follow the data schema/field schema concept (see Section 3.2.4), being formatted in its own specific way. Consequently, any device updating the temperature would make a SPARQL Update similar to the one in Listing 4.3.

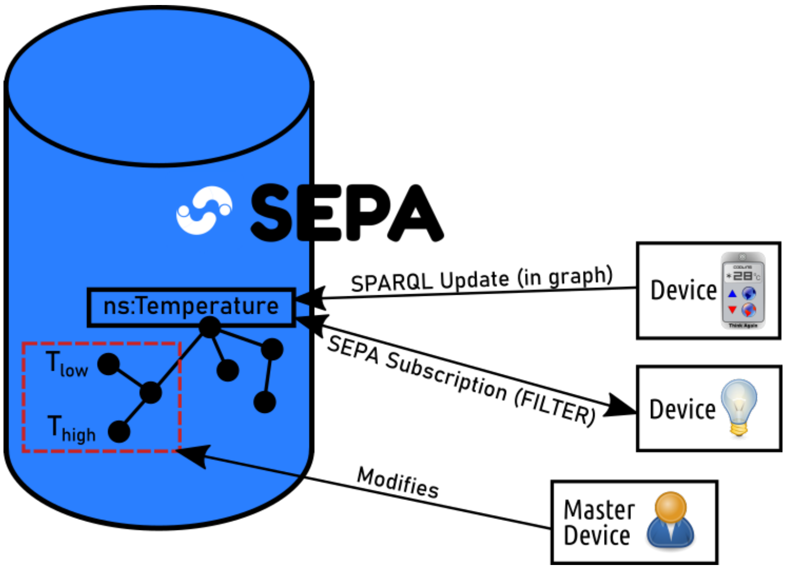


Figure 4.3 – Suggestion for including in Cocktail the concept of *Master Device*, i.e. a device dealing with the user's preferences on context control.

Listing 4.3 – SPARQL Update temperature subgraph. Refer to the previous Sections for any unexplained resource. Notice that this is currently (October 2019) an ongoing work and therefore it is not to be considered a final solution.

```
WITH ns:Temperature
DELETE {ns:RoomTemperature swot:hasValue ?v}
INSERT {ns:RoomTemperature swot:hasValue "18.4"}
WHERE {
    ns:RoomTemperature a swot:Data;
            swot:hasDataSchema ns:MyDoubleDataSchema.
}
```

The Master Device on the other hand would take care of the user preferences by modifying the appropriate part of the `ns:Temperature` subgraph. In our preliminary research we suggest to introduce a new `swot:Preference` resource type as shown in Listing 4.4, defining for the current example the lower and upper boundaries of acceptable temperatures.

Eventually, any device acting due to some temperature specific behaviors would subscribe to that same subgraph, filtering the results according to its own business logic as reported in Listing 4.5. In this way, each time the temperature falls out of the interval, SEPA triggers the relevant subscriptions based on that resource.

What is still missing in this approach is a complete theory concerning the relationships between different preferences. Are there preferences that are more or less important than others? What would be the effect of this preference ordering in the knowledge base? These are the main future directions for this research.

Listing 4.4 – SPARQL Update to temperature preferences. Notice that this is currently (October 2019) an ongoing work and therefore it is not to be considered a final solution.

```
WITH ns:Temperature
DELETE {
   ns:user_preference ns:lower ?low;
    ns:upper ?high.}
INSERT {
   ns:user_preference ns:lower "18.2";
    ns:upper "28.0".}
WHERE {
   ns:user_preference a swot:Preference.
}
```

Listing 4.5 – Suggested example for SPARQL device FILTER subscription. Notice that this is currently (October 2019) an ongoing work and therefore it is not to be considered a final solution.

```
SELECT ?current
FROM ns:Temperature
WHERE {
   ns:user_preference a swot:Preference;
    ns:lower ?low;
    ns:upper ?high.
   ns:RoomTemperature swot:hasValue ?current.
   FILTER(?current <= ?high)
   FILTER(?current >= ?low)
}
```

Summarizing:

1. Nothing changes in the PAE interaction paradigm;
2. Target entities like `ns:Temperature` would be represented as `swot:Data` within a subgraph;
3. A Master WebThing, described in the same way as other WebThings, would be able to control the user's preferences `swot:Preference` and modify the target entities accordingly;
4. The devices using the target entities would apply their (autonomous behaviour) business logic based on `FILTER`ed subscriptions related to `swot:Preference` instances.

Notice that we have formalized here a *Semantic Belief-Desire-Intention behaviour* [235]: the beliefs are the aforementioned subgraphs like `ns:Temperature`; the desires are the `swot:Preference` definitions; the intentions are the notification triggers upon `FILTER`ed subscription.

4.3.2 Habitat project example

The Habitat project acted an essential role in the development of the Semantic Web of Things as intended in this Thesis. Its main contribution, as it will be clear in the next paragraphs, is related to the flexibility in thing description and easy extensibility required to the Habitat IoT-WoT environment. They both had relevant effects in the actual Cocktail implementation.

During the project the author and his colleagues were involved in the creation of an indoor localization IoT environment with the goal of monitoring people with mental impairments. The importance of such systems can be exemplified by a typical use case connected to one of the greatest problems that modern societies are facing, namely the care for people suffering Alzheimer's disease.

Along with the cognitive decay due to the progression of the disease, it is indeed important to allow the patients to stay safely independent as much as possible so that no stress or technical malfunction could have negative effects on their everyday life. Keeping this in mind, the indoor monitoring provided by Habitat would act as a non invasive control of danger situations, like being close to the stairs, or the patient being almost out of the house without the caregiver supervision.

From a broader point of view indoor monitoring represents a functionality that has various possible applications. Given the versatility of the information retrieved, we have that smartphones apps, social networks, but also police and many others use such information to be more efficient and effective in their jobs of advertising, catching illegal activities and so on.

This is one of the main reasons why the a Semantic approach was preferred within Habitat to control the information flow. The variety of possible usages that can be done of localization information requires a shared and flexible solution for data description. As a consequence, Habitat project outlined on one hand the lower level setup (that are out of the topic of this thesis), i.e. (i) raw data measurements with radio frequency; (ii) fog computing approach retrieving medium-level information, like (x, y, z) coordinates.

On the other hand, it is important to mention that, on top of the information interoperability, a great achievement was the successful usage of a prototype of Cocktail over SEPA, realizing (iii) information aggregation; (iv) the handling of danger situations by calling the appropriate actuators.

Let us consider point (iv), that was largely addressed in the Sections describing and evaluating Cocktail. Within Habitat we considered two separate possible approaches:

- If-This-Then-That (IFTTT): the business logic of the IoT environment is realized performing the requests as a set of actions following predetermined conditions.
- Rule engines, like Drools [236]: SEPA notifications, in this case, were transformed into events triggering rule evaluation in drools, and resulting eventually in decisions over the environment state.

While Habitat project reached its end, there is still the possibility to proceed with some further research in the future. For instance, as the SEPA middleware is common to all these implementations and Cocktail, a good plan would be to study the integration of the two aforementioned techniques within the `swot:Preference` concept introduced in the previous Section.

4.3.3 Future directions

This short subsection will list a few possible ideas that could represent some future research directions connected to Cocktail, SEPA, and the applications that were mentioned in the Thesis.

Indeed, over this Thesis presentation SEPA was largely used and represented an essential tool. The following future directions can be therefore outlined concerning this architecture:

1. Study and enhance SEPA's performances. SEPA suffers of quick degradation of performances as the number and complexity of subscriptions grows. Not to mention, the number of triples contained in the knowledge base. This may be a good opportunity to study new algorithms and solutions;

2. Study the feasibility and the methodology to realize a distributed SEPA, as a single endpoint would clearly not be enough to support the Big Data revolution;

3. RDF knowledge bases can be exploited with reasoning techniques. Their effect is the modification of the knowledge base according to some rules that could have a huge impact on the whole architecture. Consider, for instance, the situation in which a triple `?subject-?predicate-?object` is enriched by a reasoner of the *inverse* predicate.

   A client could remove that inverse triple, because of its own application logic.

   The rule effect, however, once aware that the triple is missing, would be to insert it back again and again: how would complex and unexpected behaviors like these interact with the publish-subscribe and with large applications like the SWoT and the IoMusT? The SEPA architecture has not yet been tested with reasoning, and therefore it could be interesting to see how they behave when used together in an application.

Concerning Cocktail and the Semantic Web of Things, there is a need of research that include the implementation of new applications with this framework. A positive feedback of this process, moreover, would be the enhancement of the ontology itself whenever the users come across limits in the definitions or in usage. The framework, also, might be tested as a tool to realize the aforementioned Internet of Musical Things in a WoT flavour, as well as a benchmarking indicator for SEPA performances. Additionally, further research direction related to the semantic knowledge contained in the SWoT would be to extend the semantic context by using for instance DBpedia, and study how this can impact intelligent behaviours within the applications.

Table 4.1 – MIRO Report [120] of the SWOT Ontology – Part I of III

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 [134]
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.

Table 4.2 – MIRO Report [120] of the SWOT Ontology – Part II of III

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 4.1), *DvS* contents ratio, Data format impact;
E.4 Incorporation of other ontologies MUST	The ontology is stand-alone. Examples are given on how [`dul, prov, sosa`] can be included. See Table 4.4.
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`] in Fig. 2.4.
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.

Table 4.3 – MIRO Report [120] of the SWOT Ontology – Part III of III

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 `unittest`s available in the repository, allowing to check their consistency by direct usage.
G.2 Evaluation MUST	Some of the metrics reported in [151] 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 [135]. 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 [135] and [204].

SWOT ontology, and Cocktail framework in

Conclusion

Across the Chapters of this Thesis a variety of topics were discussed. First of all, we introduced the great Internet of Things revolution, that created in the last twenty years an outstanding opportunity for industry and academy to reach exceptional results. The IoT is now part of everyday life, and its effects on society can only increase in the next years.

A large discussion was then presented reporting the greatest drawback of IoT, i.e., its fragmentation. Also known as "vertical silos" fragmentation, this IoT design nightmare is substantially hindering the development of integrated systems due to the difficulty of sharing resource and data among entities that were developed at different time, with different goals, and by different programmers. The various abstraction layers that compose IoT applications need a common way to interact safely and effectively.

The term *interaction*, in this Thesis, is usually intended as *sharing information*. The Semantic Web stack was designed to provide the suitable tools capable of realizing this shared data community. That is, systems would use and design internally their own proprietary data representation, but still agree on a common way to intend resources and relationships on a higher interactive level. Consequently, the Semantic Web could represent a powerful tool to fight IoT fragmentation.

Keeping this idea in mind, we presented some applications related to the Semantic Web: a study on how to visualize it, for diagnostic and didactic purposes; an integration of multiple information sources, within the AudioCommons European project; the SPARQL Event Protocol Architecture, that is the core of the work pursued in the PhD.

The effort of creating a common playground for a fully interoperable IoT through Semantic technologies and protocols led us also to the design and the evaluation of the Internet of Musical Things Ontology. This is a major contribution of the Thesis, as we provide a large description of a shared vocabulary in an innovative and previously mostly unexplored field of the IoT. For the future, we expect to further extend step-by-step the ontology to integrate further IoT application domains in the same description.

Eventually, the Thesis outlines the possibility to realize an implementation of the Web of Things in a semantic fashion. By exploiting the SEPA, in particular, the work here presented shows that the semantic representation of information is capable to keep up with the dynamic evolution of an IoT environment. Consequently, a study is provided on how the Things and their active and passive interactions could be ontologically represented.

We designed and evaluated, therefore, the Semantic Web of Things Ontology and a framework that realizes the interaction mechanisms over our SEPA architecture. Both the framework and the ontology are an important contribution within the Thesis, and are freely available on Github. This work also led to a publication on the Journal with the highest Impact Factor in the IoT research field (IEEE IoT Journal). Their usage is expected to provide interesting results in future research, for instance by extending the Internet of Musical Things towards the Semantic Web Of Musical Things, as well as enhancing the descriptive capabilities with reasoning mechanisms, or integrating the context with the knowledge of semantic endpoints like DBpedia.

Summary of Achievements and Main Contributions I of II

IoMusT ontology	The author of this Thesis realized the ontology engineering work for the IoMusT journal publication (still under revision as for February 4, 2020). That is, the various sub-components of the ontology were studied to realize the overall integration and functionalities (Section 2.2.4). The evaluation part (Section 2.2.6) is also an achievement of the work pursued by the author in this Thesis, given the fact that there is not a standard evaluation mechanism for ontologies. Not to mention, the full `.owl` file realization through Protégé, and the provided online documentation.
SWOT ontology	The contribution of the author of this Thesis in the SWOT ontology spans over all the steps described in the previous Chapters 3 and 4. In particular, the complete analysis of the work made by the W3C, including the Thing Description conceptualization and the data representation were refactored keeping in mind the SEPA. The ontology, its `.owl` description, its documentation and the paper published at IEEE IoT Journal [136] are the main achievements to be reported here.
Cocktail framework	The author implemented and currently maintains the Cocktail framework presented in the Chapter 3 and available freely on GitHub. Cocktail is a Python3 framework SEPA compatible enabling an easy realization of Semantic WoT environments as described in [136].

Summary of Achievements and Main Contributions II of II	
Habitat Project	From mid-2016 until late-2017 the author was involved in the Habitat project: on top of semantic technologies and the SEPA, we realized a radio frequency 2D indoor localization monitoring unit, followed later on by a 3D evolution of the same device. They are both fully integrated in a smart health-care IoT application environment [25].
AudioCommons Project	As explained in the previous Section 1.2.2, the author was involved in the Internet Archive integration within the AudioCommons Mediator. The main achievement, here, was the study performed on the integration among different services addressed with semantic technologies.
Visualization of Semantic Knowledge Bases	Working with ontologies both at teaching level and at a research level outlined over the time in ARCES research group the need for tools that would help us in explaining the contents of ontologies and Knowledge Bases. The achievements reached in this field by the author are in one hand a survey over the tools already available [163] and, in the other hand, the collaboration in the realization of another tool specific for teaching purposes, named Tarsier [69]. This work was helpful to study and prepare the SWOT ontology and the IoMusT ontology.

List of Ontologies & Prefixes

`foaf` - The *Friend of a friend* ontology is a basic ontology used to describe links between people.

Developed by	FOAF project, RDF and Semantic Web community
Available at	`http://xmlns.com/foaf/spec/`
SPARQL prefix	`http://xmlns.com/foaf/0.1/`
Reference	`http://www.foaf-project.org/`

`prov` - The *Provenance of Information* ontology is a basic ontology used to describe how agents exchange information .

Developed by	W3C
Available at	`https://www.w3.org/TR/prov-o/`
SPARQL prefix	`http://www.w3.org/ns/prov#`

`ssn`, `sosa` - The *Semantic Sensor Network* ontology, and its core *Sensor, Observation, Sample, and Actuator* ontology, are the basic approach to represent semantically sensors and actuators in their environment.

Developed by	W3C
Available at	`https://www.w3.org/TR/vocab-ssn/`
SPARQL prefixes	`http://www.w3.org/ns/ssn/`
	`http://www.w3.org/ns/sosa/`

`event` - The *Event* ontology deals with sequences of events, providing the tools to identify actors and relationships among them.

Developed by	Centre for Digital Music, Queen Mary University of London
Available at	`http://motools.sourceforge.net/event/event.html`
SPARQL prefix	`http://purl.org/NET/c4dm/event.owl#`

`timeline` - The *Timeline* ontology is an ontology used to annotate any kind of time-related resource.

Developed by	Centre for Digital Music, Queen Mary University of London
Available at	`http://motools.sourceforge.net/timeline/timeline.html`
SPARQL prefix	`http://purl.org/NET/c4dm/timeline.owl#`

`mo`, `music` - The *Music* ontology is a modelization that provides a common vocabulary to address all kind of music-related information.

Developed by	Yves Raimond, Thomas Gängler, Frédérick Giasson, Kurt Jacobson, George Fazekas, Simon Reinhardt, Alexandre Passant

Available at	`http://musicontology.com/`
SPARQL prefix	`http://purl.org/ontology/mo/`
Reference	[126, 127]

`studio` - The *Studio* ontology, compared to the Music ontology, models the music production activity in a recording studio environment.

Developed by	Centre for Digital Music, Queen Mary University of London
Available at	`http://isophonics.net/content/studio-ontology`
SPARQL prefix	*not available*
Reference	[128]

`afo` - The *Audio Features* ontology describes the features of audio signals, and exploits the Event and the Timeline ontologies.

Developed by	Centre for Digital Music, Queen Mary University of London
Available at	`http://motools.sourceforge.net/doc/audio_features.html`
SPARQL prefix	*not available*
Reference	[237]

`dul` - The *DOLCE+DnS Ultralite* (DUL) ontology, with reference to the website, aims to provide a collection of upper-level concepts, to foster alignment among ontologies and semantic setups.

Developed by	Aldo Gangemi, Nicola Guarino, Claudio Masolo Alessandro Oltramari, Luc Schneider
Available at ↴	`http://ontologydesignpatterns.org/wiki/Ontology:DOLCE%2BDnS_Ultralite`
SPARQL prefix	`http://www.ontologydesignpatterns.org/ont/dul/DUL.owl`
Reference	[238]

`swot` - The *Semantic Web of Things* ontology is a work that organizes a new vision of the Web of Things, compatible with semantic technologies, to overcome the fragmentation of IoT.

Developed by	Francesco Antoniazzi, Fabio Viola
Available at	`https://w3id.org/swot#`
SPARQL prefix	`http://www.semanticweb.org/unibo/antoniazzi/2019/0/swot`
Reference	[136]

`iot`, `iomust` - The *Internet of Musical Things* ontology aims to provide the instruments to define an IoT environment for music. It is fully described in Section 2.2.

Developed by	Francesco Antoniazzi
Available at	`https://w3id.org/iomust#`
SPARQL prefix↴	`http://www.semanticweb.org/iot/ontologies/2019/5/internet_of_things`
Reference	*under review*

`collection` - The *Collection* ontology is used to define a common vocabulary to represent collections, lists, bags of items, and their relationships.

Developed by	Paolo Ciccarese, Silvio Peroni

Available at	`https://code.google.com/archive/p/collections-ontology/`
SPARQL prefix	`http://purl.org/co#`
Reference	[239]

`dc` - The *Dublin Core* ontology was defined to describe metadata. It contains, for instance, the authorship relation, and is commonly used to describe even meta information on ontologies.

Developed by	The Dublin Core Metadata Initiative (DCMI)
Available at	`https://www.dublincore.org/specifications/dublin-core/`
SPARQL prefix	`http://purl.org/dc/elements/1.1/` `http://purl.org/dc/terms/`
Reference	[240], `https://www.dublincore.org/specifications/dublin-core/dcmi-terms/2012-06-14/?v=terms`

`aco` - The *Audio Commons* Ontology provides main concepts and properties for describing audio content, both musical and non-musical, on the Semantic Web.

Developed by	George Fazekas, Miguel Ceriani
Available at	`http://www.audiocommons.org/ac-ontology/aco.html`
SPARQL prefix	`https://w3id.org/ac-ontology/aco#`
Reference	[95]

Table 4.4 – 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#`
`iot:`	`http://www.semanticweb.org/iot/ontologies/2019/5/internet_of_things/`
`iomust:`	`http://www.semanticweb.org/iot/ontologies/2019/5/internet_of_things/iomust/`
`mo, music:`	`http://purl.org/ontology/mo/`
`prov:`	`http://www.w3.org/ns/prov#`
`sosa:`	`http://www.w3.org/ns/sosa/`
`co:`	`http://purl.org/co#`
`foaf:`	`http://xmlns.com/foaf/0.1/`
`event:`	`http://purl.org/NET/c4dm/event.owl#`
`timeline:`	`http://purl.org/NET/c4dm/timeline.owl#`
`dbpedia:`	`http://dbpedia.org/resource/`
`dbpo:`	`http://dbpedia.org/ontology/`
`ns:`	*whatever personal valid namespace*

List of Figures

List of Tables

Bibliography

[1] Y. Shoham, "Agent-oriented programming," *Artificial intelligence*, vol. 60, no. 1, pp. 51–92, 1993.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[3] K. Ashton *et al.*, "That internet of things thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[4] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[5] D. Dougherty, "The maker movement," *Innovations: Technology, Governance, Globalization*, vol. 7, no. 3, pp. 11–14, 2012.

[6] L. Martin, "The promise of the maker movement for education," *Journal of Pre-College Engineering Education Research (J-PEER)*, vol. 5, no. 1, p. 4, 2015.

[7] M. B. Abbasy and E. V. Quesada, "Predictable influence of iot (internet of things) in the higher education," *International Journal of Information and Education Technology*, vol. 7, no. 12, pp. 914–920, 2017.

[8] K. Lensing and J. Friedhoff, "Designing a curriculum for the internet-of-things-laboratory to foster creativity and a maker mindset within varying target groups," *Procedia Manufacturing*, vol. 23, pp. 231–236, 2018.

[9] L. Farhan, R. Kharel, O. Kaiwartya, M. Quiroz-Castellanos, A. Alissa, and M. Abdulsalam, "A concise review on internet of things (iot)-problems, challenges and opportunities," in *2018 11th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP)*. IEEE, 2018, pp. 1–6.

[10] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, "Toward better horizontal integration among iot services," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 72–79, 2015.

[11] N. Shadbolt, T. Berners-Lee, and W. Hall, "The semantic web revisited," *IEEE intelligent systems*, vol. 21, no. 3, pp. 96–101, 2006.

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

[13] D. Zeng, S. Guo, and Z. Cheng, "The web of things: A survey," *JCM*, vol. 6, no. 6, pp. 424–438, 2011.

[14] X. Liu and O. Baiocchi, "A comparison of the definitions for smart sensors, smart objects and things in iot," in *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2016, pp. 1–4.

[15] A. M. Mzahm, M. S. Ahmad, and A. Tang, "Enhancing the internet of things (iot) via the concept of agent of things (aot)," *Journal of Network and Innovative Computing*, vol. 2, no. 2014, pp. 101–110, 2014.

[16] C. Savaglio, G. Fortino, M. Ganzha, M. Paprzycki, C. Bădică, and M. Ivanović, "Agent-based computing in the internet of things: a survey," in *International Symposium on Intelligent and Distributed Computing*. Springer, 2017, pp. 307–320.

[17] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil, and D. Barton, "Big data: the management revolution," *Harvard business review*, vol. 90, no. 10, pp. 60–68, 2012.

[18] D. Saha and A. Mukherjee, "Pervasive computing: a paradigm for the 21st century," *Computer*, vol. 36, no. 3, pp. 25–31, 2003.

[19] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[20] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, pp. 1454–1464, 2017.

[21] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[22] G. Betis, V. M. Larios, D. Petri, X. Wu, A. Deacon, and A. Hayar, "The ieee smart cities initiativeaccelerating the smartification process for the 21st century cities [point of view]," *Proceedings of the IEEE*, vol. 106, no. 4, pp. 507–512, 2018.

[23] A. Solanas, C. Patsakis, M. Conti, I. S. Vlachos, V. Ramos, F. Falcone, O. Postolache, P. A. Pérez-Martínez, R. Di Pietro, D. N. Perrea *et al.*, "Smart health: a context-aware health paradigm within smart cities," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 74–81, 2014.

[24] L. Catarinucci, D. De Donno, L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and L. Tarricone, "An iot-aware architecture for smart healthcare systems," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 515–526, 2015.

[25] E. Borelli, G. Paolini, F. Antoniazzi, M. Barbiroli, F. Benassi, F. Chesani, L. Chiari, M. Fantini, F. Fuschini, A. Galassi *et al.*, "Habitat: An iot solution for independent elderly," *Sensors*, vol. 19, no. 5, p. 1258, 2019.

[26] N. Gondchawar and R. Kawitkar, "Iot based smart agriculture," *International Journal of advanced research in Computer and Communication Engineering*, vol. 5, no. 6, pp. 838–842, 2016.

[27] K. Patil and N. Kale, "A model for smart agriculture using iot," in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*. IEEE, 2016, pp. 543–545.

[28] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (iot) communication protocols," in *2017 8th International conference on information technology (ICIT)*. IEEE, 2017, pp. 685–690.

[29] G. Mulligan, "The 6lowpan architecture," in *Proceedings of the 4th workshop on Embedded networked sensors*. ACM, 2007, pp. 78–82.

[30] F. Samie, L. Bauer, and J. Henkel, "Iot technologies for embedded computing: A survey," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 2016, p. 8.

[31] J. Guth, U. Breitenbücher, M. Falkenthal, P. Fremantle, O. Kopp, F. Leymann, and L. Reinfurt, "A detailed analysis of iot platform architectures: concepts, similarities, and differences," in *Internet of Everything*. Springer, 2018, pp. 81–101.

[32] I. Jacobson, I. Spence, and P.-W. Ng, "Is there a single method for the internet of things?" *Communications of the ACM*, vol. 60, no. 11, pp. 46–53, 2017.

[33] T. Banerjee and A. Sheth, "Iot quality control for data and application needs," *IEEE Intelligent Systems*, vol. 32, no. 2, pp. 68–73, 2017.

[34] A. Rayes and S. Salam, "The internet in iotosi, tcp/ip, ipv4, ipv6 and internet routing," in *Internet of Things From Hype to Reality*. Springer, 2017, pp. 35–56.

[35] C. Bormann, M. Ersue, and A. Keranen, "Terminology for constrained-node networks," Tech. Rep., 2014.

[36] O. Bello, S. Zeadally, and M. Badra, "Network layer inter-operation of device-to-device communication technologies in internet of things (iot)," *Ad Hoc Networks*, vol. 57, pp. 52–62, 2017.

[37] H. Mora, M. Signes-Pont, D. Gil, and M. Johnsson, "Collaborative working architecture for iot-based applications," *Sensors*, vol. 18, no. 6, p. 1676, 2018.

[38] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "Role of middleware for internet of things: A study," *International Journal of Computer Science and Engineering Survey*, vol. 2, no. 3, pp. 94–105, 2011.

[39] D. Soni and A. Makwana, "A survey on mqtt: a protocol of internet of things (iot)," in *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, 2017.

[40] M. B. Yassein, M. Q. Shatnawi *et al.*, "Application layer protocols for the internet of things: A survey," in *2016 International Conference on Engineering & MIS (ICEMIS)*. IEEE, 2016, pp. 1–4.

[41] H. Derhamy, J. Eliasson, and J. Delsing, "Iot interoperabilityon-demand and low latency transparent multiprotocol translator," *IEEE*

Internet of Things Journal, vol. 4, no. 5, pp. 1754–1763, 2017.

[42] E. Palavras, K. Fysarakis, I. Papaefstathiou, and I. Askoxylakis, "Semibiot: secure multi-protocol integration bridge for the iot," in *2018 IEEE international conference on communications (ICC)*. IEEE, 2018, pp. 1–7.

[43] M. Blackstock and R. Lea, "Iot interoperability: A hub-based approach," in *2014 international conference on the internet of things (IOT)*. IEEE, 2014, pp. 79–84.

[44] G. Vivek and M. Sunil, "Enabling iot services using wifi-zigbee gateway for a home automation system," in *2015 IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*. IEEE, 2015, pp. 77–80.

[45] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio, "A mobile multi-technology gateway to enable iot interoperability," in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2016, pp. 259–264.

[46] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The internet of things has a gateway problem," in *Proceedings of the 16th international workshop on mobile computing systems and applications*. ACM, 2015, pp. 27–32.

[47] P. Desai, A. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for iot interoperability," in *2015 IEEE International Conference on Mobile Services*. IEEE, 2015, pp. 313–319.

[48] A. Bröring, A. Ziller, V. Charpenay, A. S. Thuluva, D. Anicic, S. Schmid, A. Zappa, M. P. Linares, L. Mikkelsen, and C. Seidel, "The big iot api-semantically enabling iot interoperability," *IEEE Pervasive Computing*, vol. 17, no. 4, pp. 41–51, 2018.

[49] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, and K. Wasielewska, "Towards semantic interoperability between internet of things platforms," in *Integration, interconnection, and interoperability of iot systems*. Springer, 2018, pp. 103–127.

[50] M. Antunes, D. Gomes, and R. L. Aguiar, "Towards iot data classification through semantic features," *Future Generation Computer Systems*, vol. 86, pp. 792–798, 2018.

[51] A. D. JoSEP, R. KAtz, A. KonWinSKi, L. Gunho, D. PAttERSon, and A. RABKin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, 2010.

[52] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[53] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih, "Big data technologies: A survey," *Journal of King Saud University-Computer and Information Sciences*, vol. 30, no. 4, pp. 431–448, 2018.

[54] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money, "Big data: Issues and challenges moving forward," in *2013 46th Hawaii International Conference on System Sciences*. IEEE, 2013, pp. 995–1004.

[55] V. N. Gudivada, R. Baeza-Yates, and V. V. Raghavan, "Big data: Promises and problems," *Computer*, no. 3, pp. 20–23, 2015.

[56] D. Raggett, "The web of things: Challenges and opportunities," *Computer*, vol. 48, no. 5, pp. 26–32, 2015.

[57] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.

[58] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Semantics and content," 2014.

[59] T. Berners-Lee, R. Fielding, L. Masinter *et al.*, "Uniform resource identifiers (uri): Generic syntax," 1998.

[60] B. Cheng, S. Zhao, J. Qian, Z. Zhai, and J. Chen, "Lightweight service mashup middleware with rest style architecture for iot applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1063–1075, 2018.

[61] M. Laine, "Restful web services for the internet of things," *Online]. Saatavilla: http://media. tkk. fi/webservices/personnel/markku_laine/restful_web_services_for _the_internet_of_things. pdf*, 2012.

[62] N. F. Noy, "Semantic integration: a survey of ontology-based approaches," *ACM Sigmod Record*, vol. 33, no. 4, pp. 65–70, 2004.

[63] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[64] N. Noy, "Order from chaos," *Queue*, vol. 3, no. 8, pp. 42–49, 2005.

[65] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer *et al.*, "Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.

[66] A. Maedche and S. Staab, "Ontology learning for the semantic web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 72–79, March 2001.

[67] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi, "Storing and querying scientific workflow provenance metadata using an rdbms," in *e-Science and Grid Computing, IEEE International Conference on.* IEEE, 2007, pp. 611–618.

[68] S. G. Kobourov, "Spring embedders and force directed graph drawing algorithms," *arXiv preprint arXiv:1201.3011*, 2012.

[69] F. Viola, L. Roffia, F. Antoniazzi, D. Alfredo, C. Aguzzi, and T. Salmon Cinotti, "Interactive 3d exploration of rdf graphs through semantic planes," *Future Internet*, vol. 10, no. 9, p. 36, 2018.

[70] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "A taxonomy and survey of dynamic graph visualization," in *Computer Graphics Forum*, vol. 36, no. 1. Wiley Online Library, 2017, pp. 133–159.

[71] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker, "Cytoscape: a software environment for integrated models of biomolecular interaction networks," *Genome research*, vol. 13, no. 11, pp. 2498–2504, 2003.

[72] T. Hastrup, R. Cyganiak, and U. Bojars, "Browsing linked data with fenfire," 2008.

[73] M. Bastian, S. Heymann, M. Jacomy *et al.*, "Gephi: an open source software for exploring and manipulating networks." *Icwsm*, vol. 8, no. 2009, pp. 361–362, 2009.

[74] W. Hop, S. de Ridder, F. Frasincar, and F. Hogenboom, "Using hierarchical edge bundles to visualize complex ontologies in glow," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing.* ACM, 2012, pp. 304–311.

[75] E. Pietriga, "Isaviz: A visual authoring tool for rdf," *World Wide Web Consortium.[Online]. Available: http://www. w3. org/2001/11/IsaViz*, 2003.

[76] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphvizopen source graph drawing tools," in *International Symposium on Graph Drawing.* Springer, 2001, pp. 483–484.

[77] M.-A. Storey, N. F. Noy, M. Musen, C. Best, R. Fergerson, and N. Ernst, "Jambalaya: an interactive environment for exploring ontologies," in *Proceedings of the 7th international conference on Intelligent user interfaces.* ACM, 2002, pp. 239–239.

[78] M.-A. Storey, C. Best, and J. Michand, "Shrimp views: An interactive environment for exploring java programs," in *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on.* IEEE, 2001, pp. 111–112.

[79] D. V. Camarda, S. Mazzini, and A. Antonuccio, "Lodlive, exploring the web of data," in *Proceedings of the 8th International Conference on Semantic Systems.* ACM, 2012, pp. 197–200.

[80] S. Falconer, "Ontograf protege plugin." [Online]. Available: http://protegewiki.stanford.edu/wiki/OntoGraf

[81] A. Bosca, D. Bonino, and P. Pellegrino, "Ontosphere: more than a 3d ontology visualization tool." in *Swap.* Citeseer, 2005.

[82] M. Horridge, "Owlviz," *Available on: http://protegewiki.stanford.edu/wiki/OWLViz*, 2010.

[83] L. Deligiannidis, K. J. Kochut, and A. P. Sheth, "Rdf data exploration and visualization," in *Proceedings of the ACM first workshop on CyberInfrastructure: information management in eScience.* ACM, 2007, pp. 39–46.

[84] M. Janik and K. Kochut, "Brahms: a workbench rdf store and high performance memory system for semantic association discovery," in *International Semantic Web Conference.* Springer, 2005, pp. 431–445.

[85] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann, "Relfinder: Revealing relationships in rdf knowledge

bases," in *International Conference on Semantic and Digital Media Technologies.* Springer, 2009, pp. 182–187.

[86] E. Motta, P. Mulholland, S. Peroni, M. dAquin, J. M. Gomez-Perez, V. Mendez, and F. Zablith, "A novel approach to visualizing and navigating ontologies," in *International Semantic Web Conference.* Springer, 2011, pp. 470–486.

[87] H. Alani, "Tgviztab: an ontology visualisation extension for protégé," 2003.

[88] S. Lohmann, V. Link, E. Marbach, and S. Negru, "WebVOWL: Web-based visualization of ontologies," in *Proceedings of EKAW 2014 Satellite Events*, ser. LNAI, vol. 8982. Springer, 2015, pp. 154–158.

[89] S. Lohmann, S. Negru, F. Haag, and T. Ertl, "Visualizing ontologies with vowl," *Semantic Web*, vol. 7, no. 4, pp. 399–419, 2016.

[90] S. Lohmann, S. Negru, and D. Bold, "The ProtégéVOWL plugin: Ontology visualization for everyone," in *Proceedings of ESWC 2014 Satellite Events*, ser. LNCS, vol. 8798. Springer, 2014, pp. 395–400.

[91] M. Weise, S. Lohmann, and F. Haag, "Ldvowl: Extracting and visualizing schema information for linked data," in *2nd International Workshop on Visualization and Interaction for Ontologies and Linked Data, Kobe, Japón*, 2016, pp. 120–127.

[92] F. Haag, S. Lohmann, S. Siek, and T. Ertl, "Queryvowl: A visual query notation for linked data," in *International Semantic Web Conference.* Springer, 2015, pp. 387–402.

[93] F. Font, T. Brookes, G. Fazekas, M. Guerber, A. La Burthe, D. Plans, M. D. Plumbley, M. Shaashua, W. Wang, and X. Serra, "Audio commons: bringing creative commons audio content to the creative industries," in *Audio Engineering Society Conference: 61st International Conference: Audio for Games.* Audio Engineering Society, 2016.

[94] X. Favory, E. Fonseca, F. Font, and X. Serra, "Facilitating the manual annotation of sounds when using large taxonomies," in *Proceedings of the 23rd Conference of Open Innovations Association FRUCT.* FRUCT Oy, 2018, p. 60.

[95] M. Ceriani and G. Fazekas, "Audio commons ontology: a data model for an audio content ecosystem," in *International Semantic Web Conference.* Springer, 2018, pp. 20–35.

[96] M. Lefrançois, A. Zimmermann, and N. Bakerally, "A sparql extension for generating rdf from heterogeneous formats," in *European Semantic Web Conference.* Springer, 2017, pp. 35–50.

[97] ——, "Flexible rdf generation from rdf and heterogeneous data sources with sparql-generate," in *European Knowledge Acquisition Workshop.* Springer, 2016, pp. 131–135.

[98] A. Xambó, J. Pauwels, G. Roma, M. Barthet, and G. Fazekas, "Jam with jamendo: Querying a large music collection by chords from a learner's perspective," in *Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion.* ACM, 2018, p. 30.

[99] L. Roffia, P. Azzoni, C. Aguzzi, F. Viola, F. Antoniazzi, and T. Salmon Cinotti, "Dynamic Linked Data: A SPARQL Event Processing Architecture," *Future Internet*, vol. 10, no. 4, p. 36, 2018.

[100] J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, "Smart-m3 information sharing platform," in *The IEEE symposium on Computers and Communications.* IEEE, 2010, pp. 1041–1046.

[101] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[102] F. Morandi, L. Roffia, A. D'Elia, F. Vergari, and T. S. Cinotti, "Redsib: a smart-m3 semantic information broker implementation," in *2012 12th Conference of Open Innovations Association (FRUCT).* IEEE, 2012, pp. 1–13.

[103] D. Manzaroli, L. Roffia, T. S. Cinotti, E. Ovaska, P. Azzoni, V. Nannini, and S. Mattarozzi, "Smart-m3 and osgi: The interoperability platform," in *The IEEE symposium on Computers and Communications.* IEEE, 2010, pp. 1053–1058.

[104] I. V. Galov, A. A. Lomov, and D. G. Korzun, "Design of semantic information broker for localized computing environments in the internet of things," in *2015 17th Conference of Open Innovations Association (FRUCT).* IEEE, 2015, pp. 36–43.

[105] F. Viola, A. D'Elia, L. Roffia, and T. S. Cinotti, "A modular lightweight implementation of the smart-m3 semantic information broker," in *2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*. IEEE, 2016, pp. 370–377.

[106] ——, "Performance evaluation suite for semantic publish-subscribe message-oriented middlewares," 2016.

[107] F. Viola, A. D'Elia, D. Korzun, I. Galov, A. Kashevnik, and S. Balandin, "The m3 architecture for smart spaces: Overview of semantic information broker implementations," in *2016 19th Conference of Open Innovations Association (FRUCT)*. IEEE, 2016, pp. 264–272.

[108] H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, and H. S. Thompson, "When owl: sameas isnt the same: An analysis of identity in linked data," in *International semantic web conference*. Springer, 2010, pp. 305–320.

[109] D. Calvanese, M. Giese, D. Hovland, and M. Rezk, "Ontology-based integration of cross-linked datasets," in *International Semantic Web Conference*. Springer, 2015, pp. 199–216.

[110] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, "Internet of Musical Things: Vision and Challenges," *IEEE Access*, vol. 6, pp. 61 994–62 017, 2018.

[111] L. Turchet, "Smart Musical Instruments: vision, design principles, and future directions," *IEEE Access*, vol. 7, pp. 8944–8963, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2018.2876891

[112] L. Turchet, A. McPherson, and M. Barthet, "Real-time hit classification in a Smart Cajón," *Frontiers in ICT*, vol. 5, no. 16, 2018. [Online]. Available: https://doi.org/10.3389/fict.2018.00016

[113] L. Turchet, M. Benincaso, and C. Fischione, "Examples of use cases with smart instruments," in *Proceedings of Audio Mostly Conference*, 2017, pp. 47:1–47:5. [Online]. Available: https://doi.org/10.1145/3123514.3123553

[114] L. Turchet and M. Barthet, "Co-design of Musical Haptic Wearables for electronic music performer's communication," *IEEE Transactions on Human-Machine Systems*, vol. 49, no. 2, pp. 183–193, 2019.

[115] M. Wright, A. Freed, and A. Momeni, "Opensound control: State of the art 2003," in *Proceedings of the Conference on New Interfaces for Musical Expression*, 2003, pp. 153–160.

[116] J. Malloch, S. Sinclair, and M. Wanderley, "Distributed tools for interactive design of heterogeneous signal networks," *Multimedia Tools and Applications*, vol. 74, no. 15, pp. 5683–5707, 2015.

[117] J. Sowa, *Knowledge representation: logical, philosophical, and computational foundations*. Brooks/Cole Pacific Grove, CA, 2000, vol. 13.

[118] L. Turchet, F. Viola, G. Fazekas, and M. Barthet, "Towards a Semantic Architecture for Internet of Musical Things applications," in *IEEE Conference of Open Innovations Association (FRUCT)*. IEEE, 2018, pp. 382–390.

[119] M. Ulieru and R. Doursat, "Emergent engineering: a radical paradigm shift," *International Journal of Autonomous and Adaptive Communications Systems*, vol. 4, no. 1, p. 39, 2011.

[120] N. Matentzoglu, J. Malone, C. Mungall, and R. Stevens, "Miro: guidelines for minimum information for the reporting of an ontology," *Journal of Biomedical Semantics*, vol. 9, no. 1, p. 6, Jan 2018. [Online]. Available: https://doi.org/10.1186/s13326-017-0172-7

[121] M. Fernández-López, A. Gómez-Pérez, and N. Juristo, "METHONTOLOGY: from ontological art towards ontological engineering," in *Proceedings of the Onto- logical Engineering AAAI-97 Spring Symposium Series*. American Association for Artificial Intelligence, 1997.

[122] B. Motik, P. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, and M. Smith, "Owl 2 web ontology language: Structural specification and functional-style syntax," *W3C recommendation*, vol. 27, no. 65, p. 159, 2009.

[123] M. Uschold, "Building ontologies: Towards a uni ed methodology," in *Proceedings of 16th Annual Conference of the British Computer Society Specialists Group on Expert Systems*. Citeseer, 1996.

[124] A. De Nicola and M. Missikoff, “A lightweight methodology for rapid ontology engineering,” *Communications of the ACM*, vol. 59, no. 3, pp. 79–86, 2016.

[125] T. Wilmering, G. Fazekas, and M. Sandler, “The audio effects ontology,” in *Proc. of the 14th International Society for Music Information Retrieval Conference, ISMIR’13, November 4-8, Curitiba, Brazil*, 2013.

[126] Y. Raimond, S. Abdallah, M. Sandler, and F. Giasson, “The music ontology,” in *Proceedings of International Society for Music Information Retrieval Conference*, 2007.

[127] Y. Raimond, F. Giasson, K. Jacobson, G. Fazekas, T. Gangler, and S. Reinhardt, “The music ontology specification,” in *Online Specification Document.*, 2010. [Online]. Available: http://musicontology.com/

[128] G. Fazekas and M. Sandler, “The Studio Ontology Framework,” in *Proceedings of the International Society for Music Information Retrieval conference*, 2011, pp. 24–28.

[129] S. Kolozali, G. Fazekas, M. Barthet, and M. Sandler, “Knowledge representation issues in musical instrument ontology design,” in *Proc. of the 12th International Society for Music Information Retrieval (ISMIR’11) conference, 24-28 Oct., Miami, Florida, USA*, 2011.

[130] M. Compton, P. Barnaghi, L. Bermudez, R. Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog *et al.*, “The ssn ontology of the semantic sensor networks incubator group,” *Journal of Web Semantics: Science, Services and Agents on the World Wide Web, ISSN*, pp. 1570–8268, 2011.

[131] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, “SOSA: A lightweight ontology for sensors, observations, samples, and actuators,” *Journal of Web Semantics*, 2018.

[132] A. Haller, K. Janowicz, S. J. Cox, M. Lefrançois, K. Taylor, D. Le Phuoc, J. Lieberman, R. García-Castro, R. Atkinson, and C. Stadler, “The sosa/ssn ontology: a joint wec and ogc standard specifying the semantics of sensors observations actuation and sampling,” in *Semantic Web.* IOS Press, 2018, vol. 1, pp. 1–19.

[133] V. Charpenay, S. Käbisch, and H. Kosch, “Introducing thing descriptions and interactions: An ontology for the web of things.” in *SR+ SWIT@ ISWC*, 2016, pp. 55–66.

[134] F. Serena, M. Poveda-Villalón, and R. García-Castro, “Semantic discovery in the web of things,” in *International Conference on Web Engineering.* Springer, 2017, pp. 19–31.

[135] F. Viola, A. Stolfi, A. Milo, M. Ceriani, M. Barthet, and G. Fazekas, “Playsound.space: enhancing a live performance tool with semantic recommendations,” in *Proc. 1st SAAM Workshop (in press).* ACM, 2018.

[136] F. Antoniazzi and F. Viola, “Building the semantic web of things through a dynamic ontology,” *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 560–10 579, Dec 2019.

[137] K. Siegemund, E. J. Thomas, Y. Zhao, J. Pan, and U. Assmann, “Towards ontology-driven requirements engineering,” in *Workshop semantic web enabled software engineering at 10th international semantic web conference*, 2011.

[138] V. Braun and V. Clarke, “Using thematic analysis in psychology,” *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[139] M. Grüninger and M. S. Fox, “Methodology for the design and evaluation of ontologies,” in *Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995, pp. 6.1.–6.10.

[140] A. Bröring, S. Schmid, C.-K. Schindhelm, A. Khelil, S. Käbisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, and E. Teniente, “Enabling iot ecosystems through platform interoperability,” *IEEE software*, vol. 34, no. 1, pp. 54–61, 2017.

[141] H. Rijgersberg, M. Van Assem, and J. Top, “Ontology of units of measure and related concepts,” *Semantic Web*, vol. 4, no. 1, pp. 3–13, 2013.

[142] T. W. Narock, A. Szabo, and J. Merka, “Using semantics to extend the space physics data environment,” *Computers & Geosciences*, vol. 35, no. 4, pp. 791–797, 2009.

[143] X. Wang, X. Zhang, and M. Li, “A survey on semantic sensor web: Sensor ontology, mapping and query,” *International Journal of u-and e-Service, Science and Technology*, vol. 8, no. 10, pp. 325–342, 2015.

[144] M. A. Musen and the ProtégéTeam, "The Protégé Project: A Look Back and a Look Forward," *AI matters*, vol. 1, no. 4, pp. 4–12, 06 2015.

[145] N. R. Jennings, "Agent-oriented software engineering," in *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. Springer, 1999, pp. 1–7.

[146] P. Leitao, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, "Smart agents in industrial cyber–physical systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1086–1101, 2016.

[147] K. Kravari and N. Bassiliades, "A survey of agent platforms," *Journal of Artificial Societies and Social Simulation*, vol. 18, no. 1, p. 11, 2015.

[148] J. Hendler, "Agents and the semantic web," *IEEE Intelligent systems*, vol. 16, no. 2, pp. 30–37, 2001.

[149] J. Lin, S. Sedigh, and A. Miller, "Modeling cyber-physical systems with semantic agents," in *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*. IEEE, 2010, pp. 13–18.

[150] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao, "Prov-o: The prov ontology," *W3C recommendation*, vol. 30, 2013.

[151] 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, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 61–75.

[152] S. Lohmann, S. Negru, F. Haag, and T. Ertl, "Visualizing ontologies with VOWL," *Semantic Web*, vol. 7, no. 4, pp. 399–419, 2016. [Online]. Available: http://dx.doi.org/10.3233/SW-150200

[153] R. Shearer, B. Motik, and I. Horrocks, "HermiT: A Highly-Efficient OWL Reasoner." in *Proc. OWLED 2008*, vol. 432, 2008, p. 91.

[154] B. Parsia and E. Sirin, "Pellet: An owl dl reasoner," in *Third international semantic web conference-poster*, vol. 18. Publishing, 2004, p. 2.

[155] D. Tsarkov and I. Horrocks, "Fact++ description logic reasoner: System description," *Automated reasoning*, pp. 292–297, 2006.

[156] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa, "Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 10, no. 2, pp. 7–34, 2014.

[157] D. Guinard and V. Trifa, *Building the web of things: with examples in node. js and raspberry pi.* Manning Publications Co., 2016.

[158] L. Roffia, F. Morandi, J. Kiljander, A. DElia, F. Vergari, F. Viola, L. Bononi, and T. S. Cinotti, "A semantic publish-subscribe architecture for the internet of things," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1274–1296, 2016.

[159] M. Uschold, M. Healy, K. Williamson, P. Clark, and S. Woods, "Ontology reuse and application," in *Formal ontology in information systems*, vol. 179. IOS Press Amsterdam, 1998, p. 192.

[160] E. P. Bontas, M. Mochol, and R. Tolkorf, "Case studies on ontology reuse," in *Proceedings of the IKNOW05 International Conference on Knowledge Management*, vol. 74, 2005, p. 345.

[161] J. Z. Pan, L. Serafini, and Y. Zhao, "Semantic import: An approach for partial ontology reuse," in *Proceedings of the 1st International Conference on Modular Ontologies-Volume 232*. CEUR-WS. org, 2006, pp. 71–84.

[162] S. Karim, K. Latif, and A. M. Tjoa, "Providing universal accessibility using connecting ontologies: A holistic approach," in *International Conference on Universal Access in Human-Computer Interaction*. Springer, 2007, pp. 637–646.

[163] F. Antoniazzi and F. Viola, "Rdf graph visualization tools: a survey," in *Proceedings of the 23rd Conference of Open Innovations Association FRUCT*. FRUCT Oy, 2018, p. 4.

[164] 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.

[165] M. Rinne, E. Blomqvist, R. Keskisärkkä, and E. Nuutila, "Event processing in rdf." in *WOP*, 2013.

[166] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-P. Calbimonte, M. Riahi, K. Aberer, P. P. Jayaraman, A. Zaslavsky, I. P. Žarko *et al.*, "Openiot: Open source internet-of-things in the cloud," in *Interoperability and open-source solutions for the internet of things.* Springer, 2015, pp. 13–25.

[167] N. Seydoux, K. Drira, N. Hernandez, and T. Monteil, "Iot-o, a core-domain iot ontology to represent connected devices networks," in *European Knowledge Acquisition Workshop.* Springer, 2016, pp. 561–576.

[168] 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," *Personal and Ubiquitous Computing*, vol. 21, no. 3, pp. 475–487, 2017.

[169] D. Puiu, P. Barnaghi, R. Tönjes, D. Kümper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. Farajidavar *et al.*, "Citypulse: Large scale data analytics framework for smart cities," *IEEE Access*, vol. 4, pp. 1086–1108, 2016.

[170] 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 *2017 24th International Conference on Telecommunications (ICT).* IEEE, 2017, pp. 1–7.

[171] 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 *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT).* IEEE, 2016, pp. 442–447.

[172] 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 *2019 24rd Conference of Open Innovations Association (FRUCT).* IEEE, 2019.

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

[174] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "Iot gateway: Bridgingwireless sensor networks into internet of things," in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on.* Ieee, 2010, pp. 347–352.

[175] S. K. Datta, C. Bonnet, and N. Nikaein, "An iot gateway centric architecture to provide novel m2m services," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on.* IEEE, 2014, pp. 514–519.

[176] A. Gangemi, R. Lillo, G. Lodi, and A. G. Nuzzolese, "A pattern-based ontology for the internet of things."

[177] W. Wang, S. De, R. Toenjes, E. Reetz, and K. Moessner, "A comprehensive ontology for knowledge representation in the internet of things," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on.* IEEE, 2012, pp. 1793–1798.

[178] P.-Y. Vandenbussche, G. A. 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.

[179] 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 *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on.* IEEE, 2016, pp. 254–261.

[180] A. Gyrard, A. Zimmermann, and A. Sheth, "Building iot-based applications for smart cities: How can ontology catalogs help?" *IEEE Internet of Things Journal*, vol. 5, pp. 3978–3990, 2018.

[181] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, pp. 70–95, 2016.

[182] D. J. Wu, A. Taly, A. Shankar, and D. Boneh, "Privacy, discovery, and authentication for the internet of things," in *European Symposium on Research in Computer Security.* Springer, 2016, pp. 301–319.

[183] B. Djamaa, M. A. Kouda, A. Yachir, and T. Kenaza, “Fetchiot: Efficient resource fetching for the internet of things,” in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2018, pp. 637–643.

[184] F. Viola, L. Turchet, F. Antoniazzi, and G. Fazekas, “C minor: a semantic publish/-subscribe broker for the internet of musical things,” in *Open Innovations Association (FRUCT), 2018 23th Conference of*. IEEE, 2018, pp. 405–415.

[185] P. Waher and R. Klauck, “Internet of things-discovery,” 2018.

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

[187] S. Mayer and D. Guinard, “An extensible discovery service for smart things,” in *Proceedings of the Second International Workshop on Web of Things*. ACM, 2011, p. 7.

[188] S. B. Fredj, M. Boussard, D. Kofman, and L. Noirie, “Efficient semantic-based iot service discovery mechanism for dynamic environments,” *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*, pp. 2088–2092, 2014.

[189] M. Ganzha, M. Paprzycki, W. Pawlowski, P. Szmeja, and K. Wasielewska, “Semantic technologies for the iot - an inter-iot perspective,” *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 271–276, 2016.

[190] F. Gao, M. I. Ali, and A. Mileo, “Semantic discovery and integration of urban data streams*,” *challenge*, vol. 7, p. 16, 2014.

[191] A. Kamilaris, S. Yumusak, and M. I. Ali, “Wots2e: A search engine for a semantic web of things,” in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE, 2016, pp. 436–441.

[192] 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 *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2019, pp. 1–8.

[193] K. Dar, A. Taherkordi, R. Rouvoy, and F. Eliassen, “Adaptable service composition for very-large-scale internet of things systems,” in *Proceedings of the 8th Middleware Doctoral Symposium*. ACM, 2011, p. 2.

[194] G. Tzortzis, “A semi-automatic approach for semantic iot service composition,” 2016.

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

[196] J. Delsing, *Iot automation: Arrowhead framework*. CRC Press, 2017.

[197] 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*. Springer, 2015, pp. 137–152.

[198] A. Bassi, M. Bauer, M. Fiedler, and R. v. Kranenburg, *Enabling things to talk*. Springer, 2013.

[199] P. Barnaghi, W. Wang, C. Henson, and K. Taylor, “Semantics for the internet of things: early progress and back to the future,” *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 8, no. 1, pp. 1–21, 2012.

[200] D. Pfisterer, K. Römer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth *et al.*, “Spitfire: Toward a semantic web of things.” *IEEE Communications Magazine*, vol. 49, no. 11, pp. 40–48, 2011.

[201] M. Ruta, F. Scioscia, and E. Di Sciascio, “Enabling the semantic web of things: framework and architecture,” in *2012 IEEE Sixth International Conference on Semantic Computing*. IEEE, 2012, pp. 345–347.

[202] M. Noura, S. Heil, and M. Gaedke, “Growth: Goal-oriented end user development for web of things devices,” in *International Conference on Web Engineering*. Springer, 2018, pp. 358–365.

[203] 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.

[204] 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 *Open Innovations Association (FRUCT), 2017 21st Conference of*. IEEE, 2017, pp. 51–56.

[205] M. Swan, "Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0," *Journal of Sensor and Actuator networks*, vol. 1, no. 3, pp. 217–253, 2012.

[206] P. Gope and T. Hwang, "Bsn-care: A secure iot-based modern healthcare system using body sensor network," *IEEE Sensors Journal*, vol. 16, no. 5, pp. 1368–1376, 2016.

[207] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, "Internet of musical things: Vision and challenges," *IEEE Access*, vol. 6, pp. 61 994–62 017, 2018.

[208] M. A. Musen *et al.*, "The protégé project: a look back and a look forward," *AI matters*, vol. 1, no. 4, p. 4, 2015.

[209] P. Hitzler and F. Van Harmelen, "A reasonable semantic web," *Semantic Web*, vol. 1, no. 1, 2, pp. 39–44, 2010.

[210] M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge, "The belief-desire-intention model of agency," in *International workshop on agent theories, architectures, and languages*. Springer, 1998, pp. 1–10.

[211] B. N. Schilit, N. Adams, R. Want *et al.*, *Context-aware computing applications*. Xerox Corporation, Palo Alto Research Center, 1994.

[212] P. Dourish, "Seeking a foundation for context-aware computing," *Human–Computer Interaction*, vol. 16, no. 2-4, pp. 229–241, 2001.

[213] 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 *International symposium on handheld and ubiquitous computing*. Springer, 1999, pp. 304–307.

[214] A. Kofod-Petersen and J. Cassens, "Using activity theory to model context awareness," in *International Workshop on Modeling and Retrieval of Context*. Springer, 2005, pp. 1–17.

[215] O. B. Sezer, E. Dogdu, and A. M. Ozbayoglu, "Context-aware computing, learning, and big data in internet of things: a survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 1–27, 2018.

[216] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[217] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang, "An ontology-based context model in intelligent environments," in *Proceedings of communication networks and distributed systems modeling and simulation conference*, vol. 2004. San Diego, CA, USA., 2004, pp. 270–275.

[218] T. Broens, S. Pokraev, M. Van Sinderen, J. Koolwaaij, and P. D. Costa, "Context-aware, ontology-based service discovery," in *European Symposium on Ambient Intelligence*. Springer, 2004, pp. 72–83.

[219] M. Alirezaie, J. Renoux, U. Köckemann, A. Kristoffersson, L. Karlsson, E. Blomqvist, N. Tsiftes, T. Voigt, and A. Loutfi, "An ontology-based context-aware system for smart homes: E-care@ home," *Sensors*, vol. 17, no. 7, p. 1586, 2017.

[220] O. Cabrera, X. Franch, and J. Marco, "3lconont: a three-level ontology for context modelling in context-aware computing," *Software & Systems Modeling*, vol. 18, no. 2, pp. 1345–1378, 2019.

[221] P. C. Ccori, L. C. C. De Biase, M. K. Zuffo, and F. S. C. da Silva, "Device discovery strategies for the iot," in *2016 IEEE International Symposium on Consumer Electronics (ISCE)*. IEEE, 2016, pp. 97–98.

[222] Q. M. Ashraf, M. H. Habaebi, M. R. Islam, and S. Khan, "Device discovery and configuration scheme for internet of things," in *2016 International conference on intelligent systems engineering (ICISE)*. IEEE, 2016, pp. 38–43.

[223] Y. Zhou, G. Cherian, and S. P. Abraham, "Server-assisted device-to-device discovery and connection," May 16 2017, uS Patent 9,654,960.

[224] Y. Ding, Y. Jin, L. Ren, and K. Hao, "An intelligent self-organization scheme for the internet of things," *IEEE Computational Intelligence Magazine*, vol. 8, no. 3, pp. 41–53, 2013.

[225] N. Wanigasekara, J. Schmalfuss, D. Carlson, and D. S. Rosenblum, "A bandit approach for intelligent iot service composition across heterogeneous smart spaces," in *Proceedings of the 6th International Conference on the Internet of Things*. ACM, 2016, pp. 121–129.

[226] E. Kovacs, M. Bauer, J. Kim, J. Yun, F. Le Gall, and M. Zhao, "Standards-based worldwide semantic interoperability for iot," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 40–46, 2016.

[227] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, and K. Wasielewska, "Semantic interoperability in the internet of things: An overview from the inter-iot perspective," *Journal of Network and Computer Applications*, vol. 81, pp. 111–124, 2017.

[228] R. Murphy and D. D. Woods, "Beyond asimov: the three laws of responsible robotics," *IEEE Intelligent Systems*, vol. 24, no. 4, pp. 14–20, 2009.

[229] A. Haroon, M. A. Shah, Y. Asim, W. Naeem, M. Kamran, and Q. Javaid, "Constraints in the iot: the world in 2020 and beyond," *Constraints*, vol. 7, no. 11, pp. 252–271, 2016.

[230] M. M. Hossain, M. Fotouhi, and R. Hasan, "Towards an analysis of security issues, challenges, and open problems in the internet of things," in *2015 IEEE World Congress on Services*. IEEE, 2015, pp. 21–28.

[231] A. M. Mzahm, M. S. Ahmad, and A. Y. Tang, "Agents of things (aot): An intelligent operational concept of the internet of things (iot)," in *2013 13th International Conference on Intellient Systems Design and Applications*. IEEE, 2013, pp. 159–164.

[232] I. Kotseruba and J. K. Tsotsos, "40 years of cognitive architectures: core cognitive abilities and practical applications," *Artificial Intelligence Review*, pp. 1–78, 2018.

[233] G. Paolini, D. Masotti, F. Antoniazzi, T. S. Cinotti, and A. Costanzo, "Fall detection and 3-d indoor localization by a custom rfid reader embedded in a smart e-health platform," *IEEE Transactions on Microwave Theory and Techniques*, pp. 1–11, 2019.

[234] ——, "Anchorless indoor localization and tracking in real-time at 2.45 ghz," in *2019 IEEE MTT-S International Microwave Symposium (IMS)*, June 2019, pp. 286–289.

[235] M. Challenger, B. Tezel, O. Alaca, B. Tekinerdogan, and G. Kardas, "Development of semantic web-enabled bdi multi-agent systems using sea_ml: An electronic bartering case study," *Applied Sciences*, vol. 8, no. 5, p. 688, 2018.

[236] M. Proctor, "Drools: a rule engine for complex event processing," in *Proceedings of the 4th international conference on Applications of Graph Transformations with Industrial Relevance*. Springer-Verlag, 2011, pp. 2–2.

[237] A. Allik, G. Fazekas, and M. Sandler, "An ontology for audio features," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2016, pp. 73–79.

[238] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider, "Sweetening ontologies with dolce," in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2002, pp. 166–181.

[239] P. Ciccarese and S. Peroni, "The collections ontology: creating and handling collections in owl 2 dl frameworks," *Semantic Web*, vol. 5, no. 6, pp. 515–529, 2014.

[240] S. L. Weibel and T. Koch, "The dublin core metadata initiative," *D-lib magazine*, vol. 6, no. 12, pp. 1082–9873, 2000.