Alma Mater Studiorum – Università di Bologna

Dottorato di ricerca in: Computer Science and Engineering
Ciclo: XXXI

Settore concorsuale: 09/H1
Settore scientifico disciplinare: ING-INF/05

Semantic Web and the Web of Things: concept, platform and applications

Presentata da: Fabio Viola

Coordinatore Dottorato:
Prof. Paolo Ciaccia

Supervisore:
Prof. Tullio Salmon Cinotti

Tutor:
Prof. Luciano Bononi

Esame finale: aprile 2019

To my parents

. . .

Said he'll see me on the flip side
On this trip he's taken for a ride
He's been takin' too much on
There he goes with his perfectly unkept hope
There he goes

. . .

Pearl Jam, "Off he goes"

Abstract

The ubiquitous presence of devices provided with computational resources and connectivity is fostering the diffusion of a new ICT paradigm known as Internet of Things (IoT), where the so-called smart objects interoperate and react to the available information to provide services to the users. The IoT is the result of a three-decade evolution started with Radio-Frequency Identification (RFID) in the area of logistics and now spanning over more than fifty application domains. The pervasiveness of the IoT across so many different areas proves the worldwide interest of Researchers from both the academic and enterprise worlds. This Research has brought to the birth of a plethora of new technologies and protocols designed to address different needs of the emerging scenarios. As a result, today it is hard to develop interoperable applications, due to the diversity of the available technologies.

The Web of Things is born to address this problem through the adoption of the standard protocols responsible for the success of the Web (e.g., HTTP). But a key contribution in this sense can be provided considering also standards coming from the Semantic Web. In fact, the protocols born in the Semantic Web context grant the univocal identification of resources and the representation of data in a way that 1) information is machine understandable enabling the automatic computation; 2) information from different sources can be easily aggregated in a wider knowledge base. Semantic Web technologies can then be considered as interoperability enablers for the Internet of Things.

This Thesis investigates how to efficiently and effectively employ Semantic Web protocols in the IoT, to realize the Semantic Web of Things (SWoT) vision of a really interoperable network of smart applications. More in detail, while Part I introduces the IoT, its history and its current state, Part II investigates the algorithms to efficiently support the publish/subscribe paradigm in semantic brokers for the SWoT and their implementation in the Smart-M3 interoperability platform and its descendant SPARQL Event Processing Architecture (SEPA). Moreover, the preliminary work toward the definition of the first benchmark for SWoT applications is presented. Part IV describes the Research activity aimed at applying the developed semantic infrastructures in real life scenarios (i.e., electro-mobility, home automation, seman-

tic audio and Internet of Musical Things). Finally, in Part V, conclusions are drawn.

A lack of effective ways to explore and debug Semantic Web datasets emerged during these Research activities. Then, Part III describes a second Research branch of my PhD work aimed at the invention of a new, effective way to visualize semantic knowledge bases, based on the popular graph representation and the introduction of the concept of Semantic Planes.

Abstract (in italiano)

La presenza massiva di dispositivi dotati di capacità computazionale e connettività sta alimentando la diffusione di un nuovo paradigma nell'ICT, conosciuto come Internet of Things (IoT). L'IoT è caratterizzato dai cosiddetti smart object che interagiscono, cooperano e reagiscono alle informazioni a loro disponibili per fornire servizi agli utenti. L'IoT è il risultato di un'evoluzione iniziata circa trent'anni fa (ed ancora in atto) con l'applicazione degli RFID nelle applicazioni di logistica e si è espansa negli anni su oltre cinquanta domini applicativi. La diffusione dell'IoT su così tante aree è la testimonianza di un interesse mondiale da parte di ricercatori appartenenti sia al mondo accademico che a quello industriale. La Ricerca ha portato alla nascita di numerose tecnologie e protocolli progettati per rispondere ai diversi bisogni degli scenari emergenti. Il risultato è che oggi è difficile sviluppare applicazioni interoperabili, proprio a causa della diversità di tutte le tecnologie impiegate nell'IoT.

Il Web of Things (WoT) è nato per rispondere a questi problemi tramite l'adozione dei protocolli standard che hanno favorito il successo del Web (come ad esempio HTTP). Ma un contributo ancora più importante può venire dal Semantic Web of Things (SWoT). Infatti, i protocolli dello stack del Semantic Web permettono l'identificazione univoca delle risorse ed una rappresentazione dei dati tale che 1) le informazioni disponibili siano computabili automaticamente 2) l'informazione di differenti fonti sia facilmente aggregabile. Le tecnologie del Semantic Web possono quindi essere considerate degli interoperability enabler per l'IoT.

Questa Tesi analizza come adottare in modo efficiente ed efficace le tecnologie del Semantic Web nell'IoT per realizzare la visione del Semantic Web of Things di una rete di smart application che sia realmente interoperabile. Più in dettaglio, mentre Part I introduce l'IoT, la sua storia ed il suo stato attuale, Part II analizza gli algoritmi per supportare efficientemente il paradigma publish-subscribe nei broker semantici per il SWoT e la loro implementazione nella piattaforma Smart-M3, così come nella sua diretta discendente nota come SPARQL Event Processing Architecture (SEPA). Inoltre, viene presentato anche il lavoro preliminare che condurrà alla definizione del primo benchmark per applicazioni di SWoT. Part IV discute l'applicazione dei risultati di questa Ricerca a scenari reali appartenenti a diversi domini

applicativi (in particolare mobilità elettrica, domotica, semantic audio ed Internet of Musical Things). Infine, in Part V vengono presentate le conclusioni sul lavoro svolto.

La Ricerca su applicazioni basate su dataset semantici ha evidenziato una carenza negli attuali software di visualizzazione ed esplorazione di queste basi di conoscenza. Quindi, in Part III viene presentata una seconda attività finalizzata all'ideazione di un nuovo metodo di rappresentazione delle basi di conoscenza semantiche basato sul diffuso approccio a grafo in cui viene introdotto il concetto di Semantic Plane.

Contents

External reviewers

Prof. **Carlos Alberto Kamienski**, Universidade Federal do ABC, Brazil,
email: carlos.kamienski@ufabc.edu.br

Prof. **Johan Lilius**, Åbo Akademi, Finland,
email: johan.lilius@abo.fi

Part I

Foreword

Chapter 1

Introduction

The **Internet of Things** is a dynamic, new generation global network infrastructure composed by heterogeneous objects equipped with identifying, sensing, networking and processing capabilities seamlessly communicating with one another to accomplish some objectives [1, 2, 3]. The IoT is one of the most disrupting innovations due to the pervasiveness of its applicability: Asin and Gascon [4] identified more than 50 application domains where the IoT is being integrated in the traditional workflow.

The IoT is the result of a long process originated by a visionary article [5] of the end of the eighties. The author, Mark Weiser (a researcher at Xerox, Palo Alto), stated that *the most profund technologies are those that disappear and become indistinguishable from every day life.* This sentence, is nearly appropriate to describe the current ICT scenario but it tooks about three decades to become reality. The first step of this long chain was known as **pervasive** or **ubiquitous computing**. Pervasive computing applications deal with environments saturated with computing and communication capabilities gracefully integrated with the users [6]. Soon after the birth of pervasive computing, the importance of the context emerged. In fact, in 1994, Schillit [7] et al. proposed the first definition of context (based on examples) and introduced **context-aware computing** as a mobile, distributed computing system able to react and adapt to changes in the environment. Over the years, several definitions of **context** appeared, but the most accepted [8] is the one provided by Dey and Abowd [9]:

> *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.*

Then, a system is said to be **context-aware** (definition still provided by Dey and Abowd

in [9]) if *it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.* Several ways to model the context appeared in literature over the years [8], but the importance of the context is still unchanged. Context-aware computing is then still alive, but is now turning into the Internet of Things. The term IoT is not simply a more attracting name for context-aware computing. A subtle difference has been identified by Naito in [10] in the propensity of the IoT of fostering cooperation among different services. Nevertheless, this is where the IoT is currently failing [11].

The impressive and rapid diffusion of the IoT, a novel and not yet consolidated paradigm, originated an high amount of protocols to face different needs in the new scenarios (e.g., the need for real-time communication, the need for secure communication in scenarios composed by a huge number of heterogeneous devices, the need for protocols characterized by a small bandwidth and computational requirements [12]). Then, today one of the most challenging issues is achieving interoperability among devices as well as applications leveraging on different protocols [13]. Indeed, in literature, IoT applications appear as vertical silos, isolated structures where communication and cooperation is hard [11], resulting in a lack of interoperability. **Interoperability** involves three levels: network, syntax, and semantics. Network interoperability [13] is about protocols for exchanging information among heterogeneous devices, regardless of the content of the messages. The syntax interoperability (or messaging protocol interoperability) [13] level concerns the way messages are structured and encoded. The third level conveys the meaning of the exchanged messages and is also known as semantic interoperability that can be achieved if meaning of data can be interpreted independently from the process [13, 14, 15]. Interoperability in an IoT scenario can be achieved only if a set of standardized protocols is employed. According to Tim Berners-Lee's five-star model [16], information should be visible, structured and described according to standards and its meaning clarified by a common definition (i.e., an ontology).

A new research area was born to address the problem of interoperability: the Web of Things. It can be considered as a refinement of the Internet of Things [17], a complementary part of it. It was born to solve the extreme fragmentation of the IoT [18] world through well-consolidated technologies, the ones that made the web so popular today (e.g., HTTP, REST). Being the Web of Things in its early stages, no unanimous approaches to the design, characterization and evaluation of WoT applications have been defined. The Web of Things, definition officially coined in 2009 [19], has started gaining popularity only two years ago: in the end of 2016 in fact, the W3C created a Working Group[1] and an Interest Group[2]

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

for the Web of Things. Leading enterprises from all over the world (e.g., Siemens, Google, Samsung, Panasonic, Intel just to name a few) participate to these groups aimed at standardizing the IoT. The paradigm of the WoT is mainly centered on the concept of **Thing Description** (TD) [15, 20], a detailed profile of all the properties, events and actions exposed by a device. But, without semantics, is the WoT enough to address the problem of interoperability?

While the current specifications [21] for the Web of Things aim at finding a good trade-off between machine-understandability and ease of development, limiting the adoption of semantics to a few tasks, the **Semantic Web of Things** (SWoT) [22, 23, 24] definition is more oriented to an intensive use of Semantic Web technologies to enable horizontal integration and composition of applications over the Linked Open Data cloud [25]. The SWoT can be considered as the next step in the long path that brought from pervasive computing to the field currently known as IoT. It is a very new research area (as the first papers date back to less than ten years ago [22, 23]) and is still in its early stages. Among the research areas underlying the SWoT, a crucial role has been played by the Web of Things introduced by Dominique Guinard [26] and Vlad Trifa [27], while of course a paramount building block is represented by the Semantic Web.

The **Semantic Web** [28] was born to transform the Web from a repository of human-readable data, to a world wide network of machine-understandable information. This can be achieved through the protocols in the Semantic Web Stack: Resource Description Framework (RDF) [29] states that all the information must be represented as a set of triples (i.e., *subject*, *predicate*, *object*) where resources are univocally identified through IRIs (International Resource Identifiers). Ontologies (*formal explicit descriptions of concepts in a domain of discourse* [30]) can be represented according to RDF Schema (RDFS) [31] and Web Ontology Language (OWL) [32] and their role consists in binding meanings to RDF terms (with a set of rules expressed through RDF). Finally, SPARQL Query [33] and Update [34] languages allow retrieving and updating data in the knowledge base (KB). This set of protocols and standards allows representing the knowledge base of every application in a structured way that can be exploited to connect all the graphs into a wider structure, according to what is now known as Linked Data [25].

How can Semantic Web be applied to IoT applications? The concept of Thing Description introduced by the Web of Things can be borrowed to semantically describe all the devices involved in a SWoT application. Moreover, Semantic Web technologies permit the easy integration of different applications that can be bridged by means of proper ontologies. Then, Semantic Web technologies grant the maximum expressive power, but the price to pay is the

computational complexity. Choosing the right strategy to model the application context is usually a trade-off between the simplicity and efficiency of the representation and the expressive power of the adopted method. The diffused diffidence about the application of Semantic Web technologies to the Internet of Things is then mainly imputable to the verbosity and complexity of its formalisms and to the consequent poor level of performance that requires proper expedients [35, 36, 24, 37, 23].

My Research work is framed in the area of Semantic Web of Things with three main Research topics:

- Efficient and effective ways to employ semantics in the Internet of Things: the application of technologies borrowed from the Semantic Web to the Internet of Things is still an open Research area where a number of problems must be addressed. One of these is how to efficiently develop SWoT applications. The work presented in Chapters 3 and 4 describes my Research activity concerning the development of publish-subscribe Message-oriented Middlewares (MOMs) [38] for the Semantic Web of Things. The Research on SWoT architectures was validated on the field through the real-life use cases belonging to different application domains (i.e., electro-mobility, smart homes, semantic audio, Internet of Musical Things). These activities, carried out throughout the entire duration of the PhD, are described in Part IV.

- Evaluation of SWoT architectures: since the development of SWoT architectures is still in its early stages, there is a lack of specific benchmarks in this area. SWoT may be considered as the convergence point of several research areas (i.e., context-aware computing, IoT, Semantic Web, publish-subscribe architectures) where ad-hoc benchmarks already exist. Nevertheless, they are not suitable for this new emerging research field. During my PhD, I identified the need for a specific benchmark aimed at assessing the performance of every software component in a SWoT application, in relation to a particular SWoT scenario. In Chapter 5 I describe my Research in this area (still ongoing), whose objective is the definition of the first benchmark for SWoT applications.

- Visualization methods for semantic KBs: the development of applications based on Semantic Web knowledge bases requires effective tools to explore and debug the information shared among every software component. During my PhD, I have studied the problem of effective visualization methods for RDF graphs proposing a novel approach aimed at extracting information from complex datasets both in Semantic Web and SWoT applications. This research activity is detailed in Chapter 6.

The work described in this Thesis (except Chapter 10) has been carried out within the Advanced Research Center on Electronic Systems (ARCES) of the University of Bologna (Italy) under the supervision of Prof. Tullio Salmon Cinotti, in partnership with:

- Petrozavodsk State University[3] (PetrSU), St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences[4] (SPIIRAS), and Information Technologies, Mechanics and Optics University[5] (ITMO), Finnish-Russian University Cooperation in Telecommunications Oy (FRUCT) (activities described in Chapter 4);
- Eurotech[6], Siemens AG[7], Centro Ricerche Fiat[8] (CRF), Bitron[9] and Gewiss[10] (activities described in Chapter 8);
- ST Microelectronics TR&D SPA[11] (activity described in Chapter 9).

The work described in Chapter 10 has been carried out within the Centre for Digital Music (C4DM) of the Queen Mary University of London (QMUL) under the supervision of Prof. György Fazekas.

The research activities carried out during these three PhD years has led to the publication of six articles on international peer-reviewed journals, twelve international conference papers and four book chapters (three published, the last stil under review) and to six oral presentations as well as two demos at international conferences.

The rest of the Thesis is organized as follows. This Part (i.e., Part I) introduces my Research topics and the related background. Part II, as previously mentioned, presents my activity concerning the development of Semantic architectures for the SWoT (Chapters 3 and 4) as well as the design and implementation of a benchmark oriented at SWoT applications (Chapter 5). A novel approach to the visualization of semantic KBs is described in Part III. Part IV describes my activity regarding the application of the SWoT paradigm to electromobility (Chapter 8), home automation (Chapter 9) and sound domain (Chapter 10). Finally, in Part V, conclusions are drawn.

[3] https://petrsu.ru/en
[4] http://www.spiiras.nw.ru/en/
[5] http://en.ifmo.ru/en/
[6] https://www.eurotech.com
[7] https://www.siemens.com/uk/en/home.html
[8] https://www.crf.it
[9] http://www.bitron.net
[10] https://www.gewiss.com
[11] https://www.st.com

Chapter 2

Background

Contents

This Chapter introduces the background of this Thesis, starting from context-aware computing and Internet of Things and then moving towards the Semantic Web of Things stepping through the Semantic Web. Lastly, the reference platform for the research activities carried out during the PhD will be introduced.

2.1 Context-aware computing

Context-aware computing refers to the ability of an application of exploiting context information to adapt its behaviour and provide services without explicit user intervention. Therefore, due to the centrality of context, a definition must be provided. Many are those available in literature, but one of the most commonly accepted, as already mentioned in the Introduction, is: "*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.*" [39]. That said, context information may include location (that is the also strictly related to the first example of context-aware application [40]), time, identity, weather conditions, etc. A representative

example of context-aware applications (other than the one already mentioned) could be the work by Bardram [41] who proposed a medical environment equipped with a pill container and a bed, both able to react to the context (represented by the information of the patient and who's near the bed). Zhang et al. in [42] proposed an OSGi-based service infrastructure for smart homes where the context is acquired from a wide variety of digital and physical sources. The retrieved information is then processed to provide a set of services like fall-detection alert based on audio/video analysis or phone call forwarding if the house owner is sleeping. More recently, Wan et al. [43] proposed a context-aware architecture for vehicular cyber-physical systems encompassing vehicles, drivers, passengers and traffic authorities. Context-aware applications may rely on a wide set of platforms providing context management capabilities like the Context Toolkit [44], the COntext BRoker Architecture (COBRA) [45, 46], UBIROAD [47] or Smart-M3 [48] (that is the reference platform of the PhD research activities described in Thesis).

Several ways of modeling the context have been proposed in literature [49, 8, 50], among which the key-value model (the simplest one), the object-oriented model (exploiting the techniques used in programming, like encapsulation, inheritance and re-usability), the logic-based model (where the context is represented in terms of facts, expressions and rules) and the ontology-based [51]. The latter is based on the use of semantic technologies (that will be introduced later on in Section 2.3). Ontologies provide an uniform way for specifying the core concepts of the model as well as facts; they also enable knowledge sharing and reuse and automatic processing through reasoning engines.

Whatever the chosen context model is, the context follows a cyclic path identified by the following steps [8] (also depicted in Fig. 2.1):

- **Acquisition** – where data is collected, e.g., by measuring a physical phenomenon with a sensor;
- **Modelling** – where the collected data is represented according to an agreed format (e.g., a given ontology);
- **Reasoning** – is the act of inferring new knowledge from the collected data;
- **Dissemination** – the distribution of the context to the entities involved in an application.

According to Schilit et al. [52], context-aware applications are the product of two points along two orthogonal dimensions, represented in Tab. 2.1. Information

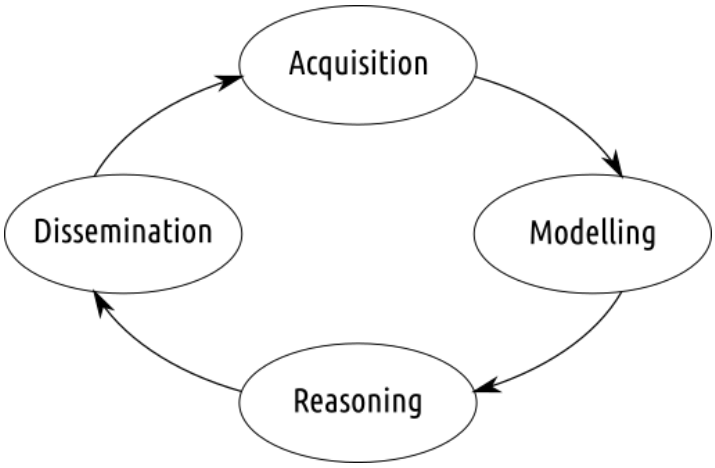


Figure 2.1: Context lifecycle

	Manual	Automatic
Information	Proximate selection	Automatic contextual reconfiguration
Command	Contextual commands	Context-triggered actions

Table 2.1: Context-awareness dimensions [7].

2.2 Internet of Things

The term IoT was first introduced by Ashton in a presentation made in 1998 [53]. Despite twenty years of Research on the IoT, a uniform definition is still missing, but it is worth mentionting:

- "*The Internet of Things allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service.*" [54]
- "*The semantic origin of the expression is composed by two words and concepts: Internet and Thing, where Internet can be defined as the world-wide network of interconnected computer networks, based on a standard communication protocol, the Internet suite (TCP/IP), while Thing is an object not precisely identifiable. Therefore, semantically, Internet of Things means a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols.*" [55]

The IoT was born as the unavoidable evolution of context-aware computing to large scale scenarios and it was mainly fostered by the fast diffusion of low cost devices and technologies like the RFID, Near Field Communication (NFC), Bluetooth Low Energy (BLE), ZigBee that opened new scenarios. What makes the IoT one of the most disrupting ICT revolutions is the

wide range of application areas where it is being applied. In fact, already in 2012, Asin and Gascon [4] more than fifty application domains that, according to Miorandi et al. [56], can be grouped into six macro areas (where boundaries are not always clear):

- **Smart Buildings** – This area includes applications aimed at enhancing the comfort level of residents (e.g., through smart entertainment systems) as well as reducing the energy consumption (e.g., with advanced policies to automatically control appliances). A prosperous areas is that of Heating, Ventilation, Air Conditioning (HVAC), also described in Chapter 9.

- **Smart Cities** – Applications related to mobility, in the sense of optimizing the use of the road infrastructure and quality of life of citizens. Interesting applications include monitoring (traffic congestion, air quality, temperature, water pressure and trash bin level [57]), smart parking (to face the problems related to the increasing need of parking spaces in large cities [58, 59]), waste management (e.g., intelligent trash bins that communicate the level of load to permit to efficiently organize the routes of the garbage trucks [60, 61, 62]), tourist recommendation (e.g., exploiting RFIDs and mobile phones to provide information to the tourists when they get close to a Point Of Interest (POI) [63, 64]).

- **Healthcare** – Healthcare is considered one of the killer applications for the IoT. Examples of healthcare IoT applications are remote health monitoring, ambient assisted living, fitness programs, just to name a few of the many applications made possible by the spread of wearable sensors. In remote health monitoring, monitored patients wear sensors that through proper Wireless Sensor Body Networks communicate their readings to central Health-Care Records where medics can supervise the most important health indicators as in [65] or [66]. Remote health monitoring facilitates elderly and disabled people, often unable to easily reach the medics and helps reducing the queue at doctor's office and hospitalization costs. It is also important for doctors to be notified in case of the so called Adverse Drug Reaction (ADR) [67, 68]. Remote health monitoring also enables collecting predictive information about diseases [69]. Aging and incapacitated individuals may benefit from IoT applications that help them feeling confident and safe in their place of living ensuring a greater autonomy. This is the purpose of Ambient Assisted Living (AAL) applications [70, 67].

- **Logistics** – This is the area where RFID were succesfully applied for the first times [71]. In fact, RFID were employed to monitor production and shipping of goods, and this

area is now a well consolidated pillar of the IoT. More specifically, in the food domain, the Food Supply Chain (FSC) is now emerging as an important IoT application domain: it is heavily distributed and complex, it has large geographical and temporal scale, complex operation processes, and large number of stakeholders. Challenges represented by traceability, visibility, and controllability will be addressed thanks to IoT technologies. The Internet of Things will be fundamental to realize the so-called farm-to-plate monitoring: from precise agriculture, to food production, processing, storage, distribution, and consuming [72].

- **Environmental Monitoring** – This area is mostly related to sensing for physical phenomena and processes to detect anomalies that can affect the environment. Environmental monitoring is also important to prevent disasters affecting other application areas, like Smart Grids [73].

- **Security** – Mainly related to surveillance, this application area is growing with a high number of applications exploiting Internet-connected cameras and sensors. Relevant examples of this application domain real-time security systems like the one proposed by Jyothi et al. in [74] or crowd-based systems like [75].

To understand the potentiality of this new Research area, it is sufficient to observe the trend of the latest years and read about the foreseen statistics: Intel[1] states that by 2020 smart devices should be about 200 billion, one hundred times more of the number of smart objects in 2006. Moreover, by 2025, the expected total global worth of IoT is expected about 6.2 trillion dollar; a relevant part of this share is for healthcare (2.5 trillion dollar) and manufacturing (2.3 trillion dollar). The European Union, in fact, with its Horizon 2020 European research and innovation programme, has invested almost 200M euros in Internet of Things research in the period that goes from 2014 to 2017[2] and many projects have been started in January 2017 (with a financial contribution of 100M euros, like ACTIVAGE (Smart living environments for ageing well), IoF2020 (Smart Farming and Food Security), MONICA (Wearables for smart ecosystems), SYNCHRONICITY (Reference zones in EU cities), AUTOPILOT (Autonomous vehicles in a connected environment). Regarding projects born under the wing of the EU, FIWARE deserves a mention. FIWARE is an open project sponsored by the Future Internet Public Private Partnership (FI-PPP) of the European Commission. The FIWARE platform is based on elements called Generic Enablers (GEs), reusable and shared modules for multiple usage areas. The back-end architecture of FIWARE is mainly based on two different

[1] https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html
[2] https://ec.europa.eu/digital-single-market/en/research-innovation-iot

modules that are the Orion Context Broker and Cosmos. The first is a context broker providing two REST API interfaces (NGSI9 and NGSI10) that allow updating, retrieving and (un)subscribing to the context. The latter is instead an the big data storage and analysis intended to deploy means for analyzing both batch and stream data. FIWARE also provides Cygnus and Short Time Historic (STH) to allow storing and retrieving historical data [76]. Among the projects realized with the FIWARE platform it is worth mentioning the SmartPort project (a web platform integrating the tools for the analysis and visualization of the sensors of the Las Palmas de Gran Canaria seaport) [77] and SWAMP (Smart WAter Management Platform) [78].

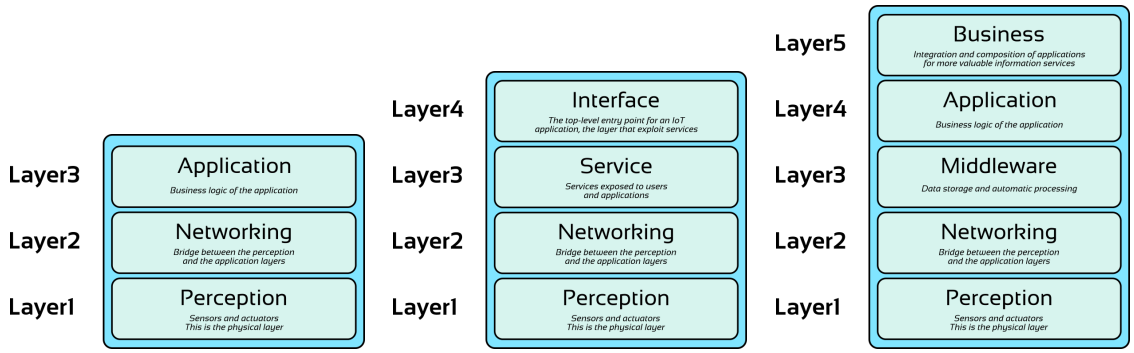
In the last years, also leading enterprises in the ICT have proposed their platforms like AWS (Amazon), Azure (Microsoft), Watson (IBM), ARTIK (Samsung), just to name a few. Enterprises usually propose a wide set of tools from devices to cloud storage and services to make building IoT applications easy also for novice users. The latters can count on a wide variety of low-cost devices like Arduino, Raspberry PI, BeagleBone Black as well as IoT platforms (e.g., ThingSpeak, Ubidots and Cayenne) to share and manipulate data, well surveyed by Singh and Kapoor in [79].

On the technical side, all the IoT applications intrinsically rely on **layered architectures** [13, 80]. The most common structure for IoT applications is made up of three layers [8] (summarized in Fig. 2.2a): the **perception** layer, the **network** layer and the **application** layer [81, 82]. This is not the only feasible partition of IoT applications, since in literature it is possible to find approaches based on four or more layers, well resumed by Al-Fuqaha et al. in [80].

The **perception layer** is crowded by sensors and actuators that constitutes the physical layer of the application. It is used to collect data from the environment as well as to control physical devices acting on it. Sensors and actuators may be directly connected to the networking layer or may be part of a Wireless Sensor and Actuator Networks bridged to the upper layer. The **networking layer** acts as a bridge between the perception and the application layers making possible for the applications to get data from devices and to control them. The networking layer is in many cases responsible of producing an high level abstraction of the underlying networks. Gateway functionalities aim at providing a uniform view of the network are part of the networking layer. The **application layer** is where the business logic of the application resides. This architecture is used by Domingo to survey many different IoT projects for people with disabilities [83]. In [84] an energy efficient architecture for the Industrial IoT (IIoT) is presented. The architecture is made up of three layers named sense layer, gateway layer and control layer. Despite the names are different they can be assimilated to the three

layers introduced at the beginning of the Section. In [85] a big health system architecture is presented and the approach chosen by authors is, again, based on a three-layered architecture: perception, transport (or network) and big health cloud which contains the applications as a sub-layer. In [86] the three-layered architecture is presented as a **generally accepted structure** with the usual perception, network and application layers as well as in [82] (but they also propose an enhanced representation based on five levels).

Despite being very easy and intuitive, this structure can be limiting to describe an IoT architecture where the application layer is, for example, made up of many interacting services that can be, themselves, split across multiple levels. This is why in literature it is possible to find approaches with more than three levels. Indeed, in [71] a four-layered architecture is proposed. The lower level is occupied by the sensing and actuating technologies. As usual, on top of the sensing layer there is the networking layer. The service layer is the third one: it creates and manages services to satisfy the user needs that access them through the fourth level: the interface layer. Fortino and Russo in [87] proposed a four-layered architecture for the IoT where the perception layer is named Smart Object, the networking layer is named Internet. Then the Middleware and the Application layers conclude the structure. Four layers were also identified by Qiu et al. in [88] as the basis for the so-called Heterogeneous Internet of Things (HetIoT). Several different research works (e.g. Khan et al. in [89], Wu et al. in [82]) proposed a five layered architecture (see Fig. 2.2b) for the IoT. In [86] the five layers (see Fig. 2.2c) are the usual couple perception-network layers followed by middleware, application and business layers. While perception, network and application do not need to be further discussed, the middleware layer deserves an introduction: it is intended for information storage and processing (for automatic decision). The output of the middleware level is used by the application layer that performs the final presentation of data. The business layer is aimed at integrating and composing different applications to provide more valuable information that can be used as a source of money (this is usually referred to as orchestration). This architecture is well described by Wu et al. in [82]. Despite introducing the three-layered architecture as a good and generally accepted structure, they consider it suitable only for the initial stage of development of an IoT application. Combining the analysis of Telecommunication Management Network and TCP/IP models they defined the subsequent five layered architecture. An even more complex architecture with seven layers has been proposed by CISCO [13].



(a) 3-layered **(b)** 4-layered **(c)** 5-layered

Figure 2.2: Layered architectures for the Internet of Things

2.3 Semantic Web

The Semantic Web was conceived by Tim Berners Lee [28] as a way to transform the Web into a repository of machine-understandable data in order to permit the development of smart applications.

In order to achieve this scope, a stack of protocols was introduced (and is still a work in progress). The main elements of the Semantic Web stack (see Fig. 2.3) are here quickly overviewed from bottom to top:

- the bottom layer is composed by **Internationalized Resource Identifier** (IRI) [**?**] that provides a way to uniquely identify resources on the web. In the bottom layer there's also **Unicode** that allows representing and manipulating text.

- **RDF** [90] is a protocol that describes how the information should be represented. According to RDF all the information should be formalized as a set of triples composed by a subject, a predicate and an object. XML (placed just below RDF) allows the serialization of triples.

- **RDFS** [31] and **OWL** [32] provide the constructs for the definition of vocabularies or ontologies. An ontology according to the definition by Noy et al [30] is:

  > *a formal explicit description of concepts in a domain of discourse (classes), properties of each concept describing various features and attributes of the concept and restrictions*

  An ontology allows writing meaningful RDF triples or to correctly interpret them. Both RDFS and OWL ontologies are formalized as sets of RDF triples.

- The **SPARQL** block provides a query [33] and an update language [34] for RDF knowledge bases.

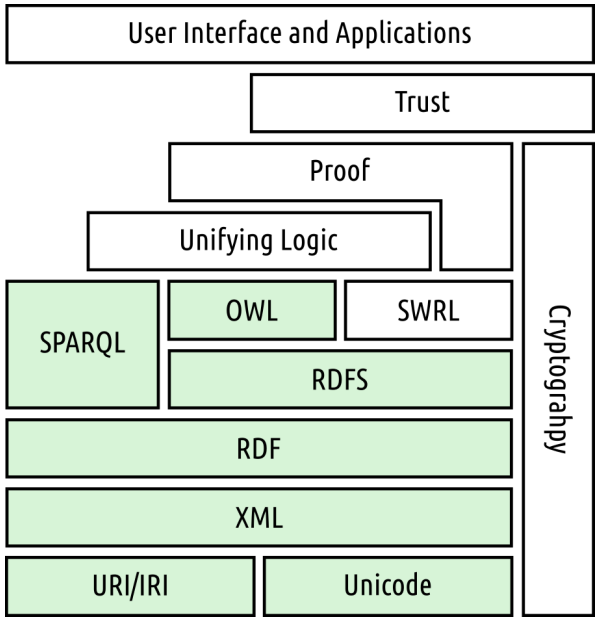


Figure 2.3: The Semantic Web stack. The color green is used to highlight the Semantic Web blocks often used in the IoT, and those I will refer to in the rest of the Thesis.

One of the most notable examples of Semantic Web applications is DBpedia [91], a semantic version of the well known website Wikipedia. DBpedia contains, as of October 2018, over 4.2M RDF resources and provides a public SPARQL endpoint to retrieve data through SPARQL queries.

2.4 (Semantic) Web of Things

The **Web of Things** aims to solve the problems of the current IoT in terms of interoperability among heterogenous devices and applications. The WoT, coined by Guinard and Trifa [19], exploits the standard protocols that made the Web popular to foster interoperability. Among these standards, a central role is played by HTTP, JSON and REST (also due to the semplicity of development of RESTful applications [92, 93]).

The **Web of Things** can be imagined as a layer standing upon the IoT [94]: no matter how things connect to the Internet, it just focuses on the way things can be accessed and programmed. An alternative vision is that of a layered architecture, partly overlapping with the IoT stack of Fig. 2.2c, as shown in Fig. 2.4.

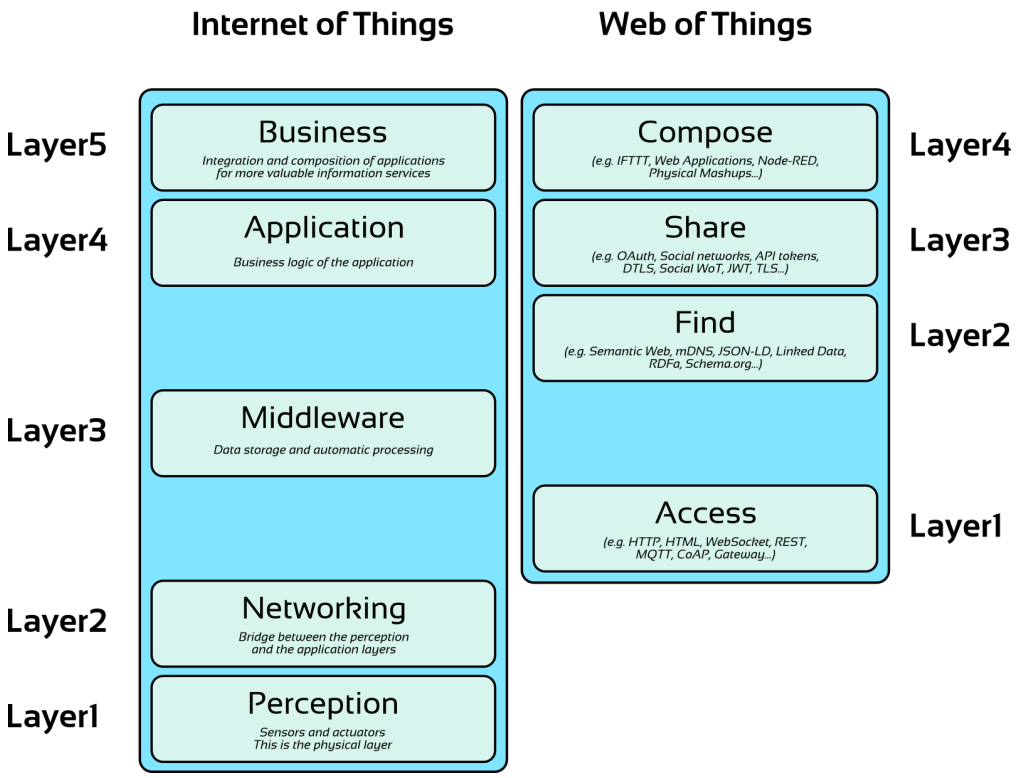


Figure 2.4: IoT and WoT: how do they relate

The WoT leverages the concept of **Thing Description** to map every smart device in an application. The smart device with its digital representation and its software agent is then defined a **Web Thing**. A Thing Description is a detailed document containing:

- **properties**: readable/writable static/dynamic data (e.g. a sensor reading) exposed by the Web Thing;
- **actions**: invokable commands of the Web Thing that may require a set of input data and may provide output data. take a certain time to complete (e.g. move the robotic arm);
- **events**: events generated by a Web Thing to notify about certain conditions (e.g., a new sensor reading is available).

The most simple scenario envisions Web Things directly connected to the network and reachable through an unique URI (that is also the entry point for the TD). This is known, in the WoT terminology, as **direct integration pattern** (Fig. 2.5). However, this is not the only possible one. In fact, devices may rely on a gateway (Fig. 2.6) that exposes its TD and an HTTP/WS interface. In this case, named **gateway integration pattern**, the

channel between the device and the gateway may be realized through any technology (e.g., Bluetooth). A third and last pattern proposed by the W3C is the so called **cloud integration pattern** cloud service that exposes functionalities (Fig. 2.7), where a cloud service exposes the Thing Descriptions of one or more devices and allows client to interact with them through a standard HTTP/WS interface. The communication between things and cloud happens through HTTP/WS.

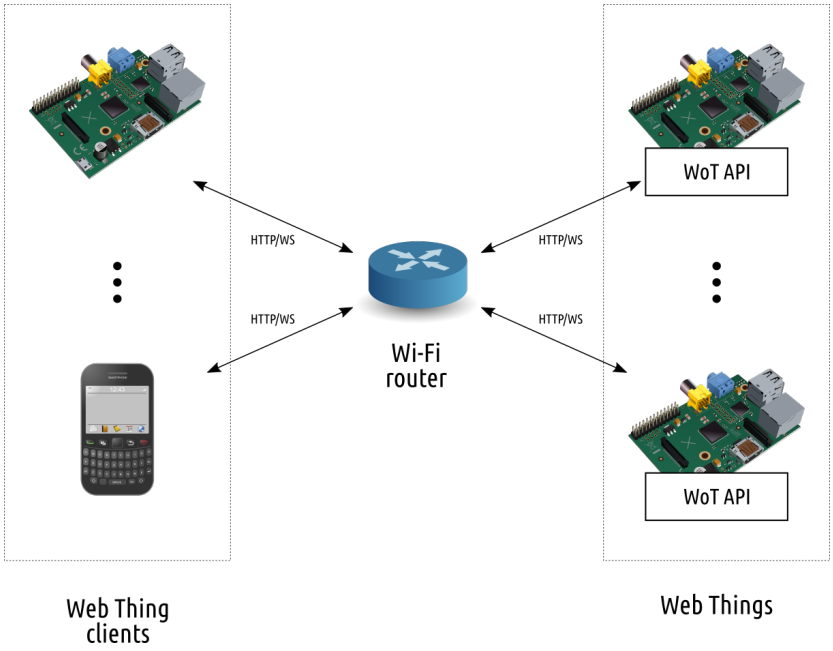


Figure 2.5: Direct integration pattern

The Web of Things, as intended by W3C Working and Interest Groups, exploits semantics for annotations [95, 96, 21]. But standards borrowed from the Semantic Web may further increase the interoperability level of WoT applications. The so called Semantic Web of Things aims at exploring this possibility [23, 22]. My Research activity is framed in this context where semantic technologies are employed to fully map Web Things' TDs to provide a powerful way to discover things. Moreover, Semantics may be exploited to support a new advanced semantic integration pattern (as will be detailed in the next Chapters).

2.5 The Smart-M3 interoperability platform

Smart-M3 is an interoperability platform developed to address the needs of semantic context-aware applications. It was initially framed in the ARTEMIS Joint Undertaking (JU) project Smart Objects for Intelligent Applications (SOFIA) and is currently developed by the ARCES

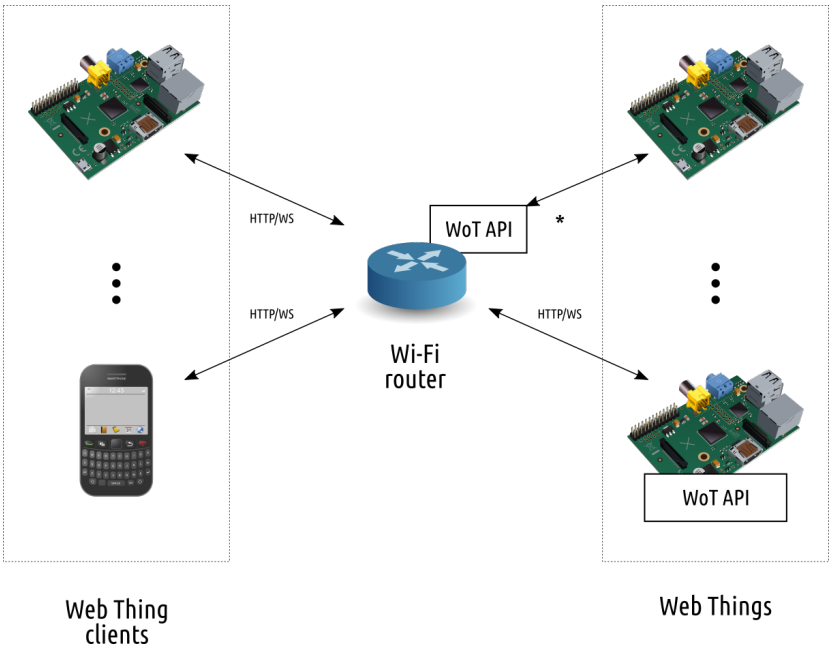


Figure 2.6: Gateway integration pattern

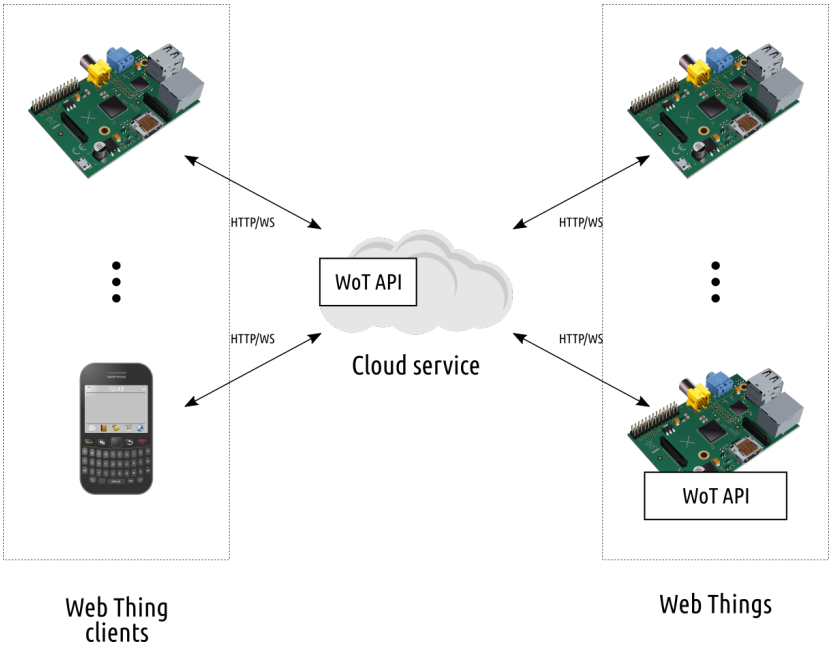


Figure 2.7: Cloud integration pattern

department of the University of Bologna together with other Russian and Finnish Universities. Smart-M3 is a client-server architecture where the server takes the burden of hosting and sharing the context through an RDF graph, providing also the ability to subscribe to changes. Applications built around the Smart-M3 platform envision two types of entities: the

server (i.e., the context broker) that is named Semantic Information Broker (SIB) and the clients, known as knowledge processors (KPs). An example of the architecture of a Smart-M3 application is reported in Fig. 4.1. KPs interoperate through the SIB by means of messages exchanged through the Smart Space Access Protocol (SSAP) and embedding SPARQL update, query or (un)subscription requests.

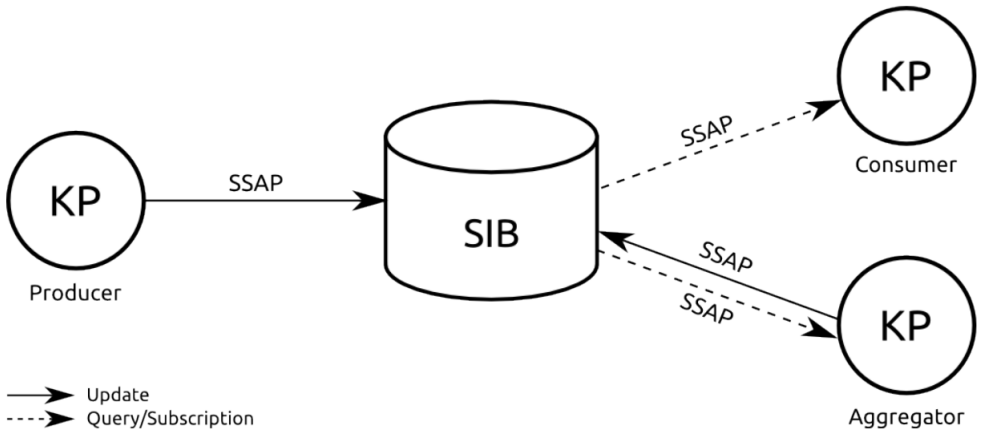


Figure 2.8: The Smart-M3 architecture

Referring to Perera's context lifecycle and using the ontology-based context modelling, the Smart-M3 platform allows modelling the information (through proper client-side libraries and a set of ontologies), disseminating the context (through the server interface) and reasoning (through client-defined primitives running on the server). Context acquisition is instead carried out by physical and virtual sensors on a lower level. The development of Smart-M3 applications takes place through a set of API developed in Python, Java, Javascript, C, Ruby and thanks to a debug and inspection dashboard.

Smart-M3 has been successfully employed in a series of European Research Projects, such as RECOCAPE (REinforcing COopeartion CAPacity of Egypt in embedded ubiquitous computing) [97], IoE (Internet of Energy) [98, 99], CHIRON (Cyclic and person-centric Health management: Integrated appRoach for hOme, mobile and clinical eNvironments) [100], ARROWHEAD [101, 102, 103]. Activities related to the latter are also described in Chapter 8. In literature, several examples of applications developed through the Smart-M3 platform are available: Smart Conference [104] (tool to assist conferencing process online) and Smart Scribo [104] (to access the blogosphere), TAIS [63] (for e-Tourism).

The Smart-M3 platform is now evolving to the SPARQL Event Processing Architecture (SEPA) to provide a better support to Big Data and SWoT applications. The evolution of the platform has been one of the main activities of the second year of PhD.

Part II

Semantic publish/subscribe middlewares

Chapter 3

Subscriptions processing: algorithms and mechanisms

Contents

3.1 Introduction

Applying Semantic Web technologies to the development of context-aware or SWoT applications means adopting a representation of the context of an application as a set of RDF triples

according to one or more ontologies. This is the previously mentioned **ontology-based context modelling** [8]. The context [9] is the building block on which every component of an application relies on. Moreover, in these applications, the context is the result of the cooperation of multiple nodes (e.g., the presence of fire in a building is the result of the information produced by every fire detection sensor installed inside it). For this reason the context should be available to all the nodes for both reading and writing, and every notification to the context should be timely propagated to the interested entities.

In the rest of the Thesis, I will focus on client-server architectures, and more specifically **broker-centric architectures**, where the **broker** (or *context broker*) is the central repository hosting and providing access to the semantic representation of the context. Then, an essential component of the context broker is an **RDF store**. The RDF store holds the knowledge base (KB) that is composed by a set of triples $T = \{(s, p, o) : s \in \{I, B\}, p \in I, o \in \{I, B, L\}$ where U is the set of all the possible IRIs, B is that of the possible blank nodes and O is the set of all the possible literals. With respect to Fig. 2.1, the context-broker is responsible for the dissemination of the context, while the clients sense the environment (i.e., acquisition phase), then represent the information according to the agreed ontology and publish it in the broker (i.e., modelling). Reasoning functionalities may be provided by both clients and brokers.

If we consider the broker as a **SPARQL endpoint** (i.e., a server holding an RDF graph and providing a standard SPARQL 1.1 interface), then the context can be produced through SPARQL 1.1 Update [34] requests and retrieved through SPARQL 1.1 Queries [33]. Detecting a change in the context by means of queries consists in polling the broker with a SPARQL query and compare results of the last execution with the previous one. This is inefficient both for the client (that should perform a repetitive set of queries and comparisons) and for the broker (that in presence of many clients is flooded by query requests). Moreover, the length of the polling interval affects the performance of the applications: if it is too short, the system is overloaded by query requests, while a time interval too long may result in slow reaction of the application.

This is why in the SWoT, I investigate the adoption of the **publish/subscribe** [105] paradigm to semantic context brokers. The publish/subscribe interaction scheme allows a client to manifest its interest in an event and be asynchronously notified when it occurs. It is ideal in large scale applications where a loose coupling between client and server is desirable. In a publish/subscribe interaction scheme, we denote with the term **Publisher** a producer of information, while a **Subscriber** is a client interested in a particular information, named **Event**. A **Notification** is the message sent by the message broker to dispatch an event to a

subscriber.

Different publish/subscribe schemes can be implemented, depending on the way the client manifests its interest. In a **Type-based** subscription system the client specifies the type of the event it is interested in. The **Topic-based** system allows to be more specific: in fact, in a system where messages also have a topic, this subscription scheme allows the client to specify the topic of the messages. **Content-based** systems provide the maximum granularity since the interest of the client can be specified in a very detailed way: if such granularity is exploited, it may allows to reduce both the number of notifications issued by the broker, and the processing required by clients to look for relevant information in the received notifications. On the other hand, it requires a more complex computation on the broker to produce and dispatch the notifications. It is then a trade off between the granularity of the event and the timeliness of the notification.

The activity presented in the rest of my Thesis is founded on content-based publish/subscribe mechanisms. In fact, clients specify their interest through a SPARQL query (indeed, subscriptions act as persistent queries) and a change of the results of the query (i.e., bindings) is the event they are interested in. The content of the notification may be the full results set of the query or just the added and removed bindings with respect to the last notification. This last strategy, that we name **delta-notification**, is of course ideal to limit the amount of data sent over the network and the processing required to the client to identify changes.

Before going further, I introduce the terminology that I will adopt in the rest of the Chapter. As previously mentioned, results of a SPARQL query are named **bindings**. Every time a new subscription request $S_i \in S$ (where S is the set of all the possible subscriptions) is issued by a client, the bindings of the corresponding query are returned. From then on, after a SPARQL Update causing modification in the subgraph interested by S_i, only the **added bindings** and the **removed bindings** are returned to the subscriber. If we name t_j the time instant of the SPARQL Update and $R(t_j)$ the bindings of the query corresponding to S_i at time t_j, then the added bindings are represented by $R_i(t_j) \setminus R_i(t_i)$, while removed bindings are $R_i(t_i) \setminus R_i(t_j)$ where t_i is a generic time instant before the execution of the SPARQL Update. Added and removed bindings are determined by a process named **subscription processing unit** (SPU). For every new subscription request, a new SPU is allocated by the context broker.

In the rest of the Chapter, I will focus on the possible algorithms to implement the publish/subscribe paradigm on top of a standard SPARQL Endpoint, starting from the most intuitive one (Subsection 3.3). Then, my contribution consisting of a set of optimizations, is presented. A first optimization is based on the use of filtering tables and local context

stores (Section 3.4). A second one, enhances the performances of the previous approach by introducing the ability to group similar subscriptions (Section 3.5), while a third optimization based on a hierarchical data structure is proposed in Section 3.6.

3.2 Related work

In literature it is possible to find several examples of semantic architectures providing a subscription mechanism. In this Section, an overview of the projects most related to my Research is presented.

3.2.1 Event-based subscriptions

In [106], Murth and Kühn have proposed SENS (Semantic Event Notification Service) a semantic publish/subscribe middleware where clients specify the subgraph of interest by means of SPARQL Construct queries. Whenever new data is added to the knowledge base, the engine re-evaluates all the subscription queries to check whether some of them return new results. At a glance, SENS is very similar to Smart-M3 (and SEPA), but there are several remarkable differences: 1) Smart-M3 and its descendent are not limited to SPARQL construct queries to specify the subgraphs of interest, but allow clients to use the whole SPARQL Query language; 2) as a result, Smart-M3/SEPA do not return triples, but bindings; 3) SENS, for every triggered subscription, returns the whole results set, while Smart-M3/SEPA adopt the so-called delta notifications where only the added and removed information is included in the notification message; 4) SENS only detects added knowledge, while Smart-M3/SEPA also takes into account removed information; 5) SENS supports reasoning mechanisms that are part of the event detection mechanism, while in the reference platform of this Thesis, reasoning mechanisms is not part of the event detection, but it's an additional feature.

Instans [107, 108, 109] adopts the algorithm Rete [110] to efficiently detect matching in SPARQL subscriptions. This algorithm is based on rules (in this case represented by SPARQL queries) and facts (i.e., events). A rule is composed by a left-hand side (the query) and a right-hand side (the handler). Rete forms a network based on the conditions to match. Instans performs continuous evaluation of incoming RDF data against multiple SPARQL queries. Every time a new event occurs, the algorithm starts a check of all the triples with a set of collaborating triples working in parallel. As SENS does, also Instans returns the whole results set every time a match is found. The same behaviour is shown also by EventCloud [111, 112] where a subset of SPARQL is used to express subscriptions on P2P content addressable networks. They propose two algorithms to handle both updates and subscriptions in different situations:

Chained Semantic Matching Algorithm (CSMA) and One-Step Matching Algorithm (OSMA). Only the second allows processing subscriptions in parallel and produces good results at the price of a heavier publication step. Instans is not the only research project exploiting Rete: Sparkwave [113] in fact, is a window-based approach that uses a modified version of Rete to provide continuous matching on RDF streams. More precisely, Rete was provided with an additional layer that determines schema entailments and a modified β-network to check time constraints.

Auer and Herre in [114] have presented an approach for the versioning of RDF knowledge bases, where detection of changes plays a crucial role. The aim of the project is to facilitate the human analysis od data, rather than automatic computation for change notification. As regards versioning of RDF datasets, it is worth mentioning the work by Fiorelli et al. [115] that analyzes the current state of the art in this area.

3.2.2 Window-based subscriptions

Another relevant category for the analysis of related work is that of windows-based or streaming semantic publish-subscribe systems. Groppe et al. [116] proposed the first SPARQL streaming engine. Their work exploits algebra to deal with subscriptions in a powerful and efficient way. This approach allows: 1) discarding irrelevant triples in the early state of processing; 2) creating indices only for triples relevant for the query; and 3) calculating partial results for the query as soon as possible. A second example is the one proposed in [117]. A window specifies the triples for which the query is executed and its size can be declared in terms of number of triples or time of execution. Still in the area of window-based RDF stream and event processing solutions, other approaches are those presented in [118] (known as continuous SPARQL or C-SPARQL), SPARQL Stream [119], event processing SPARQL (EP-SPARQL) [120], continuous query evaluation over linked data streams (CQELS) [121], and Sparkwave [113].

Three main aspects differentiate the work presented in this Chapter from the above-mentioned window-based SPARQL event processing approaches: 1) both Smart-M3 and SEPA do not use windows but are rather based on real-time evaluation of events within the whole system; 2) Smart-M3 and SEPA exploit the standard SPARQL query and update language respectively to subscribe and generate events; 3) The engines of Smart-M3 and SEPA detect how results of a specific subscription have changed from the last query results, while window-based approaches provide the whole results set whenever it is modified in any way.

There are other approaches that instead of supporting windows through modified versions of SPARQL, adopt the standard SPARQL language. An interesting one is the architecture

proposed by Groppe et al. [116]. They proposed algebra for handling RDF streams with SPARQL, along with several optimizations: they discard irrelevant triples in the early state of processing. Moreover, they create indices for triples relevant for the query and calculate partial results for the query as soon as possible. While Groppe et al. focused on streams, with Smart-M3 and SEPA I support holding a semantic store that makes the approach more suitable for IoT systems as the clients may join and leave the system dynamically at run-time. Again, the work by Groppe et al. adopts notifications containing the whole set of results, instead of *delta*-notifications.

3.2.3 Detecting changes in RDF graphs

Papavasiliou et al. in [122] highlight the importance of detecting changes in RDF knowledge bases and propose a new language for the formulation of concise and intuitive deltas (i.e., changes between versions of the same KB). Moreover they proposed a change detection algorithm with respect to that language. In the following Sections, a set of algorithm to process subscriptions will be presented. All of them pivot on the detection of changes in a graph, as proposed by Papavasileiou et al. [122] with the name of low-level deltas. The same authors, also provide a distinction among low-level and high-level deltas [123, 122], but they are not relevant to our scope, since they involve more complex algorithms that may brought inefficiency where high performance are needed.

3.3 A naive algorithm

A subscription can only produce notifications after a SPARQL Update operation. So, processing subscriptions is a task that follows the modification of the KB. A subscription S_i produces a notification for the changes caused by a generic update u, if and only if, the results of the corresponding query before and after u are different. So, for every subscription, the query results must be saved. The simplest algorithm we may think of to handle subscriptions can then be summarized with two functions:

1. the first to be executed whenever a new subscription request is received by the semantic broker, that performs the query to get and save the initial results;

2. the second to be executed when a new update must be processed.

The handler to process a new subscription request is:

```
1: function HandleNewSubscription(sub_text)
2:
3:     # Generate a random ID
4:     subID ← randomID()
5:
6:     # Create a new subscription object
7:     newSub = {}
8:     newSub["text"] ← sub_text
9:     newSub["queryResults"] ← queryKB(sub_text)
10:
11:    # Send confirm and results
12:    sendReply()
13:
14:    # Save the subscription
15:    subscriptions[subID] ← newSub
16:
17: end function
```

`HandleNewSubscription` receives the text of the query (i.e., `upd_text`) and saves it as a field of a new subscription object (i.e., `newSub`). The query is also executed and its results are saved in the `newSub` structure. It mantains the status of the subscription and is updated every time bindings change. After the creation of the `newSub` structure, a confirm message is sent to the client. Every thread responsible of managing a subscription is a Subscription Processing Unit.

Processing an update request can be made as in the following listing:

```
1: function HandleNewUpdate(upd_text)
2:
3:     # Update the KB
4:     updateDB(upd_text)
5:
6:     # Cycle over subscriptions
7:     for s ∈ subscriptions do
8:
```

```
        # Execute the query
        newRes = queryKB(s["text"])

        # Compare the results
        if newRes ≠ s["queryResults"] then

            # Find added/removed bindings
            nb = newRes \ s["queryResults"]
            ob = s["queryResults"] \ newRes

            # Send notification
            sendNotification(ob, nb)

            # Save new results
            s["queryResults"] = newRes

        end if
    end for
end function
```

Every time a SPARQL update request is received by the broker, every subscription processing unit is awakened. For every subscription, the corresponding query is executed and results are compared with those previously saved. If the presence of new bindings or the absence of old bindings is detected, a notification is produced.

Of course, subscriptions can be processed simultaneously, to speed up the execution.

3.4 Filtering and caching: LUTTs and CTSs

The algorithm presented in the previous Section is very simple, and can be strongly optimized. I implemented the optimizations proposed in this Section in the pySIB and OSGi SIB brokers presented in the next Chapter. The previous algorithm presents basically two criticalities:

1. After a SPARQL Update every subscription is awakened;

2. Every Subscription Processing Unit executes its query on the whole KB.

As regards point 1, we state that a pre-processing stage would allow detecting which subscriptions are for sure not involved by the ongoing update. In this way, the set of subscriptions to be processed may become smaller. This first optimization is described in Section 3.4.1. The second point is instead addressed through local context stores and is addressed in Section 3.4.2.

3.4.1 Look-up Triples Tables

This is achieved with a filtering table that we name Look-Up Triples Table (LUTT). Every subscription is then provided with a LUTT, containing the triple patterns involved in the SPARQL subscription text (variables are replaced by wildcards).

Every time an update is executed, the set of triples added and removed is matched against every LUTT and if a check returns an empty set, the related subscription is not further processed. Otherwise, the matching triples are added to a structure named Added Removed Triples Queue (ARTQ). Added and removed triples are determined through SPARQL Construct queries derived from the basic graph patterns included in the text of the update request.

This process may become clear with the following example. Consider two SPARQL subscriptions, the first to all the instances of the class `foaf:Person`:

```
1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
3 SELECT ?person
4 WHERE {
5   ?person rdf:type foaf:Person
6 }
```

and the second to all the instances of the same class and the age of every person:

```
1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
3 SELECT ?person ?age
4 WHERE {
5   ?person rdf:type foaf:Person .
6   ?person foaf:age ?age
7 }
```

The previous SPARQL subscriptions are characterized by the following LUTTs:

Subject	Predicate	Object
*	rdf:type	foaf:Person

and:

Subject	Predicate	Object
*	rdf:type	foaf:Person
*	foaf:age	*

Now consider adding only the following triple to the KB:

```
ns:Person1_URI foaf:age "15"
```

The new triple matches the second LUTT, but not the first one. So only the second subscription will be processed (i.e., with a query to discover added and removed bindings and the possible sending of a notification).

3.4.2 Local context stores

To address point two, a local context triple store (CTS) containing only the triples passing the LUTT may be bound to every subscription. More precisely, the CTS is the subset of the global knowledge base and is defined as the *union of all the RDF triples matching at least one of the triple patterns of the subscribe graph patterns*. Using an CTS, it is possible to reduce the size of the graph to be queried, thus reducing the time to perform the query. The CTS is related to the SPARQL endpoint as a cache is related to the main memory in a pc. Each SPU corresponds instead to a processor with its own cache in a multiprocessor system.

3.4.3 The Booster

With the system just described, every time an SPU is awakened, the related CTS is updated and the query corresponding to the subscription is performed. Then, a comparison among all the current results and the previous results should be performed. A further optimization is represented by the **booster**, that modifies this process to speed up the detection of added and deleted bindings that should be notified.

The booster performs a SPARQL query on the CTS for each triple extracted from the ARTQ. Before performing the query, the booster binds as many variables as possible using the triple content. In this way: 1) the uncertainty of the query is reduced; 2) matching triples are for sure bindings to be notified, thus removing the need to compare new query results with the old ones.

3.4.4 Discussion

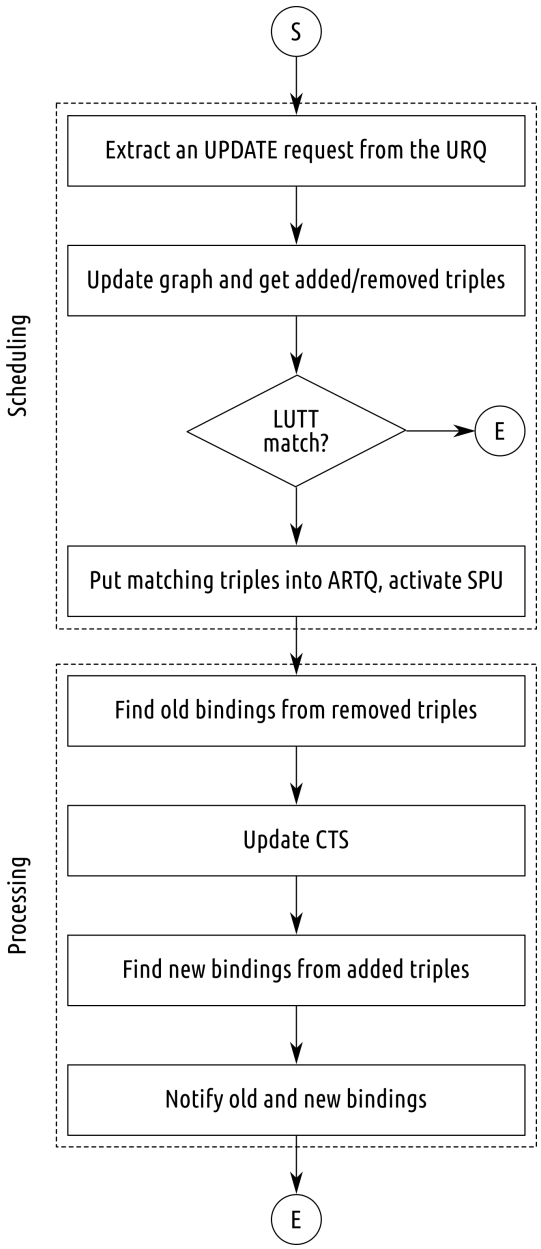


Figure 3.1: SUB Engine Workflow

Both the LUTT and CTS permit to speed up the subscription processing. The drawback is represented by the higher memory occupation to store all the LUTTs and local CTSs. The results of this research activity have been presented in [124]. In the following, the algorithm described in this Section will be referred to as LUTT-based.

As regards the complexity, a comparison between the algorithm here proposed and the naive one is now proposed. For every SPU the naive algorithm has a complexity of:

$$T_{NAIVE} = N^2 T_{CMP} + T_{QUERY}$$

where N is the number of binding results, T_{CMP} is the time required for a single comparison. T_{QUERY} is instead the time required to perform the query on the RDF store. The complexity of the LUTT-based algorithm is instead:

$$T_{ALG} \sim N_{TRIPLES} \cdot (T_{MATCH} + T_{QUERY^*} + T_{MERGE})$$

where:

- $N_{TRIPLES}$ is the number of triples in the ARTQ;
- T_{MATCH} and T_{MERGE} are the time components related to the comparison of a triple with a query (to bind variables) and merging of the results;
- T_{QUERY^*} is the time elapsed by the reduced query.

Considering the following assumptions:

- $T_{MATCH} + T_{MERGE} \ll T_{QUERY^*}$;
- $T_{QUERY} = QT_{QUERY^*}$ (where $Q \gg 1$ as the same SPARQL query with a lower number of variables is always faster than the original one);
- $N_{TRIPLES} \ll N$ (the updated triples are usually a few if compared to the number of bindings);
- the algorithm to detect events is executed twice to find both added and removed bindings;

then the speed up can be calculated as:

$$Speedup = \frac{T_{NAIVE}}{2 \cdot T_{ALG}} = \frac{N^2 T_{CMP} + QT_{QUERY^*}}{2 \cdot N_{TRIPLES} T_{QUERY^*}} = \frac{N^2 T_{CMP}}{2 \cdot N_{TRIPLES} T_{QUERY^*}} + \frac{Q}{2 \cdot N_{TRIPLES}}$$

A first impression of the speedup can be provided considering the following: an IoT application aims at detecting changes in the status of any presence sensor among 10^5 sensors of the same type. Each sensor reading corresponds to the update of one RDF triple (i.e., $N_{TRIPLES} = 1$). The bindings results returned by the query are in this case $N = 10^5$. Supposing a reasonable case where $T_{CMP} \sim 10^{-3} T_{QUERY}$ (e.g., μ s versus ms) and considering the worst case of $Q = 1$, the speedup results 5×10^6. A detailed evaluation is proposed in [124].

3.5 A centralized LUTT

In the Semantic Web of Things scenario many overlapping subscriptions may coexists at the same time. A very simple example could be an home automation application, where all the appliances subscribe to the policy selected by a manager and modify their behaviour according to the current policy.

In this cases having a LUTT for every subscription may have a double drawback: if we name n the number of these subscriptions, n will also be the number of LUTTs maintained by the broker. n is also the number of simultaneous checks that must be performed by the broker, thus affecting the performance of the broker.

This motivates my subsequent investigation on a single centralized LUTT that groups together the equivalent triple patterns. Fig. 3.2 shows how multiple LUTTs can be translated to a single centralized data structure.

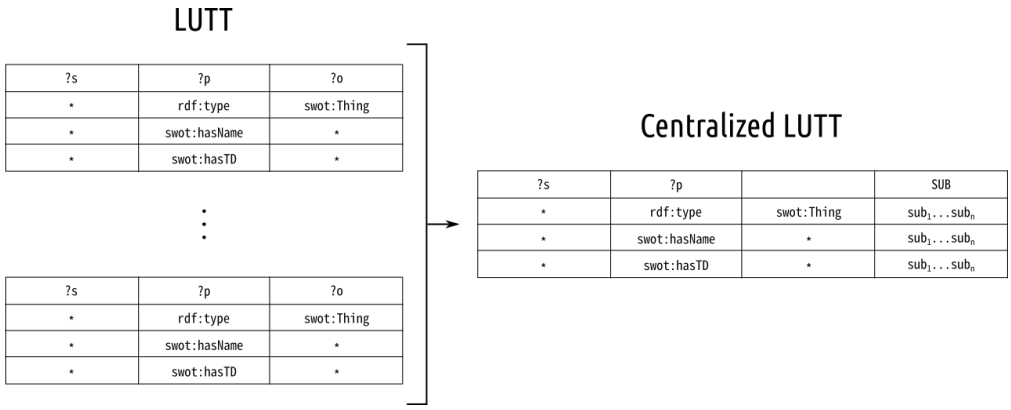


Figure 3.2: Mutiple LUTTs vs Centralized LUTT

It is easily noticeable how this approach allows saving memory not only in case of equivalent subscriptions (i.e., subscriptions sharing all the triple patterns), but also in case of subscriptions sharing only part of their triple patterns.

3.6 A centralized hierarchical look up table

I have started carrying out the study on novel ways to organize SEPA's look-up tables at the end of the second PhD year with the aim to speed up the broker by reducing the number of checks required to determine which subscriptions must be awakened. This activity, proposed in Section 3.5, has been followed by the one presented in the following lines, aimed at reducing the memory footprint of the centralized LUTT too.

In fact, a further space reduction can be achieved with a hierarchical organization of the Centralized LUTT (CLUTT): the hierarchy starts with a list of the subjects s_i (without repetitions) appearing in all the triple patterns. For every subject s_i, all the predicates (without repetitions) p_j such that s_i and p_j appears together in the same triple patterns is built. Then, the same is done for all the objects o_k. This last layer lists all the subscriptions interested by this triple pattern. An example can be observed in Fig. 3.3. In the following, this data structure will be referred to as Centralized Hierarchical LUTT (CHLUTT).

Figure 3.3 reports three sets of subscriptions $S_1 = \{s_i : i = 1 \dots m\}$, $S_2 = \{s_j : j = 1 \dots n\}$ and $S_3 = \{s_k : k = 1 \dots p\}$. With the color pink are reported the $m + n + k$ LUTTs of the subscriptions; color yellow and green are used to depict respectively the centralized and centralized hierarchical LUTTs.

3.7 Conclusion and future work

In this Chapter several algorithms to process subscriptions in a semantic context broker have been presented. The discussion has started with the most intuitive one (the naive algorithm) and then several optimizations needed to achieve a fast and scalable architecture have been introduced. While the LUTT-based algorithm has been implemented in the context broker SPS (semantic broker of the Smart-M3 platform), the CLUTT-based and CHLUTT-based have been implemented in the Python version of the SPARQL Event Processing Architecture. These brokers will be detailed in Chapter 4. A still ongoing research activity is focused on the extension of CHLUTTs to centralized hierarchical look-up quads tables (CHLUQTs), in order to keep track of the named graph owning each triple. Further works will be aimed at investigating efficient algorithms to avoid the replication of information in the local CTSs in case of triple patterns shared among the running subscriptions.

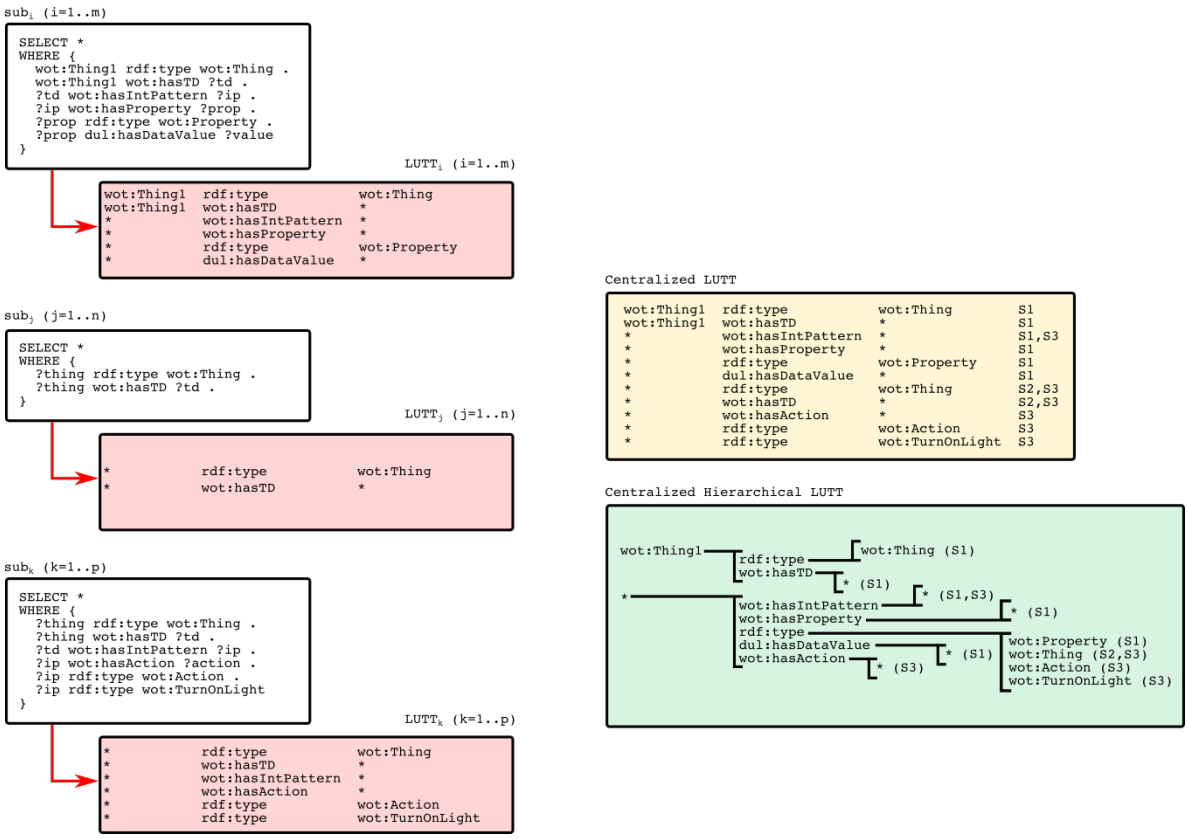


Figure 3.3: LUTTs (pink), centralized LUTT (yellow) and centralized hierarchical LUTT (green)

Chapter 4

Semantic Publish-Subscribe Engines

Contents

4.1 Introduction

Chapter 3 introduced several subscription algorithms for semantic context brokers on which I focused my PhD research. This Chapter describes my research activity related to the development of semantic publish-subscribe brokers belonging to the Smart-M3 platform (Section 4.3) and its descendent, the SPARQL Event Processing Architecture, according to the algorithms presented in the previous Chapter. More in detail, after an overview of the state of the art (Section 4.2) three implementations of Smart-M3 brokers (named SPS[1] , pySIB[2] and OSGi SIB[3], developed in the first and a half PhD year) characterized by different contributions, will be detailed in Subsections 4.3.2, 4.3.3 and 4.3.4. Section 4.4 pivots on the SPARQL Event Processing Architecture[4] (developed starting from the second PhD year), while Section 4.5[5] introduces a last architecture developed to address the needs of the Internet of Musical Things domain (activity related to the third PhD year).

[1]© IEEE, Reprinted with permission, from Luca Roffia, Francesco Morandi, Jussi Kiljander, Alfredo D'Elia, Fabio Vergari, Fabio Viola, Luciano Bononi, Tullio Salmon Cinotti. A semantic publish-subscribe architecture for the Internet of Things. IEEE Internet of Things Journal. 2016.

[2]© IEEE, Reprinted with permission, from Fabio Viola, Alfredo D'Elia, Luca Roffia, Tullio Salmon Cinotti. A Modular Lightweight Implementation of the Smart-M3 Semantic Information Broker. 2016 Proceedings of the 18th Conference of FRUCT Association. Apr. 2016.

[3]This contribution was published by Alfredo D'Elia, Fabio Viola, Luca Roffia, Paolo Azzoni, Tullio Salmon Cinotti, in Enabling interoperability in the internet of things: A OSGi semantic information broker implementation, International Journal on Semantic Web and Information Systems (IJSWIS), IGI Global, Jan. 2017

[4]This contribution was published by Luca Roffia, Cristiano Aguzzi, Fabio Viola, Francesco Antoniazzi, Tullio Salmon Cinotti, in Dynamic Linked Data: A SPARQL Event Processing Architecture, Future Internet, MDPI, Apr. 2018

[5]© IEEE, Reprinted with permission, from Fabio Viola, Luca Turchet, Francesco Antoniazzi, Gÿorgÿ Fazekas. C Minor: a Semantic Publish/Subscribe Broker for the Internet of Musical Things. 2018 Proceedings of the 23rd Conference of FRUCT Association. Nov. 2018.

4.2 Related work

The work presented in this Chapter belongs to research topics like stream reasoning [125], linked stream data processing [126], and content-based publish-subscribe [105].

The first approaches for publish-subscribe systems based on Semantic Web protocols were presented by Wang et al. [127] and Chirita et al. [128]. The first proposed an ontology-based publish-subscribe system where events are expressed with RDF graphs, while the second proposed a solution to incorporate publish-subscribe capabilities in RDF-based peer-to-peer (P2P) network. Both these research works proposed custom languages to specify events, mainly due to the immaturity of the Semantic Web at that time. The same approach can be found in [90], where the proposed language resembles SPARQL. All these projects propose very different algorithms, since they deal with different technologies. Another approach, this time based on the Semantic Web Rule Language (SWRL) is presented in [129]. In this work, differently from what I propose in this Chapter (and that has already been introduced in the previous one), a notification for a subscription includes the whole results set.

The already mentioned SENS, proposed by Murth and Kühn in [106, 130, 131, 132] is an event processing infrastructure to detect new knowledge rather than changes in the system. Subscriptions are expressed with SPARQL basic graph patterns and it is also possible to create rules (represented with a subset of SPARQL CONSTRUCT) that create new knowledge to the knowledge base when specific events occur, similarly to the Persistent Update proposed in Section 4.3.4. Differently from our approach, SENS supports only a subset of SPARQL (i.e., basic graph patterns).

Other Semantic Web based interoperability platforms are: 1) the Task Computing Environment (TCE) [133] (where the focus is the automation of users' everyday tasks); 2) the COntext BRoker Architecture (COBRA) [134] (for context-aware applications); 3) the Context Aware Platform (CAP) [135]; 4) Semantic Space [127]; 5) Semantic middleware for IoT [136]; 6) Smart objects awareness and adaptation Model (SoaM) [137]; 7) Amigo [138]; 8) SPITFIRE [22]; 9) OpenIoT [139]; 10) Orion Context Broker[6], framed in the FIWARE project.

Among the publish/subscribe systems become popular with the explosion of the Internet of Things, it is worth mentioning MQ Telemetry Transport (MQTT) [140], Constrained Application Protocol (CoAP) [141, 92] and Advanced Message Queuing Protocol (AMQP) [142]. All of them are lightweight protocols for device-to-device communication. MQTT was born in 1999, originally designed by Andy Stanford-Clark (IBM) and Arlen Nipper (Cirrus Link).

[6] `https://fiware-orion.readthedocs.io/en/master/`

It is a broker-centric protocol providing a topic-based publish/subscribe functionality: subscribers manifest their interest in a certain topic (wildcards are allowed) and are promptly notified about new messages with a matching topic. Compared to Smart-M3 and SEPA, MQTT provides a subscription mechanism allowing lower granularity. On the other hand, Smart-M3/SEPA offer support to semantics and support also the request/response paradigm. As regards CoAP [141], also in this case the protocol itself does not provide support for semantics. CoAP provides a lightweight alternative to HTTP, that is ideal to make the web accessible to resource-constrained devices [92, 143]. AMQP is a lightweight M2M protocol born in 2003 and designed to provide a reliable architecture granting security, provisioning and interoperability. Both request/response and publish/subscribe paradigms are supported by AMQP. CoAP it is basically a request/response protocol, but also supports publish/subscribe through the observation of resources. Due to the lightweight nature of CoAP and its close relation with HTTP, this protocol has also been employed for a tiny version of the SPARQL Event Processing Architecture.

As regards the Smart-M3 community, the work presented in this Chapter leverages the previous experience matured on RedSIB [144]. In particular, SPS is based on RedSIB 0.9.2. Later on, pySIB (Section 4.3.3) and the OSGi SIB (Section 4.3.4) were designed and developed by studying the application of Smart-M3 respectively on constrained devices and industrial environment. In the meantime, the Russian community proposed an alternative implementation of the Smart-M3 broker: CuteSIB [145].

4.3 Smart-M3

The M3 (Multi-device, Multi-vendor, and Multi-domain) [146, 145] architecture has been initially defined by a consortium participating to the Artemis JU funded SOFIA project and to the Finnish nationally funded program DIEM (Device Interoperability Ecosystem), working in strong collaboration with the Nokia Corporation. The Smart-M3 platform implements the M3 architecture and its first release dates back to 2009. Soon after its first release, the Smart-M3 potential was understood and applied in other European projects (e.g., about eHealth and eMobility). Smart-M3 is currently developed by several communities including the FRUCT Association, the SOFIA Community, and the ARCES department at University of Bologna. Clients of the Smart-M3 platform are named KPs and are loosely coupled. In fact, the interaction among KPs happens through the SIB. Messages between SIB and KPs are represented using the SSAP.

From the architectural point of view, as previously introduced in Section 2.5 the central

role in the Smart-M3 interoperability platform is played by the SIB. The SIB implements an information hub forming a logical rendezvous and information-level interoperability infrastructure on the top of an RDF triple-store. The SIB is the access point to the shared KB hosting the application context. The basic SIB role is to manage the read and write accesses to this graph. Moreover, the SIB implements the publish-subscribe paradigm, providing a notifications for all the updates of the graph that match the active subscriptions. Subscriptions in the Smart-M3 architecture are content-based [105], thus allowing a very high granularity. Subscriptions can be considered as persistent queries through which a KP avoids polling the knowledge base thanks to notifications.

The already introduced `subscription` primitive allows declaring the interest in a subgraph through a SPARQL 1.1 SELECT query and provides a mean for a client to be notified on specific events. When a subscription request S is issued at time t_i by a KP (let's name it $S(t_i)$), the SUB manager returns the SPARQL bindings of the analogous query performed at time t_i: $Q(t_i)$. Then, if a SPARQL Update $U(t_j)$ $(j > i)$ affects the subgraph specified by S, a notification is sent to the subscribing client. The notification does not include all the results of the analogous SPARQL query Q performed at time t_j, but contains only the added and the removed SPARQL binding results since the previous notification (i.e., as with the *Istream* and *Dstream* operators used in the SQL-based continuous query language [147]). If we name $R(t_j)$ the results of $Q(t_j)$, the notification will contain the new bindings (i.e., those appearing only in the last execution of the query) calculated as $\{R(t_j) \setminus R(t_i)\}$, and the removed bindings (i.e., those present in the previous execution of the query, but not in the last) calculated as $\{R(t_i) \setminus R(t_j)\}$.

The advantages of this approach can be appreciated by considering a simple, yet very common example: an IoT service monitoring 1M sensors. If a sensor updates its measure, the service is notified with just that value instead of receiving 1 million values and having to check all of them to discover which one has changed. Moreover, this allows to dramatically reduce the network overhead.

4.3.1 Smart-M3 primitives

The primitives implemented by the client side APIs are:

- `join`: permits a client to signal its access to the smart space;
- `leave` – launched at the end of the session, this primitive allows leaving the smart space;
- `update` – through an update request, clients can perform write operations on the graph. The update can be specified through the SPARQL update language 1.1 or through the

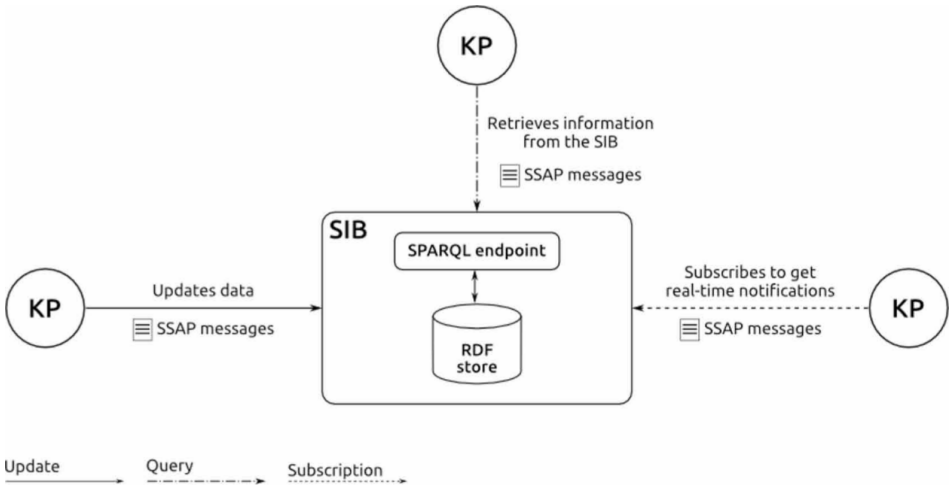


Figure 4.1: Smart-M3 architecture [148]

RDF-M3 protocol that is peculiar of the Smart-M3 platform;

- `query` – permits a client to retrieve data from the graph;
- `subscribe` – this primitive allows issuing a persistent query with two mandatory parameters: the subgraph to subscribe (specified either by means of the SPARQL Query language, either using RDF-M3);
- `unsubscribe` – the primitive allows closing an active subscription and require, as the only mandatory parameter, the id of the subscription.

All of the previous primitives are encapsulated in SSAP messages (that is an XML-based protocol). Examples of request and responses are reported in the following listings:

Listing 4.1: RDF-M3 Update Request

```
1  <SSAP_message>
2    <node_id>KP_M3_9f04076b-...</node_id>
3    <space_id>X</space_id>
4    <transaction_type>UPDATE</transaction_type>
5    <message_type>REQUEST</message_type>
6    <transaction_id>4</transaction_id>
7    <parameter name = "insert_graph" encoding = "RDF-M3">
8      <triple_list>
9        <triple>
10         <subject type = "URI">http://ns#a</subject>
```

```
11          <predicate>http://ns#b</predicate>
12          <object type = "URI">http://ns#d</object>
13        </triple>
14      </triple_list>
15    </parameter>
16    <parameter name = "remove_graph" encoding = "RDF-M3">
17      <triple_list>
18      </triple_list>
19    </parameter>
20    <parameter name = "confirm">TRUE</parameter>
21  </SSAP_message>
```

Listing 4.2: RDF-M3 Update Response

```
1  <SSAP_message>
2    <message_type>CONFIRM</message_type>
3    <transaction_type>UPDATE</transaction_type>
4    <transaction_id>4</transaction_id>
5    <space_id>X</space_id>
6    <node_id>KP_M3_9f04076b-...</node_id>
7    <parameter name="status">m3:Success</parameter>
8  </SSAP_message>
```

The field `message_type` discriminates among a request, a response and an *indication* (i.e., a notification) while the type of the specific request/response is stated in the `transaction_type`. The `transaction_id` allows keeping track of the message flows. The field `space_id` is used to specify the smart space to which the request is destined. The `node_id` contains a unique identifier of the node performing the request.

From a high-level perspective, KPs can be classified among **Producers**, **Consumers** and **Aggregators**. The first class contains KPs that only perform write operation on the knowledge base, while consumers performs only reading on the KB (through queries or subscriptions). Aggregators react to changes in the knowledge base by producing new knowledge (so they play simultaneously the role of producers and consumers). An application can, of course, exploit multiple KPs at the same time. This application design pattern, allowing a reduced instructions set, will be better explained in Section 4.3.2.

4.3.2 The SPS broker

This Section presents the Semantic Publish-Subscribe (SPS) architecture, based on the RedSIB [144] implementation of a broker for the Smart-M3 interoperability platform. SPS consists of a processing infrastructure offering a reduced primitive set composed by `update` and `subscribe`/`unsubscribe`. This is not limiting since these primitives are in principle enough to implement any application. SPS was designed to support information level interoperability in smart space applications in the IoT. The activity presented in this Section proposes three main contributions:

- The main contribution is the **novel event detection algorithm** presented in Section 3.4, that is tailored on the IoT specificities (i.e., heterogeneous events need to be detected and continuous updates of few RDF triples dominate with respect to more complex updates). Notifications produced by the architecture are related to changes in the knowledge base (i.e., from now on named *events*) and expressed in terms of added and removed SPARQL binding results since the previous notification. The algorithm also supports event negation (i.e., the not occurrence of an event within a time interval).
- A second contribution of this architecture is represented by a new primitive named **delayed SPARQL Update** that allows clients to schedule a given update at a specified time.
- Third, a novel design pattern aimed at easing the development of Smart-M3 applications is proposed. The design pattern is based on a high-level classification of Knowledge Processors among producers, consumers and aggregators.

This efficient SPARQL subscription engine (from now on *SUB Engine*), as well as the modular system architecture represent extensions to the work presented in [144] and exploit the experience matured on European Research projects (i.e., SOFIA[7], RECOCAPE[8], Flex4Grid[9], IoE[10], Chiron[11]).

Software architecture

The SPS architecture is summarized in Fig. 4.2. The SPS adopts two queues to host update and subscription requests, respectively named Update Requests Queue (URQ) and Subscription Requests Queue (SRQ). This layer, named SUB Engine, stands on top of an RDF graph.

[7] `https://cordis.europa.eu/project/rcn/106175_it.html`
[8] `https://cordis.europa.eu/project/rcn/101530_en.html`
[9] `https://cordis.europa.eu/project/rcn/194427_en.html`
[10] `https://cordis.europa.eu/project/rcn/102781_en.html`
[11] `https://artemis-ia.eu/programcall/call-2009.html`

A scheduler determines when an update or subscription request can be extracted from the queue to be processed. Fig. 4.2 also highlights the presence of an entity called SPU. The SPU, already introduced in Section 3.4, is in charge of processing a given subscription. In fact, every time a new subscription request is extracted from the SRQ, a new SPU is created. An SPU includes a LUTT, an ARTQ, a booster and a CTS, as described in Section 3.4.

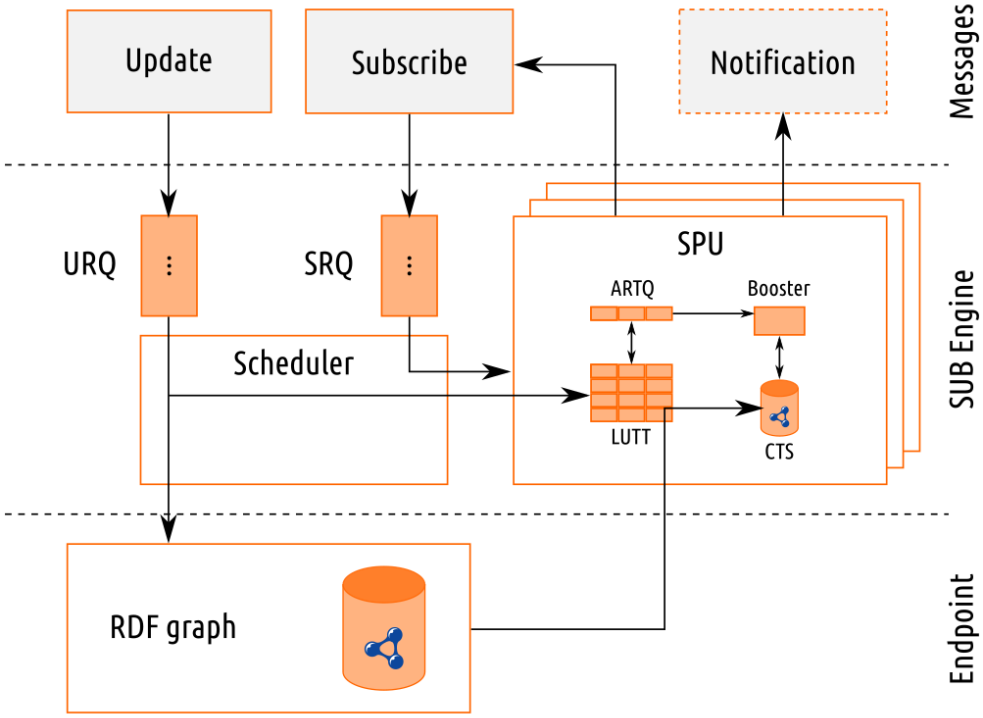


Figure 4.2: The SPS architecture

As regards SPARQL update requests, every time a new one is fetched from the URQ, the RDF graph is updated and the added and removed triples (if any) are retrieved. This triggers the event detection algorithm. To detect events, the underlying KB must not change, so only one SPARQL update can be processed at a time. Then, limiting the time needed to detect events is important also to reduce the impact on the update requests.

Added and removed triples are filtered through all the LUTTs of all the SPUs. The LUTTs (as described in Section 3.4) are a first trick to reduce the time required to process subscriptions. In fact, filtering triples against the LUTT allows avoiding awakening subscriptions that are not interested by an update. If no match is found, the processing ends, otherwise all added and removed triples matching a LUTT are inserted into the related Added Removed

Triples Queue and the associated SPU is activated. All the triples in the ARTQ are used to bind variables in the SPARQL query, that is also decomposed in n simple queries (where n corresponds to the number of triple patterns) that are executed on the local CTS (the second expedient to reduce the processing time, by reducing the size of the dataset to be queried).

Delayed SPARQL Update

The SUB engine grants time management through a SPARQL function to retrieve the current time (i.e., the unix time extended to μs). The delayed SPARQL Update is a new primitive that allows clients to schedule the execution of an update on the SUB engine side at a specified time. Through the delayed SPARQL Update primitive, the SUB engine has also the capability to handle **event negation** (i.e., the notification of events that did not occur within a specified time interval). The importance of this new feature is linked to the so-called supervising systems [149] where the "*not-occurrence*" of an expected event is itself the event to be detected and notified.

Application design pattern

A further contribution related to the work on the SPS architecture, is the introduction of a novel application design pattern based on a clean distinction of the roles of each KP and a subsequent high-level abstraction. Clients are in fact categorized in three sets:

- **Producers** – collect data from the physical world, represent it according to a given ontology and publish the resulting information in RDF graph (i.e., they perform write operations);
- **Consumers** – retrieve data from the RDF graph, either with queries or with subscriptions. In the latter case, consumers are notified by the SUB engine whenever events are detected, avoiding polling.
- **Aggregators** – bridge the functionalities provided by producers and consumers since they react to the information read from the graph producing new knowledge. Like the consumers, also aggregators may use queries or subscriptions.

The advantages of the proposed design pattern are twofold: First, this approach allows reducing the complexity of the Knowledge Processors, enabling the implementation in resource-restricted devices. Second, all the producers and consumers can be developed independently from a specific use case and shared among different applications; This, of course, leads to cost

savings also when new systems are deployed. This design pattern has been used in all the components developed during the PhD.

The SUB Manager

The SUB engine (mainly) consists of a scheduler listening for requests incoming from two FIFO queues (i.e., the `update` request queue, URQ, and the `subscribe` request queue, SRQ) and a SPU for each `subscribe` request received.

An SPU implements the event detection algorithm and notifies just the subscriber originating the request. Every SPU is responsible of maintaining a CTS containing a subset of the entire SPARQL endpoint RDF store and defined as the "*union of all RDF triples matching at least one of the triple patterns of the graph pattern specified in the subscription*". Since processing a subscription requires the execution of the analogous query to later calculate added and removed bindings, holding a CTS allows to perform the query on a smaller dataset. This results in shorter processing time. The prices is of course represented by the higher memory requirements. A CTS and the global RDF graph can be respectively compared to a cache and the main memory of a traditional computer. The SPU, could be as well compared to a processor with its own cache in a multiprocessor system.

4.3.3 pySIB

pySIB is a second research task framed in my Research on semantic architectures based on Smart-M3. During this project, I entirely designed and developed a **lightweight and portable context-broker** aimed at constrained devices. Being pySIB the first publish/subscribe broker based of the Smart-M3 family suitable for small devices, this activity can be considered an important milestone of my research that paved the way towards novel applications.

Software architecture

pySIB is a Python 2 implementation of the Smart-M3 SIB. This implementation of the SIB proposes a JSON serialization for the Smart Space Access Protocol, that resulted in a easier and faster parsing of the incoming messages. The architecture of the python SIB is depicted in Fig. 4.3 and is composed by a small set of modules:

- `Network` – handles all the incoming and outgoing TCP connections;
- `Message Parser` and `Message Builder` – respectively parse incoming messages and

build the replies. These two modules have been implemented with the `ujson` library that has been selected after an evaluation of the four most utilized libraries for JSON.

- `Store` – holds one or more RDF graphs implemented through the `rdflib` library;
- `Security Manager` – optional module to be developed in a future release.

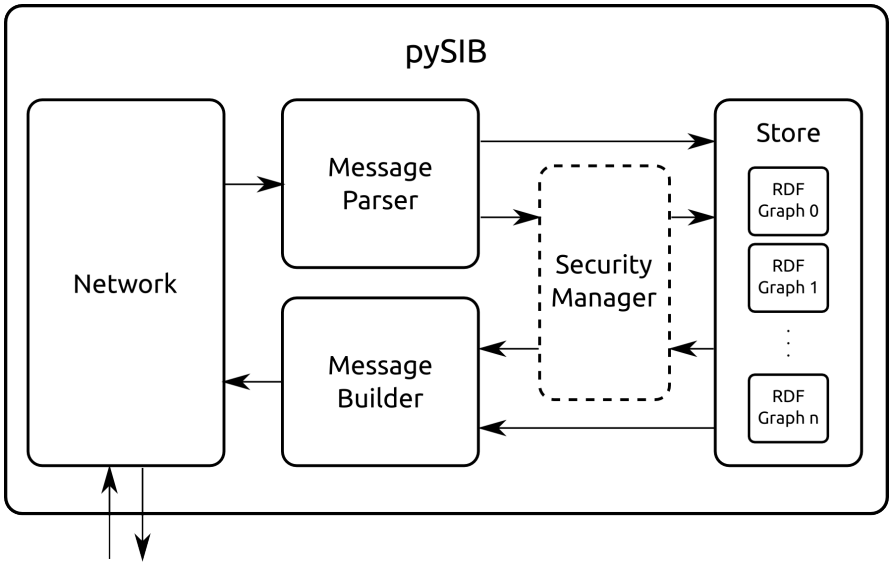


Figure 4.3: Architecture of pySIB [150]

The JSSAP Protocol

JSSAP defines the format for every request and confirm message exchanged between a KP and pySIB. It uses JSON as the default data serialization format for all the primitives in the standard SSAP specification. Remapping the SSAP protocol to a JSON encoded version allowed reducing the length of the exchanged messages from 10% (for long messages) up to 40% (for short messages). The main fields inherited from the standard SSAP protocols are:

- `node_id`: identifies the KP that performs a request (and receives the reply);
- `space_id`: an identifier for the smart space to which the KP belongs to;
- `transaction_id`: an univocal identifier for the request (and its reply);
- `transaction_type`: the transaction type identifies the kind of the request performed by the KP;

- `message_type`: the message type is used to mark a message as a request, a reply or an indication. Since the kind of message can be easily evinced by the software agent, this field is now not mandatory.

Performance evaluation

One of the objectives of pySIB was to be more effective in terms of communication, while still maintaining the possibility to be retrocompatible for legacy Smart-M3 applications. In this sense, the SSAP protocol was redesigned to reduce the message size and simplify the parsing process. In order to achieve the scope, the XML serialization has been replaced by a JSON encoding and all the redundant information transported by the protocol was removed.

The JSON-encoded version of the SSAP protocol allowed a relevant reduction of the message size (-40% for short messages, -10% for messages dispatching more than 100 triples). Moreover, a preliminary comparison among the existing Python libraries for JSON allowed the selection of the most efficient one, `ujson`, being this a critical component of the SIB.

First, the results of the comparison of the python modules named `cjson`, `json`, `ujson` and `simplejson` are reported. Tests were executed on a Lenovo Thinkpad X220 provided with an Intel(R) Core(TM) i5-2520M CPU 2.50GHz 4-core processor and 4 GB of RAM running Linux Mint 17 Qiana. The first test performed was aimed at a comparison of the JSON libraries during the encoding phase. The comparison metric is the time needed by the SIB to build a reply to a SPARQL query in relation to the number of results. The SPARQL query adopted for this purpose is the most general one:

```
1 SELECT *
2 WHERE { ?s ?p ?o }
```

This kind of test is pretty much effective and relevant to characterize the encoding time, since the reply to a SPARQL query can be very large, depending on the number of bindings to be returned. According to this test (Fig. 4.4) `simplejson` resulted the slowest module with every number of bindings. The time required by `ujson` to encode the results of a SPARQL query is always around the 30% of the time elapsed by the standard `json` module, resulting in the worst case, in a difference of more than 340 ms.

To characterize the available libraries during the decoding step, the time needed to parse a JSSAP request coming from a KP has been reported. As the decoding time depends on the JSON document tree structure, two main relevant scenarios have been taken into consideration: insertion of triples with the RDF-M3 protocol and insertion with a SPARQL Update request. The latter is way more complicated than the RDF-M3 insertion, due to the

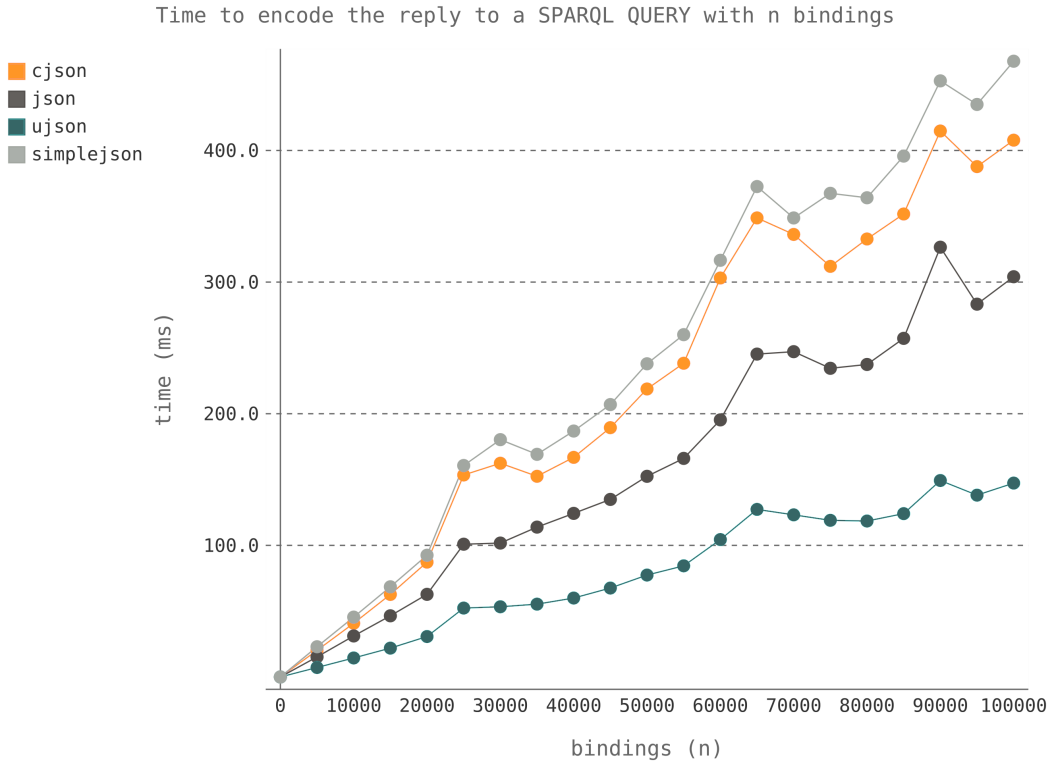


Figure 4.4: Evaluation of the python JSON libraries with SPARQL Query requests

multitude of fields forming the request. Fig. 4.5 and Fig. 4.6 show the results of the two tests. As expected, whatever library is used, the time needed to decode an RDF-M3 update results always higher than the time elapsed to parse a SPARQL update with the same number of triples since there is a very high number of field to be analyzed. For the sake of clarity, it is worth mentioning that the two diagrams utilize different scales in order to better appreciate the trend of each line. It can be observed how the default JSON module underpeforms with respect to the alternative implementations. Differently from what happens with the encoding test, here the differences among the performances of `cjson`, `ujson` and `simplejson` are negligible, since in the worst case the difference between the fastest and the slowest encoder is respectively less than 0.1 ms for a SPARQL update and less than 1.2 ms for an RDF-M3 update.

The time required to perform updates of the knowledge base, as well as the time required to retrieve information (varying respectively the number of inserted/retrieved triples) were measured on pySIB, RedSIB and the OSGi SIB (described in Section 4.3.4). In both cases (depicted in Figg. 4.7 and 4.8), pySIB outperformed the other SIB implementations.

Particular effort was also put in reducing both the memory and disk footprints of the SIB,

Figure 4.5: Evaluation of the python JSON libraries with RDF-M3 Update requests

as reported respectively by Fig. 4.9 and Table 4.1.

Table 4.1: Disk space occupation (in KiloBytes)

Disk Usage	**pySIB**	**RedSIB**	**OSGi SIB**
SIB package	25	88	13824
Dependencies	640	10832	21504
Interpreter/VM	197	0	171
Interpreter Deps	12518	0	97224
Total	13380	10920	137723

4.3.4 OSGi SIB

A third project related to the development of a semantic context broker for the Smart-M3 platform is the OSGi SIB. The motivation behind the development of this further broker is the need for a more robust and performing component (with respect to RedSIB [144]) to

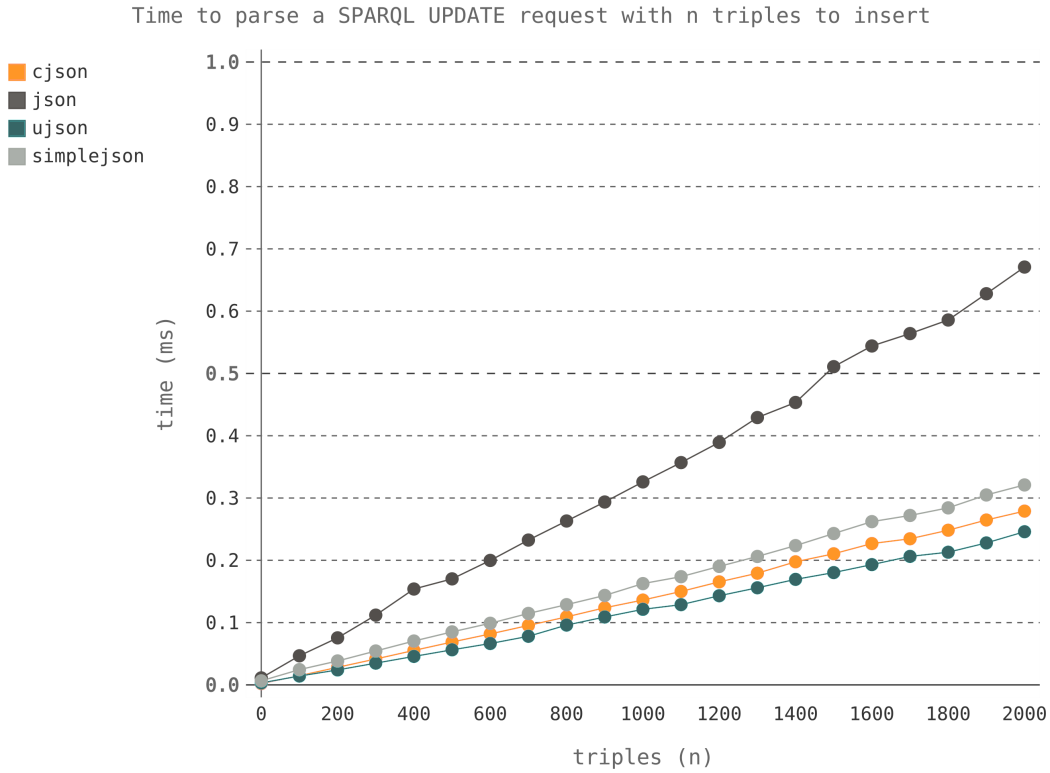


Figure 4.6: Evaluation of the python JSON libraries with SPARQL Update requests

adequately support the emerging IoT scenarios at gateway level through a modular and extensible architecture. A key requirement for the OSGi SIB was the backward compatibility with existing legacy applications based on RedSIB. The work on the novel SIB (whose architecture is detailed in Subsection 4.3.4) has also brought to the definition of a new SPARQL-based primitive to support semantic reasoning detailed in Subsection 4.3.4. Moreover, a further contribution of this research activity has been the implementation of the first Android-compatible SIB to support new Research scenarios, but this is out of the scope of my Thesis, being this activity not carried out by me.

In a second moment, a further research activity related to the implementation of a semantic last will [12] primitive has been carried out.

[12] © IEEE, Reprinted with permission, from Alfredo D'Elia, Cristiano Aguzzi, Fabio Viola, Francesco Antoniazzi, Tullio Salmon Cinotti. Implementation and evaluation of the last will primitive in a semantic information broker for IoT applications. Research and Technologies for Society and Industry (RTSI), 2017 IEEE 3rd International Forum on. 2017.

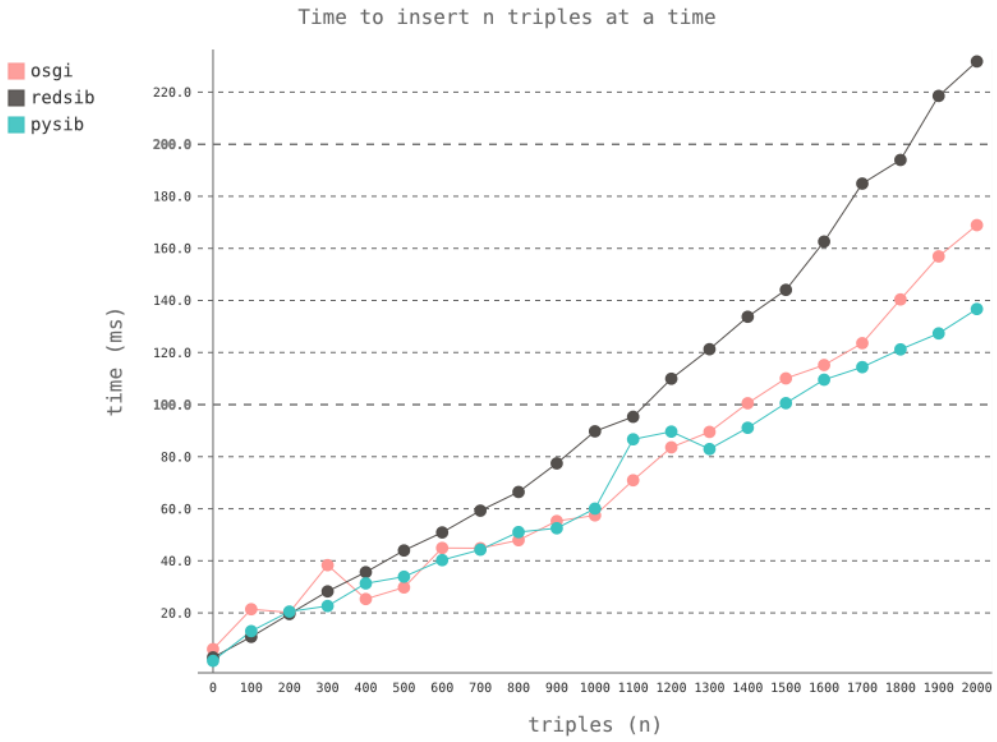


Figure 4.7: Time required to update the knowledge base

Software Architecture

As the name suggests, this SIB was developed using the OSGi framework[13]. This framework, with a service oriented architecture and a modular philosophy, allows optimizing costs and provides agility and flexibility, ensuring the ability to further evolve the system. The OSGi applications framework provides a programming model for developing, assembling and deploying modular applications based on Java EE technologies. OSGi was designed as a services gateway for set-top boxes and, since its introduction in 1998, it has become broadly adopted by industry. It has been extensively used in several application contexts (e.g., automotive, mobile and fixed telephony, industrial automation, gateways & routers, private branch exchanges) and, is now supported by many integrated development environments (e.g., IBM Websphere, SpringSource Application Server, Oracle Weblogic, Sun's GlassFish, Eclipse, and Redhat's JBoss) and key companies (e.g., Oracle, IBM, Samsung, Nokia, IONA, Motorola, NTT, Siemens, Hitachi, Ericsson).

The architecture of the broker is depicted in Fig. 4.10, where blocks correspond to bundles

[13] `http://www.osgi.org`

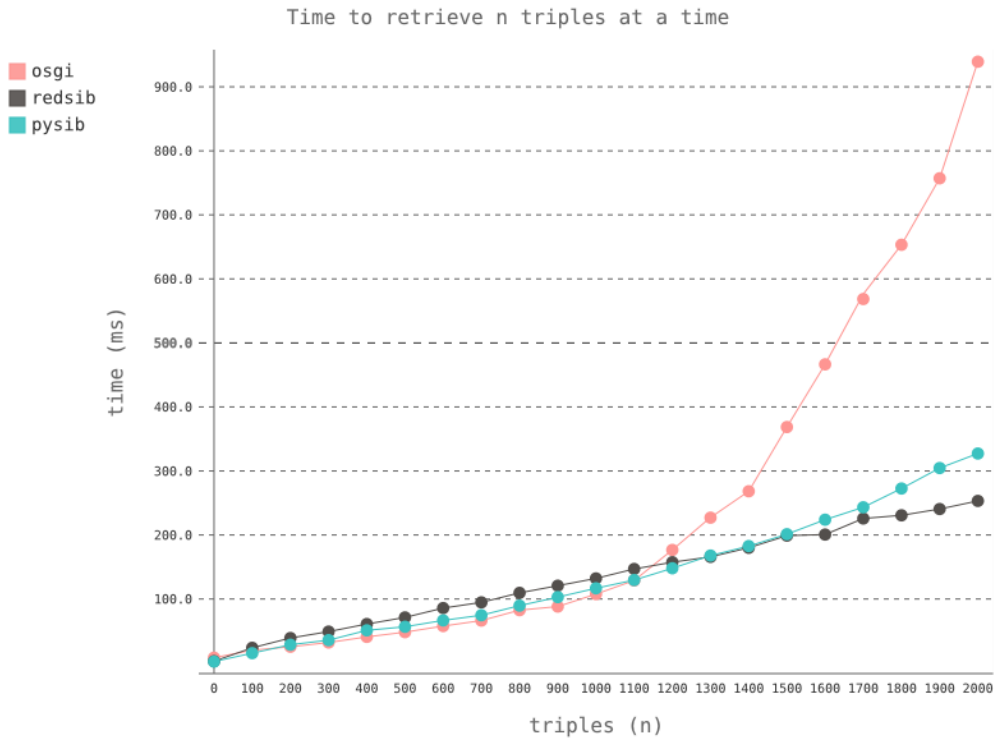


Figure 4.8: Time required to query the knowledge base

and the oriented arrows to service calls. Bundle names are indicated with a bold font, while dependencies are reported in italic.

The `TCP` bundle handles all the connections to the broker and interacts with the `Message Handler` that holds a queue of messages. The `SSAP` bundle instead, operates on a higher level, being responsible for parsing the received messages and the serialization of data for the responses. `JOIN`, `SUBSCRIPTION` and `PERSISTENT UPDATE` bundles process the namesake requests. While the `TOOLS` bundle exposes several utilities needed by the other bundles, the `TOKEN HANDLER` provides a service to get an internal identifier through which every message is marked. The `SIB` bundle holds the RDF store, exploiting the popular Jena[14] framework.

Persistent Update

The main contribution proposed by the OSGi SIB is the Persistent Update (PU) primitive. The PU is based on the SPARQL Update Language and the corresponding update is performed both when the request for a PU is first received and whenever the conditions contained in the `WHERE` section become true and require the insertion/deletion of the triples specified in the

[14] `https://jena.apache.org`

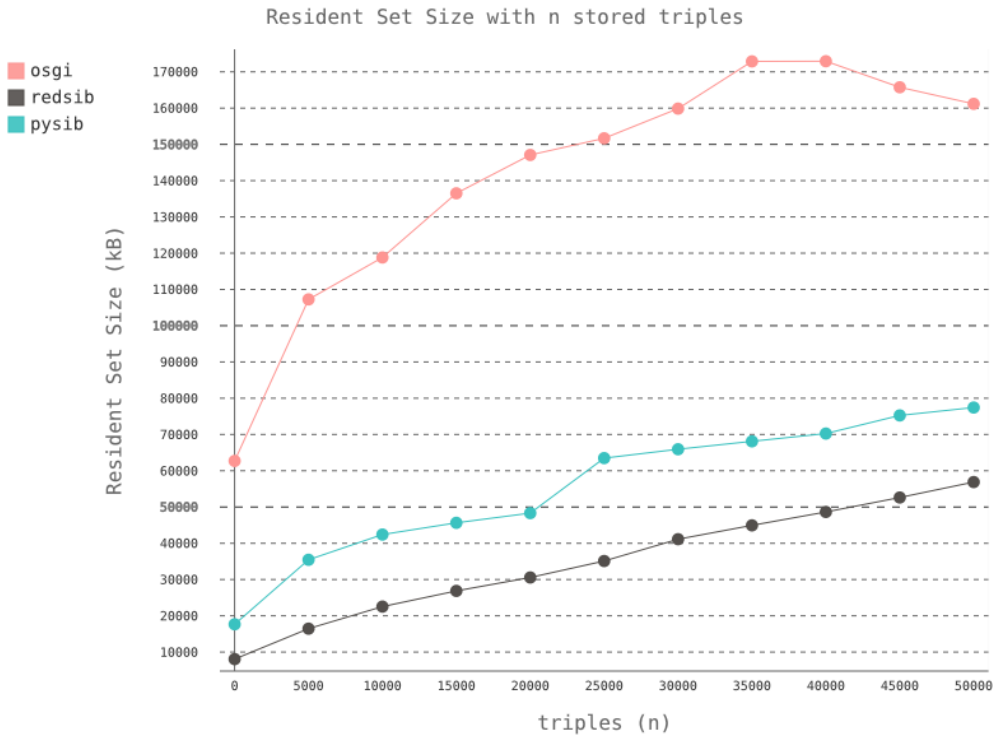


Figure 4.9: Resident Set Size (in KB) varying the number of stored triples [150]

`INSERT/DELETE` sections. This behaviour holds until the deactivation of a PU is requested by the client.

The PU answers to the following research challenge for Smart-M3 based systems: making the semantic knowledge base able to change itself depending on the context. The implementation of PU permits the definition of a set of rules persistently acting on the KB, thus realizing a SPARQL-based reasoning mechanism on server-side.

Let's consider the following SPARQL (Persistent) Update:

```
1 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
2 PREFIX ns:<http://ns#>
3 INSERT { ?f1 foaf:knows ?f2 }
4 WHERE { ?f1 ns:friendOf ?f2 }
```

Whenever a new triple matching `?f1 ns:friendOf ?f2` is put in the knowledge base, if `?f1 foaf:knows ?f2` is not present, is automatically inserted. This trivial example discloses one of the important advantages provided by the PU primitive: the ability to automatically map at runtime concepts belonging to different ontologies. The PU is handled by the proper

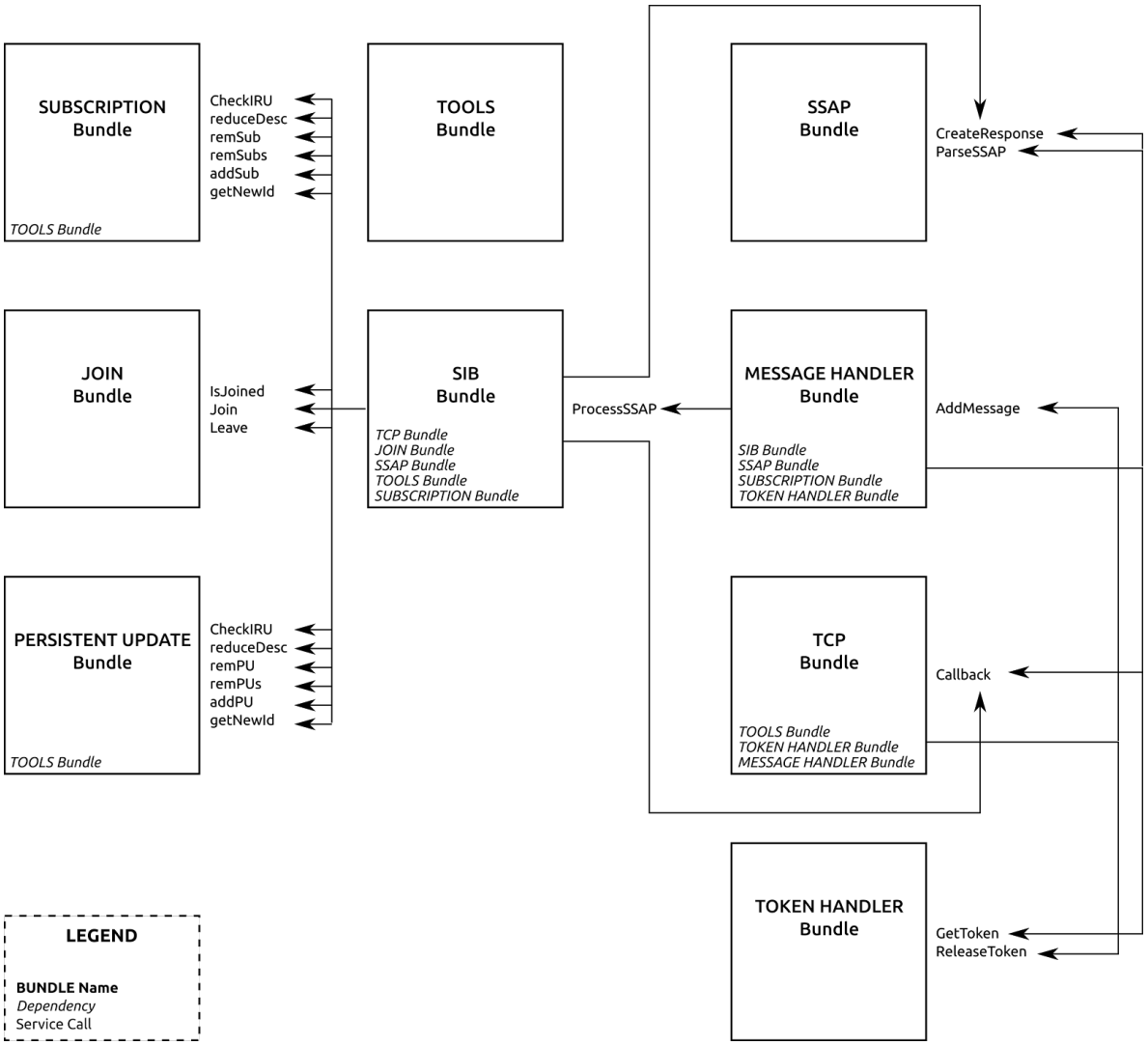


Figure 4.10: Architecture of the OSGi SIB [148]

bundle, as highlighted in Fig. 4.10. A PU is activated and deactivated respectively through the `MakePU` and `RemovePU` implemented in the Smart-M3 client libraries.

Evaluation

The performance evaluation takes into consideration separately the system behaviour and reactivity of the KB modification, of the subscriptions, the new PU primitive and KB query. A comparison between the performances of the OSGi SIB and the RedSIB [144] is proposed.

While frameworks, benchmarks and methods for performance evaluation of Semantic Web systems have been proposed in the literature [151, 152, 153, 154], these methods are not suitable to evaluate the OSGi SIB with reference to its specific features (i.e., SPARQL subscription

and SPARQL persistent update). Therefore, a benchmark inspired by a smart public lighting system (defined during the development of SPS) has been employed. A full description of the benchmark is reported in Section 5.3, while a detailed description of the KB is reported in Table 5.1.

Fig. 4.11 report the results of the first evaluation test aimed at measuring the insertion time when no subscription is active. The comparison of the insertion time on the OSGi SIB and on RedSIB highlights the better performances of the novel context broker. Fig. 4.12 contains instead the results of the second test: insertion of 1, 10 and 100 lamp-posts with n active subscriptions (n that varies between 10 and 100). Also in this case, the novel broker shows a better behaviour, especially with the insertion of 100 lamp-posts where the OSGi SIB outperforms the old implementation.

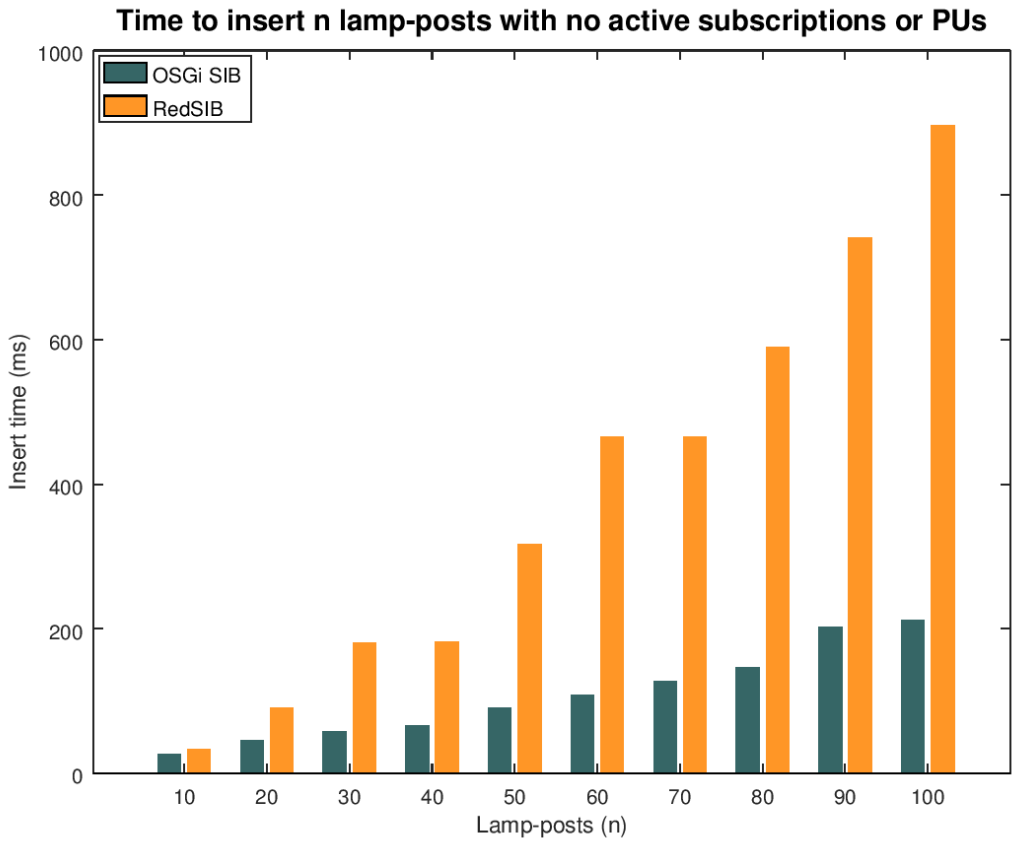


Figure 4.11: Insertion time on the OSGi SIB and RedSIB [148]

Subscriptions represent a relevant feature of a semantic broker allowing prompt reaction to context changes and reduction of the network traffic by avoiding unnecessary polling. As a drawback, subscriptions require resources and processing time that should have a negligible

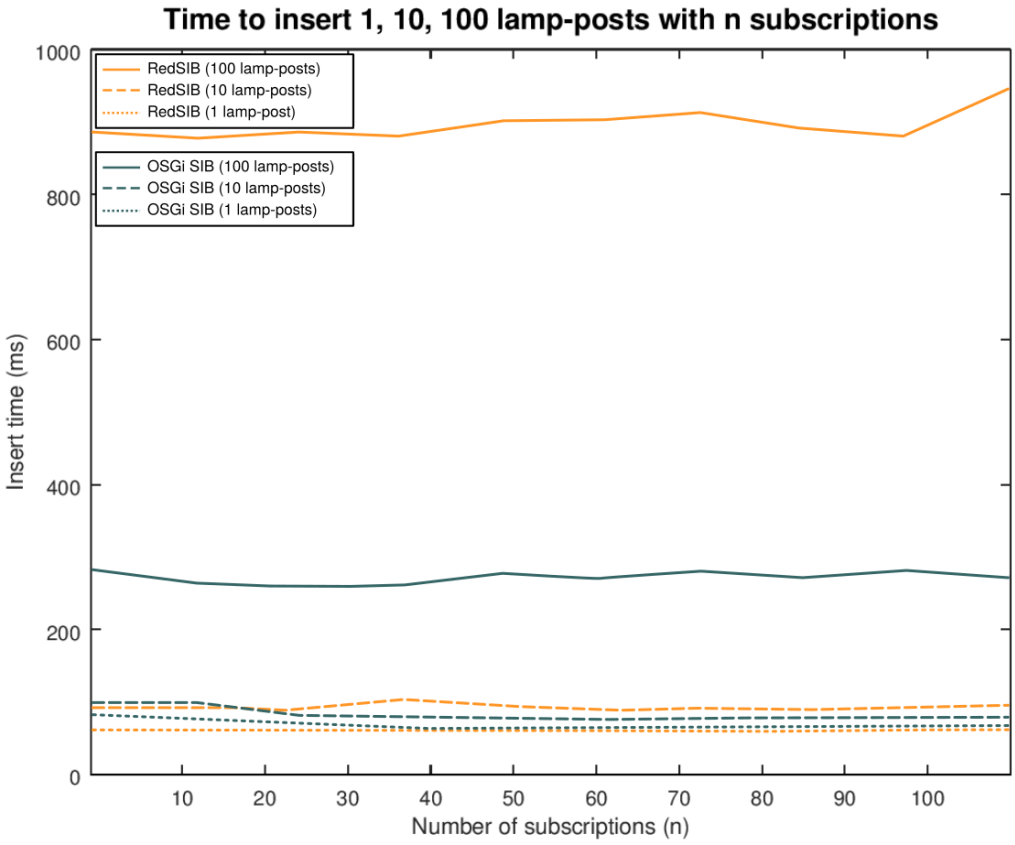


Figure 4.12: Insertion time on the OSGi SIB and RedSIB with active subscriptions [148]

impact on performances. Several updates of the dimming value of a single lamp (see Section 5.3) always triggering the same subscription (i.e., if the update changes the dimming value of `LAMP_X_Y` only the subscription to `ROAD_X` is triggered) are performed. The average latency is measured starting from when the updates are issued to the time the corresponding notifications are received on the KP side. The measure is repeated four times considering each time, as triggered subscription, a subscription to a road of different size (i.e., very small, small, medium, large). As shown in Fig. 4.13, the OSGi SIB outperforms the former implementation. If the number of active subscriptions is less than 40, no difference can be appreciated between the new and the old broker (excluding the case of RedSIB with the subscription triggered on a large road).

Further evaluation results are reported in [148] and [155], two of the dissemination outcomes related to this research activity.

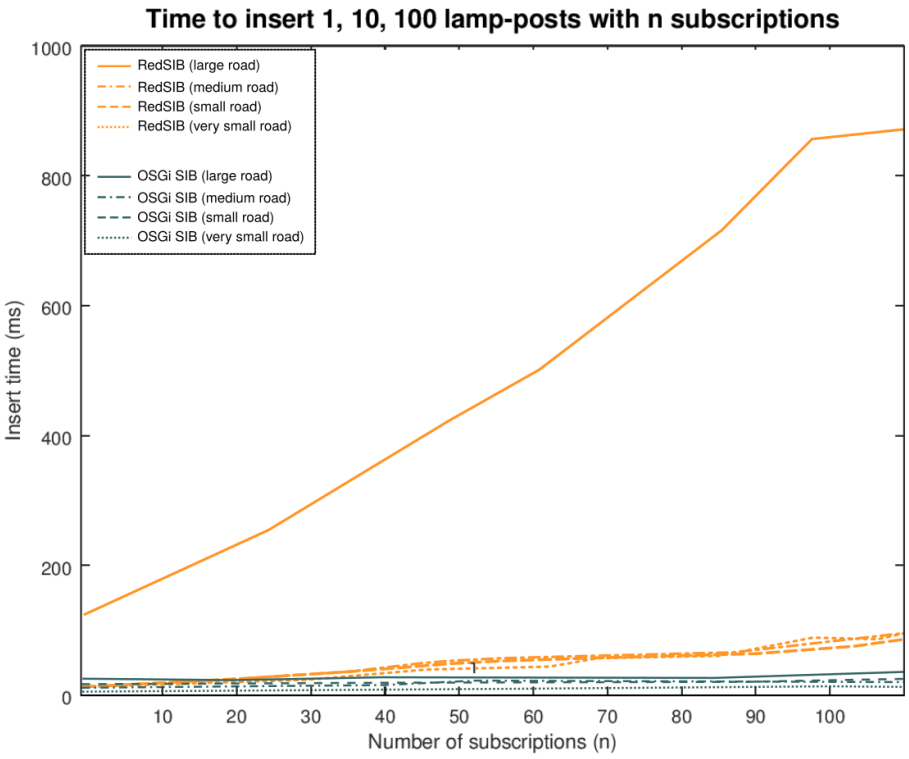


Figure 4.13: Notification latency versus number of active subscriptions [148]

The Last Will primitive

The latest research activity carried out on the OSGi SIB regards the implementation of a Last Will (LW) primitive inspired by the MQTT protocol in which it is employed to augment the tolerance to network disruptions: a client issues a special message to the broker that will be effective only when a disconnection occurs. The application of the LW to a semantic KB and its inclusion in the Smart-M3 architecture improves the overall reliability and provides new dynamics that can be exploited by developers with simpler code. The advantages are twofold: first, each agent specifying a LW, inherently instantiates a persistent connection with the central KB that can be used to verify its temporary or definitive unavailability. Second, when a connection is not working properly, the LW may automatically start a recovery procedure.

The implementation of the LW primitive in the OSGi SIB required a refactoring of the internal software architecture. The aim was to enhance the flexibility of the SIB, towards future evolution of the project. More precisely, the core of the architecture, originally partitioned in store, Message Handler and TCP, presents now new components and a different organization of the existing ones. The purpose of this refactoring is twofold: first, to have a

negligible impact on the new LW functionality; second, to avoid circular dependencies that are a common problem in growing OSGi architectures.

The LW module stores an update command for every client requesting a LW registration. This module is also activated through an event-based method as soon as a client failure is detected. As inspired by Rahman et al. [156], the failure detection relies on a periodic heartbeat signal sent by the client. A higher degree of extensibility is guaranteed by the renewed architecture of the SIB: new modules may now be added as in a plug-in system and interfaced with the event hub module.

The registration flow of the LW is pretty straightforward. The registration command, once read and parsed, is sent to the store module, which delegates the management of the request to the LW. The LW module stores the requested update operation if and only if the connection is still alive. Finally, a confirmation response is sent to the issuer client.

When a LW is already registered, four different situation may happen:

- The issuer wants to quit the system. As a result, the LW must be deleted.
- The client wants to delete its testament.
- The client amend its last will with new wills.
- A failure detection system detects the disconnection.

The first three cases are trivial and consist in the deletion/insertion of the registered command. As regards the fourth, the LW module retrieves the registered update command and sends it to MessageHandler module. In this way, the LW command (consisting of a SPARQL Update) is injected in the normal flow with minimal impact on performances.

4.4 SPARQL Event Processing Architecture

In early 2017, the SPARQL Event Processing Architecture originated from Smart-M3 interoperability platform. SEPA was born to provide a reliable architecture for the Web of Data, supporting higher data volumes than Smart-M3. Moreover, it provides a standard and transparent interface to SPARQL endpoints, instead of embedding its own RDF graph. The advantages of this architectural choice are twofold: on one hand it allows attaching SEPA to whatever SPARQL endpoint the user desires (being it a custom instance or a public endpoint like DBpedia), while on the other hand it avoids reinventing the wheel by relying on consolidated software components for the storage of triples. Then, SEPA is a decentralized Web-based software architecture that derives and extends the one presented in Section 4.3.2

and in the journal paper [124] through the use of standard Linked Data technologies and protocols: SEPA in fact, replaces the SSAP and RDF-M3 protocols typical of the Smart-M3 architecture with those promoted by the W3C (i.e., HTTP and SPARQL 1.1 Protocol) to foster interoperability.

In a SEPA application, clients exploit the W3C SPARQL 1.1 Update and Query languages to edit and retrieve data from the knowledge base. At the same time, they can express their interest in a subgraph and timely receive notification upon changes. To provide this functionality, SEPA introduces the novel SPARQL 1.1 Secure Event protocol and the SPARQL 1.1 Subscribe Language. Assuming an event as "*any change in an RDF store*", we can affirm that SEPA has been designed to enable event detection and distribution. The core element of SEPA is its broker (see Figure 4.14).

Just like in Smart-M3, the notification mechanism implemented by SEPA is delta-based: when the subscription is issued, results of the equivalent query are provided to the client; then whenever a change matching the subscription occurs, only the modified bindings are sent to the subscribers. In this way, subscribers can easily track the evolution of the query results (i.e., the context), with the lowest impact on the network bandwidth (i.e., the entire results set is not sent every time, but just the delta of the results). Another key contribution introduced by SEPA is the so-called Semantic Application Profile (SAP), a detailed description of the semantic profile of an application including the reference context broker, and all the namespaces and templates for SPARQL updates and queries/subscriptions.

After being presented at the W3C Web of Things Working and Interest Group meeting in Düsseldorf[15], SEPA was employed in the HABITAT Italian Research Project [157]. SEPA is now actively developed and is currently employed in European Research projects (i.e., AudioCommons [158] and SWAMP [78]).

This Section describes the research activity aimed at developing SEPA. In particular, Section 4.4.1 reports the an overview of the SPARQL 1.1 Subscribe Language and Secure Event Protocol designed for SEPA. In Section 4.4.2 the software architecture of the new broker is proposed, together with the vision of a SEPA-based ecosystem, while Section 4.4.3 details the Semantic Application Profile introduced by SEPA.

4.4.1 SPARQL 1.1 Subscribe Language and Secure Event Protocol

A standard protocol providing publish/subscribe functionalities on top of a SPARQL endpoint does not exist. During my PhD, two unofficial drafts have been produced to start working on a new standard. The two documents describe:

[15] https://www.wespeakiot.com/w3c-meeting-dusseldorf-another-step-towards-iot-standardization/

- **SPARQL 1.1 Secure Event Protocol**[16] – It wraps the SPARQL 1.1 Protocol to support subscriptions and secure connections. This protocol targets the application contexts where security must be supported.
- **SPARQL 1.1 Subscribe Language**[17] – It defines the content of subscribe and unsubscribe requests and responses and the format of notifications and ping messages. Every SEPA implementation must provide a JSON serialization of these messages, while other kinds of serialization formats may be optionally provided.

This Section focuses on the messages described by the SPARQL 1.1 Subscribe Language, while for a more detailed description of the language and the SPARQL 1.1 SE protocol I invite the reader to refer to the above-mentioned documents.

To request a new subscription, a client should issue a message like the following:

```
1 {
2   "subscribe" : "select * where {?s ?p ?o}",
3   "authorization" : "Bearer eyJa...",
4   "alias" : "All"
5 }
```

A confirm message is sent by SEPA:

```
1 {
2   "subscribed" : "sepa://subscription/0d057ca5-cc10-..."
3   "alias":"All"
4 }
```

The ID communicated in the confirm message should be saved by the client and used to request the closing of the subscription:

```
1 {
2   "unsubscribe" : "sepa://subscription/0d057ca5-cc10-...",
3 }
```

that is acknowledged with:

```
1 {
2   "unsubscribed" : "sepa://subscription/0d057ca5-cc10-..."
3   "authorization" : "Bearer eyJa..."
```

[16] http://wot.arces.unibo.it/TR/sparql11-se-protocol.html
[17] http://wot.arces.unibo.it/TR/sparql11-subscribe.html

```
}
```

A notification message is sent by the broker every time the results of the query bound to the subscription change. The added and removed bindings are then notified with a message like the following:

```
{
  "spuid" : "sepa://subscription/0d057ca5-cc10-...",
  "sequence" : 0,
  "results" : {
    "head": {
      "vars" : [ ... ] ,
      "link" : [ ... ]
    },
    "addedResults": {
      "bindings" : [
        {"a" : { ... } ,"b" : { ... }} ,
        {"a" : { ... } ,"b" : { ... }}]
    },
    "removedResults": {
      "bindings" : [
        {"a" : { ... } ,"b" : { ... }} ,
        {"a" : { ... } ,"b" : { ... }}]
      ]
    }
  }
}
```

A ping message is periodically sent by the broker to verify the status of the connection with the subscriber:

```
{
  "ping" : "2017-06-06T19:20:07Z"
}
```

4.4.2 Software Architecture

From a high level perspective, the SEPA platform is composed by a broker on top of which the server exposes HTTP(S) and (secure) WebSocket interfaces, respectively used to support the SPARQL 1.1 Protocol (as a standard SPARQL endpoint) and the SPARQL 1.1 Subscribe Language, both wrapped by the SPARQL 1.1 Secure Event Protocol (both introduced in the previous Section). Client-side APIs (currently available in Python3, Java, C, Javascript) provide all the required functions to interact with SEPA issuing SPARQL requests. Higher-level primitives provide an abstraction to develop KP based on their role (i.e., Producer, Aggregator, Consumer, see Section 4.3.2) by reading a JSON Semantic Application Profile (JSAP) file. The JSAP file contains the full (optionally parametric) description of the possible messages used by the application, as well as the parameters required to interact with the SPARQL endpoint.

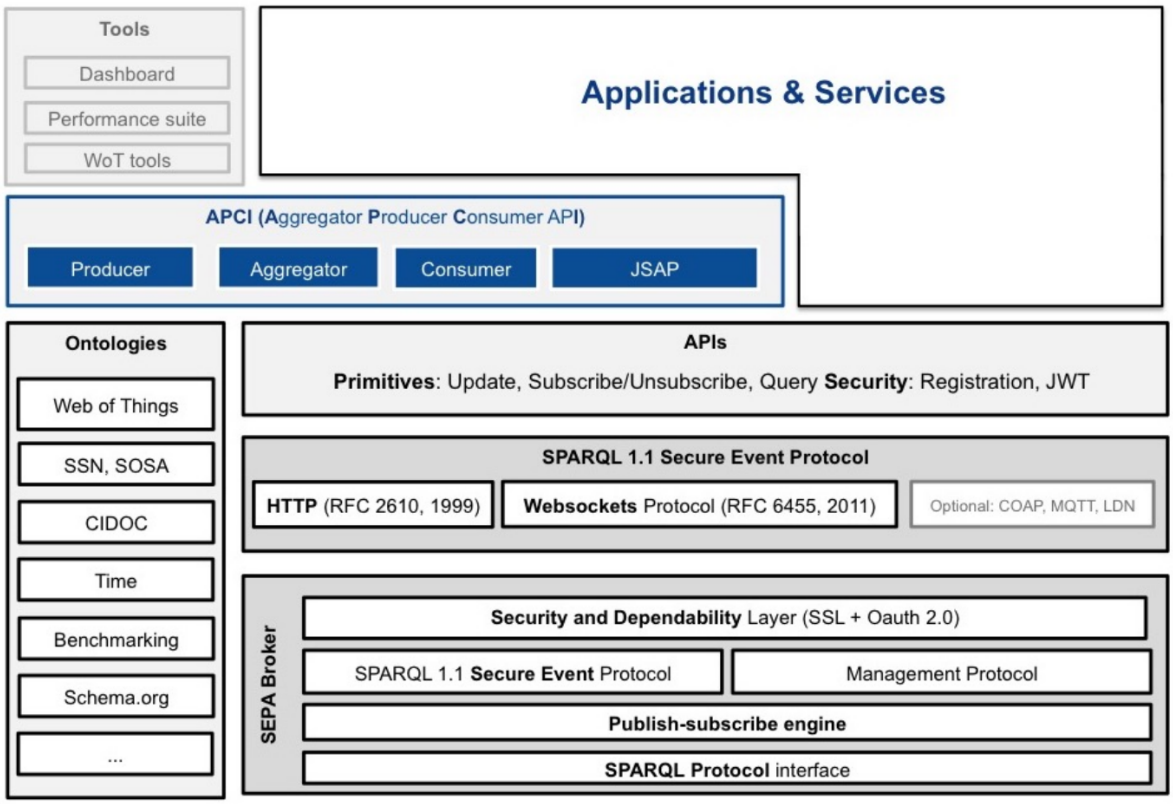


Figure 4.14: Architecture of the SEPA platform (high level) [159]

Internally (as shown by Fig. 4.15), the server of the SEPA platform dynamically creates instances of the Query Processor, Update Processor and SPU Manager when needed. In particular, an instance of the Query Processor is created whenever a new query request is

received. Since queries can be simultaneously processed without risk of data inconsistency, multiple query processors may run at the same time. Update requests instead, must be processed sequentially, so a FIFO queue hosts the Update requests to grant that only one Update Processor operates in a given time. Every time a SPARQL Update is processed, a set of Subscription Processing Units (SPU) may be triggered.

Update, Query and Subscription/Unsubscription requests are respectively handled by the (Secure)UpdateHandler, (Secure)QueryHandler and (Secure)WebSocketHandler. All of these entities can be monitored at runtime exploiting the administration panel built through the Java Management Extension (JMX), shown in Fig. 4.16 or through a web interface 4.17.

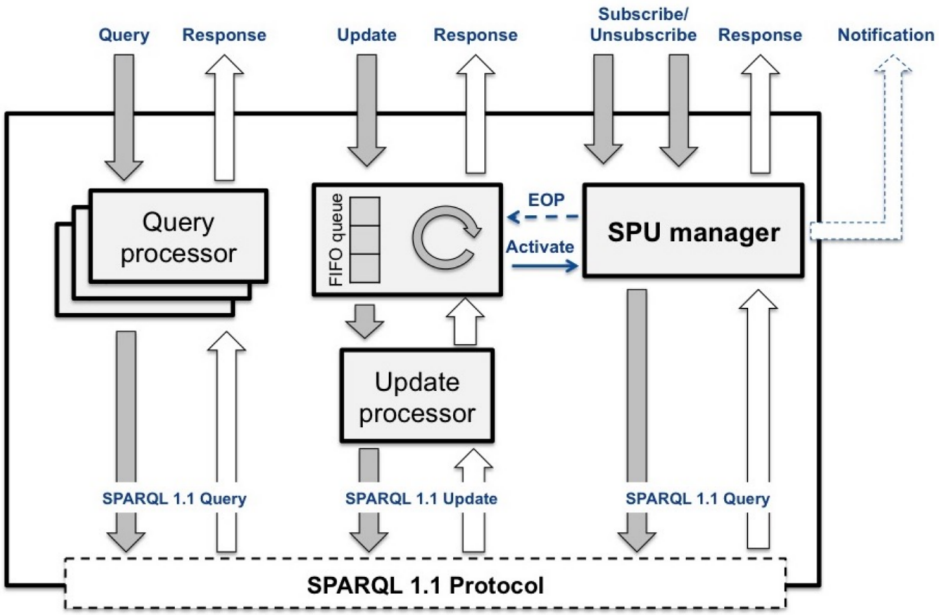


Figure 4.15: Architecture of the SEPA platform (low level) [159]

4.4.3 Semantic Application Profile

The Semantic Application Profile is a file used by a SEPA client to read the SPARQL endpoint connection parameters (in the section named `parameters`), the namespaces exploited by SPARQL updates/queries/subscriptions (section `namespaces`) and all the possible updates and queries/subscriptions (sections `updates` and `queries`). Every update, as well as every query/subscription, is specified in terms of the related SPARQL code, but also the forced bindings that represent the keys in the SPARQL template. If the SAP is encoded in JSON, then we refer to it as JSAP, YSAP if it is serialized with YAML. An example of JSON Semantic Application Profile file is proposed in the following listing.

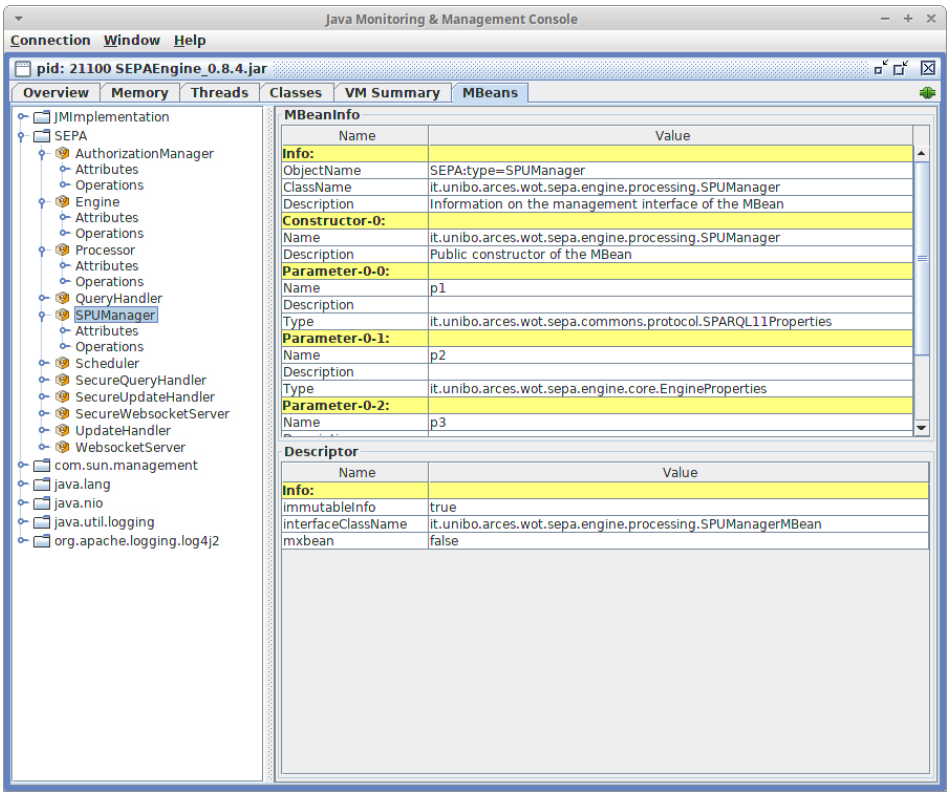


Figure 4.16: SEPA Management panel

```
{
  "parameters": {
    "host": "localhost",
      "ports": {
        "http": 8000,
        "https": 8443,
        "ws": 9000,
        "wss": 9443
    },
    "paths": {
      "query": "/query",
      "update": "/update",
      "subscribe": "/subscribe",
      "register": "/oauth/register",
      "tokenRequest": "/oauth/token",
      "securePath": "/secure"
```

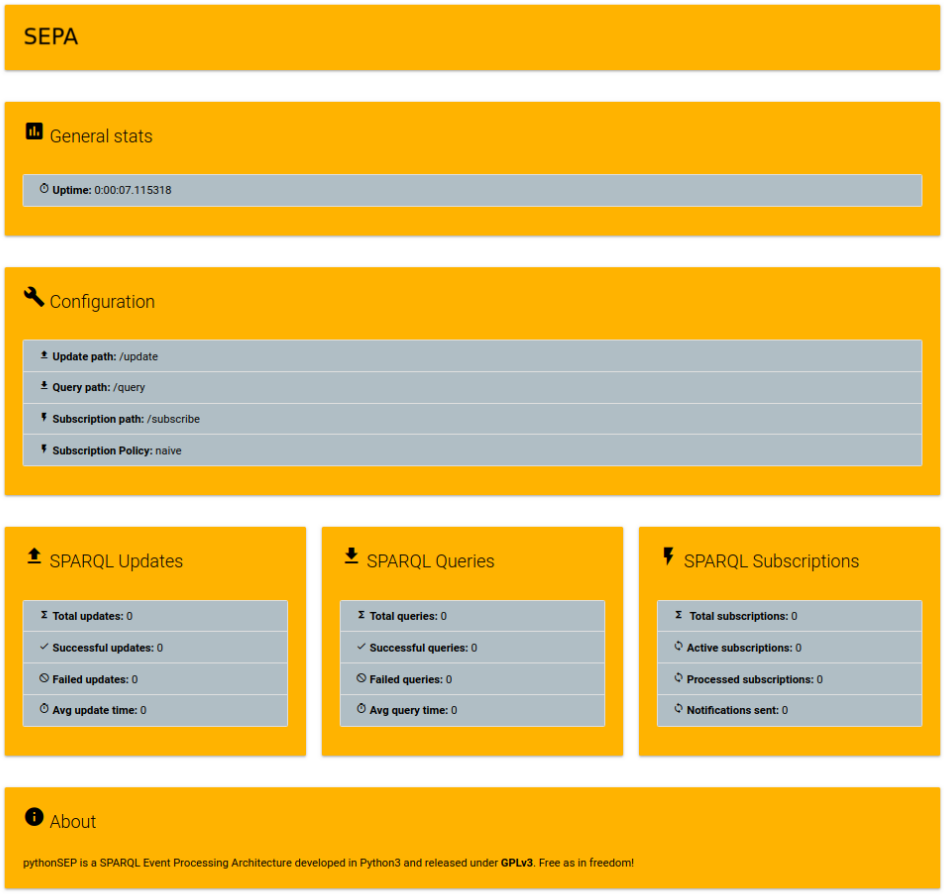


Figure 4.17: SEPA Control Panel

```
17      }
18    },
19    "namespaces": {
20      "wot": "http://wot.arces.unibo.it/sepa#",
21      "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
22    },
23    "updates": {
24      "POST_ACTION_INSTANCE_NO_INPUT": {
25        "sparql": "INSERT␣{␣?action␣a␣wot:Action␣)}"
26          "forcedBindings": {
27            "action": {
28              "type": "uri",
29              "value": ""
```

```
30            }
31          }
32        }
33      }
34    }
35    "queries":{
36          ...
37      }
```

4.5 C Minor

SEPA has been developed exploiting the experience matured on Smart-M3 with the aim to deal with large-scale scenarios and supporting web standards. A further evolution of the platform is needed to make also constrained devices directly involved in semantic ecosystems without the mediation of a gateway. In fact, these devices can be limited in terms of computational power or memory and can also be powered by a battery. These requirements conflict with the nature of the protocols adopted by SEPA. Devices with such features, require lightweight protocols that 1) minimize the number and size of exchanged messages; 2) grant a high transmission and processing speed. All these factors affects the resource requirements as well as the power consumption. Moreover, some of the IoT scenarios present severe constraints in terms of latency: an example is provided by the IoMusT that will be presented in Chapter 10. This motivates the need for a further evolution of the SPARQL Event Processing Architecture that leverages one of the popular lightweight IoT protocols that are currently gaining momentum.

This Section, presents my reasearch activity started at the Centre for Digital Music of the Queen Mary University of London where the main contribution consists in the design and development of C Minor, a SPARQL Event Processing Architecture based on CoAP [141], the first semantic publish/subscribe broker. Section 4.5.1 motivates the choice of CoAP and illustrates the process of adapting the behaviour of a SEPA to this lightweight protocol. Then, Section 4.5.2 presents the software architecture of C Minor, while Section 4.5.3 presents the primitives needed to interact with a C Minor instance. Preliminary evaluation results are presented in Section 4.5.4.

4.5.1 Evolution of the SPARQL 1.1 Secure Event protocol

Currently, among the many popular IoT protocols, the most diffused ones devoted to device-to-device communication are Advanced Message Queuing Protocol (AMQP) [142], Constrained

Application Protocol (CoAP) [141] and MQ Telemetry Transport (MQTT) [140]. CoAP was identified as the best solution due to several factors here summarized:

- CoAP was proposed by the Internet Engineering Task Force (IETF), in particular by the Constrained RESTful Environments (CoRE) subgroup, as a standard Request For Comment (RFC). So it is an open, fully documented specification [141] and it can pave the way towards interoperability in the IoT.
- Differently from MQTT and AMQP, CoAP is based on UDP [141] but still supports retransmission of lost or damaged packets (i.e., by means of confirmable and non confirmable messages). This allows CoAP to get rid of the overhead caused by the three-way handshake protocol;
- CoAP has the lowest bandwidth requirements and the lowest latency [12]; moreover its headers have a minimum impact on the message size [12, 160];
- A list of the available resources is intrinsically provided by CoAP [141], solving in this way the problem of discoverability [94].
- Lastly, the most important factor is that CoAP is designed to be easily mapped on HTTP. As highlighted by [161], the SPARQL 1.1 protocol adopts HTTP to convey requests and responses, then being able to map it on top of CoAP is a fundamental step. Since CoAP implements a subset of REST optimized for M2M computation, binding the SPARQL 1.1 Protocol [161] to CoAP is a very straight-forward process.

To develop a CoAP-based SEPA, the first step consists of defining a way to map SPARQL update and query requests (those provided by standard SPARQL endpoints):

- SPARQL Update requests: according to the SPARQL 1.1 protocol [161], a SPARQL Update should be sent with an HTTP `POST` request where the text of the update is specified in the request payload or through url-encoded parameters. A successful request may return a `2XX` or `3XX` code, while a failure is notified with a `4XX` (for wrong requests, e.g., syntax error) or `5XX` code (server issues). C Minor accepts requests including the text of the update as a payload and returns `2.04 Changed` in case of success, otherwise a `4.00 Bad Request` for a wrong request or `5.XX` code to notify problems on the server side.
- SPARQL Query requests: according to [161], queries can be performed with `GET` or `POST` requests. In the first case, the query is specified through percent-encoded parameters.

In the latter, as in SPARQL Updates, queries can be provided as a payload or through url-encoded parameters. C Minor accepts queries provided as payload of `POST` requests. The status code can be `2.04` (in case of success), `4.00` (for wrong requests) or `5.XX` (for server-side errors).

The comparison between the SPARQL 1.1 Protocol and the CoAP version proposed by C Minor is reported in Table 4.2.

Table 4.2: Mapping SPARQL 1.1 Protocol over CoAP. A summary of the implementation proposed in C Minor [162].

	HTTP	**CoAP**
Update Request verb:	`POST`	`POST`
Text specified as:	payload or url-encoded	payload
Status code for success:	`2XX` or `3XX`	`2.04`
Status code for error:	`4XX` or `5XX`	`4.00` or `5.XX`
Query Request verb:	`GET,POST`	`POST`
Text specified as:	url-encoded or payload	payload
Status code for success:	`2XX` or `3XX`	`2.04`
Status code for error:	`4XX` or `5XX`	`4.00` or `5.XX`

Then, how to deal with subscriptions? SEPA relies on Websockets to support subscriptions (i.e., to create/delete a new subscription and to transmit notifications). CoAP supports subscriptions through observations of resources. Then, to provide notifications on top of CoAP, a proper observable resource should be created. For this reason, a third CoAP route is adopted by C Minor: `/subscription`. The CoAP verbs `POST` and `DELETE` permit the creation and deletion of a new observable resource. Whenever a new observable resource is created, a new route with the specified alias becomes available to anyone in the ecosystem. This opens the way towards additional, relevant considerations: IoMusT scenarios are characterized by a high number of equivalent subscriptions running at the same time; then, the ability to group equivalent subscriptions would allow the broker to save precious resources and be more efficient. C Minor, differently from SEPA, natively supports grouping equivalent subscriptions through the creation of shared observable resources.

4.5.2 Architecture of the C Minor context broker

C Minor has been developed as a python3 server on top of the `aiocoap` framework[18], a natively asynchronous implementation of a CoAP library. C Minor can either exploit the `rdflib` to hold an internal RDF graph (thus reducing the software dependencies), or rely on an external SPARQL endpoint (in this case the Fuseki endpoint developed in the context of the Apache Jena framework[19]). Among the classes of the C Minor broker (see Fig 4.18) it is worth mentioning:

- `SPARQLQueryResource`: handles all the SPARQL query requests;
- `SPARQLUpdateResource`: handles all the SPARQL update requests and deals with the subsequent triggering of the subscriptions;
- `SPARQLSubscribeResource`: this is the class that creates or deletes subscriptions. It creates an instance of the class `SubscriptionResource` every time a new subscription is requested.
- `SubscriptionResource`: it is an observable class that permits clients to receive notifications related to a given subscription.
- `CMinorStats`: collects stats to analyze the state of the system.
- `Endpoint`: responsible of all the interactions with either the external SPARQL endpoint or the internal graph.

A class diagram is depicted in Fig. 4.18. For the sake of clarity, the diagram shows only the relationships among classes defined in the `aiocoap` framework and classes implemented in C Minor. Arrows with white head represent an inheritance relationship.

4.5.3 Interacting with C Minor

To interact with C Minor, the following primitives were implemented in client-side python libraries:

- **update** – C Minor, just like SEPA, exposes a proper route (i.e., `/update`) to handle update requests. The only difference with SEPA is represented by the adoption of CoAP instead of HTTP, but with the same verb (i.e., `POST`). Fig. 4.19 depicts the sequence of steps to successfully perform an update.

[18] `https://aiocoap.readthedocs.io`
[19] `https://jena.apache.org/documentation/fuseki2/`

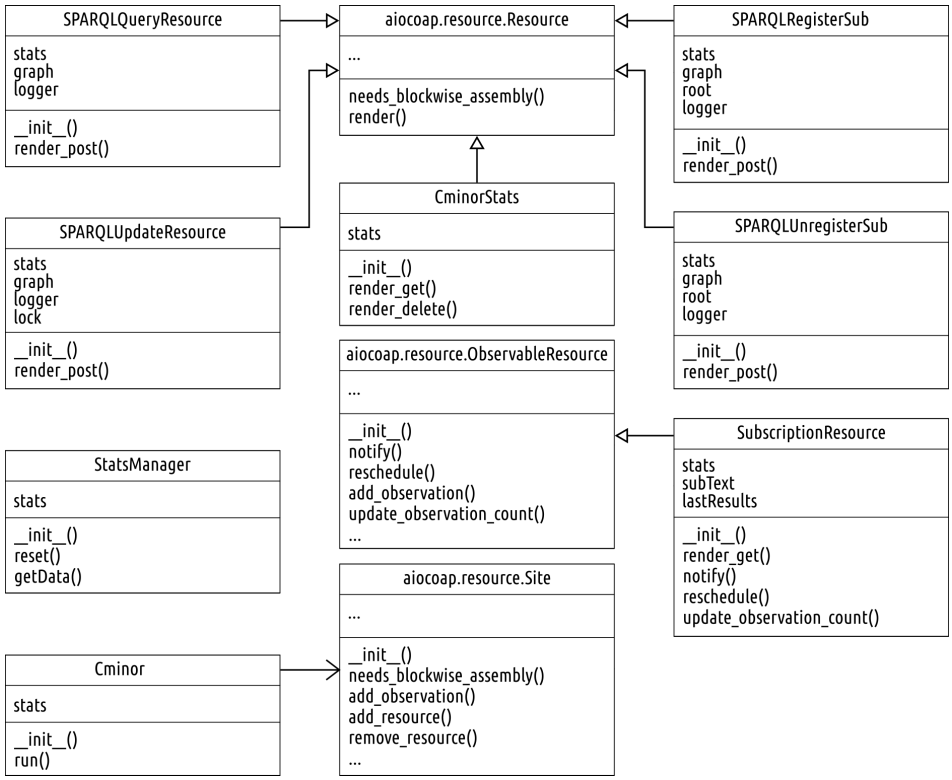


Figure 4.18: Class Diagram showing the relationship between C Minor and aiocoap classes [162].

- **query** – A `POST` request (with CoAP instead of HTTP) is used to deliver the query to the `/query` route of the server. This is very close to what happens in SEPA. Fig 4.20 depicts the process to perform a SPARQL query.

- **regSubscription** – As previously mentioned, every subscription corresponds to an observable resource. A proper primitive is needed to permit the creation of these resources. For example, a `POST` request with the following payload:

```
1 {
2   "query":"SELECT * WHERE { ?s ?p ?o }",
3   "alias":"all"
4 }
```

  creates a new observable resource with URI `/all` that provides notifications every time the results of the query changes (i.e., every time the underlying knowledge base is modified).

- **observe** – due to the way subscriptions work in C Minor, client-side libraries should provide the ability to observe resources. A `GET` request with the `observe` option set is

needed to start receiving notifications. The URI to observe a resource is the one specified during the `POST` request to `/subscription` (this process is shown in Fig. 4.21). If the client wants to observe a resource initialized by another client, then the URI can be discovered through the proper primitive `discovery` described below.

- **unregSubscription** – This primitive allows one to unregister a subscription (i.e., to delete the related observable resource). This primitive would perform a `DELETE` request to the route `/subscription` with a payload like this:

```
1 { "alias":"all" }
```

- **discover** – The resource `/.well-known/core` enables the discovery on the available resources, as specified by the CoAP protocol. A discovery request allows one to get the URIs of the routes to update and query the knowledge base, as well as the routes to register or unregister a new subscription. More importantly, the discovery allows one to get the list of the observable resources corresponding to active subscriptions, as shown by Fig. 4.22

- **stats** – C Minor holds an internal data structure containing relevant statistics, such as the number of the performed updates and queries with the average elapsed times. Moreover it maintains a list of the active subscriptions, with the number of generated notifications. These statistics can be retrieved with a `GET` request on the `/stats` URI, or reset with a `DELETE` on the same URI. Supporting the access to statistics in client-side APIs is not mandatory.

A list of the URIs exposed by the C Minor server is proposed in Table 4.3.

Table 4.3: Resources of the C Minor server [162]

URI	Verb	Payload
`/query`	`POST`	the plain SPARQL query
`/update`	`POST`	the plain SPARQL update
`/subscription`	`POST`	JSON (keys `query` and `alias`)
`/subscription`	`DELETE`	JSON (key `alias`)
`/<SUB>`	`GET`	-
`/.well-known/core`	`GET`	-
`/stats`	`GET`	-
`/stats`	`DELETE`	-

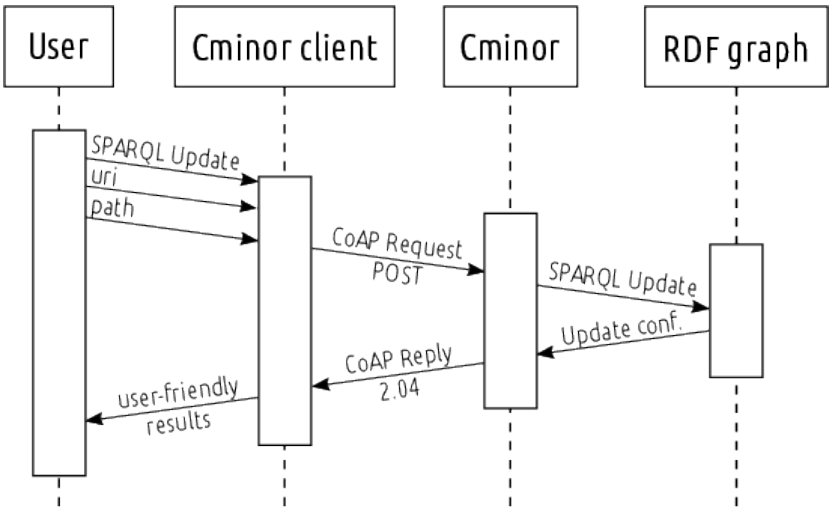


Figure 4.19: Sequence diagram for SPARQL updates [162]

4.5.4 Evaluation

A preliminary evaluation of C Minor was carried out in order to characterize the behaviour of the broker. Tests reported in the following sections were executed on two C Minor instances: the first relying on an RDFlib store and the latter on a non-persistent instance of Fuseki. As in [163] we made two assumptions: 1) limited size of the knowledge base, hosting only the current context. This is not limiting since in the envisioned IoMusT scenarios, the amount of producers is small if compared to the expected number of consumers; 2) the KB hosts only assertional data (i.e., the ontology is not stored in the context broker but used by client-side libraries). Again, this is a technical decision that allows limiting the size of the knowledge base.

According to Gray [164], the evaluation tests should be relevant for the analyzed use case, simple to understand, portable, and scalable to assess the performance of small as well as large systems. Based on these pillars, the evaluation of C Minor was carried out through the typical operations of the IoMusT domain (see Section 10.2). The evaluation has been carried out on a Dell Alienware 17 R2 laptop hosting both the semantic context broker and the clients to minimize the impact of the network.

Evaluation of the Update and Query primitives

Fig 4.23 and Fig 4.24 show the behaviour of the broker with respect to SPARQL Update requests with both Fuseki and RDFlib. The test consists in performing SPARQL Updates causing the simultaneous insertion of n discrete audio features ($\forall n \in [1, 25]$) semantically represented according to the Audio Commons Ontology (ACO) [165]. The scenario envisioned

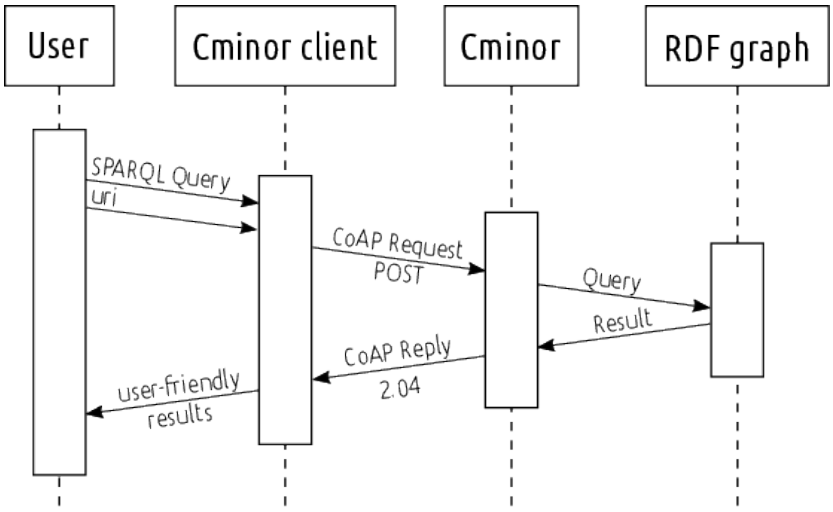


Figure 4.20: Sequence diagram for SPARQL queries [162]

in this test is that of a set of performers with Smart Instruments sharing the collected audio features with the musical things owned by the audience. With RDFlib the time required to perform the update linearly grows with the complexity of the request. It is easily noticeable that Fuseki outperforms RDFlib fulfilling every request in a nearly constant time inferior to 10 ms. Both the charts report the results of the client-side evaluation (i.e., including the CoAP request and response messages), as well as the time employed by the underlying engine to store data. The average time required by the protocol to dispatch request and response is 3.16 ms.

RDFlib performs better when responding to query requests, as shown by Fig 4.26 and Fig 4.25. This test was aimed at retrieved the whole context (i.e., n discrete audio features semantically mapped according to the Audio Commons Ontology, with $n \in [1, 25]$).

SPARQL queries on Fuseki require a higher amount of time due to the high presence of white spaces and newline characters included by Fuseki when requesting the serialization of results based on JSON. This ends up with longer messages that require a higher number of UDP segments to be dispatched. The optimization of this step is one of the future works.

Evaluation of the subscription mechanism

The behaviour of the C Minor when dealing with subscription is shown by Fig. 4.27. The related test was aimed at quantifying the time needed by the server to detect and notify a change to the observers after a SPARQL udpate. The context is composed by an average number of ten audio features. A SPARQL Update request modifies the value of a single audio feature. With a subscription to all the audio features running on C Minor, every time the

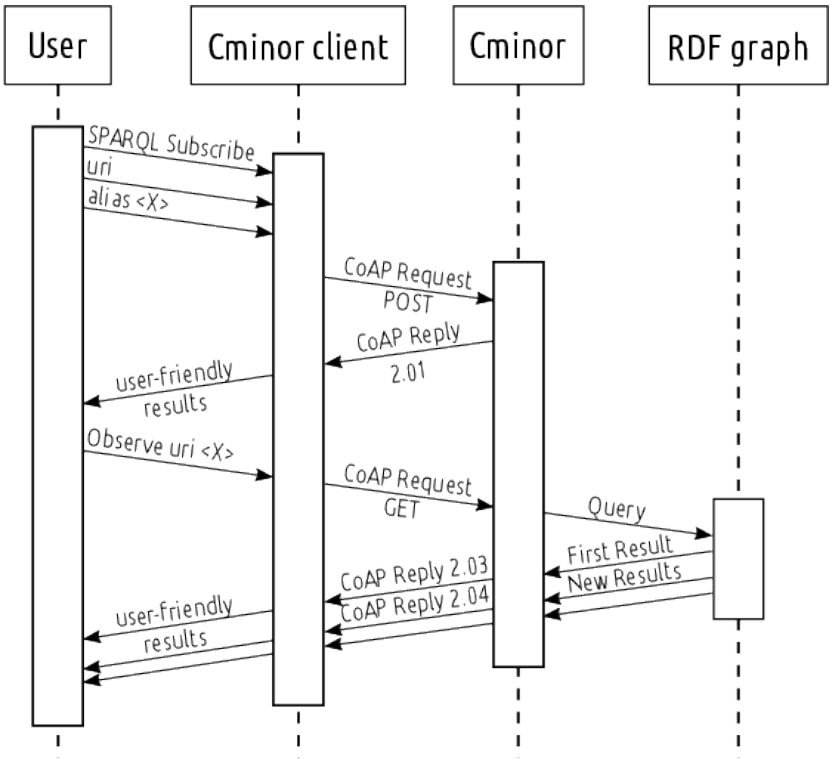


Figure 4.21: Sequence diagram for registration of SPARQL subscription and subsequent observation [162]

value of a feature changes, the server must update the state of the resource monitoring the related subgraph and then notify the change to all the observers. In this test the number of observers is n, with $n = 10 \cdot i$ (with $i = 1 \dots 10$).

Evaluation of the latency

This Section proposes the results of a first assessment of the latency of C Minor, and in particular the latency of the CoAP protocol, therefore the query, update or subscription mechanisms were not involved in these measurements. The evaluation was carried out triggering 100 CoAP requests to the C Minor broker, each n bytes long ($n = 20 \cdot i; i = 1, \dots, 25$), and measuring two of the metrics proposed in [166]: the Flow Completing Time (FCT) and the CoAP Round Trip Time (C-RTT). The first consists of the time interval between the sending of first request and the receiving of the last response, while the latter, is the average elapsed time between the sending of the original CoAP request and receiving of the CoAP response. Fig. 4.28 shows that, the time required to complete the flow of requests is not influenced by the length of the messages. This result is also confirmed by Fig. 4.29 with average CoAP Round Trip Time inferior to 5 ms.

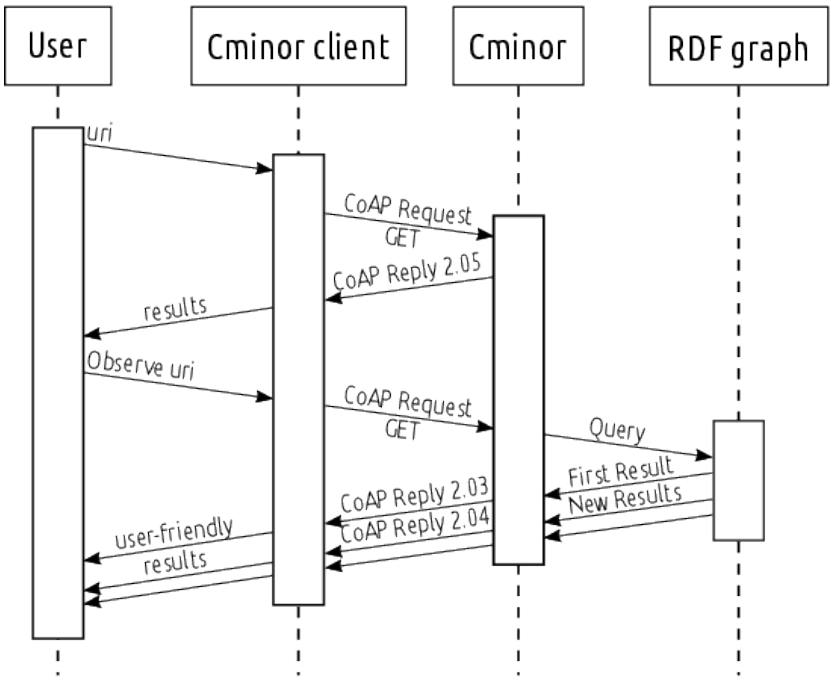


Figure 4.22: Sequence diagram for discovery of SPARQL subscriptions and subsequent observation [162]

4.6 Conclusion

In this Chapter, I described my Research in the field of semantic context brokers for context-aware and SWoT applications. This work was framed in two main areas: the Smart-M3 interoperability platform, and its direct descendant SEPA. As regards Smart-M3, three different implementation of semantic context broker were proposed, each of them with different Research contributions: SPS [124] introduced an efficient algorithm to process subscriptions, a novel primitive (i.e., the Delayed SPARQL Update) as well as a novel design pattern (i.e., based on the distinction of clients among producers, consumers and aggregator). A second SIB developed during my PhD is pySIB [150], characterized by a novel implementation of the SSAP protocol designed to be fast and efficient. Third, the OSGi SIB [148, 155] is an implementation based on the Java framework OSGi, designed to be employed in industrial environments. It introduced a novel primitive named Persistent Update. Moving on the SEPA front, this work was aimed at the development of a new platform oriented at supporting Big Data applications in the Semantic Web of Things through Web standards [159]. The latest contribution in this Research area was inspired by SEPA and is oriented at highly constrained environments, such as the Internet of Musical Things. The resulting platform (i.e., C Minor [162]) exploits a lightweight IoT protocol to permit the use of semantics also on the

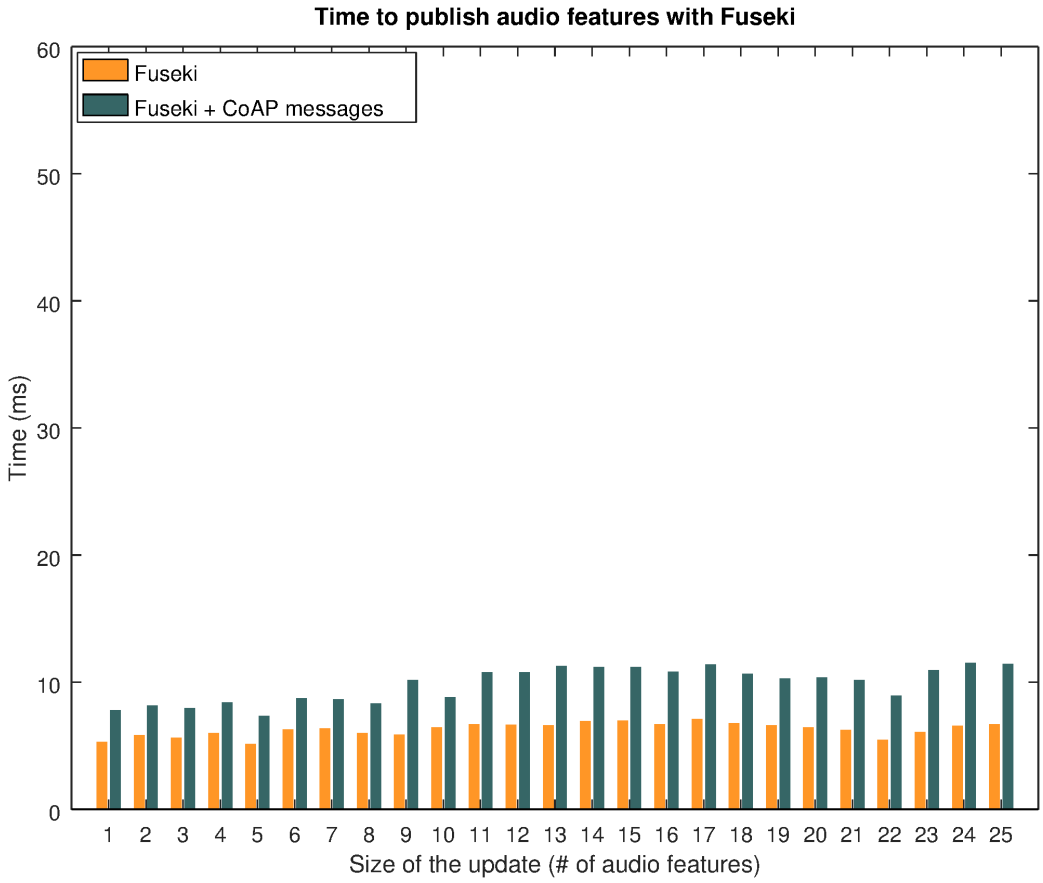


Figure 4.23: Time to publish a context composed by n audio features with a SPARQL Update on C Minor + Fuseki ($n \in [1, 25]$) [162].

lowest application level, directly on constrained nodes.

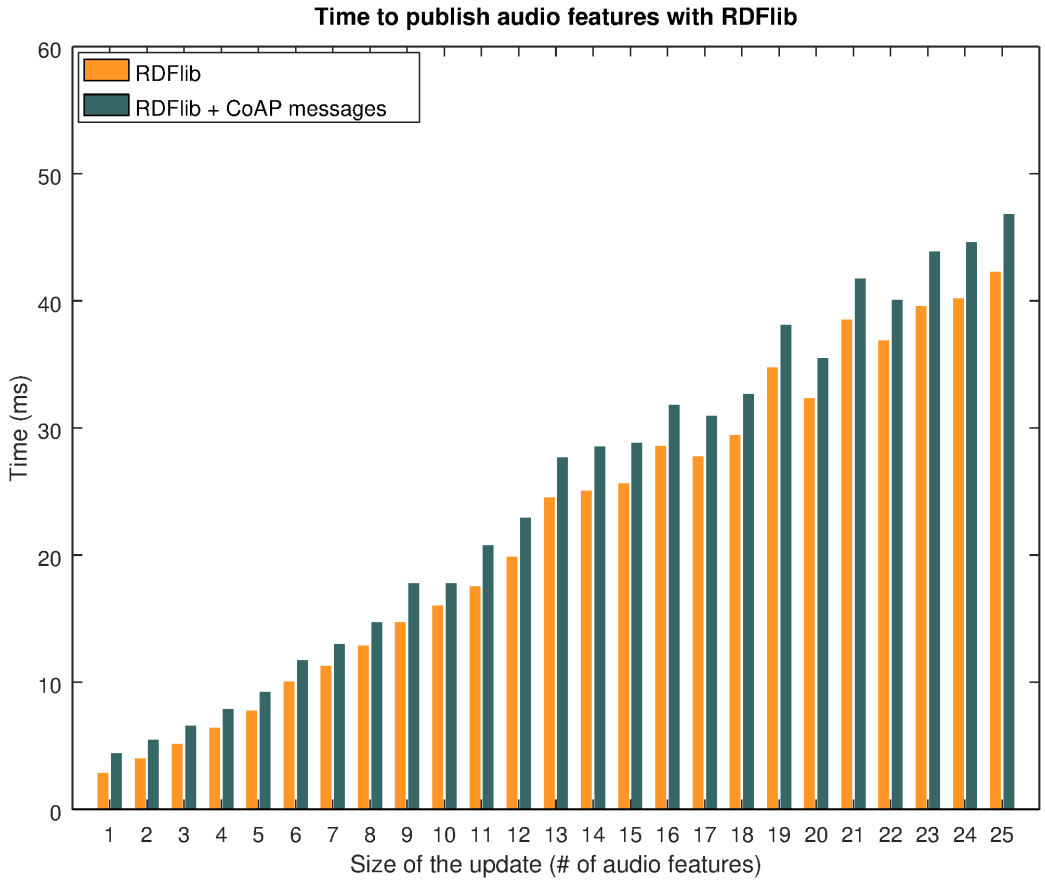


Figure 4.24: Time to publish a context composed by n audio features with a SPARQL Update on C Minor + RDFlib ($n \in [1, 25]$) [162].

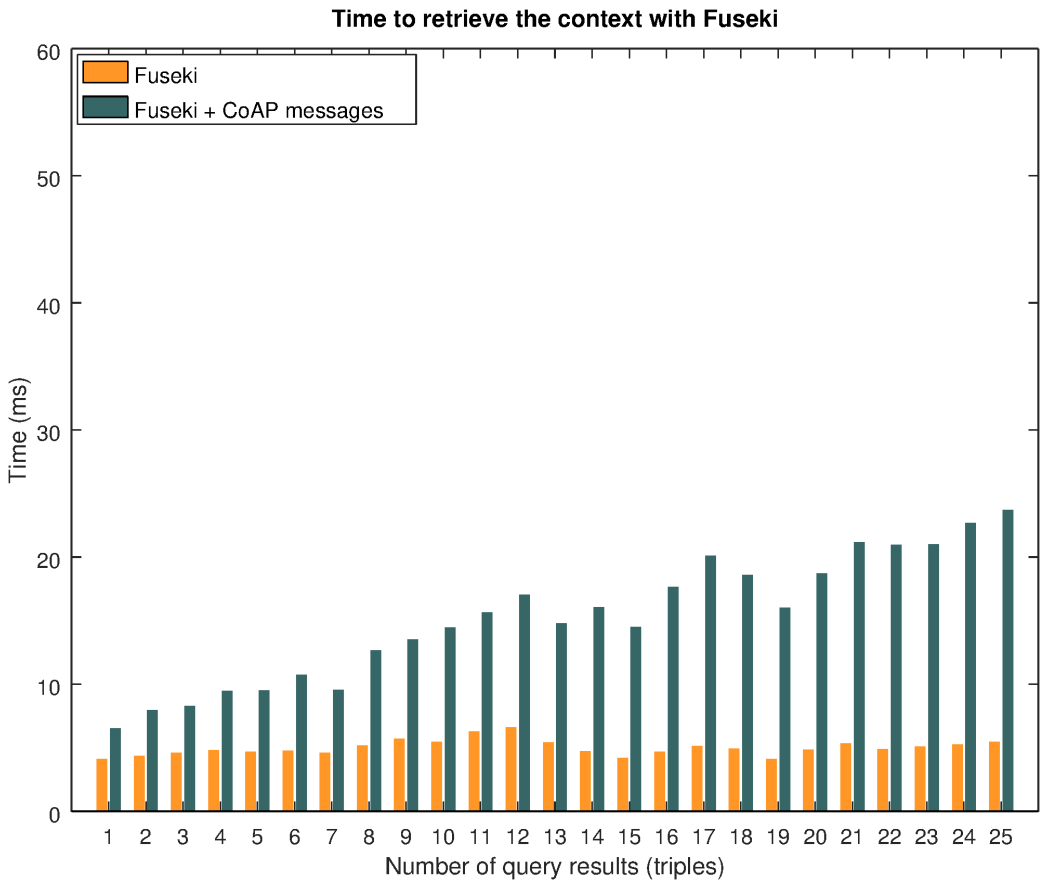


Figure 4.25: Time to perform a SPARQL Query on C Minor with Fuseki [162].

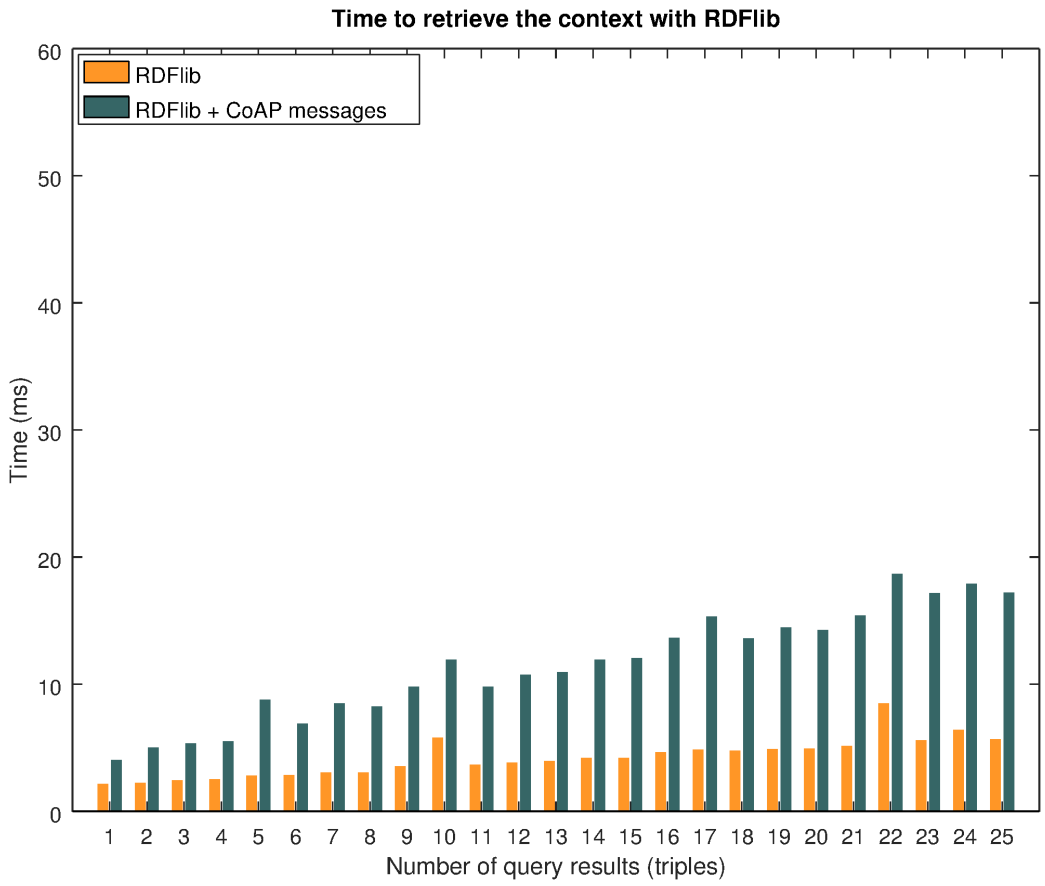


Figure 4.26: Time to perform a SPARQL Query on C Minor with RDFlib [162].

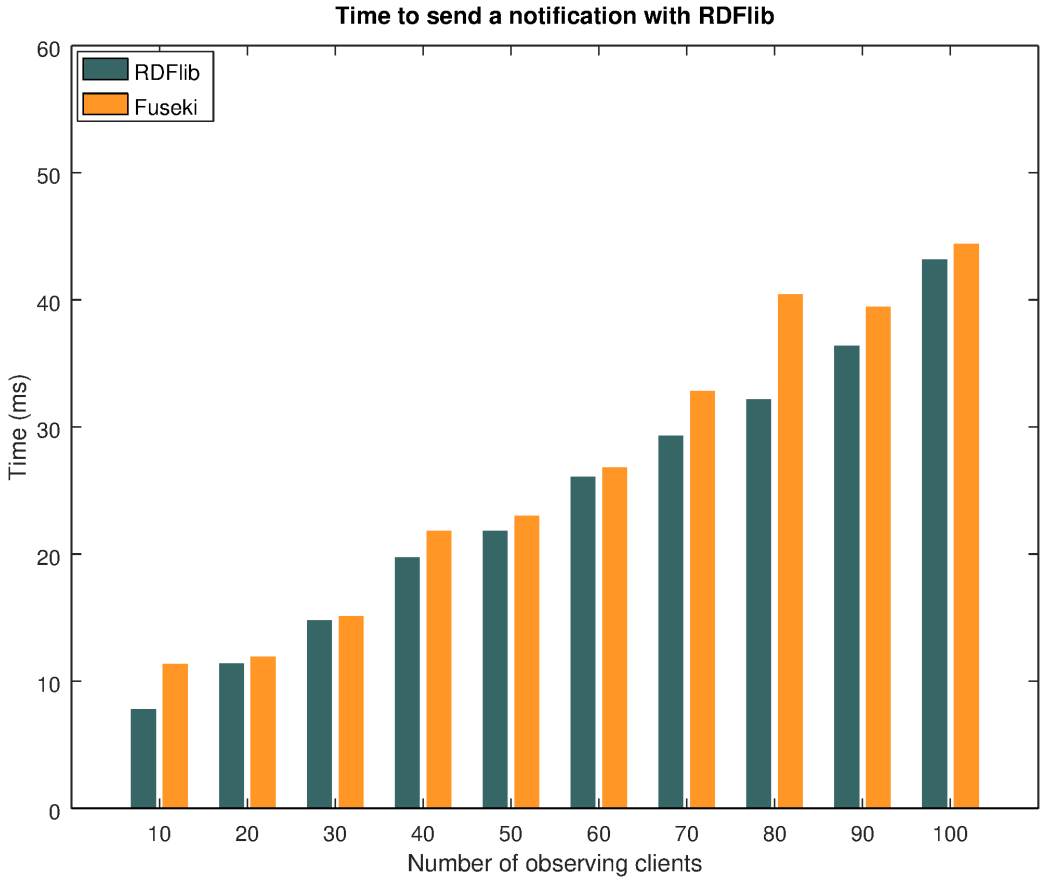


Figure 4.27: Time to send a notification to n observers ($n = 10 \cdot i, i = \{1, \ldots, 25\}$) [162].

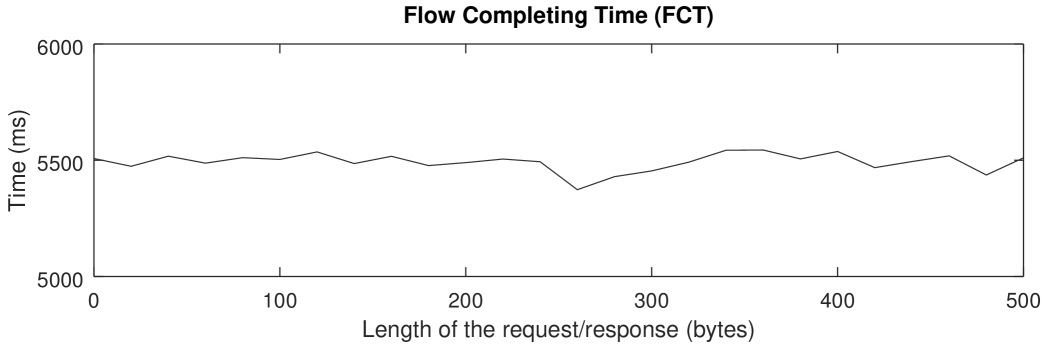


Figure 4.28: Flow Completing Time on C Minor[162].

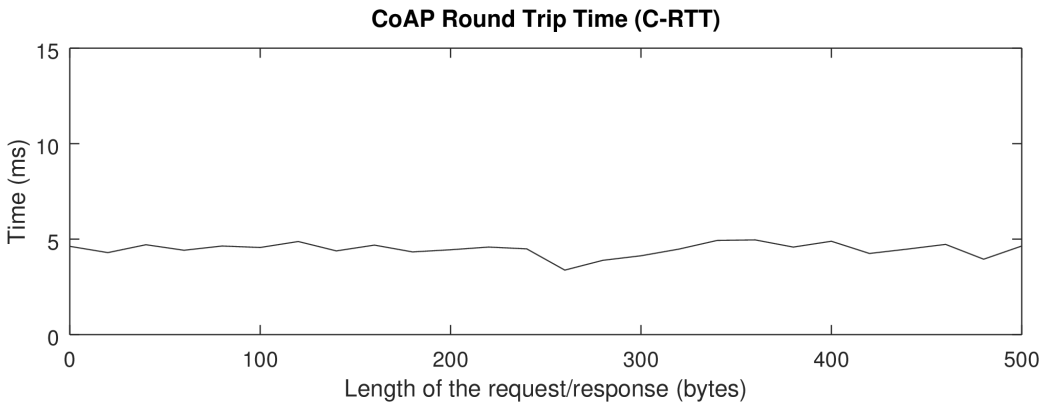


Figure 4.29: CoAP Round Trip Time on C Minor [162].

Chapter 5

Benchmarking semantic publish/subscribe middlewares

Contents

5.1 Introduction

This Chapter presents my research activity concerning the benchmarking of semantic publish/subscribe middlewares. More in detail, this Chapter will present a first activity related to a benchmark for the Smart-M3 platform (the lamp-post benchmark developed to assess the performances of the SPS broker). Then, a platform for the execution of benchmarks for semantic publish/subscribe middlewares named Performance Evaluation Suite (PES) is presented. Lastly, an ongoing activity concerning the design, development and evaluation of a SWoT-specific benchmark, named **SWoT_Bench** will be introduced. The rationale behind this benchmark is that modern Internet of Things applications can rely on Semantic Web technologies to solve the discoverability problem [24, 94], i.e., to discover and interact with *things* in an easy way. A semantic approach to the discovery of things requires a **central directory** (e.g., a semantic broker) hosting the description of the things. Then, SWoT_Bench is framed in this specific context, providing a systematic way to assess the performance of a semantic broker during the discovery of Web Things. This benchmark is then being designed to evaluate the behaviour and performance of the central node with a specific attention to the **subscription mechanism**. As will be discussed in detail in Section 5.2, none of the existing benchmarks fits the needs of this scenario.

5.2 Related work

Since the Semantic Web of Things is a very new research area, to the best of my knowledge, no specific benchmark exists yet. Anyway, this area, basically relies on three different research areas (i.e., Semantic Web, publish/subscribe mechanisms, Internet of Things), so the current state of the art will be analyzed by separately looking at benchmark specifically designed for these categories.

Benchmarking Semantic Web applications The first research area that deserves to be carefully analyzed is the one related to benchmarking Semantic Web applications. In this category we may find many different benchmarks. SPARQL Performance Benchmark (SP^2B) [154] is one of the most popular. It is a language-oriented benchmark that aims to evaluate the performance of SPARQL endpoints with respect to a given set of queries and has been designed to be exhaustive. In fact, SP^2B is made up of seventeen queries designed to test

every construct of the SPARQL Query Language. Despite being one of the best benchmarks in literature, SP^2B does not fit the SWoT scenario where publish/subscribe architectures should be evaluated. Furthermore the test scenario is founded on the DBLP Computer Science Bibliography, a test case very different from the Semantic Web of Things one. Very similar considerations can be made about the Berlin SPARQL BenchMark (BSBM) [151], a use-case driven benchmark focused on e-commerce. It is oriented at measuring performances of SPARQL endpoints not taking into consideration the publish/subscribe mechanism. Three main principles guided the design of the Leigh University BenchMark (LUBM) [152]: 1. it is based on extensional queries rather than intentional ones; 2. arbitrary scale of data to verify the behaviour of the system; 3. ontology of moderate size and complexity since the focus is on data. The first two principles also guided the development of the benchmark that will be presented in Section 5.5. In our case the proposed ontology has a moderate size, but that was not one of our requirements. Semantic Web applications also involve operations like the update of the knowledge base. The SPARQL Update language provides a formalism to properly update the knowledge base with a syntax similar to that of SPARQL Query language. Concerning the SPARQL Update language, it is worth mentioning SPARUB [167]. In fact, SPARUB is a very recent (2017) benchmark for the SPARQL Update language that tries to complement SPARQL Query analysis with the evaluation of the update of a knowledge base. Based on this review of the main benchmarks related to Semantic Web applications, it is possible to affirm that none of the existing suites considers publish-subscribe information systems, but are rather focused on static knowledge bases.

Benchmarking publish-subscribe applications The analysis of the state of the art proceeds having a look at publish-subscribe applications. jms2009-PS [168] is the first benchmark for publish-subscribe Message-oriented Middleware. It is based on the SPECjms2007 workload, the first industry-standard benchmark. Being related to Java Message Service and focused on industry (and specifically to a supermarket supply chain) it is too specific to fit the Semantic Web of Things scenario. Furthermore, as highlighted in [132], it implements a pure store-and-forward mechanism. If we consider semantic publish-subscribe systems instead, it is worth mentioning the work by Murth et al. [132]: the main drawback of this paper is the number of metrics considered that is really poor. Only the notification time and the publication throughput have in fact been introduced. In this Thesis I extend the set of Performance Indicators to measure through a benchmark. Despite being an extension of LUBM specifically designed for semantic publish-subscribe systems, the use case is still too far from the application domain of the Semantic Web of Things.

Benchmarking IoT applications In [169] a benchmark toolkit for IoT Big Data scenarios is presented: IoTABench. It is focused on large volumes of synthetic sensor data with realistic properties, and the evaluation is performed on a real life use case (smart metering). Unfortunately, this benchmark does not consider using Semantic Web technologies for data representation. Shukla and Simmhan in [170] propose a benchmark including 13 common IoT tasks forming micro-benchmarks. Later on, the same authors extended their benchmark with 14 new tasks, proposing RIoTBench [171]. As the previous work, also this one is not intended for applications using Semantic Web knowledge bases. [172] is aimed at benchmarking RDF stream processing systems.

5.3 Smart-M3 lamp-posts benchmark

The work carried out on the development of the SPS broker brought to the definition of a benchmark inspired by a public lighting system of a small city with large, medium, small, and very small roads (i.e., roads with up to 100, 50, 25, and 10 lamp-posts). In order to describe the benchmark, the following Sections will propose details about: 1) the metrics to be measured; 2) the ontology and knowledge base; 3) the updates and subscriptions to be tested; 4) the test procedure.

5.3.1 Metrics

Five Performance Indicators (PIs) have been defined to assess the performance of the SPS platform:

- **Average number of updates per unit of time**:

$$Ups = \frac{n}{T_{TOTAL}}$$

- **Average number of subscriptions processed per unit of time**:

$$Sps = m \cdot Ups$$

- **Average number of triples processed per unit of time**:

$$Tps = Nu_{AVG} \cdot Sps$$

- **Engine to SPARQL Endpoint impact factor**:

$$E2E = \frac{T_{TOTAL} - T_{UPDATE}}{T_{UPDATE}}$$

- **Minimum Notification Latency**:

$$NL_{min} = \min\{tl_{i,j} + tb_{i,j} | i = 1, \ldots n, j = 1, \ldots m \wedge te_{i,j} \neq 0\}$$

- **Maximum Notification Latency**:

$$NL_{max} = \max\{tl_i + tb_i + te_i | i = 1, \ldots n \wedge te_i \neq 0\}$$

- **Notification Latency Range**:

$$NL = [NL_{min}, NL_{max}]$$

Ups is an indicator of how many updates are processed per unit time in average, while *Sps* measures how many subscriptions are processed per unit time and therefore it is directly related to the subscription profile cardinality. Since *Ups* and *Sps* do not consider events complexity, that depends on the number of triples processed, *Tps* is introduced to provide an indication of the computational load in terms of average number of triples processed per unit time.

E2E is motivated by the consideration that the engine stands on top of a SPARQL endpoint. Then, this PI measures the overhead introduced by the semantic event detection and notification capability to a SPARQL endpoint. The closer this PI is to zero, the lower is the overhead introduced by the SUB engine on the underneath SPARQL endpoint.

The notification latency range NL is proposed as a measure of the time span between updates and notifications when events are detected. The lower bound (NL_{min}) can only be reached when the engine works in parallel mode. On the contrary, NL_{max} occurs when the last SPU notifies its client in sequential mode.

5.3.2 The knowledge base

Table 5.1 provides the details about the ontology and the RDF store size (i.e., the number of triples). The city has 9500 lamp-posts represented by 334k RDF triples and each post is equipped with a lamp and two sensors (i.e., temperature and presence).

Each road and each lamp of a road are identified by a URI, respectively, in the form: `ROAD_URI_X` and `LAMP_URI_X_Y`, where `X` is a road identifier (i.e., in the range 1..310), while

Table 5.1: Benchmark knowledge base

OWL Ontology T-Box content				
Classes			27	
Individuals			26	
Object properties			16	
Datatype properties			8	
OWL Ontology A-Box content				
Road types	**N_{LAMP}/Road**	**Roads**	**Lamp-posts (Sensors)**	**RDF Triples**
Very Small	10	100	1K (2K)	35K
Small	25	100	2.5K (5K)	88K
Medium	50	100	5K (10K)	175K
Large	100	100	1K (2K)	35K
Total		**310**	**9.5K (19K)**	**334K**

`Y` is a lamp identifier within a road (i.e., `Y` varies from 1 to `N_LAMP` , where `N_LAMP` is the amount of lamp-posts in road `X`). Each lamp is characterized by a status (i.e., `ON`, `OFF`, and `BROKEN`), a dimming value (percentage) and a type (i.e., whether it is a led or a traditional lamp). Each post is identified by its geographical position (i.e., latitude and longitude), while each sensor is represented by a set of properties: the type, the unit of measure, the value and a timestamp.

5.3.3 Experiments

The benchmark designed for SPS is based on two types of SPARQL updates (U_{LAMP}`(X,Y)` and U_{ROAD}`(X)`) and subscriptions (S_{LAMP}`(X,Y)` and S_{ROAD}`(X)`).

The first SPARQL Update (i.e., U_{LAMP}`(X,Y)`) is used to set to 100% the dimming value of lamp `Y` of road `X`:

```
1 DELETE {
2   LAMP_URI_X_Y ns:hasDimmingValue ?dimming
3 }
4 INSERT {
```

```
5   LAMP_URI_X_Y ns:hasDimmingValue "100"
6 }
7 WHERE {
8   LAMP_URI_X_Y ns:hasDimmingValue ?dimming
9 }
```

The SPARQL update $U_{ROAD}(X)$ is used to set to 100% the dimming of all the lamps of road X and is defined as:

```
1  DELETE {
2    ?lamp ns:hasDimmingValue ?dimming
3  }
4  INSERT {
5    ?lamp ns:hasDimmingValue "100"
6  }
7  WHERE {
8    ?lamp ns:hasDimmingValue ?dimming .
9    ?post ns:hasLamp ?lamp .
10   ?road ns:isConnectedTo ?post .
11   FILTER(?road = ROAD_URI_X)
12 }
```

The first subscription $S_{LAMP}(X,Y)$ is a fine-grain one aimed at detecting a change of the dimming value of lamp Y of road X:

```
1 SELECT ?dimming
2 WHERE {
3   LAMP_URI_X_Y ns:hasDimmingValue ?dimming
4 }
```

while $S_{ROAD}(X)$ is a coarse-grain subscription sensitive to the update of the dimming value of any lamp placed on road X:

```
SELECT ?lamp ?dimming
WHERE {
  ?lamp ns:hasDimmingValue ?dimming .
  ?post ns:hasLamp ?lamp .
  ?road ns:isConnectedTo ?post .
  FILTER(?road = ROAD_URI_X)
}
```

Based on these updates and subscriptions, two experiments named `LAMP` and `ROAD` were defined. Before proceeding with the description of the two tests, it is worth analyzing the LUTT content and `cts` size of the two subscriptions, as summarized in Table 5.2

Table 5.2: LUTT content and CTS size for fine- and coarse-grain subscriptions

$S_{LAMP}(X,Y)$	**LUTT**	ROAD_URI_X	ns:isConnectedTo	*
	CTS			1 triple
$S_{ROAD}(X)$	**LUTT**	ROAD_URI_X	ns:isConnectedTo	*
		*	ns:hasLamp	*
		*	ns:hasDimmingValue	*
	CTS			$\approx 19K$ triples

A subscription profile `S` has been defined, including 1000 fine grain subscriptions (50 for very small, 100 for small, 150 for medium and 700 for very large roads) and 4 coarse grain subscriptions (one for each road type). The subscription profile is better detailed in Table 5.3. The table shows that with this subscription profile, a notification is triggered whenever any of the dimming value changes.

Road Type	**Sub.** $S_{LAMP}(X,Y)$	**Sub.** $S_{ROAD}(X)$	**Monitored Lamps**
Very small	50	1	60
Small	100	1	125
Medium	150	1	200
Large	700	1	800
Total	**1000**	**4**	**1185**

Table 5.3: Subscription Profile `S`

A formalization of the subscription profile is:

$$
\begin{aligned}
S = \{ & \\
& S_j \equiv S_{LAMP}(X,Y) \mid j = 10(X-1)+Y, X \in \{1\dots5\} \wedge Y \in \{1\dots10\}\} \\
& \cup\, S_j \equiv S_{LAMP}(X,Y) \mid j = 100+25(X-101)+Y, X \in \{101\dots104\} \wedge Y \in \{1\dots25\}\} \\
& \cup\, S_j \equiv S_{LAMP}(X,Y) \mid j = 200+50(X-201)+Y, X \in \{201\dots203\} \wedge Y \in \{1\dots50\}\} \\
& \cup\, S_j \equiv S_{LAMP}(X,Y) \mid j = 300+100(X-301)+Y, X \in \{301\dots307\} \wedge Y \in \{1\dots100\}\} \\
& \cup\, S_j \equiv S_{ROAD}(X) \mid (j,X) \in \{(1001,6),(1002,105),(1003,204),(1004,308)\} \\
\} &
\end{aligned}
\tag{5.1}
$$

Two update profiles $\mathtt{U_{LAMP}}$ and $\mathtt{U_{ROAD}}$, both made by 310 updates have been defined. The first profile updates one lamp per road, resulting then in 310 lamps updated. The second profile instead, is composed by a set of producers updating the dimming value of all the lamps on a road. The second profile updates 9.5k lamps. The two update profiles can then be formalized as:

- $U_{ROAD} = \{U_i \equiv U_{ROAD}(i) | i \in \{1\dots310\}\}$
- $U_{LAMP} = \{U_i \equiv U_{LAMP}(i,1) | i \in \{1\dots310\}\}$

For both the update profiles, the number of triples per update is shown in Table 5.4.

Table 5.4: Numbers of triples per update of the two experiments

ROAD	LAMP
10 $(1,\dots,100)$ 25 $(101,\dots,200)$ 50 $(201,\dots,300)$ 100 $(301,\dots,310)$	1

If we define the average number of triples updated by a single update primitive within an experiment as:

$$Nu_{AVG} = \frac{1}{n}\sum_{i=1}^{n} Nu_i$$

and the LUTT Hit Ratio (LHR) as:

$$LHR(\%) = \frac{100}{m \times n}\sum_{i=1}^{n}\sum_{j=1}^{m} h_{i,j}$$

then we can affirm that the ROAD experiment is computationally heavier than LAMP, since:

$$Nu_{AVG}(ROAD) = 31 > Nu_{AVG}(LAMP) = 1$$

and:

$$LHR(ROAD) = 0.72\% > LHR(LAMP) = 0.40\%$$

Given the update and subscribe profiles, the following amounts of notifications are sent to the subscribers during the reported experiments:

- ROAD: 1004 notifications (all S_j are triggered)
- LAMP: 23 notifications (19 S_{LAMP} subscriptions and 4 S_{ROAD} subscriptions are triggered).

5.3.4 Test process and evaluation

The benchmark has been executed to assess the performance of the SPS architecture presented in Section 4.3.2. The experiments ROAD and LAMP have been executed on a test bed consisting of:

- A machine (Intel Core i7-2630QM CPU @ 2.00 GHz × 8 cores, 8 GB RAM) hosting both the SUB engine and the SPARQL endpoint (i.e., Virtuoso).
- A remote multithreading C# client application running on a Virtual Box machine (4 GB RAM, 1 CPU, execution cap 100%). The client machine is a MacBook Pro, Intel Core i7 2.2 GHz, 16 GB RAM. Network connection through a 100 Mb/s LAN.

As a first initialization step, the ontology is loaded on the RDF store. After that, the simulator starts all the 1004 subscriber threads. Once all the subscribers are up and running, the simulator sequentially issues all the Update requests to the engine that is configured to run in sequential mode (i.e., one core is used for both the scheduler and all the SPUs). In this way, at the end of each experiment, it is possible to extract the timing profile, logged by the SUB engine, related to a single request. Each experiment has been repeated several times and average values of the timing components have been calculated to evaluate the parameters of the performance model.

Results of the execution of the two exeperiments repeated with and without using LUTTs are reported in Tables 5.5 and 5.6

Results reported in Tables 5.5 and 5.6 shows that the impact of the LUTT is similar for both the experiments (improvement of two orders of magnitude). In fact, in both the

Table 5.5: Performance Indicators of the two experiments `LAMP` and `ROAD` executed with and without LUTT

Experiment	LUTT	Ups	Sps	Tps	Nl_{max}	E2E
`LAMP`	✔	68	68K	68K	0.09s	1
`LAMP`	✖	0.55	557	557	1.80s	245
`ROAD`	✔	3.8	3.8K	117.3K	0.54s	1.8
`ROAD`	✖	0.03	29.4	901	129s	367

Table 5.6: Timing component for the two experiments `LAMP` and `ROAD` with and without LUTT

Experiment	LUTT	$T_{booster}$	T_{LUTT}	T_{total}
`LAMP`	✔	2s	0.3s	4.6s
`LAMP`	✖	9min	-	9min
`ROAD`	✔	50.2s	2.1s	81.3s
`ROAD`	✖	176min	-	176min

experiments 1000 out of 1004 LUTTs stop nearly all the triples from progressing to the Booster stage, resulting in a dramatic reduction of the computational load.

5.4 Smart-M3 performance evaluation suite

This Section describes the activity carried out during the 1st PhD year to design and develop a Performance Evaluation Suite (PES) for Semantic Publish-Subscribe MOMs.

The resulting implementation was realized taking Smart-M3 as a reference platform: publish and subscribe primitives are then expressed using both SPARQL 1.1 (i.e., respectively as SPARQL Update and SPARQL Query) or through an RDF triple pattern serialization formalism named RDF-M3. In both cases, requests are encapsulated in SSAP messages. This is not limiting since, as will be detailed later on, porting PES to another semantic publish/subscribe MOM would involve only one of the modules of the architecture.

The main contribution of this work consists in a portable set of tools and methods to quickly prototype experiments (like the ones proposed in Section 5.3.3) to measure relevant performance metrics of the target platform. Moreover, PES has been designed to allow creating new experiments specifically designed for the target domain, as well as relying on existing

benchmarks (e.g., SP^{2}B [154] or LUBM [152]). The definition of a benchmark includes a set of updates and queries, the optional subscriptions to activate, along with the related RDF datasets (e.g., OWL, N3) to initialize the system. In fact, PES provides a data loader to populate the knowledge base prior to run the experiments. The evaluation outcome is in the form of graphical representations of the main results (i.e., SVG or PNG files) and includes the statistical analysis on the measured timing components (e.g., median, variance, maximum and minimum values included in a CSV file).

5.4.1 Software architecture

PES is a set of software modules released under the GNU General Public License 3.0 aimed at benchmarking semantic publish-subscribe middlewares. The entire suite is developed with the Python programming language. PES is multiplatform, so it supports all the major operating systems. Its software architecture is depicted in Fig. 5.1and described in the following lines.

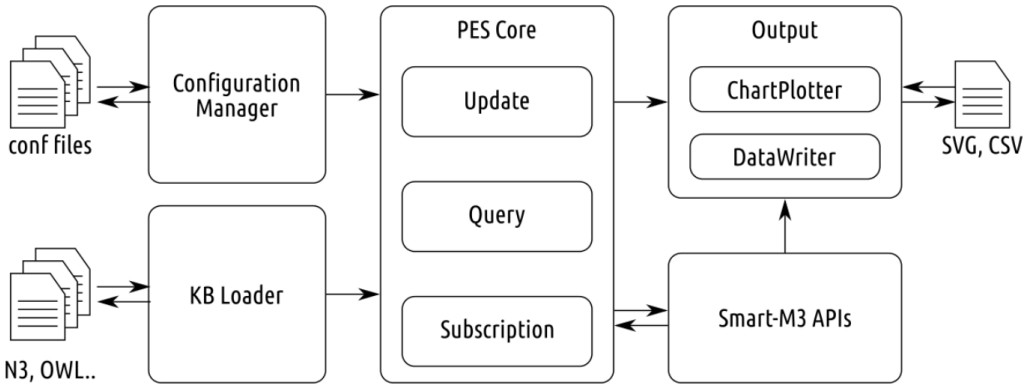


Figure 5.1: Software architecture of the Performance Evaluation Suite

- **The Configuration Manager**: the behavior of the suite depends on the directives specified in its configuration files (compliant with the specifications contained in [173]) and from the command line. The main parameters specified from the command-line or through the global configuration file are the list of the SIBs to be tested (composed by IP address and port and by the required interaction protocol, e.g., SSAP [174] or JS-SAP [150]) and the type of test to be performed (e.g., a *query test*). Other configuration files are test-specific and are used to configure the desired benchmark. A benchmark is defined by proper configuration files. Each of these configuration files allows specifying the initial knowledge base, the number of iterations to perform, the desired output format for the chart (i.e., SVG or PNG) and if the CSV output file should be produced or

not. Depending on the type of test to be performed, the configuration file may include different sections.

- **The KB Loader**: The KB Loader is used to load the triples that constitute the initial knowledge base when a performance test is started. This component supports the N3 and the OWL serialization formats. The first grants the compatibility with all the exisiting benchmarks adopting this format (e.g., the SP^{2}B benchmark [154]). The KB Loader sends n triples at a time to the SIB, where n is a user-defined parameter depending on the trade-off between KB size, the number of operations to load them and efficiency of the target broker to process large input files.

- **Core**: The core of PES is composed by a set of test modules, among which it is worth mentioning:

  - **Update Test**: allows measuring the performance of an update request with either SPARQL or RDF-M3. For all the SIBs to be tested, the module performs a series of insertions of n triples where n ranges from n_{MIN} to n_{MAX} with step s. Each of these parameters is configured exploiting the Configuration Manager. Every test is repeated i times, where i is the number of iterations needed to obtain sufficient statistical samples. The mean value, the minimum and maximum and the variance are then calculated. The time elapsed to perform the update operation is measured at client side, so it can be considered as the sum of the following components:

    $$t_{update} = t_{kp\_req} + t_{net\_req} + t_{sib\_req} + t_{sib\_elab} + t_{sib\_rep} + t_{net\_rep} + t_{kp\_rep}$$

    where:

    - t_{kp_req} and t_{kp_rep} respectively represent the time needed by the Knowledge Processor to encode the request and parse the reply;
    - t_{net_req} and t_{net_rep} are the number of milliseconds used to transfer the request and response packets over the network;
    - t_{sib_req}, t_{sib_elab} and t_{sib_rep} represent the time used by the context broker to parse the received request, elaborate the request and produce a reply.

    The current implementation of the PES only measures t_{update}. Measuring the time elapsed to perform an update allows assessing whether or not the SIB is able to timely store and share the information sent by the KP. The module can be configured to run with active subscriptions to evaluate their impact on the platform.

- **Query Test**: this module measures the performances of the RDF store. For each formalism, two kinds of tests can be performed:
  - **Simple test**: the knowledge base is loaded, then the query is performed;
  - **Complex test**: the knowledge base is loaded in several steps and at the end of each step the specified query is performed.

  The module can be configured, as for the updates, exploiting the Configuration Manager previously described. The parameters used to set the behavior of the module are:
  - The type of query test to perform (i.e., simple or complex);
  - The files containing the knowledge base to load together with their format (i.e., N3 or OWL) and the desired step;
  - The query to perform together with its type (i.e., SPARQL or RDF-M3);
  - The number of iterations to perform.
- **Subscription Test**: represents the most significant contribution since none of the existing benchmarks for semantic publish-subscribe systems permits to properly characterize the performance of the subscription engine.

  This test allows subscribing to a given triple pattern (using RDF-M3) or to a subgraph (by means of the SPARQL Query language), then to perform updates of the knowledge base and measure the time in milliseconds required by the KP to receive the expected notification. The Subscription Test can also be used to instantiate a variable number n of KPs, each one with the same subscription, in order to calculate a notification loss ratio or to perform stress tests. The Subscription Test can be configured with a dedicated configuration file that states the initial knowledge base (a list of N3 or OWL files to load), the subscriptions and the updates to perform and the desired number of iterations.

- **The Output Module**: this module plots the results using the pygal library that allows rendering the charts on SVG or PNG files (an example is proposed by Fig. 5.2). Moreover, all the measured values are registered in a proper CSV file.

  The following listing reports an example of a CSV file produced during the execution of a subscription test. The first field is the name of the target SIB. The following fields contains a list of the collected notification times, the mean value, minimum and the maximum values and the variance. All the values are expressed in milliseconds:

```
OSGi,2.819,...,2.986,1.792,3.948,0.281
```

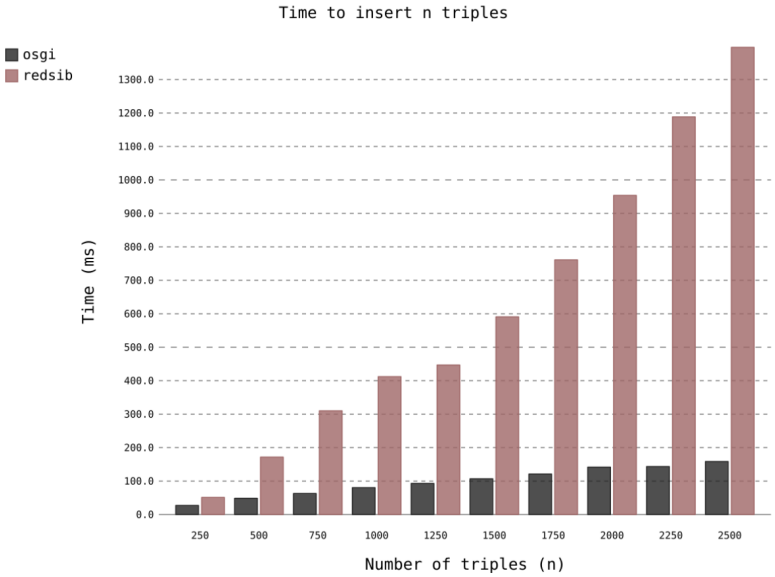


Figure 5.2: Example chart plotted by PES

```
SPS,3.789,...,2.538,1.381,3.789,0.57
pySIB,1.054,...,2.392,1.003,3.51,0.673
```

- **The Smart-M3 APIs**: even though PES is not constrained to a specific platform, the first implementation provides support for Smart-M3. This is not a loss of generality, since it is sufficient to replace this module and its calls with the proper APIs to evaluate the performance of other Semantic Publish-Subscribe MOMs.

  The KPs forming PES have been developed exploiting the Smart-M3 APIs that make possible the interaction with the SIB. These libraries were not developed ad-hoc for the purpose of this project, but are external modules included into PES. Since PES is developed in Python, the APIs adopted by the suite are the Python Smart-M3 APIs (including the one providing support for the JSSAP introduced by pySIB [150]).

5.4.2 Conclusion and future work

This Section has presented PES, a suite designed to measure the performance of semantic publish/subscribe middlewares with respect to user-defined or existing benchmarks. The suite, implemented taking Smart-M3 as a reference platform, has demonstrated to be a useful tool to assess the performance of the SIBs: the charts plotted by PES allowed characterizing the performance improvement obtained by subsequent implementations of the broker.

Future work concerning the Performance Evaluation Suite will be aimed at including support for the new generation of the Smart-M3 platform, known as SEPA. Moreover, the

benchmark will be extended to support the description of the experiments through SAP files (see Section 4.4.3) and to support measuring all the Performance Indicator foreseen by the SWoT benchmark (see Section 5.5).

5.5 SWoT_Bench

This Section introduces the first benchmark designed to assess the performance of Semantic Web of Things applications. More specifically, the focus is mainly on the broker hosting the Thing Descriptions of the Web Things. Section 5.5.1 introduces the scenario of the test. On the technical side, Section 5.5.2 reports a detailed overview of the SWoT Ontology playing a central role in the benchmark, while the SPARQL Updates, Queries and Subscriptions composing SWoT_Bench are reported in Section 5.5.3.

5.5.1 Scenario

From a high-level perspective, SWoT_Bench is oriented at evaluating brokers in Semantic Web of Things scenarios. But the SWoT research area involves very different operations and very different application domains. In literature, it is possible to find many examples of the application of Semantic Web technologies to the Web of Things, but one of the areas where this combination is really promising is to solve the problem of **discoverability** [24, 175]. Generally speaking, discoverability is the ability to discover the URIs of the resources available in a smart space. In the WoT, this task is more challenging if compared to the traditional Internet. In fact, in the WoT, many instances of the same resource are usually available at the same time (e.g., multiple devices with similar capabilities) and their physical and digital location, as well as their lifetime is highly mutable. Then, the discoverability in the Semantic Web of Things is the ability to **dynamically** discover Web Things (and their capabilities).

5.5.2 Ontology

The ontology used by SWoT_Bench is the SWoT Ontology developed with my collegues in the ARCES research center. This ontology leverages the previous work by Serena et al. [175]. In this paper, authors propose a Web of Things ontology based on the requirements highlighted by the W3C Web of Things Working and Interest Groups. The resulting ontology is aimed at mapping *what*, *where* and *how* things can be discovered and accessed. Then, the Web of Things Ontology[1] defines all the classes required to semantically map a **Thing Description** [20]. The main classes are then:

[1] http://iot.linkeddata.es/def/wot/index-en.html

- `Thing` – a Thing is any thing which has a distinct and independent existence and can have one or more web representations [175]. In our domain, we can declare a Thing (or *Web Thing*) as a (physical or virtual) device exposing a set of properties and/or actions and/or events. Examples of Web Things are a temperature sensor or a valve controlling a radiator.

- `InteractionPattern` – This class is used to map all the possible ways to interact with a Web Thing. This class is subclassed by `Property`, `Action` and `Event`.

- `Property` – Properties describe a readable and/or writeable attribute of the Web Thing. The value of a property is defined according to a proper data schema. Considering a temperature sensor as the reference Web Thing, a property could be represented by the brand of the sensor. While this is obviously a static property, a dynamic property is the frequency of sensing that can be modified at runtime, for example, to extend the battery life.

- `Action` – The class `Action` is used to map all the actions provided by a device. Both input and output of an action, if present, are defined according to a data schema. Possible actions for the Web Thing realizing a radiator valve are *open* and *close*. While the input of these actions determines how much the valve should be opened/closed, the output is a physical reaction, so a data schema is not used.

- `Event` – Events are used to notify a particular condition (e.g., a critical battery level detected by the temperature sensor).

- `DataSchema` – As previously mentioned, a data schema is used to define the input or output of an action as well as the value of a property or event.

An overview of the ontology is reported in Fig. 5.3 [175]. SWoT Ontology extends this work by defining the classes `ActionInstance` and `EventInstance`. Through these classes, a SPARQL Event Processing Architecture can be used not only to solve the discoverability problem, but also to provide a novel way to interact with things (e.g., to invoke actions through SPARQL updates). To achieve this scope, the SWoT ontology also provides the classes `InputData` and `OutputData` to map also the real input and output of an action or event.

It is worth mentioning that, for the purpose of the benchmark presented in this Thesis, one could also employ the WoT Ontology designed by Serena et al. [175]. In fact, as previously mentioned, the main difference between their work and the extension proposed together

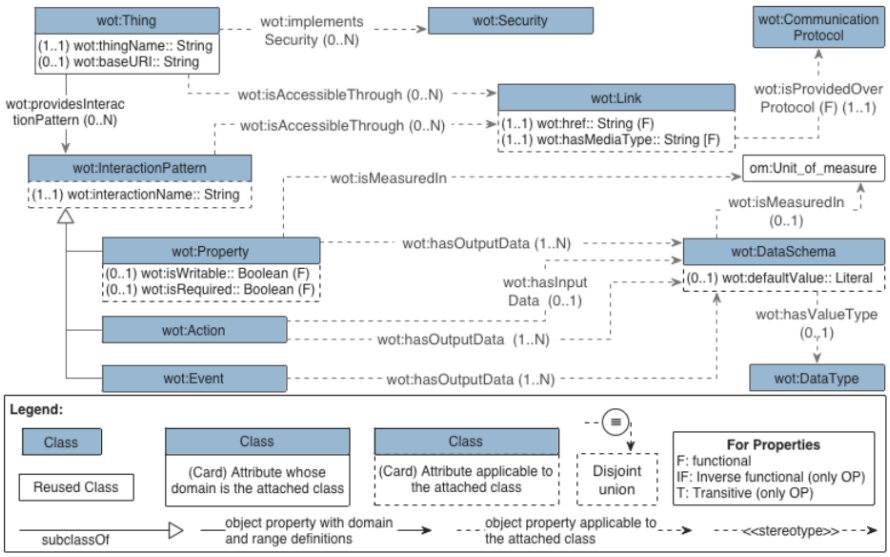


Figure 5.3: Overview of the WoT Ontology [175]

with my research group is mostly focused on the interaction with the Web Things (see Section 7.1.3), that is not part of the current benchmark yet. Nevertheless, a future extension of SWoT_Bench will be focused on controlling Web Things through a SEPA, and, this motivates the adoption of the extended ontology.

In the following examples, I will also refer to the Smart Appliances REFerence (SAREF) ontology [176] to describe the category of WT.

5.5.3 SPARQL updates and subscriptions

The SPARQL subscriptions adopted in this benchmark are aimed at discovering WTs. Four subscriptions are presented in the following lines, all characterized by an increasing level of selectivity (i.e., constraints imposed on the discovery).

SUB_0 Performed to get notifications about all the Web Things joining/leaving the ecosystem. This is the subscription with the minimum selectivity among the ones presented in this Section.

```
1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
3 SELECT ?thing ?thingName
4 WHERE {
5   ?thing rdf:type wot:Thing .
```

```
6   ?thing swot:hasName ?thingName
7 }
```

SUB_1 Adds three constraints to SUB_0 in order to get notifications about Web Things providing at least an action:

```
1  PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
3  SELECT ?thing ?thingName ?action ?actionName
4  WHERE {
5    ?thing rdf:type swot:Thing .
6    ?thing swot:hasName ?thingName .
7    ?thing swot:hasInteractionPattern ?action .
8    ?action rdf:type swot:Action .
9    ?action swot:hasName ?actionName .
10 }
```

SUB_2 An additional constraint is imposed on the type of the Web Thing (in this case the Web Thing must be an instance of the class `saref:Switch`).

```
1  PREFIX saref:<https://w3id.org/saref#>
2  PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
4  SELECT ?thing ?thingName ?action ?actionName
5  WHERE {
6    ?thing rdf:type swot:Thing .
7    ?thing rdf:type saref:Switch .
8    ?thing swot:hasName ?thingName .
9    ?thing swot:hasInteractionPattern ?action .
10   ?action rdf:type swot:Action .
11   ?action swot:hasName ?actionName .
12 }
```

SUB_3 lastly, SUB_3 is the most selective subscription. In fact, this is not intended to look for a set of devices, but for a specific Web Thing (i.e., `wot:Thing1_URI`).

```
PREFIX saref:<https://w3id.org/saref#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
SELECT ?thingName ?action ?actionName
WHERE {
  swot:Thing1 rdf:type swot:Thing .
  swot:Thing1 rdf:type saref:Switch .
  swot:Thing1 swot:hasName ?thingName
  swot:Thing1 swot:hasInteractionPattern ?action.
  ?action rdf:type swot:Action .
  ?action swot:hasName ?actionName .
}
```

Four SPARQL Updates triggering different subsets of the previous subscriptions compose the benchmark:

U_0 The first SPARQL Update proposed by SWoT_Bench should trigger all the subscriptions proposed in the previous lines. This update in fact, produces the registration of a new Web Thing `wot:Thing1` belonging to the class `saref:Switch` and providing an action.

```
PREFIX saref:<https://w3id.org/saref#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
SELECT ?thingName ?action ?actionName
WHERE {
  swot:Thing1 rdf:type swot:Thing .
  swot:Thing1 rdf:type saref:Switch .
  swot:Thing1 swot:hasName ?thingName
  swot:Thing1 swot:hasInteractionPattern ?action.
  ?action rdf:type swot:Action .
  ?action swot:hasName ?actionName .
}
```

U_1 Update U_1 differs from U_0 for the URI of the new Web Thing. This affects subscription SUB_3 that should not be triggered and, of course, should neither produce a notification.

```
PREFIX saref:<https://w3id.org/saref#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
INSERT DATA {
  swot:ThingN rdf:type swot:Thing .
  swot:ThingN rdf:type saref:Switch .
  swot:ThingN swot:hasName "ThingN" .
  swot:ThingN swot:hasInteractionPattern swot:ActionN .
  swot:ActionN rdf:type swot:Action .
  swot:ActionN swot:hasName "Action N" -
}
```

U_2 Produces the registration of a new Web Thing not belonging to the class `saref:Switch`. Then, the set of triggered subscription comprises only SUB_0 and SUB_1.

```
PREFIX saref:<https://w3id.org/saref#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
INSERT DATA {
  swot:ThingN_URI rdf:type swot:Thing .
  swot:ThingN_URI swot:hasName "ThingN" .
  swot:ThingN_URI swot:hasInteractionPattern swot:Action1_URI .
  swot:ActionN_URI rdf:type swot:Action .
  swot:ActionN_URI swot:hasName "Action N"
}
```

U_3 This one produces the creation of a new Web Thing where the interaction pattern is an event, rather than an action. Then, the only triggered subscriptions should be those of the category SUB_0.

```
PREFIX saref:<https://w3id.org/saref#>
```

```
2  PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
4  INSERT DATA {
5    swot:ThingN_URI rdf:type swot:Thing .
6    swot:ThingN_URI swot:hasName "ThingN" .
7    swot:ThingN_URI swot:hasInteractionPattern swot:Event1_URI .
8    swot:EventN_URI rdf:type swot:Event .
9    swot:EventN_URI swot:hasName "Event N"
10 }
```

U_var Finally, I propose a fourth SPARQL Update request characterized by a variable complexity. The update presented in the following listing, contains in fact a section named <INT_PATTERNS> used to specify a variable number of interaction patterns that determine the complexity of the Web Thing (thus affecting the number of triples to be inserted).

```
1  PREFIX saref:<https://w3id.org/saref#>
2  PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
4  INSERT DATA {
5    swot:ThingN_URI rdf:type swot:Thing .
6    swot:ThingN_URI swot:hasName "ThingN" .
7    <INT_PATTERNS>
8  }
```

where <INT_PATTERNS> are built according to the following schema (variables whose names are prefixed by a dollar represent the *forced bindings*):

```
1    swot:ThingN_URI swot:hasInteractionPattern $ip .
2    $ip rdf:type $ip_type .
3    $ip swot:hasName $ip_name .
```

5.5.4 Metrics

The following is a list of metrics that can be measured with the SWoT_bench to characterize the behaviour of the context broker in a Semantic Web of Things scenario:

Overhead It is a measure of the impact of a SPARQL Event Processing Architecture on a standard SPARQL endpoint. Given a SPARQL Update request u, the overhead is determined by:

$$Overhead(u) = \frac{t_{SEPA}(u) - t_{Endpoint}(u)}{t_{Endpoint}(u)}$$

where $t_{SEPA}(u)$ is the time spent by SEPA to perform the update u, while $t_{Endpoint}(u)$ is the time required by the underlying endpoint to satisfy the request. Intuitively, it depends on the complexity of the SPARQL Update request. This metric is derived from the lamp-posts benchmark presented in Section 5.3.

Global CTS Size If the broker adopts a subscription policy relying on context triple stores, it is interesting to evaluate its size (that depends on the sequence of updates triggering the subscription). Then, if $S = \{s_i\}$ is a set of subscriptions, the global CTS size is:

$$GCS = \sum_i size(CTS_i)$$

Global LUTT Size Given a set of subscriptions, this indicator determines the amount of memory occupied by all the Look-Up Triples Tables (if any).

$$GLS = \sum_i size(LUTT_i)$$

In presence of many equivalent or partly equivalent subscriptions, is the broker able to group them and minimize the required memory?

Awakening ratio Given a SPARQL Update u, a set of subscriptions S. If $S_{trig} \subseteq S$ is the subset of the triggered subscriptions and $S_{notif} \subseteq S$ is the subset of the subscriptions that must produce a notification, then the awakening ratio can be defined as:

$$AR = \frac{|S_{notif}|}{|S_{trig}|}$$

In the ideal case, all and only the subscriptions that must issue a notification are triggered (i.e., $AR = 1$). The closer this value is to 1, the higher is the efficiency of the LUTT. If $AR < 1$ (the most common case), the LUTT mechanism is loose. If $AR > 1$, this is a signal of a malfunctioning LUTT system that prevent some important subscriptions to be awakened. This indicator depends, of course, on the given S and u.

Awakening Time This is the time spent during a SPARQL Update to wake up the subscriptions involved by this modification of the graph. For algorithms based on a LUTT, this time may include the time to parse the available LUTTs and determine the subscriptions to be triggered.

Time to Notification Given a SPARQL Update u and a subscription producing a notification s ($S_{notif} \subseteq S$, $|S_{notif}| = 1$) the time to notification can be defined as:

$$TTN(u, s) = t(u) - t_{notif}(s)$$

Average Time to Notification Given a SPARQL Update u, a set of active subscriptions S, and a subset of subscriptions producing notifications $S_{notif} \subseteq S$, the average time to notification can be defined as:

$$ATTN = \overline{TTN(u, s_i)}, \quad s_i \in S_{notif}$$

Variance This is a measure of the variability of the time to notification and is indentified by the symbol $\sigma^2(TTN(u, s))$.

Notification completeness A measure of the number of results provided by a notification with respect to the total number of expected results. It is a ratio of the received bindings to the expected ones. If lower than 1, the notification is incomplete. If greater than 1, the notification includes unattended bindings. In both cases it highlights a malfunctioning in the event processing engine. This measure is borrowed from LUBM [152] and adapted to the publish-subscribe paradigm. Can be measured both at client and server side.

Notification soundness Can be considered as a complementary measure to the completeness. This indicator is not focused on the completeness of the results, but on its correctness. The total number of (added and removed) bindings included in the notification is compared with the number of correct bindings. A ratio lower than 1 highlights a wrong behaviour of the SPARQL Event Processor. Can be measured both at client and server side.

Notification success It is a combined metric that measures if an update is successfully notified to a client. If both $\sigma_s = 1$ and $\gamma_s = 1$ and if the notification time is below a certain threshold (that depends on the specific application), then the notification success can be

considered as *True*. Otherwise, and also if the time limit is hit, the notification success is *False*.

5.5.5 Tests

This Section proposes a list of tests that, through given combinations of the SPARQL Updates and Subscriptions described in Section 5.5.3 allows to determine the value of each of the metrics proposed in Section 5.5.4.

Test 1a – Overhead with no running subscriptions

To the best of my knowledge, none of the existing semantic publish/subscribe services are natively provided by SPARQL Endpoints. This means that SEPA architectures are currently developed as additional layers placed on top of SPARQL endpoints. An unavoidable overhead is then introduced and it is important to quantify it in all the possible situations. *Test 1a* aims to quantify the overhead when no subscription is running, then when a SPARQL Update request should be simply forwarded from the SEPA to the underlying endpoint. This test is then characterized by $S = \emptyset$. This test should be performed with SPARQL Updates of increasing complexity. U_var can be used to fulfill this task.

Example: an example configuration has been used to compare four different SEPA engines in terms of the overhead on update requests. The four instances were running with different subscription algorithms (i.e., Naive, LUTT, CLUTT and CHLUTT), and all of them were running on top of a persistent instance of Blazegraph. The SPARQL Update requests used to measure the overhead on the time was U_var with n interaction patterns, with n assuming all the values in $[0, 20]$. Results (available in Fig. 5.4) show that, when no subscription is running, the overhead is negligible for all the analyzed algorithms.

Test 1b – Overhead with no notifying subscriptions

This test is intended to assess the performance of different subscription algorithms when a number of equivalent subscriptions are running on the system and none of them should produce a notification after a SPARQL Update u (so $|S| > 0, S_{notif} = \emptyset$).

For this test, a user-defined number n of subscriptions belonging to the class SUB_3 is adopted and an update U_0 is performed. The initial knowledge base is represented by the n thing descriptions of the devices subscribing to the new Web Things (so the size of the initial graph is not null). A SPARQL update of class U_var with a variable (i.e., increasing) number of patterns i is performed.

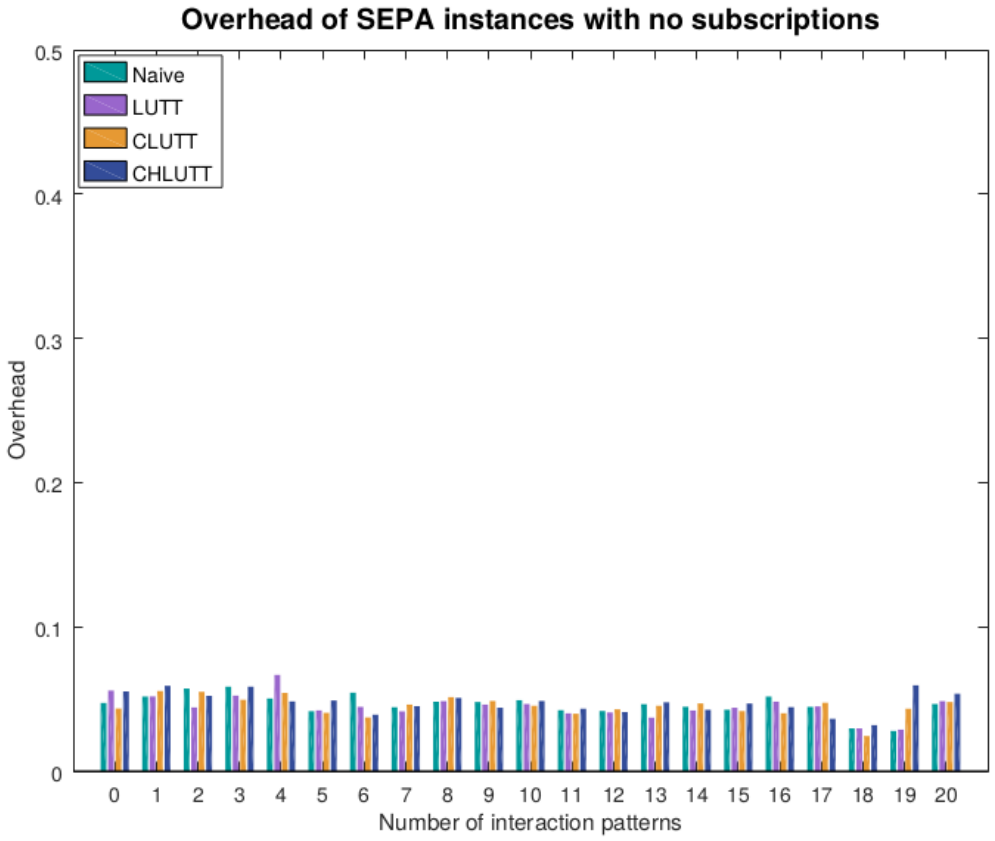


Figure 5.4: Overhead of the update requests with no subscriptions

To plot the results of the overhead, the x axis should host the number of interaction patterns i of the Thing Description to be inserted. The y axis should host the measured overhead. n and i should be sized based on the scenario of interest. An example configuration could be represented by $n = 20$ subscriptions of class SUB_3 and a set of updates with i interaction patterns (where i assumes all the values in $\{10 \cdot k : \forall k \in [1 \dots 10]\}$).

Example: Fig. 5.5 reports the results of the execution of Test 1b with four SEPA instances running on top of Blazegraph. The test has been executed with $n = 20$ and $i = \{10 \cdot k : \forall k \in [1 \dots 10]\}$. Results show that the Naive algorithm introduces a higher overhead if compared to the others. This happens because the Naive algorithms wakes up all the subscriptions and and all of them perform a query on the whole knowledge base to detect whether or not a notifications should be sent.

The example scenario is based on a Smart Space where a given number of devices is present and all of them are subscribed to a specific kind of device. Moreover, the Thing Description of

all these devices is put into SEPA. The registration of a new device, with variable complexity, does not produce (in this case) any notifications, due to the high level of selectivity of the subscriptions.

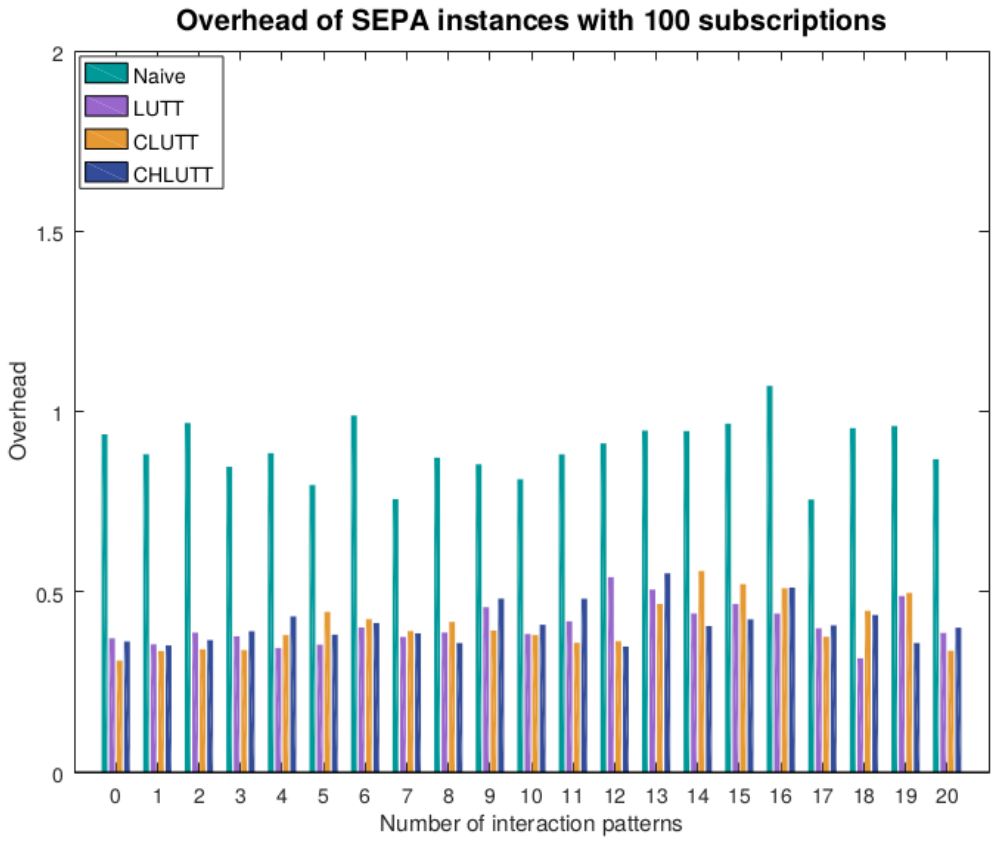


Figure 5.5: Overhead of the update requests with 20 non-notifying subscriptions

Test 1c – Overhead with notifying subscriptions

This test is intended to assess the performance of different subscription algorithms when a number of equivalent subscriptions are running on the system and all of them should produce a notification after a SPARQL Update u ($S_{notif} \equiv S$, $|S| > 0$). The test is then composed by a user-defined number n of subscriptions (e.g., with $n = 10 \cdot k, k = 0, \ldots, 100$) belonging to the same class SUB_i, $i \in [1, 4]$. The initial knowledge base is represented by the n thing descriptions of the devices subscribing to the new Web Things. In this test we consider that all the subscriptions are triggered, in order to simulate the worst case. To plot the results of the overhead, the x axis should host the number of running subscriptions n, while on the y axis the measured overhead. n should be sized based on the scenario of interest.

Test 2 – Variation of the GLS Look-up tables are used to speed up the detection of the subscriptions interested by a SPARQL update, achieving then a higher level of performance. The price is a greater occupation of memory. This test allows to verify which of the subscription algorithms behaves better in the common case of many (partly) equivalent subscriptions. In this test n subscriptions equally distributed among the classes SUB_1, SUB_2, SUB_3 and SUB_4 are running. What happens varying n?

Example: A first test is aimed at measuring the impact of the different algorithms on the memory, considering only the look-up tables. Fig. 5.6 reports the results of the execution of this test with n instances of subscription SUB_i ($\forall i \in [1,4]$), where $n = 10 \cdot k$ ($\forall k \in [0,10]$). It is immediately noticeable the difference in terms of memory occupation among the different subscription management algorithms. The naive algorithm is not shown, since it does not relies on look-up tables. When multiple equivalent subscriptions are running, the Centralized LUTT outperforms its predecessor (the LUTT algorithm [124]). A further improvement in this sense is granted by the Centralized Hierarchical LUTT, whose memory occupation is negligible if compared to the one of the LUTT.

5.6 Conclusion and future work

This Chapter presented three main contributions, one of which still in progress:

- The first contribution is related to a benchmark for the Smart-M3 platform that allowed measuring the impact of the LUTTs to timely process subscriptions. This benchmark highlighted the high efficiency of the LUTT: this structure permits an efficient analysis of the triples added or removed by a SPARQL Update, in order to avoid awaking subscriptions not involved by a modification of the graph.

- A second contribution is represented by the Performance Evaluation Suite designed to provide an effective way to describe and run performance tests on semantic publish/-subscribe middlewares. The platform has been grounded on Smart-M3, but is easily portable to novel architecture (e.g., the SPARQL Event Processing Architecture).

- The last contribution refers to an ongoing activity aimed at designing the first benchmark for the Semantic Web of Things. This work is motivated by the absence of benchmarks for SWoT applications where devices are described by a Thing Description formalized according to a given ontology. This benchmark has been carefully described in [124].

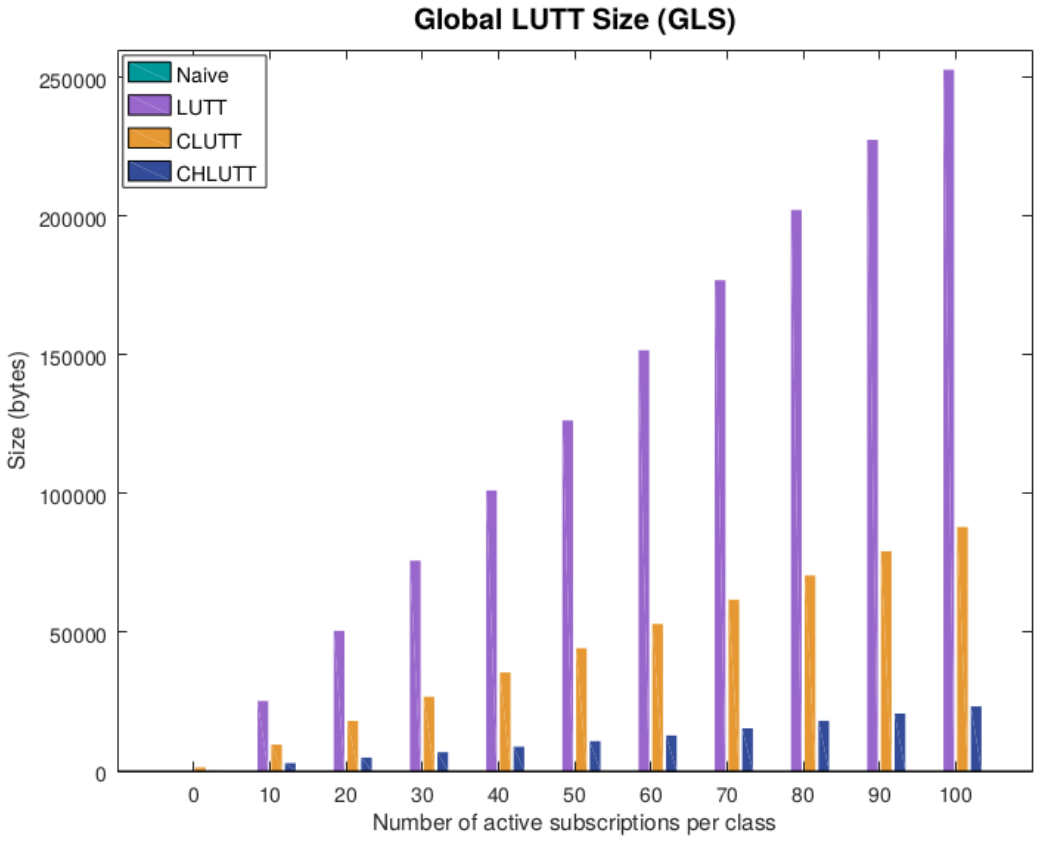


Figure 5.6: Global Lutt Size varying n (Test 3)

This benchmark (named SWoT_Bench), pivoting the discoverability problem, is based on the SWoT ontology developed in my department to permit the semantic representation of the Thing Description of a device, as well as the interaction with the device through a set of SPARQL requests. SWoT_Bench provides an extensive set of metrics and a set of SPARQL subscriptions and updates to evaluate them. First preliminary results highlighted the ability of the test in quantifying the differences of the performances of existing algorithms for SEPA archictecures. This activity brought to the publication of a conference paper and a subsequent oral presentation at the UBICOMM conference [177].

Future works on this topic will be aimed at the completion of the benchmark. Furthermore, the research activity will go on towards the extension of the benchmark outside the boundaries of the discoverability problem. In fact, as previously mentioned, the SWoT Ontology allows controlling and orchestating WTs through SEPA. This new integration pattern should be carefully evaluated and compared to the existing ones proposed by the W3C (i.e., direct integration pattern, gateway and cloud integration patterns).

Part III

Visualization of semantic knowledge bases

Chapter 6

Visualization of RDF graphs

Contents

Dealing with applications relying on Semantic Web technologies requires effective tools for inspection and debugging of data. Furthermore, it is also important for developers to be quickly able to understand the nature of data and the overall structure (this is known in literature as "sensemaking" [178]). Nowadays, this is particularly true since SW technologies are gaining momentum. This is testified, for example, by the impressive growth of Linked Open Vocabularies (LOV)[1], a catalog of ontologies and vocabularies. In [179], authors propose an interesting overview of the impressive growth of the repository: in approximately four years (March 2011 - June 2015) the number of vocabularies hosted by LOV grew from less than 100 to 511 (66.14% of which developed in English). As of August 2018, the number is 650.

The state of the art of visualization tools for RDF knowledge bases proposes software that makes difficult for the user to isolate a subgraph and verify the way it is connected with the rest of the knowledge base. Analyzing a knowledge base and understanding its structure as well as discovering errors results challenging. Due to this gap, my research activity has been extended towards the study of new visualization methods for RDF datasets and resulted in the introduction of semantic planes aimed at effective visualization of small/medium-sized KBs for both novice users and experts[2].

In the rest of this Chapter, this research activity is presented: first, motivations are introduced in Section 6.1; An overview of the state of the art is provided by Section 6.2 [3] . Then, a tool for the exploration of RDF KBs (and based on the concept of semantic planes) is presented in Section 6.3. The tool is demonstrated against four use cases in Section 6.4 while a preliminary analysis of the user experience and the performance is presented in Section 6.5. Finally, Section 6.6, concludes the Chapter with final remarks.

6.1 Background and motivation

Visualization of RDF datasets is a challenging task. The most intuitive way to achieve the scope is a single table with three columns (i.e., subject, predicate, object), but this solution is not scalable. Also a set of tables, one for each property, is a possible approach. But, even though relational tables have been widely used to optimize data storage and retrieval [180],

[1] https://lov.linkeddata.es/dataset/lov

[2] This contribution was published by Fabio Viola, Luca Roffia, Francesco Antoniazzi, Alfredo D'Elia, Cristiano Aguzzi, Tullio Salmon Cinotti, in Interactive 3D Exploration of RDF Graphs through Semantic Planes, Future Internet, MDPI, Aug. 2018

[3] © IEEE, Reprinted with permission, from Francesco Antoniazzi and Fabio Viola. RDF Graph Visualization Tools: a Survey. 2018 Proceedings of the 23rd Conference of FRUCT Association. Nov. 2018.

this is not as effective to visualize RDF data.

The most diffuse way to graphically represent an RDF KB is a graph: RDF data can be represented as a directed and labeled graph where subjects and objects of each statement are nodes linked by an edge labeled with the predicate. Depending on the number of triples in the KB, the graph can be very complex. Then, a tool for its visualization should address a set of issues and requirements that can be summarized as:

p0 Pre-Filtering – A graphical representation of a large number of triples is usually both ineffective (hard for the user to retrieve the desired information) and inefficient (computationally heavy). Then, a pre-filtering mechanism allows extracting the subgraph that is really relevant for the user from the full knowledge base.

p1 Node placement – Node positioning should be smart enough to avoid overlapping with other graphical elements. The complexity of this task is directly proportional to the size of the knowledge base. If possible, the principle of proximity should be respected (i.e., linked resources should be placed close to each other to easily gather as much information information as possible in a glimpse).

p2 Incremental approach – The portion of the KB that needs to be inspected is usually limited and the desired visualization may require a series of steps to be achieved. Then, an effective tool should support the incremental building of the view.

p3 Filtering – Filtering must be as flexible as possible in order to hide/show/highlight information. Providing powerful filtering features in a user-friendly way is often a difficult task.

p4 Support for RDFS and OWL – Both RDFS and OWL should be supported. This allows selecting and filtering the graph content by means of concepts like class, domain and range of properties, datatype and object properties.

p5 Domain-agnostic – Semantic Web and Linked Data technologies may be applied to very different and heterogeneous domains even within the same application. The datasets in the Linked Open Data cloud mainly belong to seven domains (cross-domain, geographic, media, life sciences, government, user-generated content, and publications) [181], while in the IoT, where SW technologies are often applied, the application domains, as mentioned in the Introduction, are more than fifty [4].

As will be detailed in Section 6.2, tools for visualization of semantic knowledge bases do not address all the requirements and lack proper strategies to effectively analyze data. This

motivates my research activity, that brought to the design of a new approach to the visualization of data: Semantic Planes. A semantic plane can be defined as a set of RDF terms sharing a common meaning and can be created directly or indirectly through standard SPARQL 1.1 queries. This metaphor has been implemented in Tarsier, a visualization tool developed during my PhD that fulfills all the above mentioned requirements overcoming the limitations of existing software. This approach helps to understand and debug data by following a common mental approach: splitting the KB among planes, each of them related to a specific concept. The user can incrementally build a view by adding and/or removing information according to his/her actual needs and splitting the information among planes. Relationships among resources in different planes are in such a way emphasized. Nevertheless, the user still maintains a view on the rest of the knowledge base, when needed.

6.2 Related work

Semantic knowledge bases take the form of directed labelled graphs, so the tools for the interactive visualization of these datasets could be classified as:

- tools for the visualization and exploration of all kinds of graphs with the support of plugins/extensions to import semantic data;
- tools specifically designed to visualize Semantic Web knowledge bases and ontologies. It is worth mentioning that the graph is not the only approach to the graphical visualization of RDF data. This is demonstrated, for example, by Gallego et al. [182] that propose a method to visualize RDF data based on a 3D adjacency matrix. Then, in this category it is possible to find solutions based on a graph visualization and approaches based on other different ideas.

Generally speaking, tools belonging to the first category usually include sophisticated functions for the analysis of graphs and are aimed at expert users, while the second category includes tools with a reduced set of functions, suitable for Semantic Web users. The formers are presented in Section 6.2.2, while tools specific for the Semantic Web are discussed in Section 6.2.2 with a detailed analysis of software based on the graph representation.

6.2.1 Graph drawing algorithms and tools

Many algorithms to layout graphs and interact with them exist in literature. Among all, I provide here an overview of those most related to the scenario presented in this Chapter.

Gansner et al. [183] proposed a method for drawing directed graphs based on four steps: 1) ranking; 2) ordering; 3) positioning; 4) making splines. Later on, Gansner and North [184] proposed a graph visualization software along its application in several fields. In [185], algorithms for positioning nodes and routing edges in order to maximize the readability of circular layouts are presented. An algorithm for drawing labelled nodes removing overlapping and minimizing, at the same time, the drawing area is instead the focus of the paper by Gansner and Hu [186]. Binucci et al. [187] focused on the problem of drawing arrows in directed graphs, while an algorithm and the related tool for grouping nodes in non-overlapping regions based on node attributes and allow user to interactively filter the results are presented by Shneiderman and Aris [188]. The main algorithms presented in [189] are implemented by the GraphViz, a famous open source graph visualization software [190]. As will be shown in the next Section, GraphViz is one of the most used libraries for graph drawing.

In [191] a new idea and a model specifying graph visualization techniques is presented. The aim was to provide a new model for graph exploration along with the potential of discovering new network visualization techniques. In [192] a list of considerations are made about the methods commonly used to build visualization systems. For instance, the limited flexibility of some tools is highlighted, regarding their specific context of use, or their lack of extensibility towards interactive use. Moreover, authors introduced a novel language, called DeVIL, that is able to correlate user interactions with the database views in a variety of ways. The DeVIL program is translated into a workflow that creates the interface and listens to user's direct and indirect requests. While DeVIL requires that the users already knows the structure of the database, Tarsier allows the free exploration of data and tries to extract the structure of the knowledge base to guide the user in the Sensemaking [178] process.

As regards 3D visualization instead, some problems related to the representation are discussed by Brandenburg et al. in [193]. A totally different approach was proposed by Gansner et al. [194]: by focusing on object properties, the authors presented a tool for visualizing relational data with geographic-like maps.

Gephi [195] is one of the most recent and powerful tools. It is designed to represent not only semantic graphs, but every kind of graph or network. Two external plugins, VirtuosoImporter[4] and SemanticWebImport[5] (this one developed by INRIA) provide support for Semantic Web ontologies and knowledge bases. Thanks to these plugins, Gephi is able to retrieve data from SPARQL endpoints or RDF files and allows one to apply filters through SPARQL queries. The look of the graph visualized by Gephi is fully customizable, in terms of colors and layouts.

[4] `https://github.com/avens19/virtuosoimporter`
[5] `https://github.com/gephi/gephi/wiki/SemanticWebImport`

Furthermore, Gephi supports grouping similar nodes and this helps achieving better results when dealing with very complex graphs. Fig. 6.1 reports an example of graph retrieved from DBpedia by using a SPARQL `CONSTRUCT` query. Unfortunately, as shown by Fig. 6.1, it is quite difficult to get the overall idea of the composition. Although there is the possibility to add the labels of nodes and edges, the output is not reader-friendly, and the research in it is a rather impossible task. Eventually, a number of statistical functions can be applied to the network, like the *Network Diameter*, the *Density* and the *Average Path Length*.

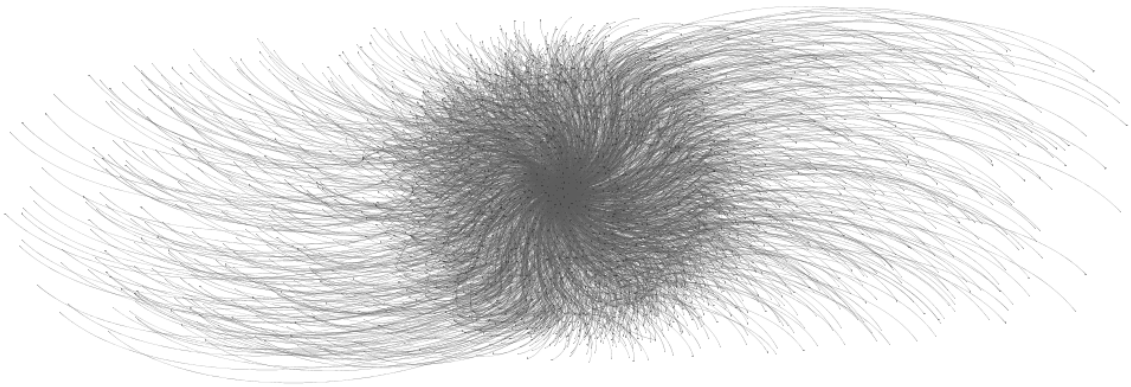


Figure 6.1: Gephi [195] is capable to query DBpedia and show the resulting graph, in this case made by 6529 triples. Source: [196].

6.2.2 Visualization tools for semantic web knowledge bases

This Section proposes an analysis of the tools specifically designed to visualize data of the Semantic Web domain. In particular, Section 6.2.2 focuses on the graph-based tools, while other approaches are presented in Section 6.2.2.

Graph-based visualization

Many are the research works available in literature that focus on the visualization of semantic knowledge bases through graphs. This Section proposes a detailed analysis of the main tools belonging to this category. The tools presented in this Section are reported in alphabetical order. Among the tools presented, several are available as plugins for the popular ontology editor Protégé.

CytoScape – Cytoscape [197] is a tool for network data integration, analysis and visualization. A set of extensions hosted on CytoScape's App Store provide support for seman-

Figure 6.2: With Gephi [195] 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". Source: [196].

tic web datasets: General SPARQL[6], SemScape[7] and Vital AI Graph Visualization[8]. General SPARQL allows navigating Semantic Web KBs through an extensible set of pre-defined queries. The plugin is pre-configured to retrieve and visualize data from public endpoints (e.g., Reactome, Uniprot, HGNC, NCBI Taxonomy, Chembl). SemScape permits the interaction with remote SPARQL endpoints and allows one to visualize the results of a SPARQL query. Vital AI Graph Visualization instead, is not limited to semantic databases, but provides access also to SQL and NoSQL databases as well as Apache Hadoop instances. The limit of Cytoscape is represented by the possibility to visualize only data compatible with the BioPAX format.

Fenfire – Fenfire [198] was a tool for the visualization and editing of RDF graphs (development stopped in 2008). The aim was the interactive exploration of semantic graphs. Authors faced the problem of scalability by limiting the exploration of the graph to one thing at a time. The visualization in facts, displays only one central node (based on `foaf:primaryTopic`

[6] `http://apps.cytoscape.org/apps/generalsparql`
[7] `https://apps.cytoscape.org/apps/semscape`
[8] `https://apps.cytoscape.org/apps/vitalaigraphvisualization`

if present, otherwise selected by the user) and its surroundings. 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.

GLOW – Glow [199] was a visualization plugin for the ontology editor Protégé that provided force-directed, node-link tree and inverted radial tree as layout algorithms. The items are arranged automatically with every layout, and cannot be moved. Being, a plugin for Protégé, the tool was aimed at representing a set of ontologies, with optional visualization of their individuals.

IsaViz – IsaViz [200] was a 2.5D tool based on GraphViz [201] for the visualization of RDF graphs. It was originally developed by E. Pietriga (INRIA) in collaboration with Xerox Research Centre Europe. IsaViz is able to load data from RDF/XML, Notation 3 and N-Triple files (and these file formats are also employed by the export function, along with the `png` and `jpg` formats). The UI is based on three views: *Graph view* (showing the current portion of the graph), *Radar* (presents an overview of the graph, since the graph view may contain only a portion of it) and *Property Browser* (to select resources and access a textual list of properties).

Jambalaya – Jambalaya [202] was another Protégé plugin developed with support from the National Center for Biomedical Ontology (NCBO) [9]. The main characteristic of Jambalaya is the integration of the Simple Hierarchical Multi-Perspective (SHriMP) [203] visualization technique designed to improve the user experience while browsing, exploring, modelling and interacting with complex information spaces (technique originally born to help programmers understanding software). The tool proposes a nested graph view and the nested interchangeable views. Nesting is used to represent the sub-class 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 current ontology.

LOD Live – LOD Live [204] is an active project aimed at providing a web-based tool for the incremental navigation of Linked Data available on a selected SPARQL Endpoint (e.g., DBpedia). Endpoints can be configured through a JSON map containing all of their parameters. Differently from the other tools described in this Section, the purpose of LOD

[9] `https://www.bioontology.org/`

Live is to demonstrate that the powerful SW standards are also easy to understand, and then fostering 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. Fig. 6.3 shows how LOD Live performs a task to the one in Fig. 6.2: exploring data is easier, but there is no way to perform requests based on a user-provided SPARQL query.

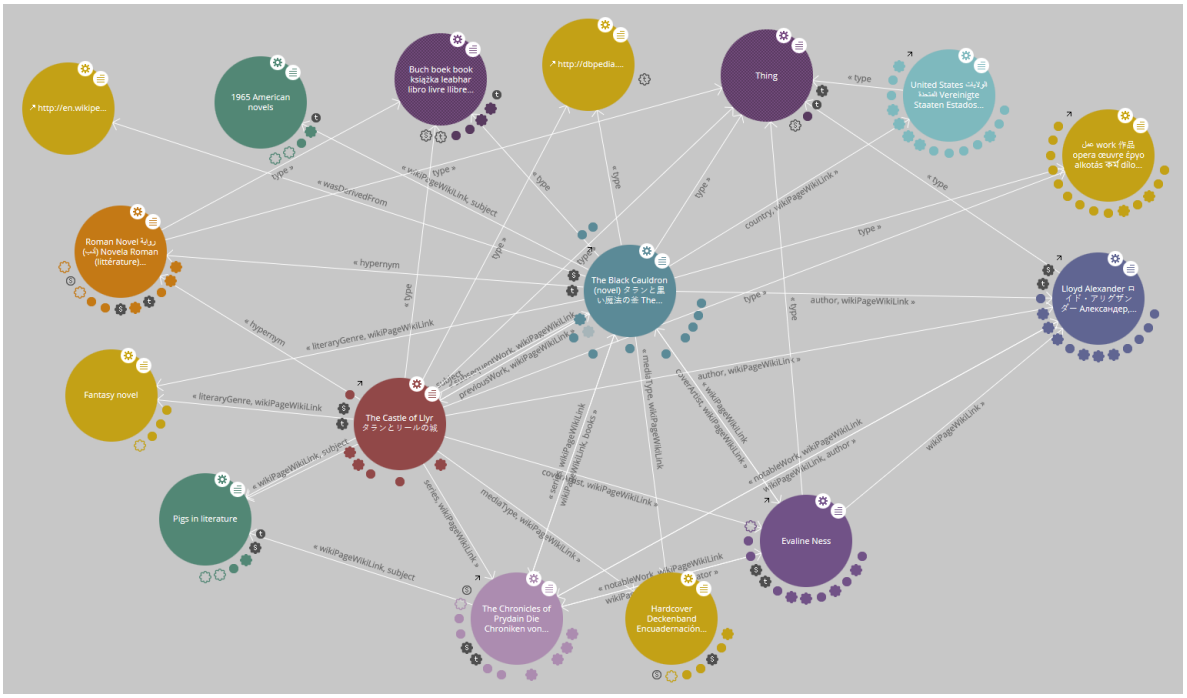


Figure 6.3: To use LOD Live [204] a resource must be fixed. Then, the knowledge related to the resource can be expanded as shown. Like in Figure 6.2, the example here is based also on L. Alexander's novel "The Black Cauldron". Source: [196].

Ontograf – Ontograf [205] is yet anothervisualization tool provided by Protégé. It allows building a custom visualization of the ontologies loaded in 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. The only way to visualize individuals of a class is through 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 is based on the layout library provided by Jambalaya.

An example of Ontograf is shown in Fig. 6.4 that depicts a graph created using the DBpedia ontology. Classes `work` and `written work` were initially selected. Then, a double click on the latter allowed expanding it and visualizing all the subclasses (solid blue line), and all the classes linked to it by means of an object property (dashed lines).

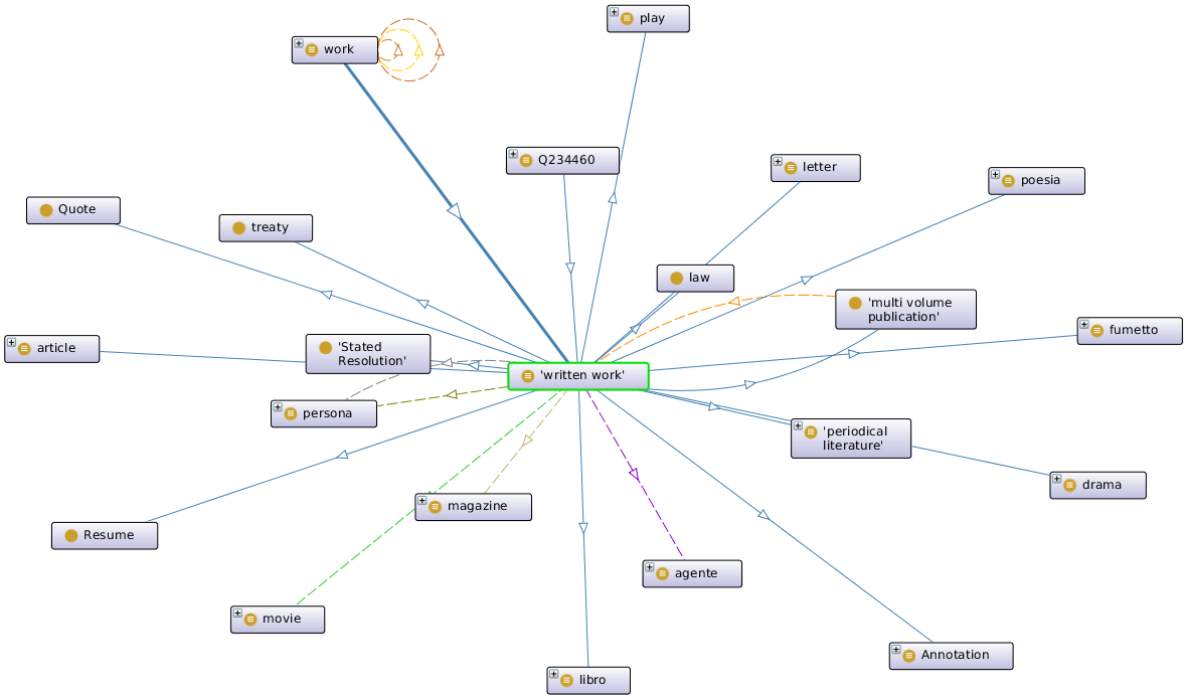


Figure 6.4: A portion of the DBpedia ontology visualized in Ontograf [205]. Source: [196].

Even though the last version dates back to April 2010, Ontograf is still included in the last stable version of Protégé (the 5.2.0, as of October 2018). To summarize, the tool results useful to select and visualize (a small number of) classes from the ontologies loaded in Protégé and the existing relationships, but uncomfortable when dealing with vaste ontologies.

OntoSphere – OntoSphere [206] is the only tool proposing a three-dimensional visualization of the graph. The rational 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 permit to easily convey information about the different nature of represented items. All of the previous considerations play a fundamental role in the design of Tarsier that will be detailed later on. Furthermore, OntoSphere (like Tariser) is aimed at representing both terminological and assertional statements.

OntoSphere proposes three scene types to fulfill different requirements: 1) the `RootFocus` scene (shows all the concepts and their relationships on a sphere); 2) the `TreeFocus` scene

(draws the tree originating from a concept); 3) 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 development of OntoSphere stopped in 2008.

OWLViz – OWLViz [207] is a plugin for Protégé enabling the incremental visualization of the classes in the class hierarchy. Also this tool, like IsaViz, is based on the famous AT&T library GraphViz and allows exporting the visualized graph as `png`, `jpeg` and `svg`. Through OWLViz is easy to visualize classes and `is-a` relationships. Like Ontograf, OWLViz is not developed anymore, but is still included in the last version of Protégé (as of October 2018).

Paged Graph Visualization – Paged Graph Visualization (PGV) [208] was a Java software aimed at the visualization of RDF graphs, based on Brahms [209], a high performance store. With PGV, the exploration starts from a point of interest and then incrementally includes more data. The point of interest can be selected from a list or through a SPARQL query. The user is able to explore nodes by double-clicking on them.

Deligiannidis et al. [208] 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.

RelFinder – RelFinder [210] is a web tool developed using Adobe Flex. An online instance configured to access DBpedia is available for tests on the homepage of the project[10].

RelFinder differs from the other tools analyzed in this Section, since it is aimed at visualizing all the paths connecting two resources, being then a special purpose tool. 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.

Fig. 6.5 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. Fig. 6.6 shows the filtering panel proposed by RelFinder to show/hide elements in the visualization. Paths can be filtered by length, class of the RDF terms, property and connectivity level.

[10] `http://www.visualdataweb.org/relfinder.php`

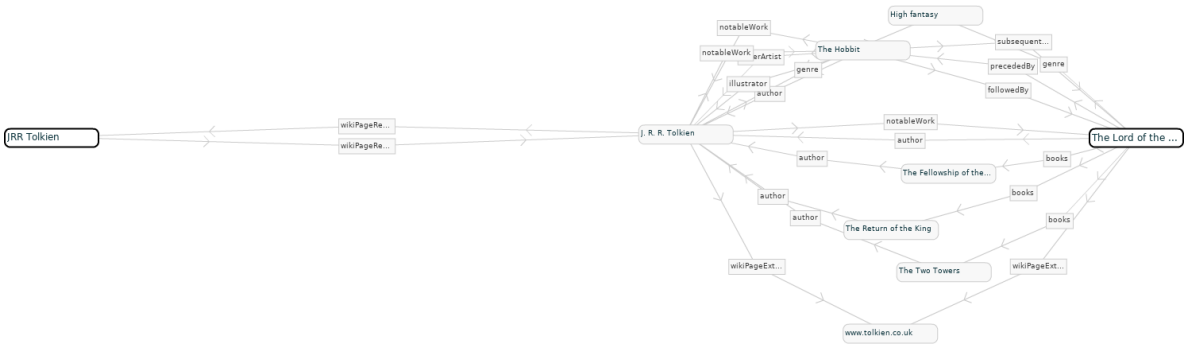


Figure 6.5: RelFinder [210] showing all the paths from "JRR Tolkien" to "The Lord of the Rings". Source: [196].

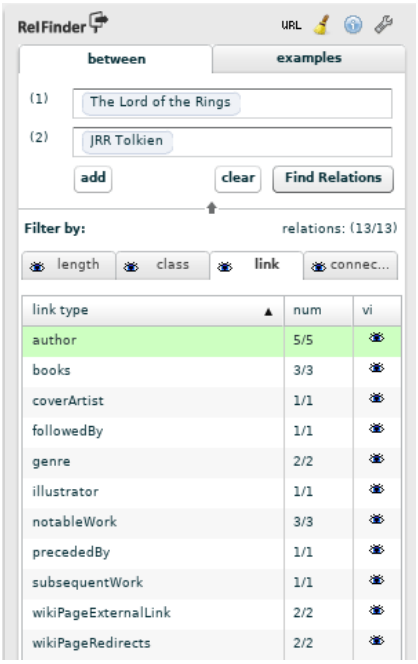


Figure 6.6: RelFinder [210] filtering panel. Source: [196].

TGVizTab – Also TGVizTab [211] is a visualization plugin for the ontology editor Protégé designed to be lightweight and support both T-Boxes and A-Boxes visualization. TGVizTab exploits TouchGraph[11], an open source Java environment aimed at creating and navigating network graphs in an interactive way. The tool adopts a spring layout to draw the graph: similar nodes are drawn close to each other. Like other tools (e.g., Fenfire), TGVizTab 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,

[11] `http://www.touchgraph.com`

view). Then TGVizTab allows building the desired visualization incrementally.

VOWL – VOWL (Visual OWL) [212] is a set of visualization tools providing: 1) a web-based tool (WebVOWL [213]); 2) a plugin for Protégé (ProtégéVOWL [214]); 3) a tool to directly interact with Linked Data endpoints (LD-VOWL [215]); 4) a visual query language tool, QueryVOWL [216]. As the name suggests, all of these tools are designed to graphically represent ontologies. These tool propose a force-directed graph layout. The basic representation rules adpoted by VOWL consists in:

- Classes are represented by circles; 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 depicted with black solid lines with, respectively, light blue and green labels, while RDFS properties have purple labels.
- Relationships `subClassOf` are identified by dashed lines.

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. The latter also report the number of individuals of the selected class. Unfortunately this is the only information about individuals. VOWL provides a basic support to filters to show/hide object/datatype properties, solitary classes, class disjointness and set operators. VOWL is actively developed and an online instance is available[12]. An example based on the DBpedia ontology is proposed in Fig. 6.7.

Other approaches to the visualization of RDF data

In [217], Lomov and Shishaev propose cognitive frames as a novel approach to the visualization of ontologies. Cognitive frames convey to the user the knowledge of a target concept related to the visualized fragment of ontology. The approach proposed by Lomov and Shishaev is more focused on terminological data, rather than assertional data. Heim and Steffen in [218] combine scatter plots (i.e., to support the visual identification of linear correlations, clusters, patterns, and extreme values) with the interaction metaphor of magic lenses [219]. Among the most recent approaches to the visualization of semantic KBs, it is worth mentioning [220] and [221]. The first investigates on different approaches for exploratory discovery and analysis of Linked Data provided by a set of tools. the latter instead focused on presenting a tool based

[12] http://www.visualdataweb.de/webvowl

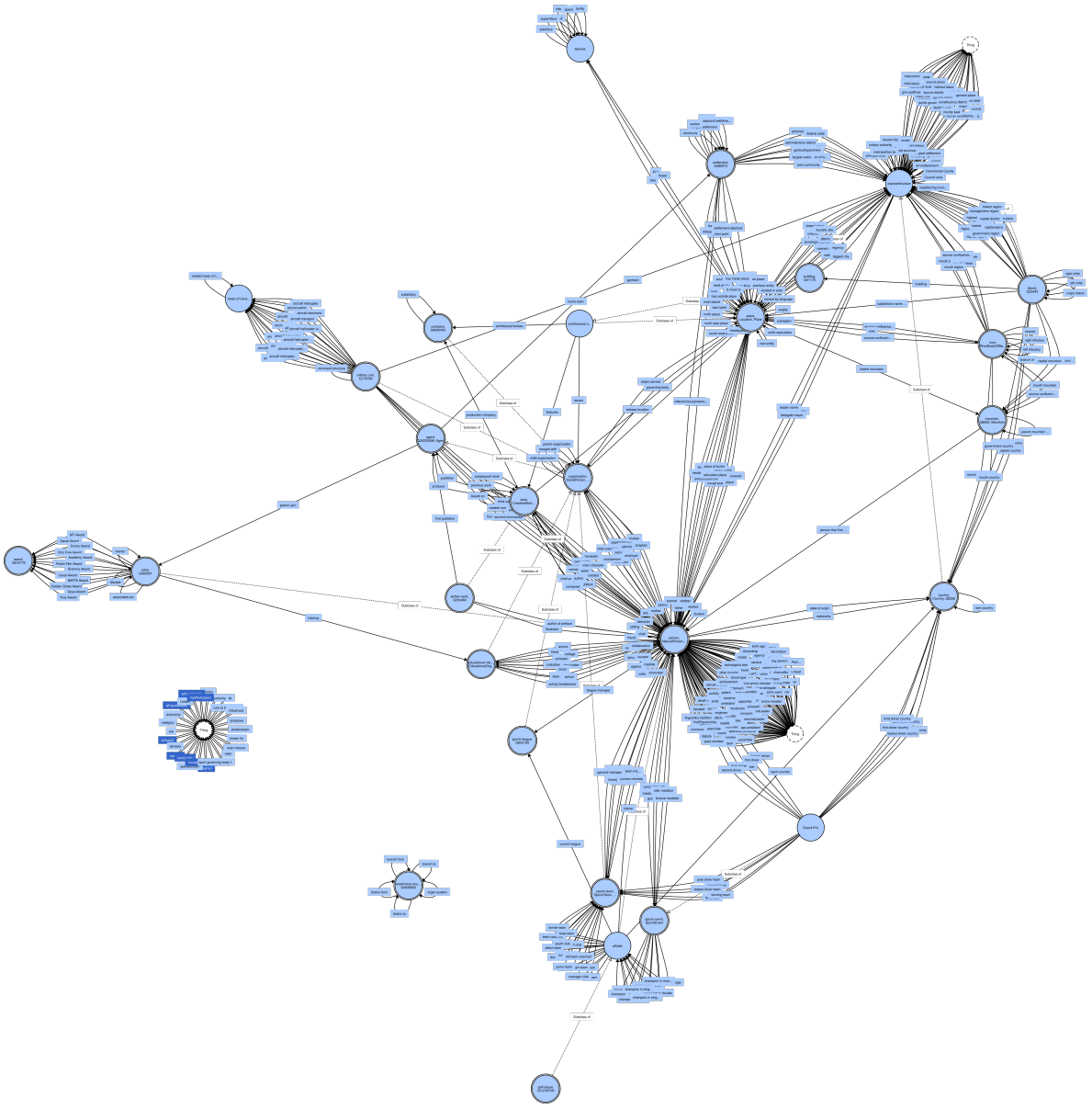


Figure 6.7: Overview of the DBpedia ontology in WebVOWL2 [213]. Source: [196].

on a novel approach to Linked Data exploration named Encyclopedic Knowledge Pattern (EKP).

6.3 Tarsier: 3D exploration of RDF knowledge bases

This Section presents Tarsier and the approach based on the concept of Semantic Plane. The approach is described in Subsection 6.3.1. The architecture of the software is presented in Subsection 6.3.2, while implementation details are in Subsection 6.3.3. The full list of features

is detailed in Subsection 6.3.4. The mechanism exploited by Tarsier to identify the elements in a graph is shown in Subsection 6.3.5, while Subsection 6.3.6 describes the User Interface of the tool.

6.3.1 Semantic planes

Semantic planes represent an innovative approach to the visualization of RDF graphs. Semantic planes group together RDF terms sharing a common user-defined common set of semantic features. The meaning conveyed by a semantic plane may be very simple (e.g., all the resources belonging to the class `foaf:Person`), or the result of a more complex filtering (e.g. the set of resources belonging to the class `foaf:Person` that work on the same project but do not know each other). This layered visualization allows:

- focusing on the information that is relevant for a given task, while still preserving a non intrusive view on the rest of the knowledge base;
- visualizing incoming and outgoing edges of a subgraph (i.e. semantic connection between planes).

Semantic planes are the results of filtering operations. This feature is accessible for both newcomers and advanced users. In fact, semantic planes may be created by selecting items in the lists of properties (i.e. datatype or object), classes, instances and blank nodes or through SPARQL queries. Filtering operations can be iterated multiple times and combined to refine the content of semantic planes.

6.3.2 Software architecture

Tarsier has been designed as a client-server architecture (see Fig. 6.8) due to the need to pre-process a potentially very large set of data (i.e., to subdivide RDF Terms among classes, instances, datatype and object properties) while still having light clients. Server-side, the main components are:

- A config manager (to configure the server).
- A client for SPARQL endpoints, to retrieve data from the desired endpoints.
- A Cache Manager. Since Tarsier is intended to be used also with dynamic systems where the KB evolves quickly (e.g., IoT applications), the application creates a snapshot of the knowledge base. This avoid disruptions to the user process of analysis due to changes in

the KB. Moreover, in this way Tarsier builds a local cache that speeds up every query to data. The user is able to update the local storage producing a new snapshot.

- A data extractor that identify the role of every RDF term; The resulting information is then organized in a data structure allowing clients to easily retrieve all the elements needed to draw and apply the filters selected by the user. The data extractor perform its tasks through a set of SPARQL queries detailed in Section 6.3.5.
- The HTTP interface through which client and server communicate.

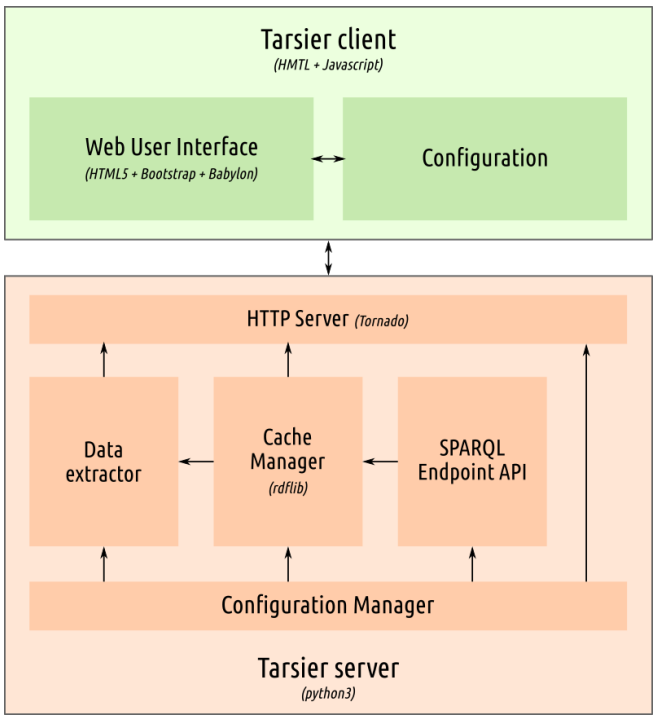


Figure 6.8: Software architecture of Tarsier. Implementation details are reported with the italic font. [222]

6.3.3 Implementation

Tarsier server is a Python 3 application exploiting the framework Tornado[13]. It provides an HTTP interface to receive requests from the clients. All the features of the server can be configured through a proper YAML file containing the port of the server and all the SPARQL queries needed by the Data Extractor. An important component of the Tarsier server is the already mentioned cache: it is implemented with the Python implementation of `rdflib`

[13] `http://www.tornadoweb.org/en/stable/`

On client-side, Tarsier is an HTML5 + Javascript application. Tarsier pivots on the canvas element to build a 3D representation of the knowledge base. The whole UI is based on the Bootstrap[14] framework, while Babylon JS[15] is responsible for drawing the 3D graph. Babylon JS was selected because of its support to hardware acceleration. While the client is started with a default configuration, all the parameters can be overridden by loading a YAML file hosting the parameters for a set of SPARQL endpoints and all the settings to customize the drawing and modified at run-time through the UI. The configuration file can also be used to store the SPARQL queries most frequently used.

6.3.4 Features

The main features of Tarsier are summarized in the following list:

Initial Knowledge Base – RDF graphs may be very large, hosting a high number of triples too difficult to represent in an effective way. For this reason, Tarsier provides a prefiltering mechanism to define an initial KB by means of SPARQL Construct queries (addressing in this way the requirement *p0*). Queries may also be loaded from a proper YAML configuration file. This allows the user to dominate the complexity of the underlying knowledge base focusing only on the information considered relevant for the current task. An example is shown in Fig. 6.9.

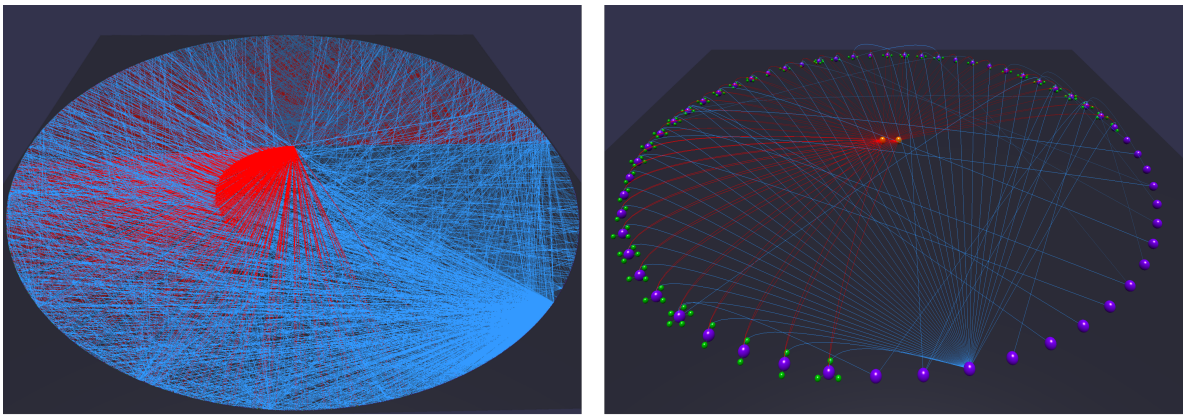


Figure 6.9: RDF graphs can host a number of triples too high to be effectively and efficiently visualized (subfigure *a*), but a prefiltering stage can help to visualize only a subgraph of interest (*b*) [222].

Support for RDFS and OWL – Despite being ontology agnostic (to adapt to different use

[14] https://getbootstrap.com/
[15] https://www.babylonjs.com/

cases, as suggested by point *p5*), Tarsier, through its data extractor (see Section 6.3.5) is able to identify classes, datatype and object properties by means of RDFS and OWL constructs (requirement *p4*). Comments and labels are also retrieved with the proper predicates.

Visualization techniques – Being able to quickly distinguish classes from other resources, datatype from object properties and `rdf:type` relationships above all may significantly speed up the analysis process, as demonstrated by Fig. 6.10, subfigure *a*. Therefore, Tarsier adopts a classification algorithm based on a set of SPARQL queries (detailed in Section 6.3.5) to identify classes, instances of classes, blank nodes, object and datatype properties and `rdf:type` relationships and paint each of them with a different color. Moreover a smart placement algorithm is employed to face requirement *p1*. This algorithm places items through the following scheme:

- Classes represented as equidistant spherical meshes on a circumference of radius r_{class};
- Individuals are represented as equidistant spherical meshes on a circumference of radius $r_{ind} > r_{class}$. The circle dedicated to classes is the innermost, since usually the number of concepts is less than the number of instances;
- Blank nodes lay on a third external circumference of radius $r_{blank} > r_{ind}$;
- Datatype properties of an instance are equidistant spheres placed on a circumference centered on the related instance.

This layout scheme is represented in Fig. 6.10, subfigure *c*). Different arrangement methods will be studied as future enhancements for this tool.

Filtering – The filtering mechanism (Fig. 6.11) implemented by Tarsier allows users to select items through UI or SPARQL queries and decide what action to perform. The selection can be related to any kind of RDF term: classes, instances, datatype or object properties as well as literals, RDFs or blank nodes. The action consists in showing/hiding selected meshes, or moving them across layers. Every filter applies to the current visualization, allowing then, the incremental filtering. A visualization that fits the user needs can then be achieved even by novice users. The filtering mechanism here described, faces the requirements identified in points *p2* and *p3*.

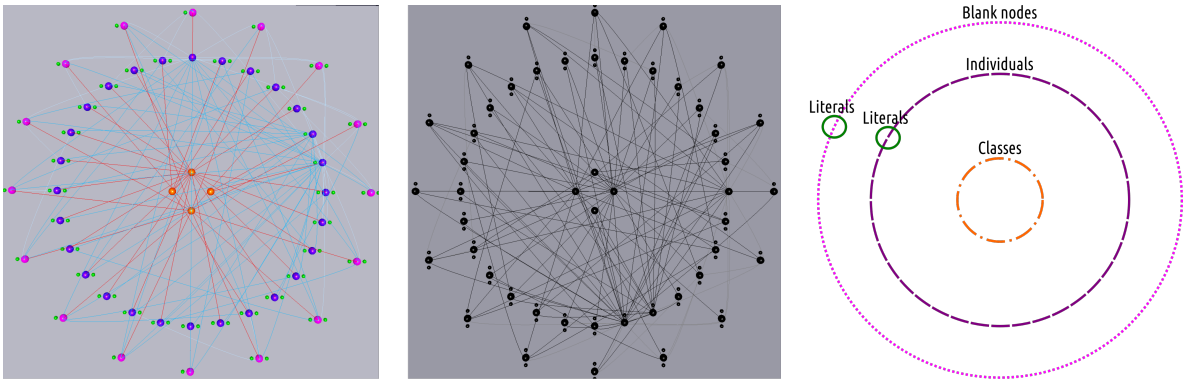


Figure 6.10: The classification of RDF terms among blank nodes, individuals, classes or literals as well as data and object properties, bound to using colours provide a more intuitive visualization (*a*), if compared to a monochrome one (*b*). In subfigure *c*, the drawing strategy adopted by Tarsier [222].

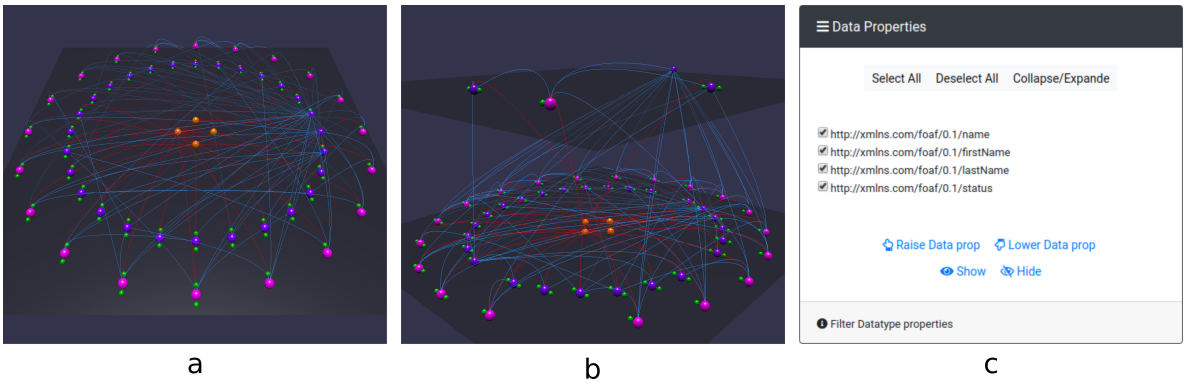


Figure 6.11: Filtering helps to gradually build the desired visualization of data. An example knowledge base is shown in subfigure *a*, while the result of filtering in subfigure *b*. Subfigure *c* shows one of the UI boxes through which filtering can be applied [222].

6.3.5 Data extractor

A specific software component is responsible for the classification of the RDF terms extracted from the KB. This is done through a set of SPARQL queries. In Listing 6.1 is reported the SPARQL query to identify classes (and optional details), while Listings 6.2 and 6.3 report the SPARQL queries used to respectively detect datatype and object properties.

Listing 6.1: Classes

```
1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX owl:<http://www.w3.org/2002/07/owl#>
4 SELECT DISTINCT ?class ?label ?comment
```

```
WHERE {
  { ?resource rdf:type ?class .
    OPTIONAL { ?class rdf:label ?label } .
    OPTIONAL { ?class rdf:comment ?comment }
  }
  UNION {
    ?class rdf:type owl:Class .
    OPTIONAL { ?class rdf:label ?label } .
    OPTIONAL { ?class rdf:comment ?comment }
  }
  UNION {
    ?class rdf:type rdfs:Class .
    OPTIONAL { ?class rdf:label ?label } .
    OPTIONAL { ?class rdf:comment ?comment }
  }
}
```

Listing 6.2: Datatype Properties

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?prop ?domain ?range ?label ?comment
WHERE {
  { ?prop rdf:type owl:DatatypeProperty .
    OPTIONAL{ ?prop rdfs:range ?range } .
    OPTIONAL{ ?prop rdfs:domain ?domain } .
    OPTIONAL{ ?prop rdfs:label ?label } .
    OPTIONAL{ ?prop rdfs:comment ?comment }
  }
  UNION {
    ?s ?prop ?o .
    FILTER isLiteral(?o)
  }
}
```

Listing 6.3: Object Properties

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?prop ?domain ?range ?label ?comment
WHERE {
  { ?prop rdf:type owl:ObjectProperty .
    OPTIONAL { ?prop rdfs:range ?range } .
    OPTIONAL { ?prop rdfs:domain ?domain } .
    OPTIONAL { ?prop rdfs:label ?label } .
    OPTIONAL { ?prop rdfs:comment ?comment }
  }
  UNION
  { ?s ?prop ?o .
    FILTER (isIRI(?o) || isBlank(?o))
  }
}
```

The data classifier also identifies resources, literals and blank nodes, but this data is not extracted through SPARQL queries. In fact, since this information is already available through the underlying Python RDFlib, no further computation is needed.

6.3.6 User Interface

Fig. 6.12 shows the User Interface of Tarsier. The top left hand side panel allows the user to load the YAML configuration file and shows all the parameters read from it. Below, is reported a list of the (customizable) colors set for the 3D graph and other parameters related to the view. Among these parameters it is worth mentioning the level of details (LOD) that allows to set the quality of the representation to find the best trade-off between resource usage and appearance. On the right hand side there is the canvas where the graph is drawn. Below the canvas, a text box shows information about the clicked elements.

On the bottom of the UI it is possible to notice a control panel containing the following eight cards:

- **Classes**: presents a lists of the classes identified by the data extractor. Every class can be selected/deselected through a checkbox. The selection allows toggling the visibility of classes and/or their instances, or move items across layers.

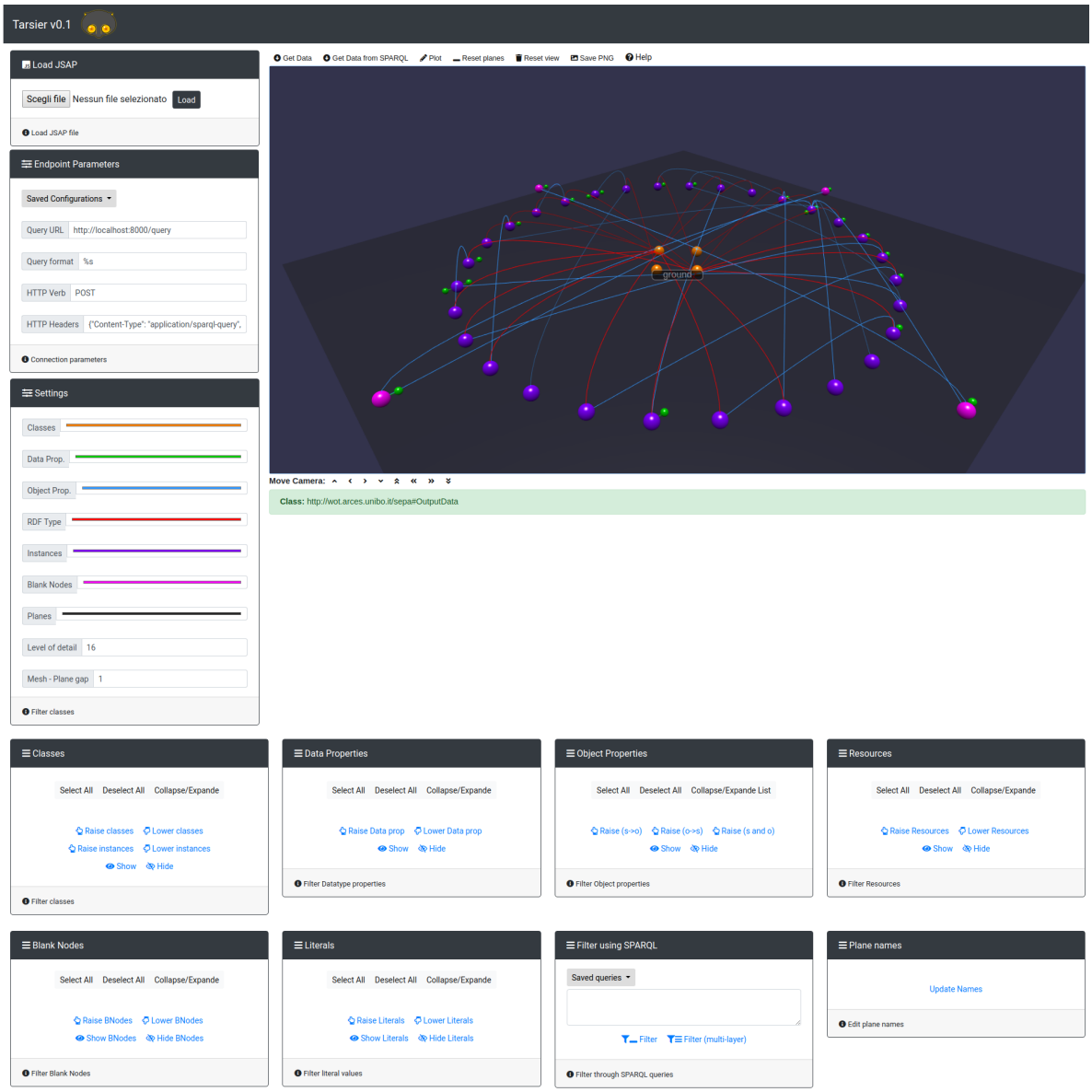


Figure 6.12: UI of Tarsier [222].

- **Resources**: proposes a list of referents (i.e. IRI resources). The user is allowed to select/deselect items and modify visibility and the layer the items belongs to.
- **Blank Nodes**: this panel presents the list of blank nodes found in the knowledge base. As with the previous boxes, the user can select items and change visibility and layer.
- **Object Properties**: this box shows the list of the object properties detected by the data extractor. Item can be selected and then shown/hidden or moved across layers.
- **Data Properties**: this box proposes the list of all the data properties.

- **Literals**: this box reports the list of literals (i.e. values of the datatype properties) found in the KB. The user can move them across layers or toggle their visibility.
- **Filter Using SPARQL**: this box is for the advanced filtering through the SPARQL query language, that permits more complex analysis than the standard filtering. The results of the SPARQL query are shown on a new semantic plane or on a set of semantic planes (i.e., one for each variable in the variable list of the query). A second important function of this text area is to specify queries for the pre-filtering stage.
- **Plane names**: this last box is used to rename the semantic planes according to their meaning.

6.4 Examples

This Section proposes four examples showing Tarsier and semantic planes in action. The first example is a didactic scenario based on the FOAF ontology. This is often the first ontology that students meet while learning how to deal with Semantic Web technologies. In the second one, Tarsier is used to visualize data extracted from DBpedia. A third example is shown based on the reification pattern. Lastly, a use case pivoting the knowledge base of the activities described in Chapter 8 is proposed.

6.4.1 Use Case #1: Teaching through FOAF

From the didactic point of view, Tarsier may help facing the steep learning curve of Semantic Web technologies. Tarsier is aimed at the visualization of an RDF graph, being it an ontology or the content of a store. It allows users to isolate the concepts of interest, while still maintaining a view to the rest of the data and it is not intended to build or modify RDF stores, but rather to explore and debug. Then, Tarsier can be considered as part of a student or developer toolkit, together with ontology editors, dashboards and APIs.

The default color scheme, also visible in the following examples, is based on the one proposed by the well-known ontology editor Protégé[16]: classes are represented with the orange color, datatype and object properties with respectively green and blue; individuals and blank nodes are painted purple and pink. Moreover, Tarsier adopts the color red to mark the property `rdf:type` that is very important to quickly identify the relationship between a class and its instances.

The example proposed in this Section is based on the FOAF ontology[17] (one of the first

[16] https://protege.stanford.edu/
[17] http://xmlns.com/foaf/spec/

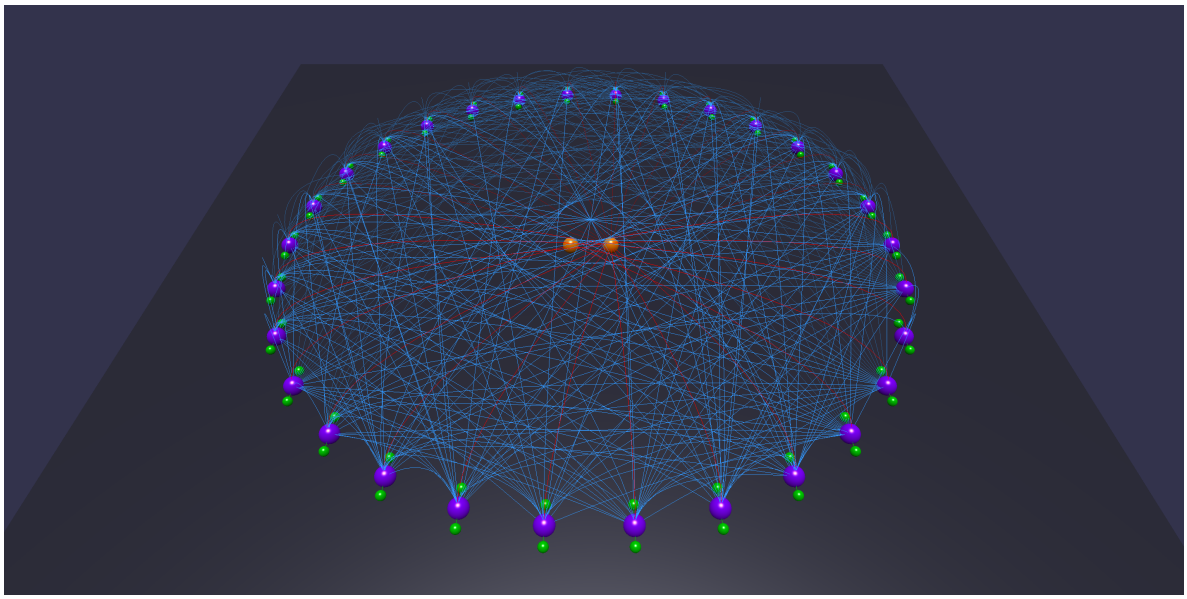


Figure 6.13: Full knowledge base of first the use case [222].

met by students approaching Semantic Web technologies). The datasets contains people, projects and relations among them. For the sake of clarity, the size of the knowledge base will be kept small, but this is not limiting, since the UI proposes intuitive filtering functions to hide unwanted items and navigate the 3D space. Displaying only a small-sized graph is an expedient to have better screenshots.

Table 6.1 briefly summarizes the content of the knowledge base. The graphical representation of the full graph is instead proposed by Fig. 6.13.

Table 6.1: Use Case #1: Summary of the knowledge base [222]

OWL Ontology T-Box Content	
Classes (`Person`, `Project` ∈ `foaf`)	2
Object Properties (`knows`, `currentProject` ∈ `foaf`)	2
Datatype Properties (`name`, `surname`, `status` ∈ `foaf`)	2
OWL Ontology A-Box content	
Persons	25
Projects	5
Links among persons (i.e. `foaf:knows`)	250
Links persons-projects (i.e. `foaf:currentProject`)	125

We may want to find an answer for the following questions:

1. Is there a person without friends?

2. Is there any unassigned project?

3. Do *Person1* and *Person2* share any projects?

Question 1 – *Is there a person without friends?* An answer to this question can be obtained in multiple ways. For example through a SPARQL query like the following one:

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3  SELECT ?p1
4  WHERE {
5    ?p1 rdf:type foaf:Person .
6    ?p2 rdf:type foaf:Person .
7    FILTER NOT EXISTS { ?p1 foaf:knows ?p2 } .
8    FILTER NOT EXISTS { ?p2 foaf:knows ?p1 } .
9    FILTER(?p1 != ?p2)
10 }
```

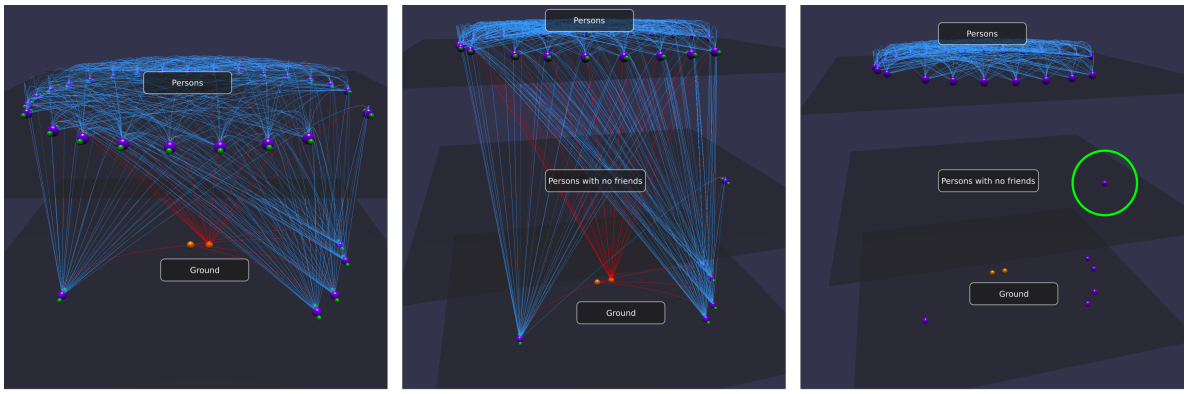
However, Tarsier provides an even simpler way to achieve this scope through the creation of the following semantic planes. In order, the user should:

a) Create a first semantic plane hosting the instances of the class `foaf:Person` (that are then moved above the rest of the knowledge base).

b) Create a second semantic plane hosting all the instances of the class `foaf:Person` that are involved in a friendship relationship (i.e. being either the subject or the object of a `foaf:knows` triple). In this way all the persons without friends, if any, remain on the previously created plane. This can be done selecting the object property `foaf:knows` and clicking on *Raise (S and O)*.

c) In the end, just to achieve a cleaner view, it is possible to hide unwanted information (e.g., all the datatype properties and all the object properties except `foaf:knows`).

The previous steps are all graphically shown by Fig. 6.14. Semantic planes allow one to immediately notice the existence of an isolated instance of the class `foaf:Person` that stands on the mid-plane renamed as "*Persons with no friends*". To enhance the readability, datatype properties and object properties other than `foaf:knows` were hidden through the UI commands.



a b c

Figure 6.14: Use Case #1, question 1: one instance of the class `foaf:Person` has no incoming or outgoing `foaf:knows` edges [222].

Question 2 – *Is there any unassigned project?* One could answer this question through the following query:

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3 SELECT ?p
4 WHERE {
5   ?p rdf:type foaf:Project .
6   ?person rdf:type foaf:Person .
7   FILTER NOT EXISTS { ?person foaf:currentProject ?p }
8 }
```

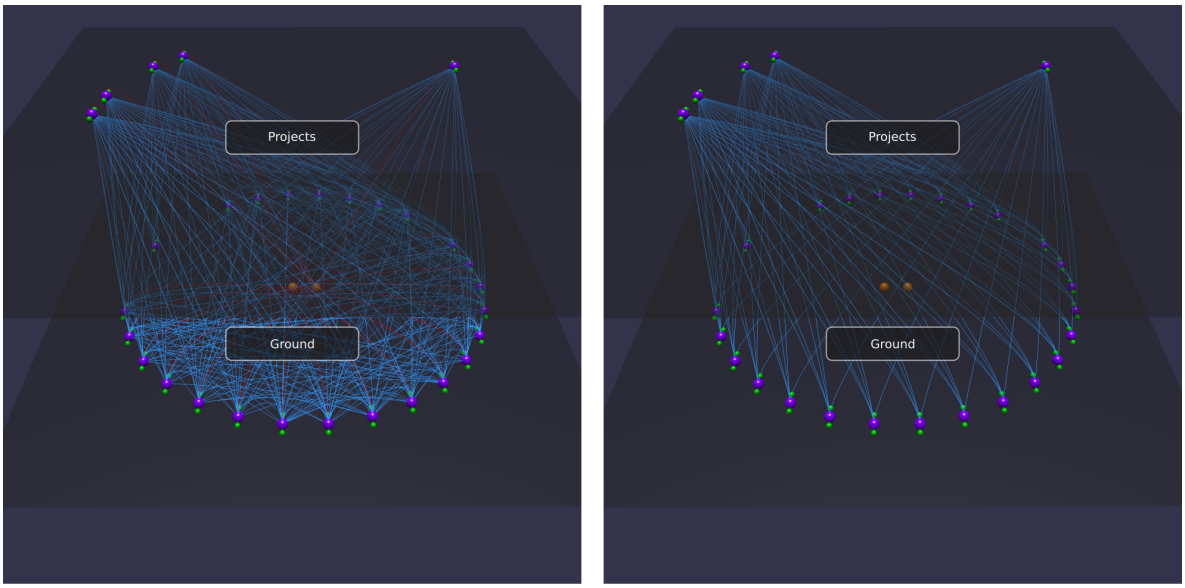
But, Tarsier allows retrieving and showing the same information without having to know the SPARQL query language. Again, the user may draw upon semantic planes by (in order):

a) Creating a semantic plane containing all the projects (i.e., selecting the class `foaf:Project` and clicking on *Raise instances*);

b) Hiding all the data properties and all the arcs related to `foaf:knows` and `rdf:type`.

The result of these actions is shown in Fig. 6.15a (or Fig. 6.15b where unwanted data has been hidden). The user may immediately notice that all the existing projects are assigned to instances of the class `foaf:Person`.

Question 3 – *Do Person1 and Person2 share any projects?* The third question can, once again, be answered through a SPARQL query:

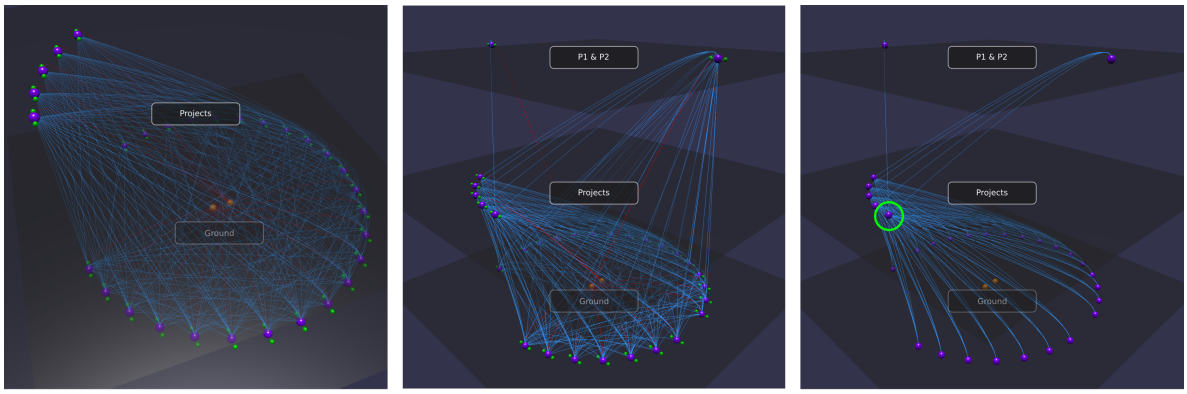


a b

Figure 6.15: Use case #1, question 2: the semantic plane of the projects clearly highlight that all the projects are bound to at least one person (a). Undesired data can be hidden from the proper UI commands (b) [222].

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3 ASK {
4   foaf:Person1 foaf:currentProject ?p .
5   foaf:Person2 foaf:currentProject ?p
6 }
```

Once again, the user may avoid typing a SPARQL query creating a semantic plane for the projects (step *a*) and a semantic plane hosting only the resources `foaf:Person1` and `foaf:Person2` (step *b*). Hiding object properties different from `foaf:currentProject` and all the datatype properties (step *c*), it is easy to notice that one of the projects (hosted by the middle semantic plane) presents two incoming edges from the topmost plane (the one related to the selected persons). Then, the previous question has an affirmative answer. If needed, a click on the project reveals further information. These steps and the results are visualized in Fig. 6.16.



a b c

Figure 6.16: Use Case #1, question 3: Do `foaf:Person1` and `foaf:Person2` work on at least a common project? From the semantic planes defined, it is easy to identify a project where `foaf:Person1` and `foaf:Person2` work together [222].

6.4.2 Use Case #2: Exploring DBpedia

Tarsier is able to retrieve data from any standard SPARQL endpoint. DBpedia[18] is one of them. It is a public data infrastructure for a large, multilingual, semantic knowledge graph. As previously mentioned, a tool for the visualization of RDF graphs should provide a way to declare the portion of the knowledge base that the user intends to inspect. This is of paramount importance with DBpedia, since visualizing a graph containing 6.6M entities (this is the size of the last official release) can be both computationally heavy and uneffective from the point of view of the results. Then, when using Tarsier to deal with DBpedia, the user can define the subgraph of interest through a proper SPARQL CONSTRUCT query. The example proposed in this Section allows extracting from DBpedia the following graph:

> *Artists born in Bologna between 1000 a.C. and 2000 a.C. and people who inspired them.*

The desired information can be retrieved with a SPARQL query like the following:

```
1 PREFIX : <http://dbpedia.org/resource/>
2 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
5 PREFIX dbo: <http://dbpedia.org/ontology/>
```

[18] http://dbpedia.org

```
SELECT ?artist ?artBirthD ?artDeathD ?artBirthP ?artName
?ins ?insName ?insBirthD  ?insDeathD ?insBirthP ?insDeathP
WHERE {
  ?artist rdf:type dbo:Artist ;
          rdf:type foaf:Person ;
          foaf:name ?artName ;
          dbo:birthDate ?artBirthD ;
          dbo:birthPlace :Bologna .
  OPTIONAL {
    ?artist dbo:deathDate ?artDeathD } .
  OPTIONAL {
    ?artist dbo:deathPlace ?artDeathP } .
  OPTIONAL {
    ?artist dbo:influencedBy ?ins .
    ?ins rdf:type foaf:Person ;
         dbo:birthPlace ?insBirthP ;
         dbo:birthDate ?insBirthD .
    OPTIONAL {
      ?ins dbo:deathPlace ?insDeathP ;
           dbo:deathDate ?insDeathD }} .
  FILTER (?artBirthD > "1000-01-01"^^xsd:date).
  FILTER (?artBirthD < "2000-01-01"^^xsd:date)
}
```

A SPARQL CONSTRUCT query can be easily derived from the original SELECT, in order to define a graph with the same meaning:

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>
CONSTRUCT {
  ?artist rdf:type dbo:Artist ;
```

```
8           rdf:type foaf:Person ;
9           foaf:name ?artName ;
10          dbo:birthDate ?artBirthD ;
11          dbo:birthPlace :Bologna ;
12          dbo:deathDate ?artDeathD ;
13          dbo:deathPlace ?artDeathP ;
14          dbo:influencedBy ?ins .
15  ?artDeathP rdf:type dbo:Place .
16  :Bologna rdf:type dbo:Place .
17  ?ins rdf:type foaf:Person ;
18       dbo:birthPlace ?insBirthP ;
19       dbo:birthDate ?insBirthD ;
20       dbo:deathPlace ?insDeathP ;
21       dbo:deathDate ?insDeathD .
22  ?insDeathP rdf:type dbo:Place .
23  ?insBirthP rdf:type dbo:Place .
24 }
25 WHERE {
26  ?artist rdf:type dbo:Artist ;
27          rdf:type foaf:Person ;
28          foaf:name ?artName ;
29          dbo:birthDate ?artBirthD ;
30          dbo:birthPlace :Bologna .
31  OPTIONAL {
32    ?artist dbo:deathDate ?artDeathD } .
33  OPTIONAL {
34    ?artist dbo:deathPlace ?artDeathP } .
35  OPTIONAL {
36    ?artist dbo:influencedBy ?ins .
37    ?ins rdf:type foaf:Person ;
38         dbo:birthPlace ?insBirthP ;
39         dbo:birthDate ?insBirthD .
40    OPTIONAL {
41      ?ins dbo:deathPlace ?insDeathP ;
42           dbo:deathDate ?insDeathD }} .
```

```
43    FILTER (?artBirthD > "1000-01-01"^^xsd:date).
44    FILTER (?artBirthD < "2000-01-01"^^xsd:date)
45  }
```

Fig. 6.17 shows the subgraph resulting from the previous step.

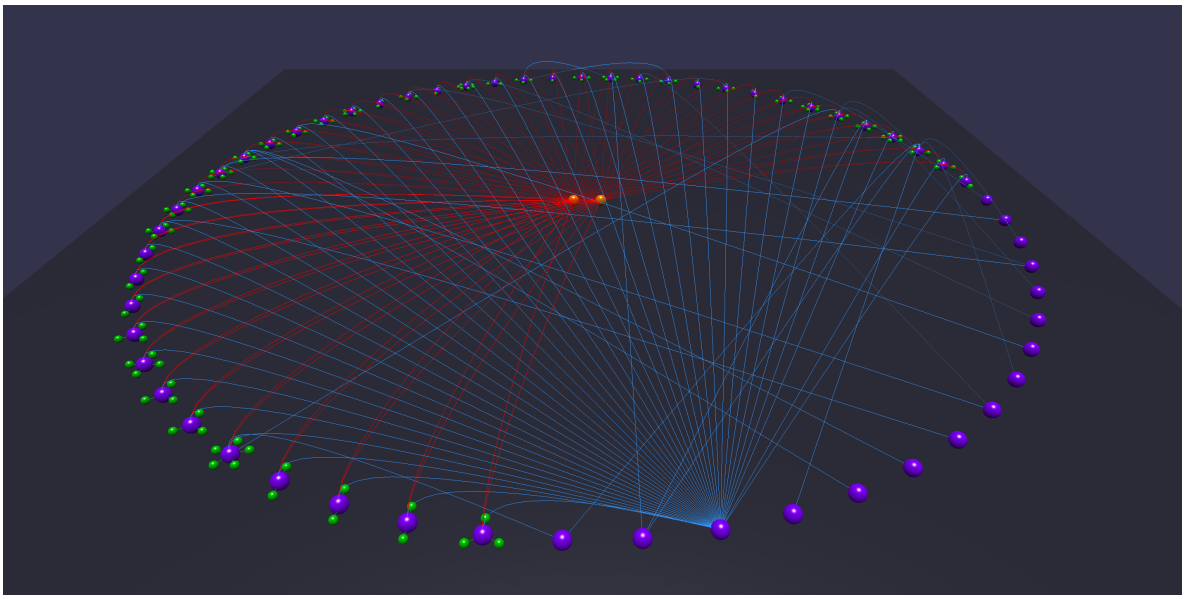
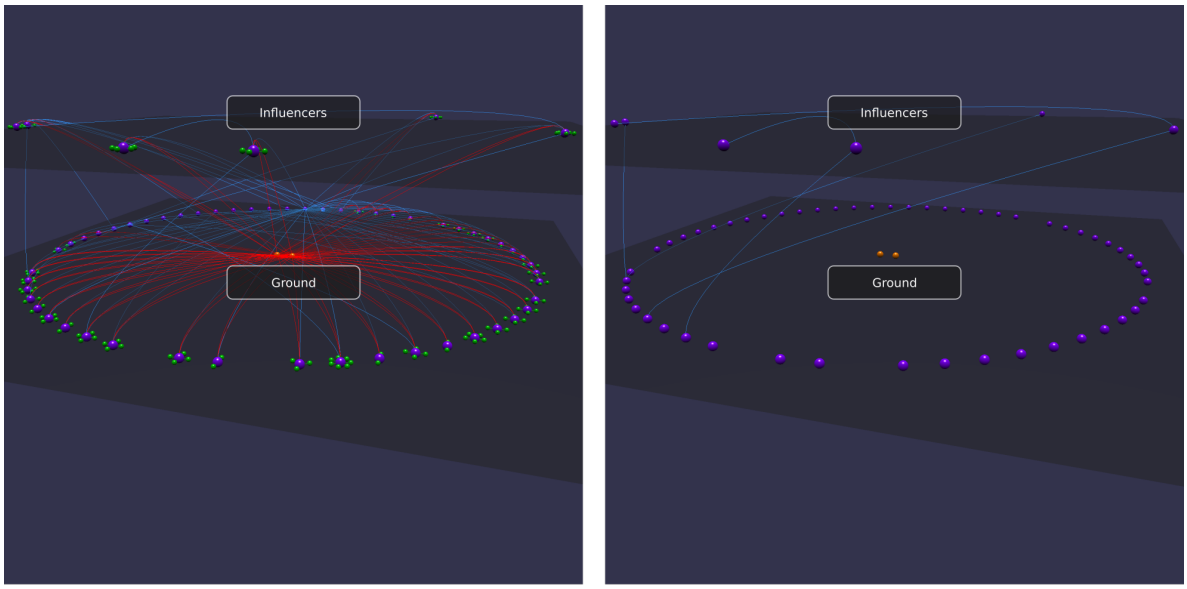


Figure 6.17: Visualization of the graph extracted from DBpedia, containing all the artists born in Bologna between 1000 a.C. and 2000 a.C. and people who inspired them [222].

The subgraph extracted with the CONSTRUCT can be browsed by using the mouse and clicking on spheres and edges to see the related information. Tarsier can then be used to answer questions about the visualized data, through the use of semantic planes. Keeping the previous graph as reference, three possible questions that we could answer using Tarsier are:

1. Are there any relations among *influencers*?
2. Are there any connections between living artists and influencers?
3. Are there any living artists?

Question 1 – *Are there any relations among influencers?* Semantic planes allows one to easily get a response. In fact, it is sufficient to create a semantic plane hosting influencers (step *a*) and hide unwanted information (step *b*), to notice the presence of two links among influencers (Fig. 6.18a). A click on these links shows that Carlo Cignani was influenced by Francesco Albani, while Ludovico Carracci was influenced by Annibale Carracci. In Fig. 6.18b



a b

Figure 6.18: Use Case #2, question 1: a semantic plane showing the influencers, placed above the semantic plane with the rest of the KB

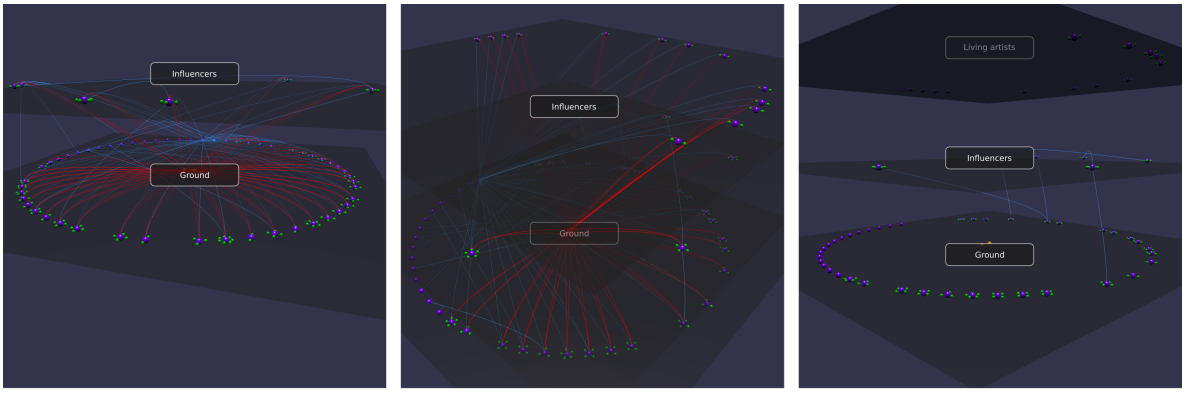
all the other object properties and all the data properties were hidden through the proper controls.

Question 2 – *Are there any connections between living artists and influencers?* The answer to this question derives from two steps that correspond to the creation of two semantic planes: the first one dedicated to influencers, the second to living artists. A semantic plane dedicated to influencers can be created as in the previous example, while the second plane can be achieved with a simple SPARQL query:

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX dbo: <http://dbpedia.org/ontology/>
3 SELECT ?art
4 WHERE {
5   ?art rdf:type dbo:Artist .
6   FILTER NOT EXISTS { ?art dbo:deathPlace ?dp }.
7   FILTER NOT EXISTS { ?art dbo:deathDate ?dd }
8 }
```

A third optional step is to hide unwanted information. These steps and the resulting visualization are depicted in Fig. 6.19. It is easy to notice the absence of connections between the two new semantic planes. So, at least according to DBpedia, none of the living artists

born in Bologna has been influenced by another artist.



a b c

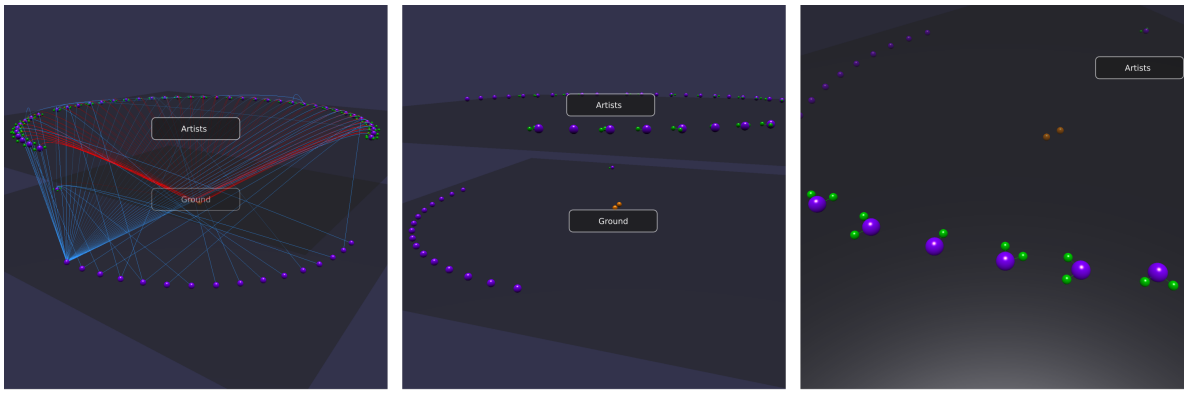
Figure 6.19: Use Case #2, question 2: A semantic plane containing the living artists standing above the semantic plane of the influencers. No links between these two planes. On the bottom, the rest of the knowledge base [222].

Question 3 – *Are there any living artists?* To answer this question is sufficient to move all the artists to a dedicated semantic plane. Then, hiding all the data properties except `dbo:deathDate` a more clean visualization is achieved. These two steps are depicted in Fig. 6.20, subfigures *a* and *b*. All the resources not connected to a green sphere are living artists. Subfigure *c* shows a close-up on the second plane: resources without visible data properties can be considered as living artists. Through this close-up, the user may notice something strange in the knowledge base: many resources have more than one visible green ball: i.e., many artists died more than once! This is the case, for example, of Alessandro Tiarini, for which the death date is reported as `"1668-02-08"` and `"1668-2-8"` (clearly the same date with different formatting) or the case of Domenichino which instead has two death dates that differ not only for the formatting, but also for the value of the year (`"1641-4-6"` and `"1648-04-06"`).

Therefore, the wrong use of a functional property can be easily identified through Tarsier.

6.4.3 Use Case #3: Reificated KBs

In this Section, a third use case still based on FOAF (and in particular on the classes `Person`, `Project` and `Organization`) is proposed. The knowledge base adopted in this use case is a very simple one showing relationships among people and projects. Organizations confirm the relationships adding a start date and the envisioned end date (if any). In this sample KB, the confirmation of a relationship between a person and a project is expressed through the reification pattern. In this specific case, the standard reification [223] that involves the



a b c

Figure 6.20: Use Case #2, question 3: Green spheres refer to the `dbo:deathDate` property. Having more than one of these spheres means that the related artist has more than one death date [222].

`rdf:Statement` class. A summary of the knowledge base is proposed in Table 6.2.

Table 6.2: Use Case #3: Summary of the Knowledge Base [222].

OWL Ontology T-Box Content	
Classes (`Person`, `Project`, `Organization` ∈ `foaf`, `Statement` ∈ `rdf`)	4
Object Properties (`currentProject` ∈ `foaf`, `ackBy`, `startDate`, `endDate` ∈ `ns`, `subject`, `predicate`, `object` ∈ `rdf`)	6
Datatype Properties (`name`, `surname`, `status` ∈ `foaf`, `object` ∈ `rdf`)	5
OWL Ontology A-Box content	
Persons	20
Projects	5
Organizations	2
Links persons-projects (i.e. `foaf:currentProject`)	20
Links organizations-persons (i.e. `foaf:member`)	20
Acknowledged statements (i.e. `ns:ackBy`)	20
Statements time-stamped with `ns:startDate`	20
Statements time-stamped with `ns:endDate`	10

The description of the example deserves a brief introduction to standard reification. RDF allows representing all the information as a set of triples (subject, predicate and object) like:

```
foaf:Person1 foaf:currentProject foaf:ProjA
```

This may be not enough, for example to state something about a given triple. The reification pattern allows solving this issue. A triple $t = (s, p, o)$ is broke down in four triples:

the first is used to declare a statement, the others to express its components (i.e., subject, predicate and object). Based on the previous triple, the reification pattern can be applied as follows (`ns` is a custom namespace):

```
ns:St1 rdf:type rdf:Statement
ns:St1 rdf:subject foaf:Person1
ns:St1 rdf:predicate foaf:currentProject
ns:St1 rdf:object foaf:ProjA
```

Additional information on the triple can be expressed by simply referring to the statement:

```
ns:St1 ns:ackBy foaf:Organization1
ns:St1 ns:startDate "..."
ns:St1 ns:endDate "..."
```

Back to the Use Case #3, the unfiltered content of the knowledge base described in Table 6.2, is presented in Fig. 6.21. Fig. 6.22 shows one of the present statements on a dedicated plane. This visualization allows identifying the subject, the predicate and the object composing the triple. In the described scenario, an instance of the class `rdf:Statement` is used to link a person to its current project. Organizations may acknowledge the triple and append information to each statement as the start and end date of the collaboration. Link outgoing from the statement (i.e., the pink sphere, since in this case the statement has been defined as a blank node) represent the information appended by the Organization: only a data property is bound to the statement, so no end date is envisioned for the collaboration of the person with that project. The presence of the underlying semantic plane (i.e., the ground) name allows maintaining a view on the rest of the KB, even when looking at a single statement. It is possible to notice, in the specific example, that the predicate is linked to the ground by a high number of links, so many other statements may have this predicate; the project only has three links with the ground (a click on them reveals that are links of type `foaf:currentProject`).

In Fig. 6.23 another example based on the same KB is proposed. All the statements were moved to a proper semantic plane (step *a*), while one of the organizations (i.e., instances of `foaf:Organization`) was moved to the topmost plane (step *b*). Then, all the `rdf:type` edges have been hidden for the sake of readability. This allows identifying the relationships among the selected organization and all the instances of `rdf:Statement`. This view shows that this organization took the burden of acknowledging all of the present statements.

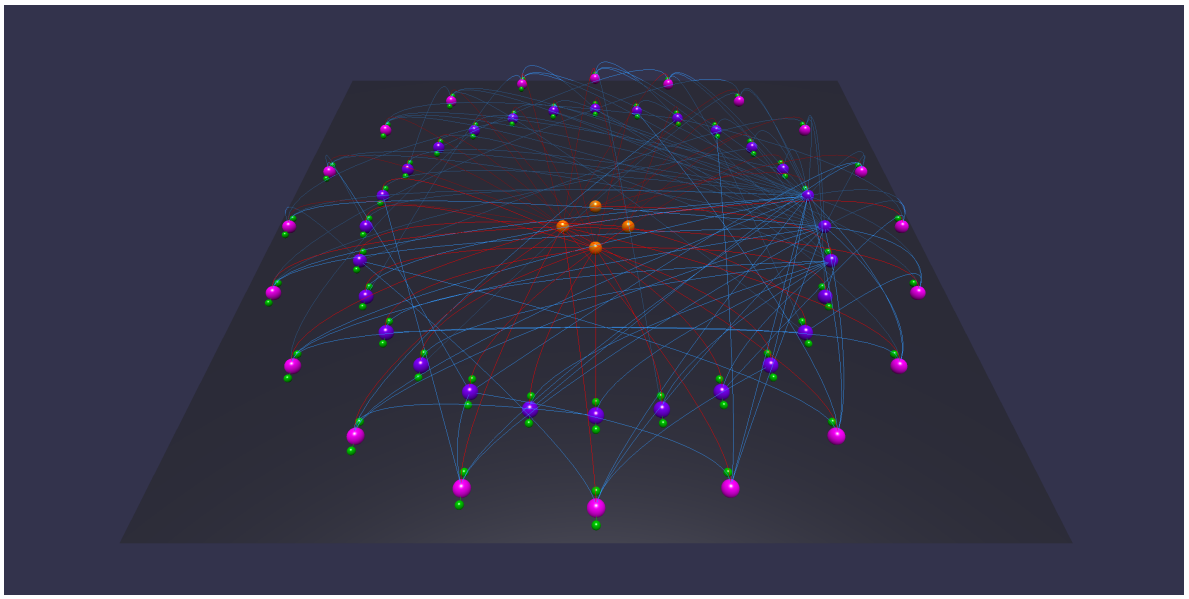


Figure 6.21: Use Case #3: The unfiltered knowledge base of the reification use case [222].

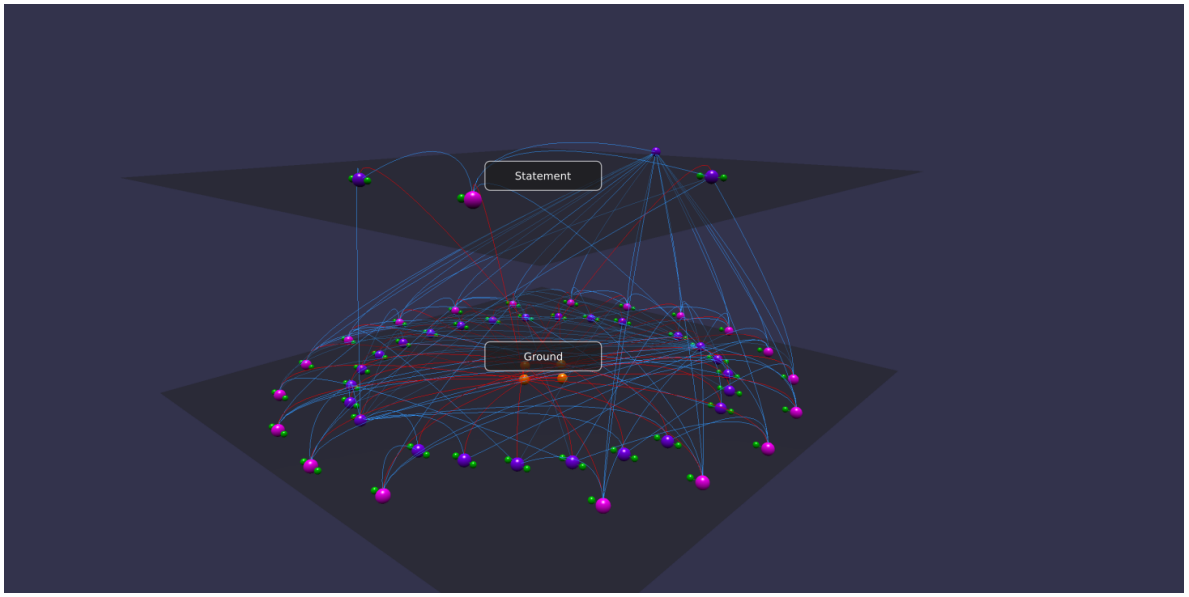
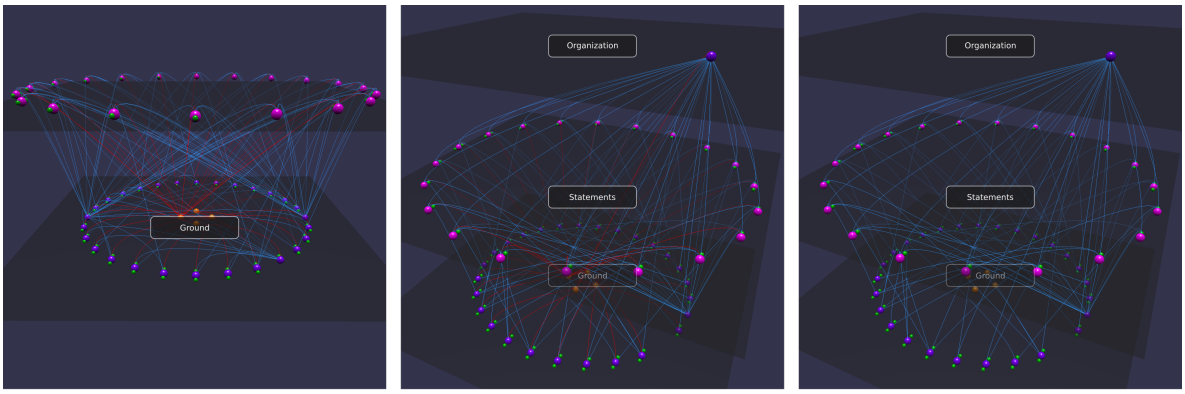


Figure 6.22: Use Case #3: A statement has been moved to a dedicated plane [222].

6.4.4 Use Case #4: Debugging an IoT application

A fourth use case where Tarsier demonstrated its effectiveness is the debug of a set of real IoT applications. The applications are related to the Internet of Energy (IoE) and Arrowhead EU Research projects and concern the co-simulation of the Smart Grid and electric vehicles in the city of Bologna. These applications, better described in Chapter 8, are the result of the



a b c

Figure 6.23: Use Case #3: A multi-planar view with a topmost semantic plane dedicated to the Organization 1, a second plane with all the statements and the third with the rest of the KB [222].

collaboration of several developers from different departments. While testing the reservation app, a series of tricky bugs appeared. Identifying the bugs through the classic exploration of the knowledge base with queries and tables proved to be a slow and ineffective process. The visual approach proposed by Tarsier, thanks to the use of Semantic Planes, allowed us to identify some erroneous triples with a few clicks. An example of these erroneous triples is visible in Fig. 6.25 (while the full knowledge base is shown in Fig. 6.24). The query presented in the following Listing, permits the creation of three semantic planes to host respectively the instances of the `ChargeProfile`, the instances of the class `VoltageData` and those of the class `UnitOfMeasure`. Hiding unwanted information, to visualize only the links between the two topmost planes, is easy to notice an instance of the class `VoltageData` that is not connected to the proper unit of measure. A further inspection allowed discovering the cause of the bug: a *typo* in the name of the predicate (i.e., `hasUnityOfMeasure` instead of `hasUnitOfMeasure`).

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX ioe:<http://www.m3.com/2012/05/m3/ioe-ontology.owl#>
3  SELECT ?cp ?volt ?uom
4  WHERE {
5    ?cp rdf:type ioe:ChargeProfile .
6    ?cp ioe:hasVoltage ?volt .
7    OPTIONAL {
8    ?volt ioe:hasUnitOfMeasure ?uom .
9      }
10 }
```

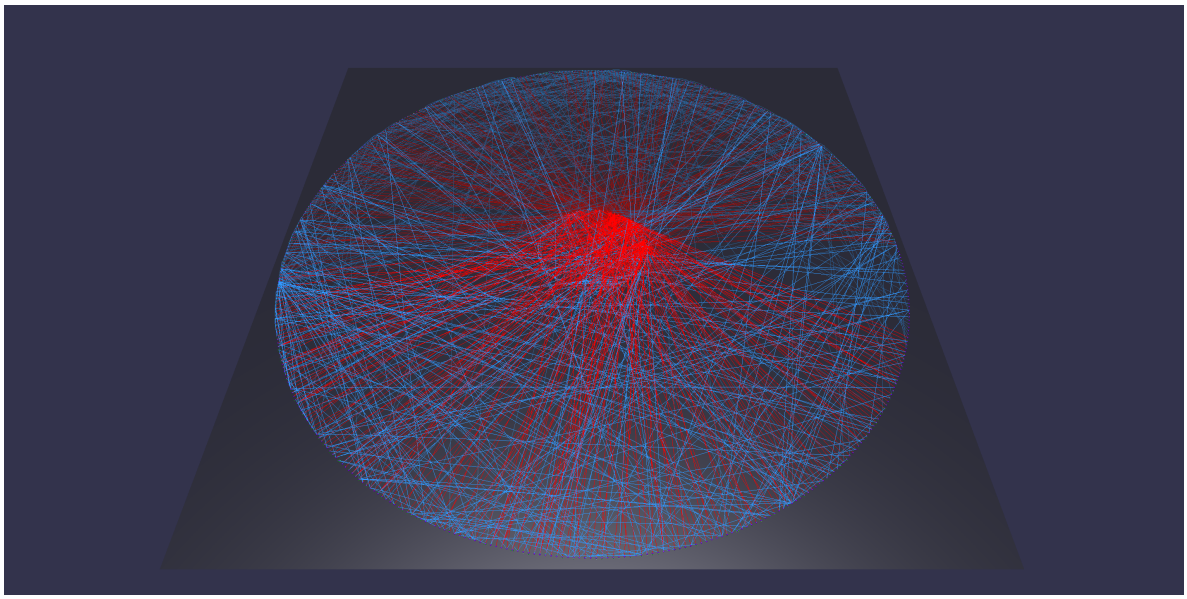


Figure 6.24: Use Case #4: The whole IoE dataset loaded in Tarsier

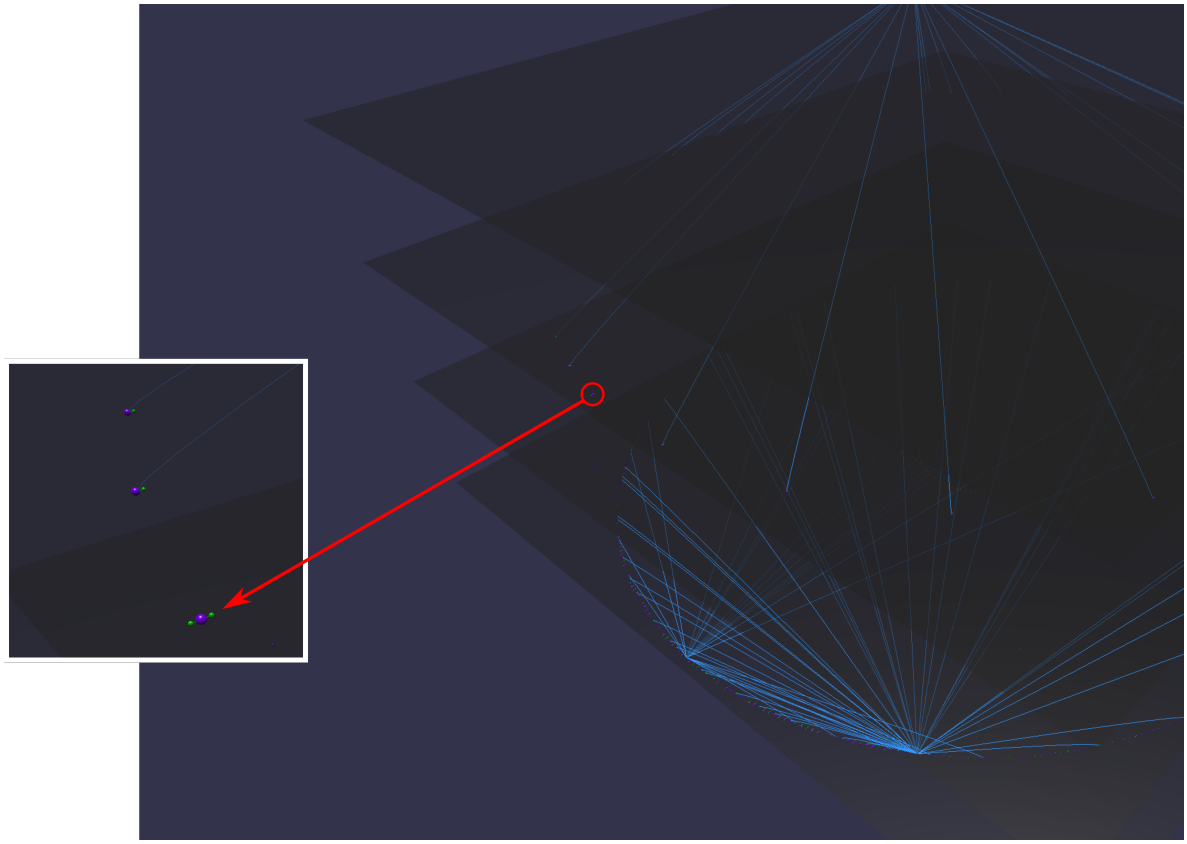


Figure 6.25: Use Case #4: A SPARQL filter applied to the IoE knowledge base

6.5 Evaluation

A preliminary analysis of the user experience has been carried out to assess the validity of the approach. Moreover, the performances of the tool have been measured to characterize its behaviour. The first activity is described in Section 6.5.1), while the computational assessment of the performance is detailed in Section 6.5.2.

6.5.1 User evaluation

After the first implementation of Tarsier, a test of the User Experience was performed. The purpose of the test was to study the behaviour of the users to assess the validity of the approach and identify possible improvements.

Sixteen participants were selected among the students attending the course "*Interoperability of Embedded Systems*" held at the Computer Engineering faculty of the University of Bologna. This allowed me to assess the validity of the approach (and the tool) in the didactic with students dealing with Semantic Web technologies for the first times (only six of them have had previous minor experience with semantic technologies). A second user experience test will be carried out in the future with expert users.

The evaluation was based on a set of tasks to assess the efficiency and effectiveness. The evaluation was preceeded by a short presentation of the tool that lasted 10 minutes: during this presentation, the basics of the UI and the aim of the tool were discussed. Then, the experimenters were free to explore the tool and perform a feature walkthrough for other 10 minutes. The concurrent think-aloud protocol (CTA) algorithm was applied to gather the insights of users' cognitive processes during both the free experimenting phase and the execution of five assigned tasks (of increasing complexity).

As regards the tasks, in the first three, the users were asked to interact with a local SPARQL Endpoint based on SEPA [224], the same adopted by students for their final assessment project. The remaining tests were about the visualization of data contained in DBpedia. Participants were free to allocate the desired amount of time to carry out the assigned tasks for a total time of one hour.

After the test, students filled a survey with two sets of questions: the first task-specific, intended to understand how the user carried out each task, while the second aimed at the overall evaluation of the tool. Students were also allowed to write a short sentence after each question to express opinions and advices. The test was intended to assess the level of usability and the learning curve, the overall feedback and the perceived level of utility and novelty. The final test was based on ten 5-points Likert items (selected scale: *Strongly disagree*, *Disagree*,

Neutral, *Agree*, *Strongly Agree*).

Fig. 6.26 summarizes the results of the final test. These preliminary results suggest that the tool is useful for newcomers and effective to understand data. The idea of semantic planes and the filtering mechanism were judged positively by all the participants. Students suggested "*adding more visual tips and feedbacks*" and this request was promptly accepted in the subsequent release. Moreover, to guide newcomers, a help screen was added to the tool and three introductive videoclips were realized[19] to showcase some examples.

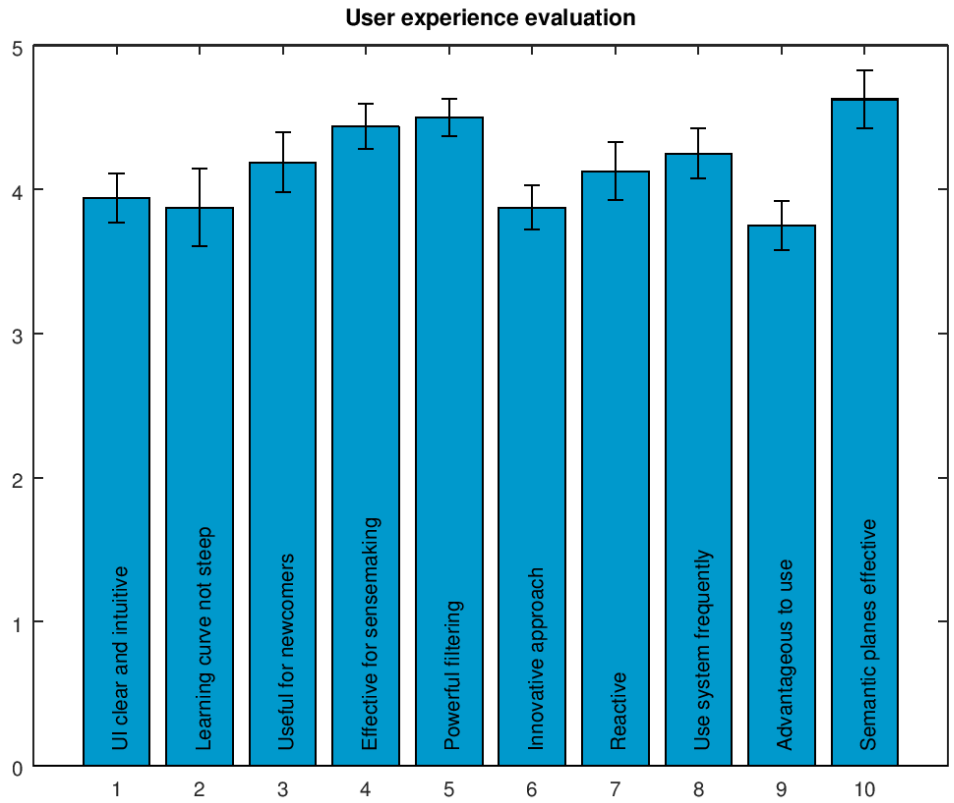


Figure 6.26: Mean and standard error of the mean (SEM) of the results of the questionnaire items [222].

6.5.2 Performance evaluation

Performances are not a primary aspect of inspection tools. Nevertheless, in this Section I present the results of preliminary evaluation tests carried out on Tarsier. Intuitively, the time

[19]The three introductory videoclips are available on the GitHub page `https://github.com/desmovalvo/tarsier`

needed to draw the graph depends on three main factors:

- The amount of data to be traced (i.e., the number of meshes);
- The requested LOD (configurable by the user);
- Whether or not, the 3D scene has been initialized (i.e., this condition is defined *cold start*).

To assess the performances of the visualization tool, I utilized a generic dataset with 5 different sizes. Datasets are identified with the labels DS#i with $i = 1, \ldots, 5$; Dataset DS#i contains $200 \cdot i$ triples, while the full representation requires $200 \cdot i + 1$ spheres (for classes and class instances) and $200 \cdot i$ bezier curves (adopted for datatype and object properties). Due to the nature of this test (i.e., evaluation of the drawing component), the specific ontology used to represent triples in the knowledge base does not influence the evaluation.

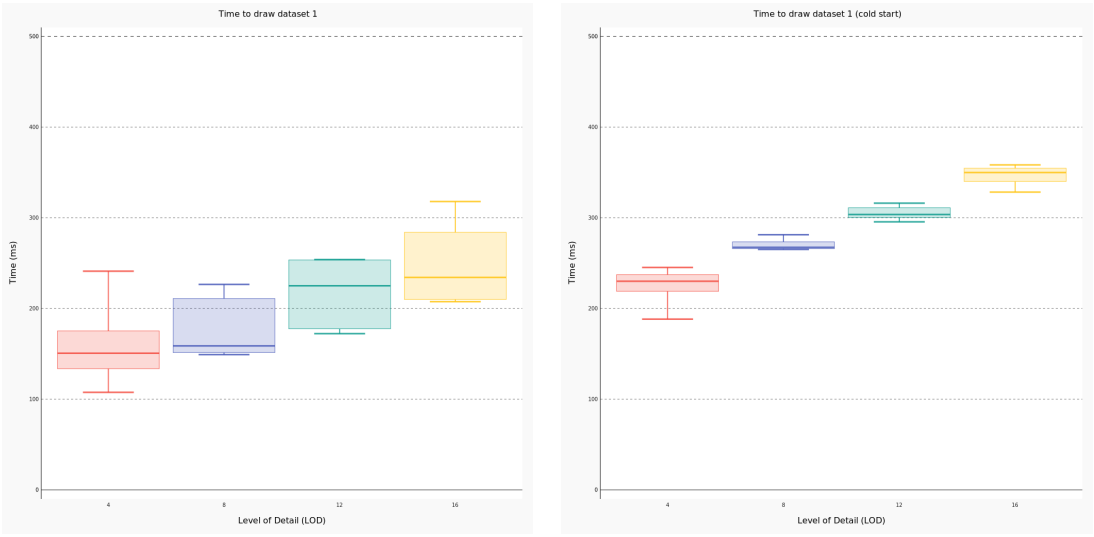
Every dataset was tested in a cold start condition and with an already initialized scene. To evaluate how the behaviour of the system changes with respect to the quality of the representation, four LOD values were tested (4, 8, 12 and 16). All the tests were performed on a Dell Alienware with 8-core Intel(R) Core(TM) i7-4720HQ CPU 2.60GHz and 8 GB RAM. Both server and client were running on the same machine using Google Chrome 64.0.

Results of tests executed on Tarsier in a cold-start situation are reported in Figg. 6.27a, 6.28a, 6.29a, 6.30a and 6.31a, while Figg. 6.27b, 6.28b, 6.29b, 6.30b and 6.31b refer to an already initialized instance of the scene.

The charts confirm the expected behaviour of the application, since time needed to draw the graph grows linearly with the size of data and with the requested level of details. Moreover, a sensibly higher number of milliseconds is required to complete the drawing if the scene has not been initialized.

6.6 Conclusion

In this Chapter I presented a novel approach to the visualization of RDF graphs based on the metaphor of Semantic Planes that allow grouping all the RDF terms sharing common concepts. To demonstrate the validity of the approach, I developed Tarsier, an interactive tool for the visualization of RDF KBs with support for RDFS and OWL. Tarsier permits the creation and editing of semantic planes, and their further split through a set of UI controls or through SPARQL 1.1 queries. The use of semantic planes in Tarsier allows to 1) support students learning how to deal with Semantic Web data representation formats 2) support



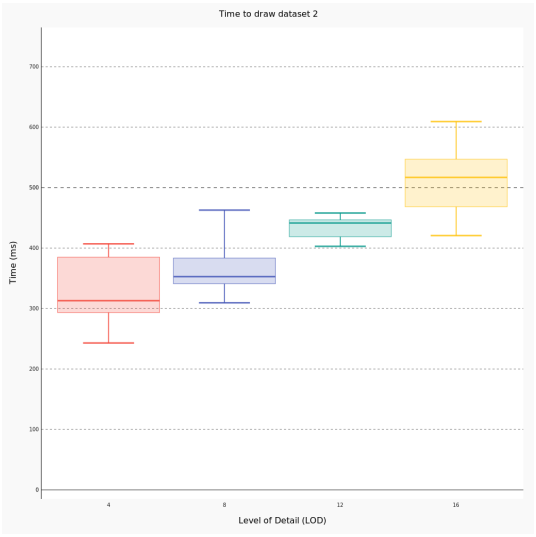
(a) DS#1 (Cold Start) **(b)** DS#1

Figure 6.27: Time to represent DS#1 [222].

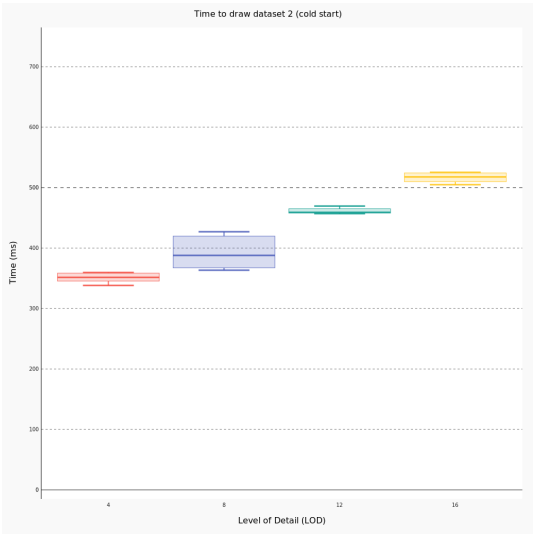
software developers during the debugging of applications with RDF stores (in fact, through the pre-filtering phase, it is possible to extract a subgraph from a very large dataset).

The preliminary user evaluation tests on the application suggest that the three-dimensional visualization of small and medium-sized knowledge bases, combined with the approach of semantic planes and a powerful filtering mechanism helps newcomers to understand the nature of data and its structure.

The development of the tool will continue with a particular focus on 1) the support for the whole set of SPARQL constructs (e.g. to allow using aggregation functions); 2) Alternative arrangement methods for meshes in the 3D space ; 3) Support for real-time visualization of data through SPARQL Event Processing Architectures (e.g. [224, 124, 225]); 4) Advanced statistics related to the knowledge base items.

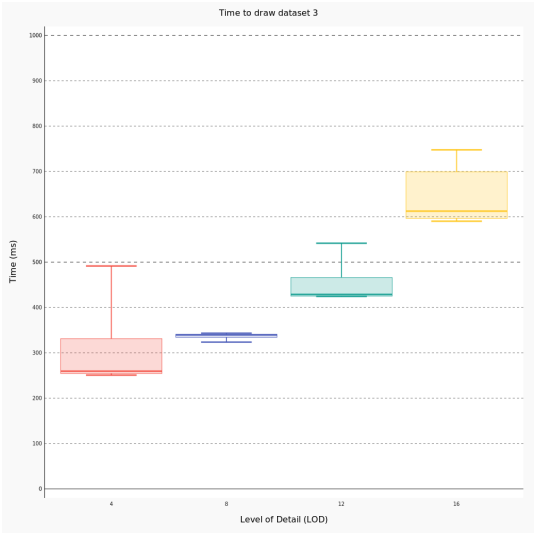


(a) DS#2 (Cold Start)

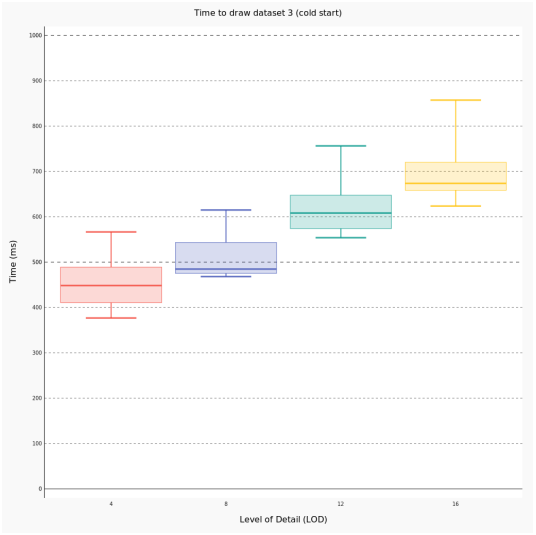


(b) DS#2

Figure 6.28: Time to represent DS#2 [222].

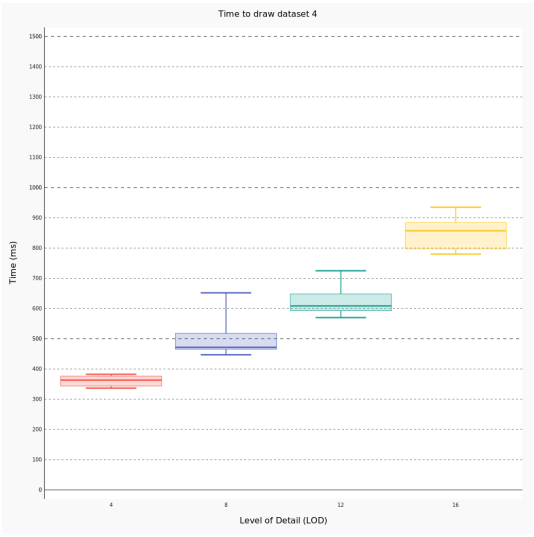


(a) DS#3 (Cold Start)

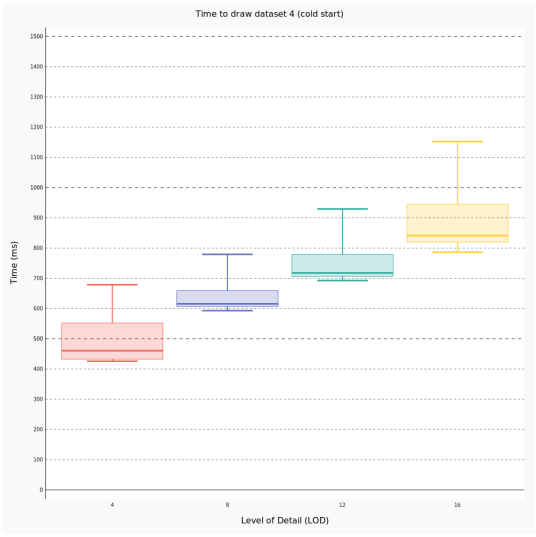


(b) DS#3

Figure 6.29: Time to represent DS#3 [222].

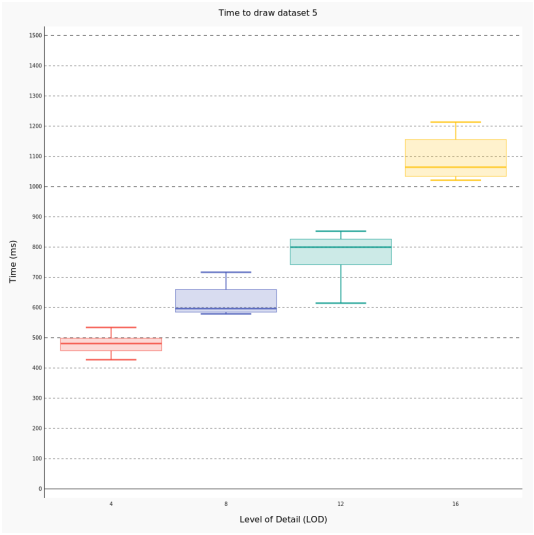


(a) DS#4 (Cold Start)

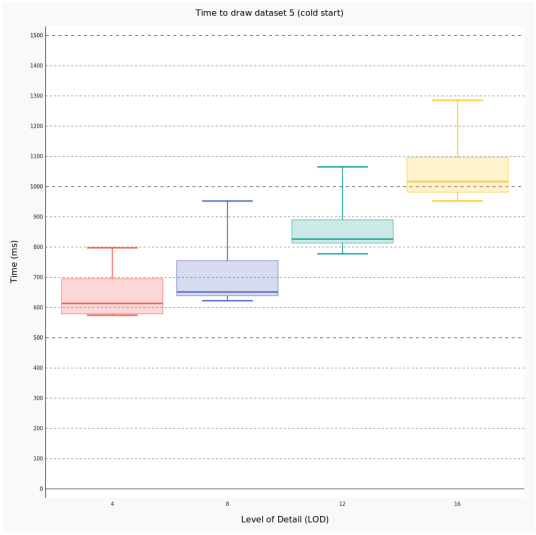


(b) DS#4

Figure 6.30: Time to represent DS#4 [222].



(a) DS#5 (Cold Start) [222].



(b) DS#5

Figure 6.31: Time to represent DS#5 [222].

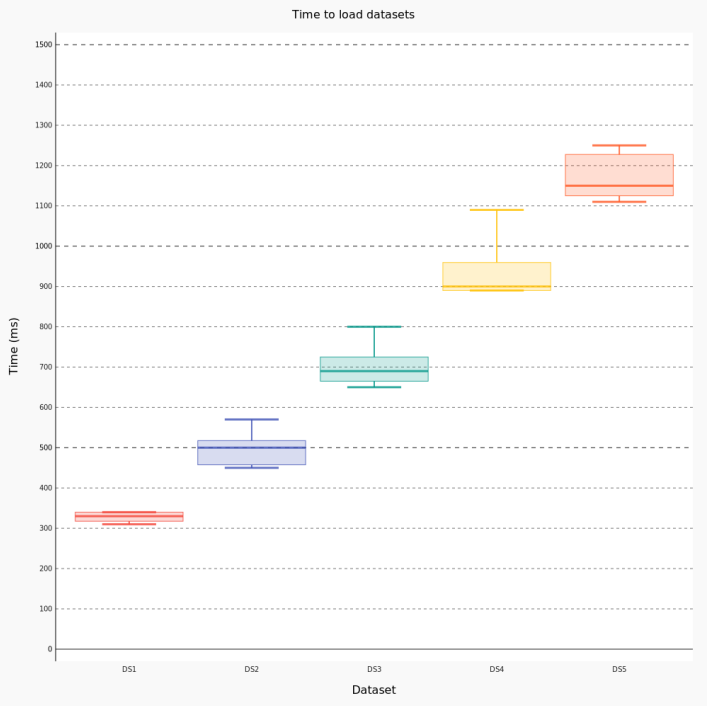


Figure 6.32: Time to analyse data depending on the dataset [222].

Part IV

Applications

Chapter 7

Applications development framework

Contents

A research on semantic infrastructures for the Internet of Things cannot be satisfactorily carried out without a constant validation of the work on real application domains. Then, Part IV presents the research activities concerning the application of the semantic platforms described in the previous Chapters to three domains. Before going further in detail, this Chapter describes the overall infrastructure built on top of the Smart-M3 and SEPA context brokers adopted to develop the Electro-Mobility and home automation applications as well as those pertaining the Semantic Audio and Internet of Musical Things. All of these applications will be presented respectively in Chapters 8, 9 and 10.

7.1 Smart-M3/SEPA Framework at a glance

Developing SWoT applications with the Smart-M3/SEPA platforms requires a set of common tools that form the model depicted in Fig. 7.1. This onion structure highlights the centrality of the context broker.

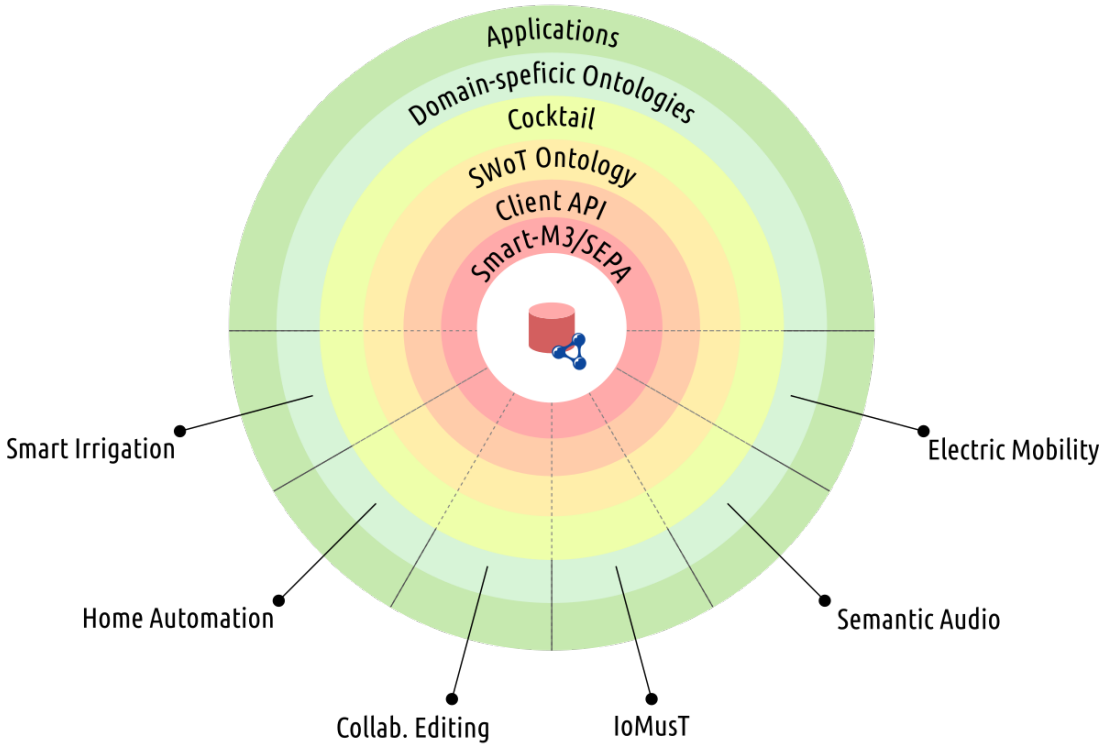


Figure 7.1: Smart-M3/SEPA Framework at a glance. The onion structure pivots on the semantic context broker. A set of APIs allows interacting with the broker to push Thing Descriptions according to the SWoT ontology (through the Cocktail libraries). Other domain-specific vocabularies permits the creation of applications pertaining different application domains.

All the layers of this onion structure are detailed in the following Sections.

7.1.1 Smart-M3/SEPA

SEPA (formerly Smart-M3) is the layer implementing the publish/subscribe paradigm on top of a standard SPARQL Endpoint. In all the SWoT applications developed exploiting Smart-M3, this component was represented by the SIB (i.e., SPS or OSGi SIB or pySIB).

In the new generation of the Smart-M3 interoperability platform, now known as SPARQL Event Processing Architecture, this central node is represented by either the Java or Python implementation of a SEPA. The development of all these semantic context brokers has been carried out throughout the whole duration of my PhD.

7.1.2 Smart-M3/SEPA API

Applications are composed by KPs, then by clients of the Smart-M3/SEPA platform. For this reason, a proper set of client-side libraries is needed. Smart-M3 APIs are available for Python2 and 3, Java and C. SEPA APIs are currently available for the following programming languages: Python2 and 3, Java, Ruby, Javascript, C. Python2 and 3, Ruby and Javascript implementation of the APIs have been developed during the 2nd PhD year.

7.1.3 SWoT Ontology

The semantic context broker hosts a set of RDF graphs, so it holds multiple sets of RDF triples. Triples represented without respecting the rules of a proper set of ontologies are meaningless. Beyond all the ontologies that can be exploited by the applications to address the need of a specific domain, there is one ontology that is the fundamental building block for SWoT applications: the SWoT ontology.

The SWoT ontology (described in Section 5.5.2) allows the semantic represention of the Thing Description of a Web Thing. This ontology (still under development) is based on the WoT Ontology presented by Serena et al. in [175] and exploits the concepts first introduced by Guinard and Trifa in [94] and now adopted also by the W3C Working and Interest Group on the WoT [20]. I contributed to the development of the ontology during the second and third years of my PhD.

SEPA integration pattern

The SWoT Ontology allows a new integration pattern with those described in Section 2.4: the **SEPA integration pattern** depicted in Fig. 7.2. In this integration pattern:

1. Web Things publish their Thing Descriptions on the SPARQL Event Processing Architecture through a SPARQL Update request;

2. subscribe to their interaction patterns through SPARQL subscriptions issued over WS according to the SPARQL 1.1 Secure Event Protocol.

3. Web Things can be discovered through SPARQL queries/subscriptions performed according to the SWoT Ontology.

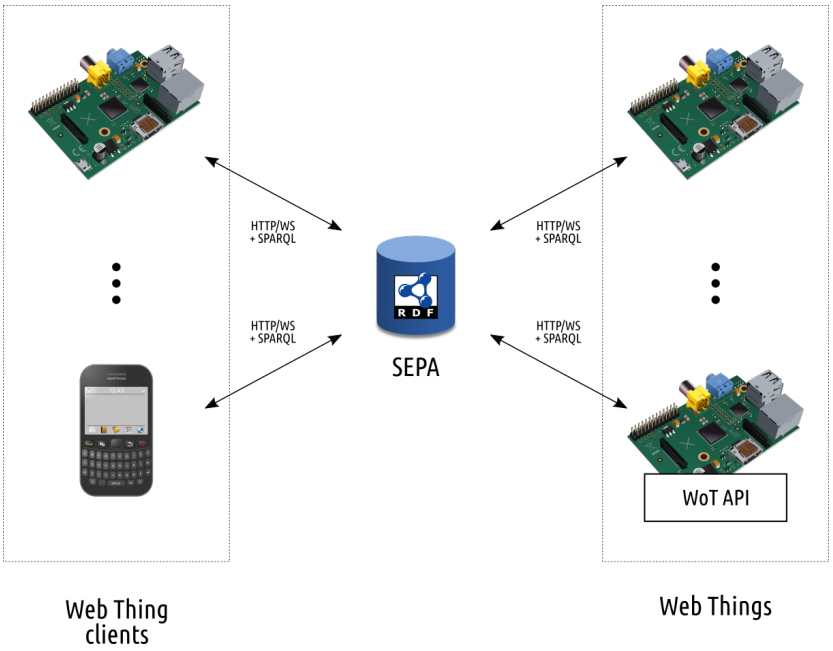


Figure 7.2: SEPA integration pattern

This integration pattern exploits the standard protocols HTTP and WebSocket to permit the interaction with/among Web Things. Still in terms of standards, the SPARQL Update and Query languages can be used to publish/retrieve Thing Descriptions. A partial deviation from the adoption of standards consists in the subscription mechanism, but it is motivated by the need for a subscription mechanism with a high granularity that is envisioned by the W3C WoT Working and Interest groups in the WoT Client API [226], but still not formalized.

It is worth analzing the point 2 of the previous list: the SEPA integration pattern in fact allows invoking actions through the semantic broker. A Web Thing willing to publish its Thing Description to a SEPA would issue a SPARQL Update like the following one:

```
1 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
3 INSERT DATA {
4   swot:FooThing rdf:type swot:Thing .
5   swot:FooThing wot:hasName "Foo Thing" .
6   swot:FooThing wot:hasInteractionPattern swot:FooAction .
7   swot:FooAction rdf:type swot:Action .
```

```
8   swot:FooAction wot:hasName "FooAction"
9 }
```

The Web Thing publishing this very simple Thing Description, exposes only one action (i.e., `swot:FooAction`). A second Web Thing willing to invoke this action through SEPA, is simply required to perform another update creating an instance of the class `swot:Action-Instance` linked to the action `swot:FooAction`:

```
1  PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
3  INSERT {
4    swot:FooActionInstance rdf:type swot:ActionInstance .
5    swot:FooActionInstance swot:hasRequestTimeStamp ?newATS .
6    swot:FooActionInstance swot:requestedBy swot:InvokingThing .
7    swot:FooAction swot:hasActionInstance swot:FooActionInstance .
8  }
9  WHERE {
10   BIND (NOW() AS ?newATS)
11 }
```

For this reason, in the envisioned integration pattern, a Web Thing subscribes to its interaction patterns: to be timely notified of new requests to be fulfilled. A very simple subscription achieving this scope is the following:

```
1  PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX swot:<http://wot.arces.unibo.it/ontology/web_of...>
3  SELECT ?thing ?actInstance {
4    ?actInstance rdf:type swot:ActionInstance .
5    ?actInstance swot:requestedBy ?thing .
6    swot:FooAction swot:hasActionInstance ?actInstance
7  }
8  WHERE {
9    BIND (NOW() AS ?newATS)
10 }
```

The interaction with the Thing or a Gateway/Cloud serving one or more Things may be mediated by SEPA or not. In the latter case, the SEPA is only used to discover Web Things through a powerful query language; once the Web Thing is discovered, the interaction takes place according to the Direct, Gateway or Cloud Integration Patterns.

7.1.4 Cocktail

Cocktail[1] is a set of APIs designed to speed up the development of SEPA applications. Through the Cocktail APIs it is possible to quickly generate a Semantic Application Profile describing the whole information flow of a SWoT application. The development of the Cocktail framework is not part of my PhD activities.

7.1.5 Domain-specific ontologies

Excluding the SWoT Ontology whose role is to represent the Thing Description of Web Things, the other ontologies are used in SWoT applications to map the context. Ontologies should then be specific to cover all the concepts considered relevant for a given scenario. In the following Chapters a set of application domains will be introduced.

7.1.6 Applications

Finally, the most external layer is represented by the applications. Fig. 7.1 shows how different SWoT applications can be structured in the same way according to the tools developed in this Smart-M3/SEPA Framework.

In the following Chapters, I present my research activity related to the application of the semantic architectures developed during the three PhD years over a set of different application domains:

- Chapter 8.1 presents the activity framed in the EU Research Project Arrowhead, regarding the application of the Smart-M3 interoperability platform in the Electro-Mobility area.
- Chapter 9 proposes a home automation application exploiting autonomous sensors and actuators to control an HVAC system. Energy management is then a central topic also in this case.

[1] `https://github.com/fr4ncidir/Web_Of_Things/`

- Chapter 10 presents the research activity carried out during my period as a visiting researcher at the Centre for Digital Music of the Queen Mary University of London in the areas of Semantic Audio and Internet of Musical Things.

7.1.7 Debugging tools

The Smart-M3/SEPA framework cannot be considered complete without two tools for the inspection of the knowledge base:

- **SEPA Dashboard**: a control panel developed using HTML5 + CSS3 + Javascript to perform SPARQL updates, queries and subscriptions. The SEPA dashboard also allows loading a JSAP file containing the parameters to interact with a SEPA instance, a list of saved namespaces, updates and queries. A screenshot of the dashboard is reported in Fig. 7.3.

- **Tarsier**: the general purpose visualizer described in Chapter 6, born to graphically inspect the knowledge base of a standard SPARQL Endpoint through a navigable three-dimensional space and the use of Semantic Planes.

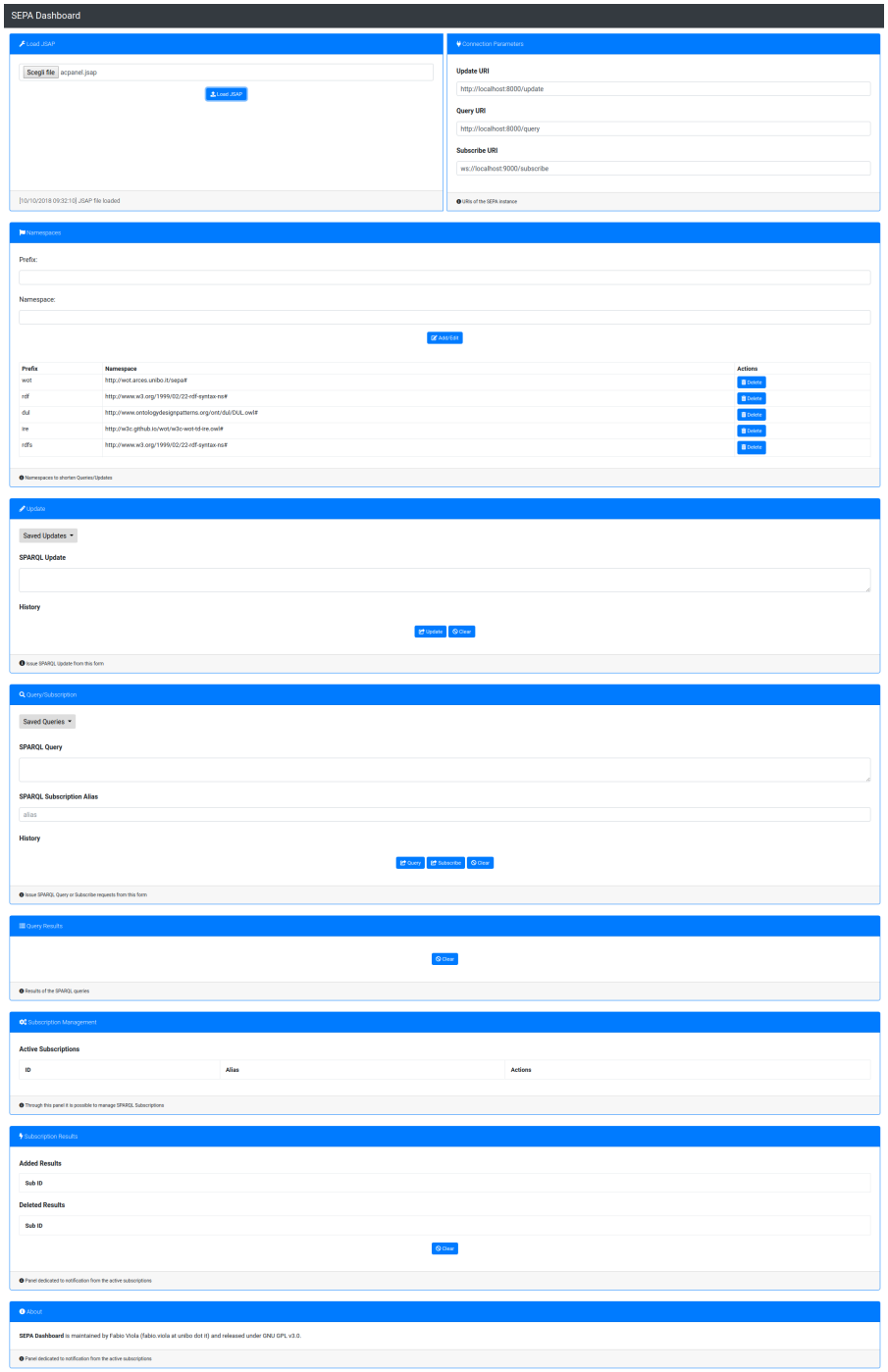


Figure 7.3: SEPA Dashboard

Chapter 8

Energy Management in Smart Cities

Contents

This Chapter summarizes the activity related to the application of the semantic technologies developed during the PhD (and described in Chapters 3 and 4) to the Electro-Mobility (EM) domain. This research has been supported in part by the Artemis JU Innovation Pilot Project Arrowhead, in part by Knowledge and Innovation Community of the European Institute of Innovation and Technology DIGITAL Activity entitled "*Planning Tool for EV Deployment and Related User Centric Services*" (Grant 14053), in part by ENIAC JU Project 2011-1/296131 entitled Energy to Smart Grid (E2SG), and in part by Artemis JU Project Internet of Energy (IoE). More in detail, my contributions in this area are related to the Arrowhead project. After a brief introduction to the Arrowhead project (Section 8.1), two main

research contributions framed in this context and related to Electro-Mobility are presented, both pivoting the Smart-M3 platform.

8.1 Arrowhead

The aim of the European project Arrowhead[1] was to address the technical and application issues associated with cooperative automation based on Service Oriented Architectures [227]. Arrowhead targeted five business domains: 1) Production (process and manufacturing); 2) Smart Buildings and infrastructures; 3) Electro mobility; 4) Energy production; 5) Virtual Markets of Energy. The biggest challenge of the Arrowhead project was to enable interoperability between systems relying on different technologies. More in detail, the Arrowhead project was aimed at finding an answer to the following questions:

1. How does a service provider make its services known to potential consumers?
2. How does a service consumer discover services it wants to consume?
3. How does a service provider determine if a consumer should be authorized or not?
4. How to orchestrate system of systems (where systems can be both producers and consumers)?

The answer to these questions is the Arrowhead Framework. The framework provides a common solution for core functionalities pertaining Information Infrastructure, Systems Management and Information Assurance. It also includes design patterns, documentation templates and guidelines for developers to develop Arrowhead-compliant services. An overview of the framework is proposed by Fig. 8.1.

The frameowrk includes a set of Core Services, among which it is worth mentioning:

- **Discovery** – deals with the registration of services in the Arrowhead ecosystem and provides discovery functionalities to the consumers;
- **Authorization** – responsible for the authentication of service consumers;
- **Orchestration** – provides the ability to orchestrate different services;
- **System Status** – manages the overall infrastructure.

[1] `http://www.arrowhead.eu/`

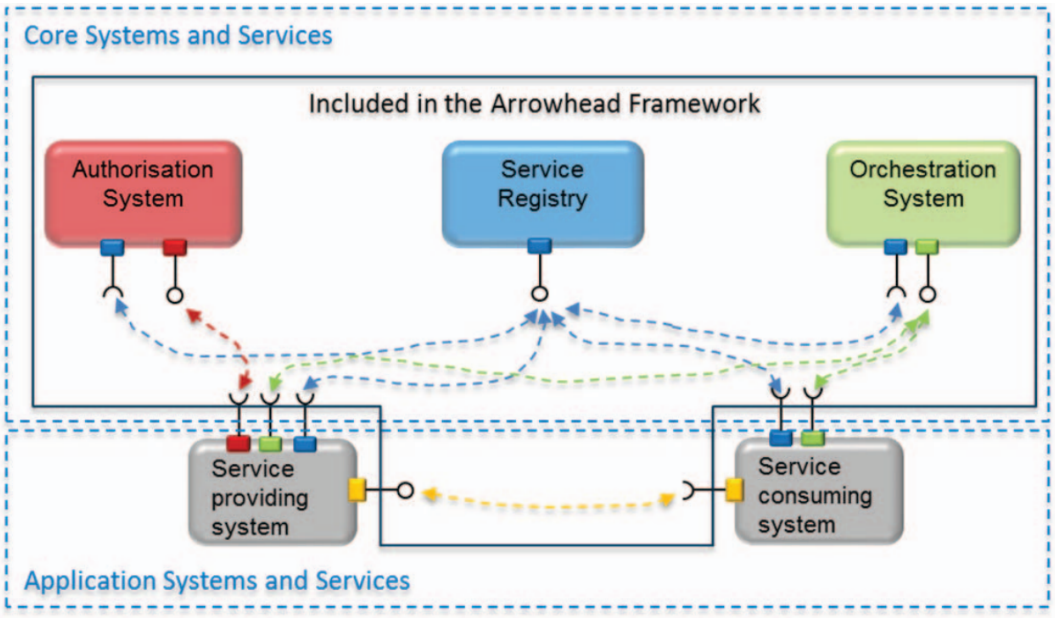


Figure 8.1: Arrowhead Framework overview [227]

The Arrowhead Framework has then been utilized to carry out mainly two research activities during my PhD. The first[2] is related to simulation of a fast recharge infastructure for rural areas [103]. The second is instead related to a wider interdisciplinary project for the co-simulation tool for traffic and power network for electro mobility [101]. This second scenario[3] is characterized by an ecosystem of services operating over two interoperable communication and a service management architectures.

8.2 Fast recharge infrastructure for rural areas through the Arrowhead Framework

The Electro-Mobility (EM) is a novel research area focused on ecosystems where electric vehicles recharge their batteries from a network of charging stations connected to the power grid or powered through renewable sources (e.g., solar panels). EM involves a radical change from the current mobility model based on fossil fuel combustion, with an expected impact on society, economy, transportation and environment that determines major investments from

[2]© IEEE, Reprinted with permission, from Alfredo D'Elia, Fabio Viola, Federico Montori, Paolo Azzoni, Matteo Maiero. Electro Mobility automation through the Arrowhead Framework. Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE. 2016.

[3]© IEEE, Reprinted with permission, from Alfredo D'Elia, Fabio Viola, Federico Montori, Marco Di Felice, Luca Bedogni, Luciano Bononi, Alberto Borghetti, Paolo Azzoni, Paolo Bellavista, Daniele Tarchi, Randolf Mock, Tullio Salmon Cinotti. Impact of Interdisciplinary Research on Planning, Running, and Managing Electromobility as a Smart Grid Extension. IEEE Access. Nov. 2015.

governments and companies. Not only EM requires the construction and deployment of a distributed charging infrastructure, but there's also to consider the problem of user acceptance that can be faced through a trusted network of surrounding interoperable services. One of the contributions of my PhD research in the area of the Electro-Mobility, framed in the Arrowhead project, is the design and development of a solution for EM automation based on a service-oriented, IoT and cloud-centric ecosystem of charging stations. The proposed solution was then evaluated against a simulated use case of a fast recharging infrastructure for a rural area.

The objective of this research is to demonstrate how IoT and cloud technologies could help the integration and automation of the EM scenario. This can be achieved through a service-oriented management system based on a Device to Cloud (D2C) approach that controls the data flow from Electric Vehicle Supply Equipment (EVSE) to end user or third party services and vice versa. The main contributions provided by this research be summarized as:

- secure data flow from and to the EVSEs with low bandwidth usage;
- semantic cloud storage for a scalable access to big data with a high level of abstraction;
- a service oriented architecture that simplifies and rationalizes the full automation of the Electro-Mobility scenario;
- network of trusted public services to provide specific information from the cloud to the end users or other services (i.e., *publishing services*);
- a network of trusted public services to perform actions on the recharge infrastructure (e.g., supply energy to an authenticated users), named *control services*.

This research was carried out in collaboration with Eurotech and with the support of Bitron.

8.2.1 From charging station to cloud

The proposed solution is based on the Eclipse Kura[4] framework, an open source tool aimed at providing a standard solution for the easy deployment and configuration of a high number of embedded systems on the field. Kura offers an OSGi-based container for M2M applications running in service gateways and is a programming environment that wraps the complexity of low-level device management with high level constructs. In this way, low level calls are translated to services, simplifying and speeding-up the software development. The charging

[4] `http://www.eclipse.org/kura/`

stations exploit Kura services for a pervasive integration with the cloud platform. Kura runs on a control unit placed inside the charging station that, acting as a multi-service gateway, provides:

- a hardware abstraction layer that simplifies the business logic development on the edge;
- wide support for data collection from the field;
- edge computing services for local data processing;
- efficient and secure MQTT-based [228, 140] cloud client;
- remote management of the charging stations.

The integration between the charging stations and the cloud platform is realized through a set of Kura bundles intended to simplify data collection, remote monitoring and service provisioning. A specific Kura bundle exposes a cloud service that simplifies the communication of the charging stations with the cloud platform. The bundles rely on the MQTT protocol for an efficient implementation of the previous functionalities.

MQTT is a publish-subscribe broker-centric protocol [228, 140] highly diffused in M2M applications where is both efficient and easy to integrate. This protocol provides both transport security and reliability. Differently from the already mentioned Smart-M3 platform, the publish-subscribe paradigm implemented by MQTT is topic-based and not content-based. Examples of the topics related to data collected from the charging stations are:

```
acme/123456/chargingStation123/chargingprocess/status
acme/123456/chargingStation123/chargingprocess/power
acme/123456/chargingStation123/chargingprocess/level
```

8.2.2 The cloud platform

Eurotech Everyware Cloud (EC) is a M2M/IoT integration platform adopted to simplify managing charging stations and collecting data through a set of cloud services. In terms of services, it is responsible for the EM Management Service that is published on the Arrowhead Framework. Among the functionalities provided by EC, it is worth mentioning:

- remote control of Kura instances running on the charging stations;
- collection of the information related to the charging stations and to the charging processes;

- cooperation with the EM Booking Service to manage booking of the recharges;
- data analytics;
- REST API to access the acquired data.

The architecture of EC, depicted in Fig. 8.2, shows that the M2M protocol (i.e., MQTT) is exploited by all the components. The charging stations, via their MQTT clients, may subscribe to a given topic (i.e., a recharges booking list) and receive a notification whenever a message with the same topic is published into the cloud. The Rule Engine is instead based on SQL: it is responsible for processing incoming data. Statistical rules are applied over the data in real-time; examples of actions generated by the rules include sending an e-mail, an SMS, a Twitter notification, generating a field protocol publish event, or issuing a REST API call. To ease the management of high amounts of data, the EC platform adopts a non-relational database for data storage.

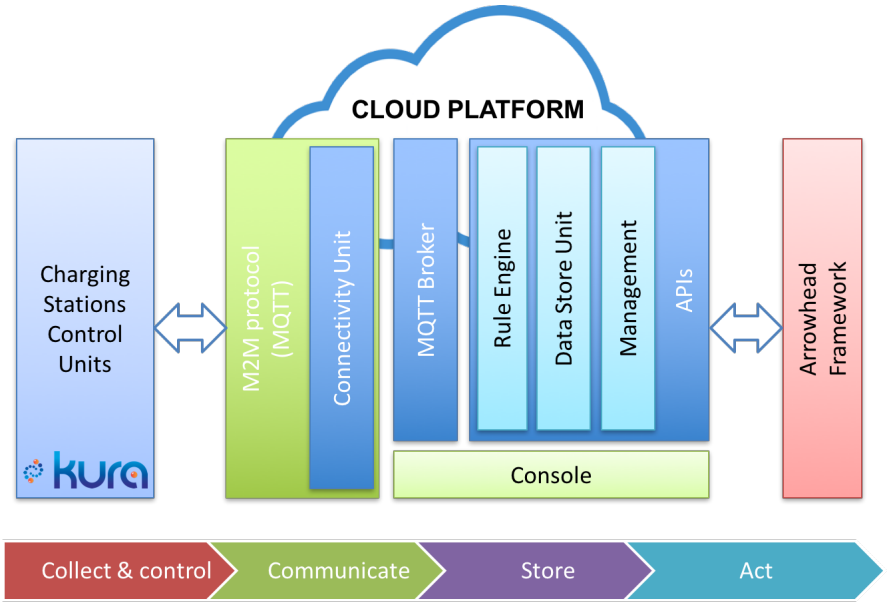


Figure 8.2: Software architecture of the Eurotech Everyware Cloud platform [103]

8.2.3 From cloud to EM Services

The EC Service Abstraction (ECSA), depicted in Fig. 8.3, is a layer interacting with Kura to completely hide the complexity of both the charging stations and the cloud infrastructure. This service abstraction is used to implement one of the services published in the Arrowhead Framework: the Electro-Mobility Management System (EMMS). The ECSA introduces two

types of MQTT topics: *publish* and *control*. The first is used by a charging station to publish data in the cloud, while the second is used by the EM application or by the cloud platform to send data or commands to the charging stations. Control topics can be further classified in two categories: those representing a control channel to a specific charging station, and those related to a control channel to all the charging stations belonging to an account. Each information collected from the charging stations must be described following the ECSA data model and a specific data acquisition service to collect information must be defined.

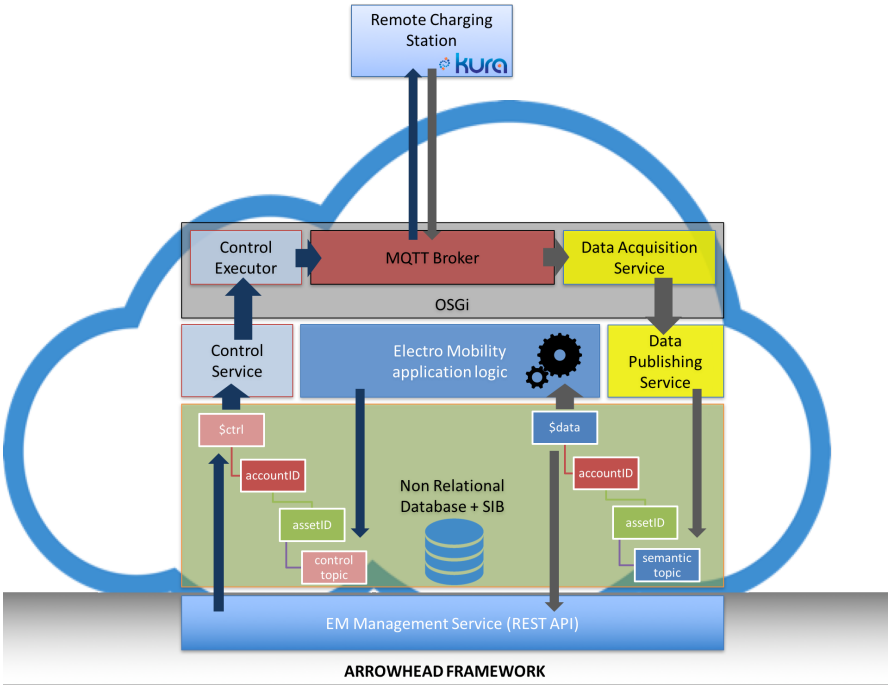


Figure 8.3: The EC Service Abstraction [103]

In the proposed architecture, several data acquisition services were introduced to monitor charging stations: the status, the charging progress, its power, its battery level, and others. Every time these services receive new data, they send it to the data publishing service which, according to the application logic, elaborates aggregates and finally publishes the data to the corresponding publishing topic. Moreover, a set of control executors were implemented to perform the retroaction activities required by the remote management of the charging stations: authentication confirmation, start/stop a recharge and go offline.

The EMMS completely hides the complexity of this structure by exposing a REST API with all the functionalities offered by the ECSA. This REST API is exposed as a simple Arrowhead Service registered and published in the Arrowhead Framework.

8.2.4 Simulated use case: fast recharge in a rural area

The proposed approach has been evaluated through a simulated scenario related to a public charging station (i.e., an EVSE) installed in a rural area, allowing only a limited number of recharges throughout a single day. The EVSE has two energy sources, both connected to a local energy storage system: the power grid and the Photo-Voltaic (PV) panels. A recharge process may take place at two different speeds:

- **fast recharge** (50kW), if the local storage is not empty;
- **slow recharge** (3.3kW), if the storage is exhausted.

In the latter case, the vehicle is recharged directly from the power grid. The EVSE owner may recharge his/her local using solar energy (for free) or using the energy provided by the grid. The integration of this EVSE in an ICT application allows employing a smart policy to recharge the storage: this policy takes into account the number of recharge reservations and the weather conditions. These data is provided by proper Arrowhead services. Without an automated system exploiting these information, the only two possible policies would be 1) always recharge using all the available sources (i.e., sun and grid) or 2) never recharge from the grid. From now on, I will refer to these policies by naming them respectively *always* and *never*, while the smart policy exploiting Arrowhead services will be referred to as *smart*.

This scenario was implemented as a fully configurable Python test suite simulating an EVSE and its local storage. To simulate the effects of cloudiness on the quality of service and on costs, a proper weather forecast service was included in the simulator. The test suite also includes a vehicle generator that, according to a chosen probability distribution, simulates the arrival of new vehicles with a random state of charge. Another module is responsible for plotting the final charts. The test suite works as an event based simulator calculating, for each simulated second, the energy balance of the local storage between sources (i.e., PV and power grid) and sinks (i.e., vehicles).

Fig. 8.4 reports the results of the simulation of three days with three different arrival frequencies when the policy is set to "Always". With only one vehicle, the amount of energy in the local storage quickly reaches the maximum and this is mainly due to the use of the power grid (i.e., there is no solar energy available in the first six hours of the day). With five vehicles per day, the level of charge is always close to the maximum. With nine, the storage is never completely exhausted. What happen if the policy is set to "Never"? It is easy to get the storage empty (Fig. 8.5), both with five or nine vehicles. With only one vehicle, the charge value returns to its maximum value, but the recharge takes a very long time. Why is

it so important to avoid having the storage empty? Because the unpredicted arrival of a new EV would require charging it from the grid that is both expensive and slow. On the other hand, also recharging the storage always from the grid is expensive. So a good compromise can be achieved only using a smart policy (see Fig. 8.6: in this case, the amount of energy available in the local storage at the arrival time of every vehicle is then enough to provide a fast recharge requiring a lower amount of energy from the grid (i.e. smaller costs) with respect to the policy "Always".

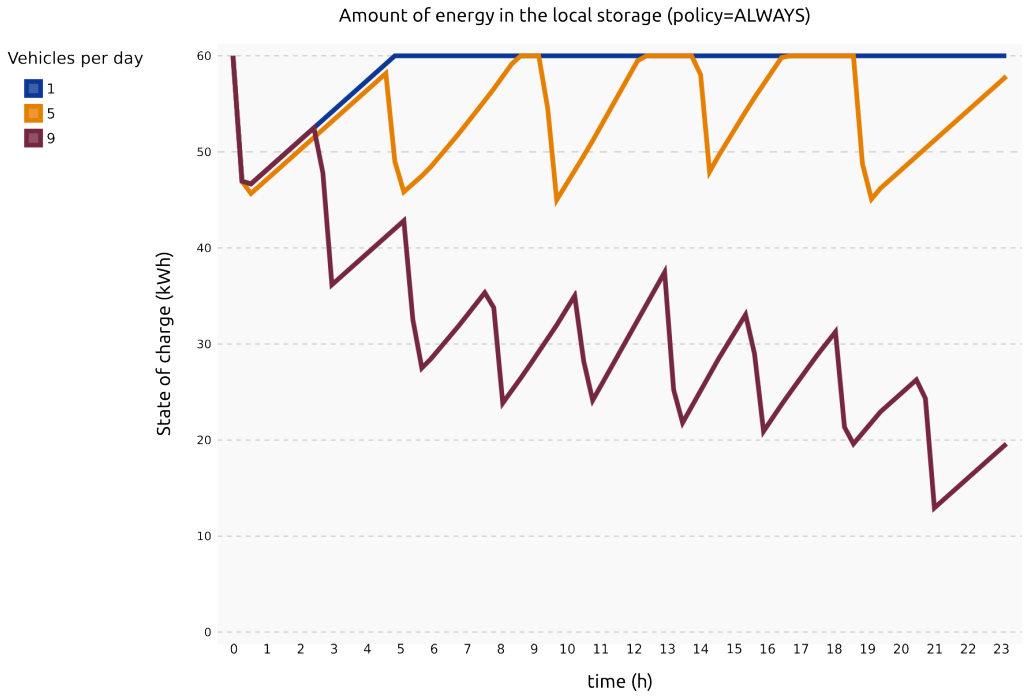


Figure 8.4: amount of energy in the local storage (simulation with policy "Always" [103]

The different slopes of the sections of the curves during the recharge of the storage depend on the variation of the intensity of the sun light during the day. In our setup, the power gathered from the surface of the solar panels, in good weather conditions, ranged from fractions of kW in the first hours of the day to 3 kW at midday. Different weather conditions affect the recharge speed of the EVSE's local storage, but these changes can be foreseen and managed through the weather forecast service (e.g., in cloudy days the smart policy will more likely rely on the power grid in addition to the PV). Fig. 8.7 confirms that the policy smart is the best in terms of energy consumption from the grid.

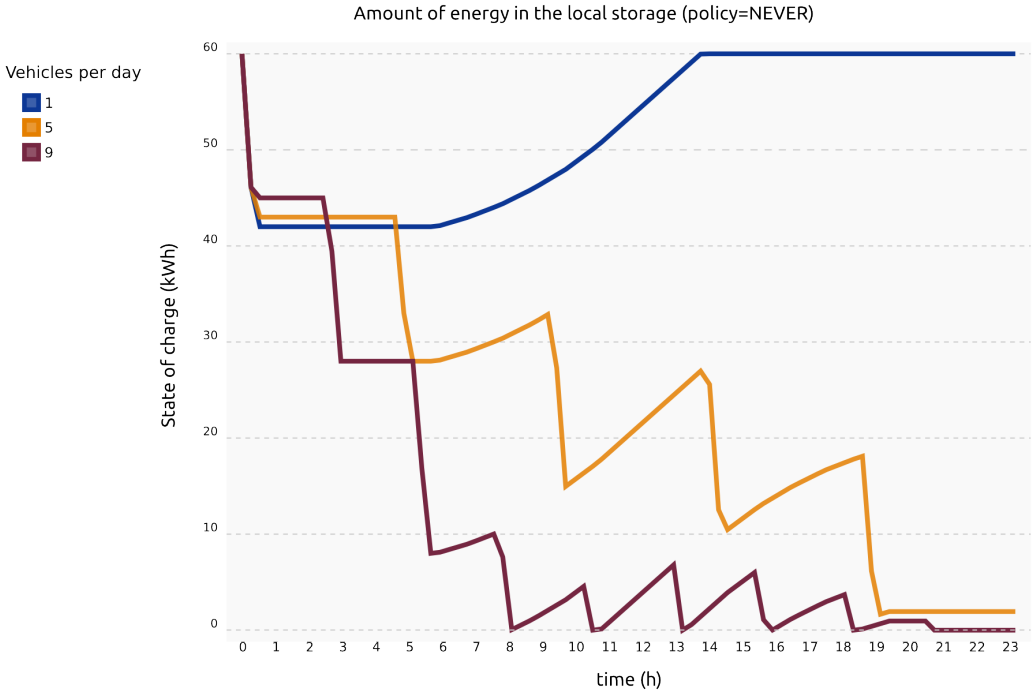


Figure 8.5: amount of energy in the local storage (simulation with policy "Never" [103]

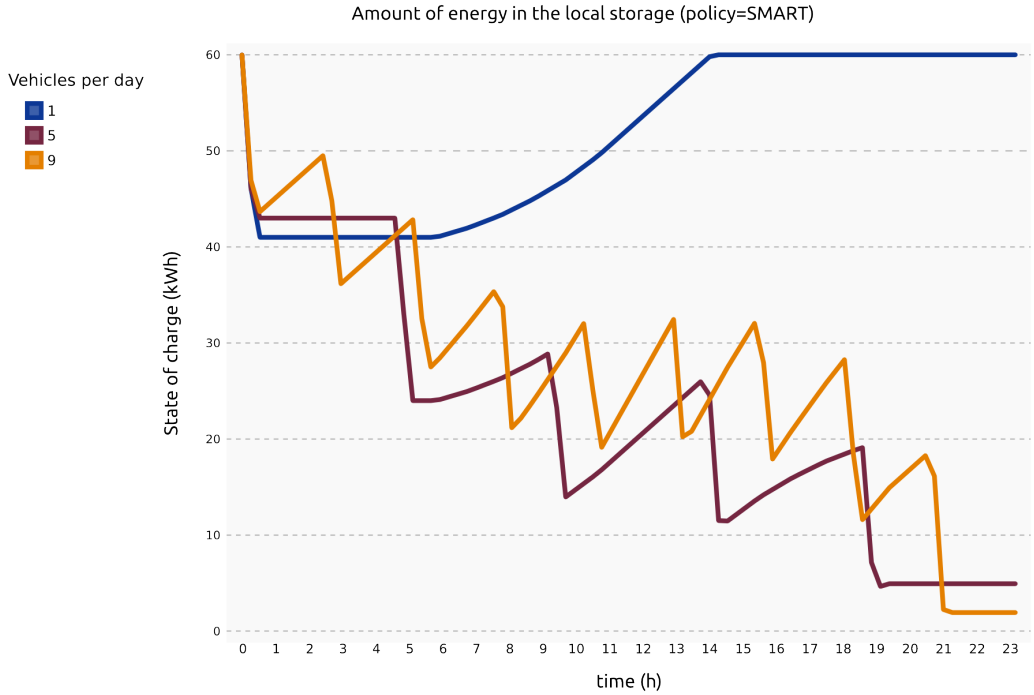


Figure 8.6: amount of energy in the local storage (simulation with "Smart" [103]

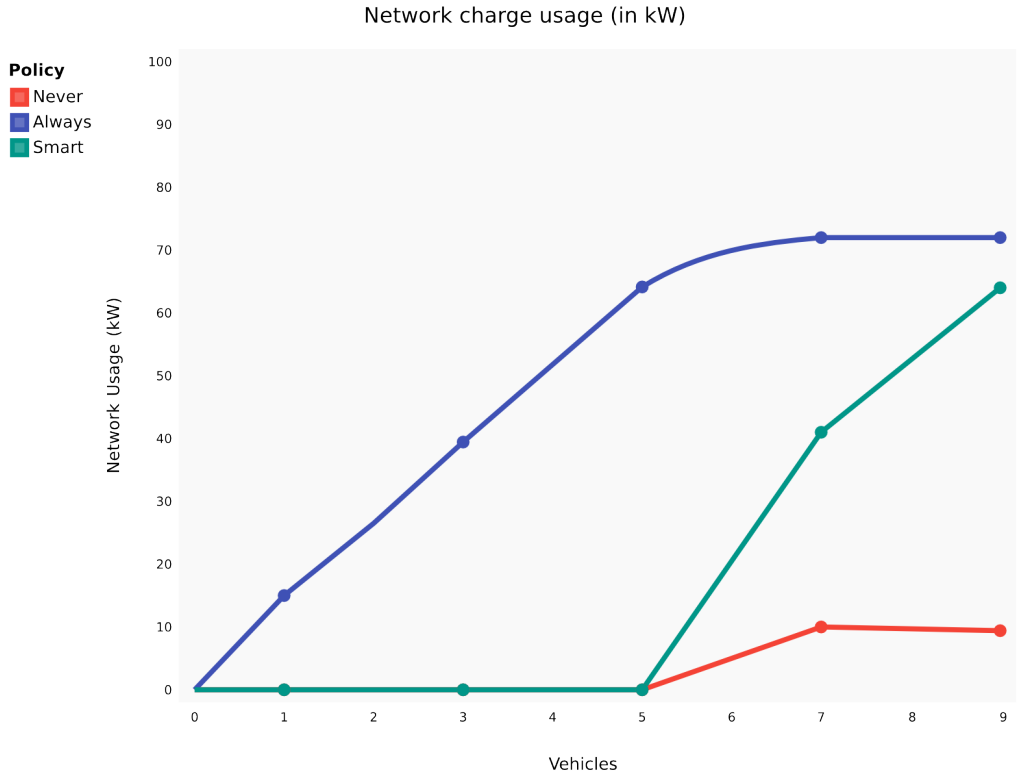


Figure 8.7: amount of energy in the local storage (simulation with "Smart" [103]

8.3 Interdisciplinary research in the Electro-Mobility

In the last decade, we assisted to the convergence of several independent research areas into what is commonly referred to as Smart Grid (SG), a large scale energy management infrastructure involving all actors related to energy production, distribution, storage and usage as well as the surrounding service ecosystems and information management infrastructures.

This Section presents the results of a study aimed at assessing the advantages of a multi-domain approach over a single-domain one in the domain of the SGs. The multi-domain approach allows to face more complex challenges and study more complex problems. For example: only with an integrated scenario we could answer to the questions "*how much a charging spot reservation service affects the traffic?*" or "*does a reservation made during energy consumption peaks require an energy amount affordable for the power grid?*". We could answer these questions only considering scenarios where SGs communicates with the recharging infrastructure and with electric vehicles. Then, this Section presents a set of concepts, software artifacts and simulation environments belonging to different fields but developed according to a multi-domain scenario. Collaboration with industrial partners like

ENEL, Siemens, Centro Ricerche Fiat (CRF) and Eurotech helped us to be grounded on the specific industrial needs and requirements for a realistic simulation environment.

8.3.1 The platform at a glance

Fig. 8.8 reports a schematization of the implemented test suite where it is possible to notice three cooperating frameworks:

- the information management and communication framework;
- the Electro Mobility and power network co-simulation framework;
- the service layer.

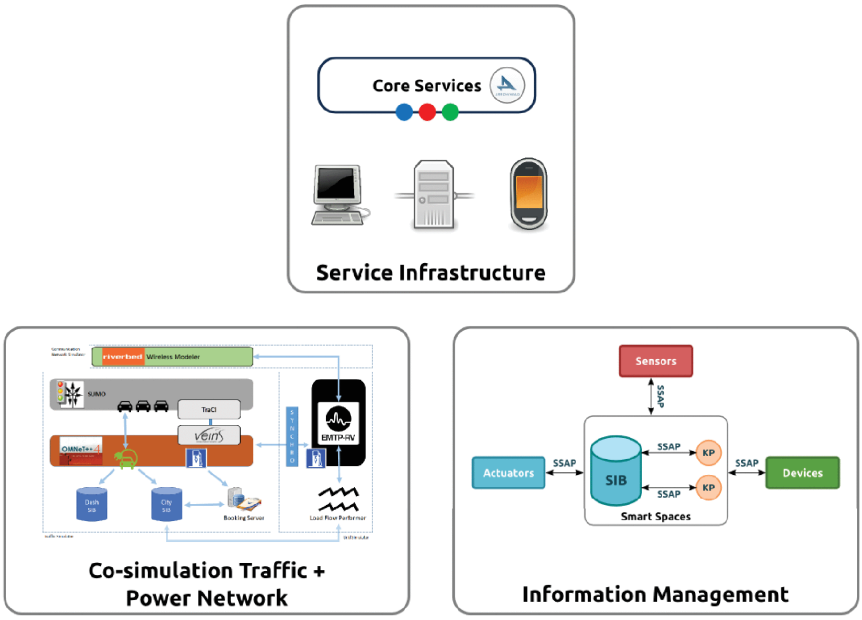


Figure 8.8: The implemented infrastructure at a glance [101]

Among the results that could not have been achieved without the co-operation of inter-disciplinary teams, the most relevant are those related to the interaction between the electric vehicles, the power network and the service infrastructure. In fact, the proposed infrastructure allowed us to:

- Assess the impact of Electro-Mobility on the power network;
- Consider the constraints imposed by the grid to the mobile services (e.g., variable price of reservations due to the estimated available power in the reserved time slot);

- Analyze different power network configurations and sizing in relation to Electro-Mobility and renewable sources (i.e., pre-deployment analysis);
- Assess the impact on the traffic of mobile services simplifying the discovery and usage of the recharging infrastructure (e.g., reservation services and route planners);
- Assess the usefulness of Vehicle to Grid and local storage facilities in the application of regulation services or other countermeasures to prevent power grid congestion during request peaks.

My research contribution in this multi-disciplinary research project carried out by different teams is related to the development of the information management and communication framework and on the development of the services. Therefore, only details about this part will be reported in the following Sections. The reader could refer to [101] for further information.

8.3.2 Information management and communication framework

In order to transform the power grid into a smart grid, a proper technological infrastructure must be developed. The infrastructure must face the heterogeneity of the scenario and be scaleable to support the growth of the connected entities. It should also be extensible to support new development without disrupting changes. Message-oriented Middlewares are emerging as a reasonable choice for smart grids, due to high scalability, loose coupling between entities, ability to provide synchronous and asynchronous communication, and support for differentiated priority levels. Among the exitsting MOMs, we leveraged the interoperability platform Smart-M3 (see Chapter 4.3). Smart-M3 acts as an interoperability enabler, since very different nodes are able to communicate through the broker by means of messages represented according to a shared set of ontologies. Moreover, as previously mentioned, Smart-M3 implements the publish-subscribe paradigm, a base feature for reactive systems. Lastly, thanks to Smart-M3, the information model is easily extensible, so novel software components can be developed without disrupting the existing code. In the presented infrastructure Smart-M3 is used by all the interacting entities (i.e., vehicular simulator, power network simulator and services).

8.3.3 Service platform

One of the challenges to be faced by Electro-Mobility is the diffidence of new users. In this context, the role of mobile, in-vehicle and context-aware services is to simplify the transition to the new infrastructure by both reducing the impact on the end users and encourage the

transition to EVs. We developed a general framework for the deployment of interoperable mobile services, providing all the main functionalities requested by EV drivers: profiling, route planning and charging reservation. The latter is the one on which I focused during the project.

It is worth mentioning that all the services in this infrastructure rely on a common ontology (i.e., the powergrid ontology [99]) shared among all the actors of the scenario. This ontology allows defining: 1) all the physical entities (e.g., EVs, EVSEs, Connectors); 2) abstract entities (e.g. Data, ChargeProfile); 3) service specific terminology (e.g. ChargeRequest, ChargeResponse, Reservation, and so on). All data produced by the actors is then collected through Smart-M3 SIBs.

The need for a reservation service is motivated by the longer recharge times needed for EVs, compared to the time needed by refueling. When the EV drivers need to recharge, instead of driving directly to the closest EVSE, they look for an available charging opportunity in a target area, and during a preferred time frame. The first implementation of the Reservation Service [98, 229] was then further modified and adapted to achieve:

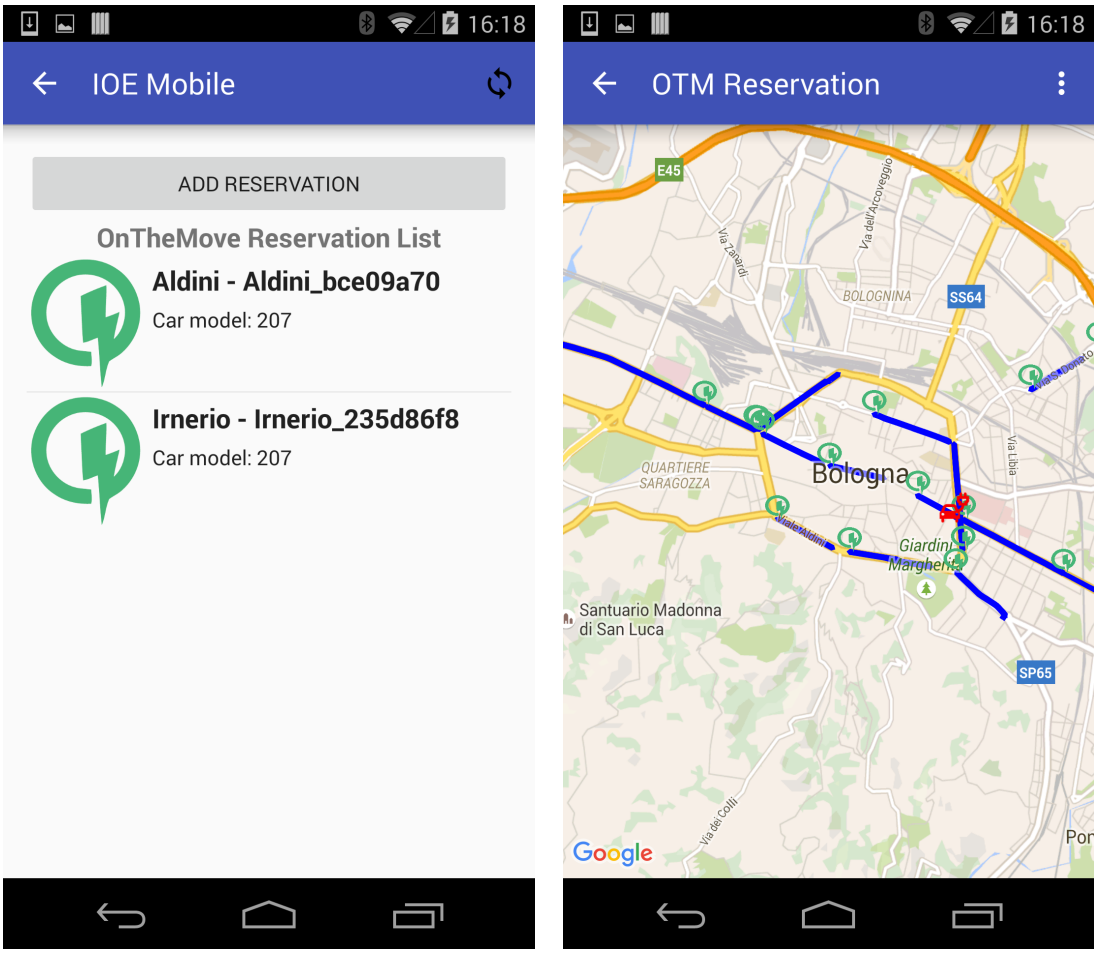
1. Integration in the Arrowhead ecosystem through registration to the Core Services infrastructure;

2. Support of an additional (and still unpublished) scenario: the recharge **on-the-move** (OTM).

8.3.4 Discussion

The integration between Electro-Mobility and the power distribution network requires careful, simulation-based, pre-deployment analysis of the recharging infrastructure and of the associated services. The goodness of the resulting ecosystem is strictly related to the fulfillment of requirements related to user satisfaction, energy efficiency, communication and power network qualities. All these requirements can be satisfied only considering a set of complex factors like the EV penetration, the associated travel, traffic and recharging patterns, power, density and distribution of recharging spots, renewable energy sources, energy storage units, user services, user behaviour prediction, control capabilities of the power delivered by the power network feeders and business models of the entire value chain. This complex and heterogeneous scenario requires large multidisciplinary teams that join forces towards this sustainable development.

With this research, we focused on information management and communication, co-simulation frameworks and services for the smart grids. To achieve the scope, several de-



(a) Existing OTM reservations **(b)** Overview of the reserved path

Figure 8.9: The mobile app during OTM recharge reservations

partments from different research areas worked together. The purpose of the project was to enable the Smart Grid to leverage on research results from the areas of 1) Software Defined Networks; 2) Semantic Interoperability; 3) Big Data management and cloud-based services.

The SG was considered only from the point of view of its interplay with the EM. Nevertheless, this case study demonstrated the emerging need for interdisciplinary infrastructures and approaches in research.

8.4 Conclusion

This Chapter presented two research activities belonging to the area of Electro-Mobility where semantic technologies (and in particular the Smart-M3 platform on which my research is focused) have been applied.

In the first activity (carried out in collaboration with Eurotech and with the support of Bitron) we demonstrated how a semantic service infrastructure exploiting information about reservations can be exploited to optimize the service provided by recharge stations. This problem in particular has been studied with respect to rural environments characterized by a low number of daily recharge requests.

The second activity presented in this Chapter was related to a highly interdepartimental study involving Siemens AG, Eurotech, several departments of the University of Bologna (i.e., the Advanced Research Center on Electronic Systems, the Department of Computer Science and Engineering and the Department of Electrical, Electronic and Information Engineering). In fact, Electro-Mobility is a young research area that is intrinsically inter-disciplinary. As demonstrated by the activities described in this Chapter (and better detailed in the related publications [101, 230, 102]), it involves researchers belonging to the areas of information science (and in particular experts of semantic interoperability, cloud computing, service oriented architectures), to the automotive and infrastructure industry, experts of mobile systems and electrical engineers as well as researcher from embedded systems industry in line with the emerging Internet of Things vision. The research activity carried out on the Arrowhead project demonstrated the power of the semantic interoperability platform Smart-M3 as a timely communication infrastructure shared by all the heterogeneous components of the Electro-Mobility ecosystem.

Chapter 9

Energy management in smart homes

Contents

The Internet of Things, and of course the Semantic Web of Things, strongly rely on Wireless Sensor and Actuator Networks (WSANs) to sense the environment and react to changes in the conditions according to user-specified policies. Pervasive interconnected electronic devices, in the so-called smart spaces, cooperate to simplify human life and enhance the perceived comfort, without any particular need for maintenance or direct control.

One of the challenges in this research area is related to the power requirements of constrained devices: it is not uncommon, for both the sensing and actuating devices, to be powered by batteries. In fact, sensors and actuators are often installed also in places where an energy plug is not available. Batteries provide a lifespan proportional to their capacity, and then also to their cost and volume, but proper policies allow making their amount of charge last longer. Moreover, harvesting is increasingly being employed to harness energy from the environment [231] and extend the battery life.

In the research work presented in this Chapter[1] , we proposed the first prototype of an

[1] © IEEE, Reprinted with permission, from Alfredo D'Elia, Luca Perilli, Fabio Viola, Luca Roffia, Francesco Antoniazzi, Roberto Canegallo, Tullio Salmon Cinotti. A self-powered WSAN for energy efficient heat distribution. Sensors Applications Symposium (SAS). 2016.

IoT-ready network of autonomous WSAN consisting of low power sensors and actuators accessible through an IoT gateway. The WSAN was designed to solve the problem of controlling a radiator-based heat distribution through autonomous, unobtrusive thermovalves. The prototype is intended to become a platform for novel user-centric, automatic and energy efficient heat distribution systems. This activity has been carried out in collaboration with ST Microelectronics TR&D SPA[2]. My contributions in this research activity consists of the overall design of the system, development of the IoT gateway entirely founded on the Smart-M3 platform.

9.1 Scenario and system architecture

The use case driving this work is an environment pervaded by two types of autonomous nodes: sensing nodes and actuating nodes. Both the sensing and actuating nodes sense the temperature and communicate their state, while the actuating node also drives a valve and tunes the water flux in a typical radiator. The end user, through his/her smartphone, checks the temperature of the monitored environments and sets the target temperature. The app is developed for Android and a screenshot is visible in Fig. 9.1.

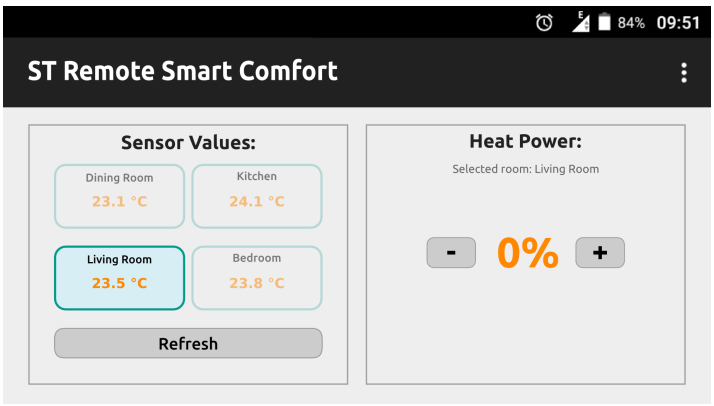


Figure 9.1: Screenshot of the mobile Android application [232]

The architecture is depicted in Fig. 9.2. All the WSAN nodes are connected through a DASH7 network to a node named **coordinator**. The latter is plugged to a home gateway connected to power supply and to the Internet. In the gateway an HTTP server and a message dispatcher based on the semantic interoperability platform Smart-M3 are running. The server is accessible from remote devices and uses the dispatcher module to convert the high level request into low level DASH7 messages for the WSAN coordinator. This translation allows

[2] https://www.st.com

to move from a verbose user-friendly representation to an efficient byte representation. The power management module tracks and performs a moderation on the user requests to the various nodes to deny, delay or merge them to preserve a fully operative WSAN where nodes never discharge completely.

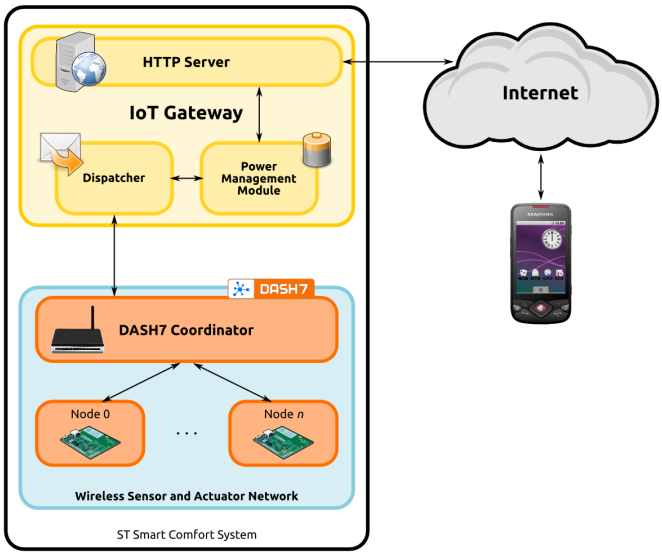


Figure 9.2: System architecture [232]

9.2 Sensor and actuator nodes with harvesting

The WSAN consists of autonomous smart nodes. All of them include:

- a temperature sensor;
- a low power microcontroller;
- a power management unit with transducers for collecting Photo-Voltaic (PV) or thermoelectric energy;
- a subGHz radio device for data communication.

The power management unit draws energy from the battery and the harvesters and supplies it to the node microcontroller, radio, sensors and actuators. If the harvested power exceeds the demand, the remaining energy is used to recharge the battery. The design goal is to have a positive energy balance (average harvested energy greater than average system node consumption) so that the battery does not need to be replaced.

To achieve this capability a characterization of the energy supplied by the harvesters transducers in different environmental conditions and the measurement of the nodes power consumption in different working conditions is required. The harvesters are based on off-the-shelf solar cells designed for indoor purposes (AM-1801[3]), or on a thermo-electric generator (TEG)[4].

The transducers are connected to a DC-DC buck-boost converter which transfers the harvested energy to the energy storage element. It also integrates an Maximum Power Point Tracking (MPPT) algorithm [233] that keeps the input voltage at the transducers optimum value in order to maximize power transfer efficiency. The converter is followed by a rechargeable battery required to store the surplus energy extracted from the sources and makes it available to the load when the harvested power is lower than system consumption.

The characterization of the AM-1801 based power unit in typical indoor lighting conditions (i.e., between 200 and 1000 lux) is shown in Fig. 9.3, while Fig 9.4 shows the current harvested by the TEG module installed on a hot pipe of a radiator. As a heat source is not usually available near a temperature sensor, a PV harvester is used with the sensing node. On the other side the TEG is used with the thermo-valve actuator because the heat source is available while it is not so obvious to have a light source of comparable output current near every radiator. The sensor and actuator nodes are shown respectively in Fig. 9.5 and Fig. 9.6.

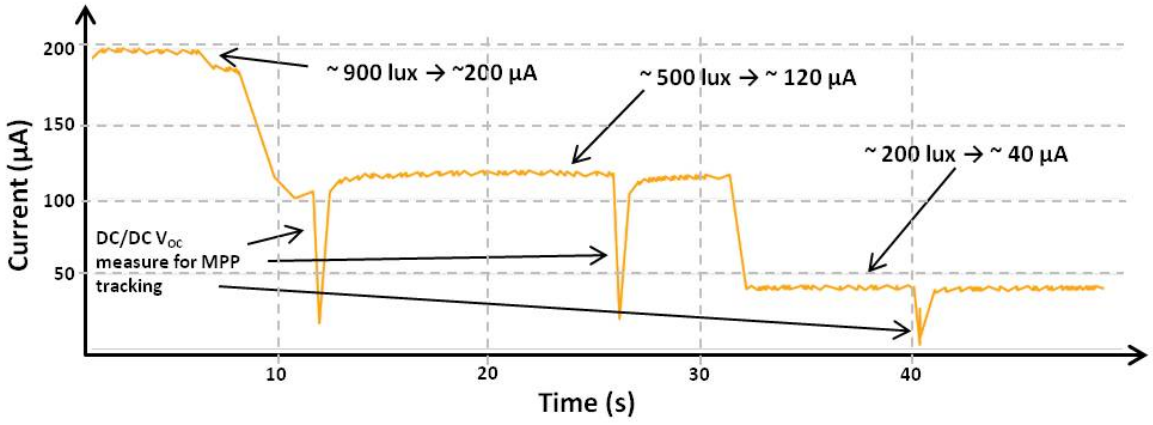


Figure 9.3: Harvested current from PV cells in different light conditions [232].

The sensing nodes include a temperature sensor, a microcontroller unit (MCU) and a low power radio. The microcontroller is the ultra low power STM32L1, based on an ARM Cortex-M3 core. It acquires the temperature data and implements the communication data

[3] http://docs-asia.electrocomponents.com/webdocs/0d10/0900766b80d10cf4.pdf
[4] http://www.micropelt.com/downloads/data-sheetthermogeneratorpackage.pdf

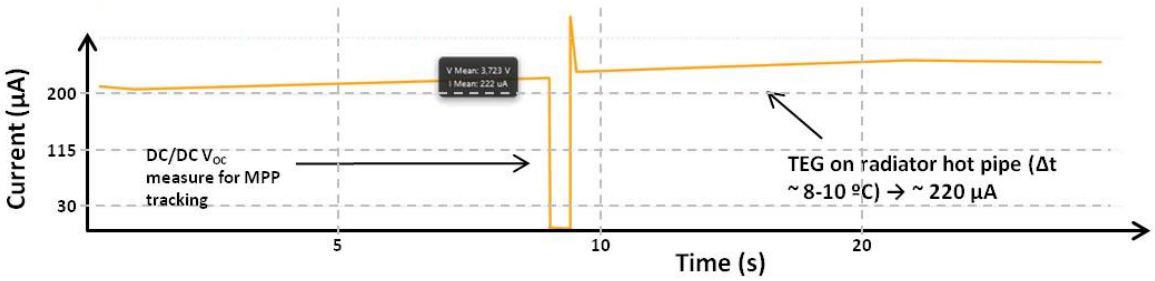


Figure 9.4: Harvested current from TEG [232].

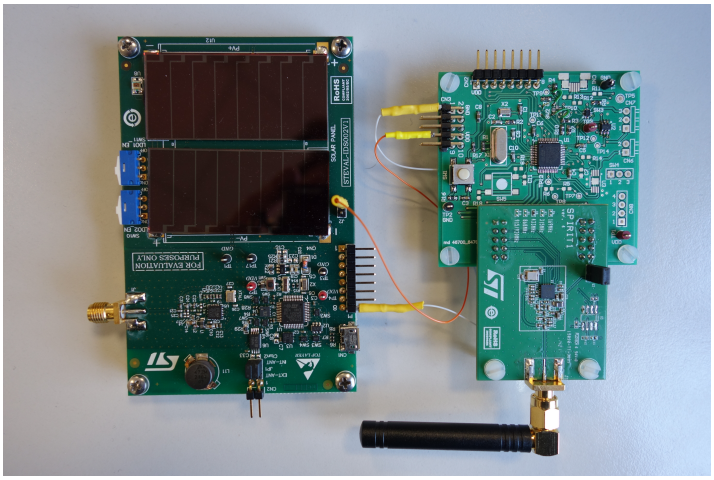


Figure 9.5: Sensor node with PV harvesting board [232]

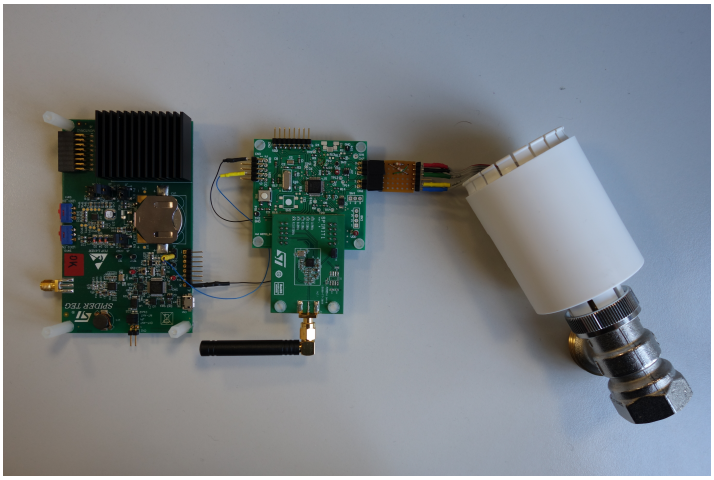


Figure 9.6: Actuator node with TEG hervesting board and valve connection [232]

link based on DASH7 standard. The radio device is the 433 MHz ST device SPIRIT1 ultra low power subGHz RF transceiver[5].

[5] http://www.st.com/web/en/resource/technical-/document/datasheet/dm00047607.pdf

The current consumption of the node in sleep mode is 5 μA, while it is 12 mA when the radio operates in receive mode. Modern thermo-valves include an electronic module powered by a battery that, through an electric motor, control the aperture of a valve regulating the heating flow according to a temperature target.

The designed prototype aims to 1) make the thermo-valve autonomous; 2) remove the need for manual control of individual radiators; 3) ensure unobtrusive installation (i.e., no wired connections); 4) enable its remote control via a web interface. To meet these requirements, the TEG based harvester supplies a WSAN node which includes a low voltage motor driver (DRV8830) controlling a brushed DC motor (RF-300EA) with a gear box for thermostatic radiator valve control. The energy consumption of the valve component (electronic and motor) is 20 mA during the 24 s of aperture and increases to 27 mA during the 26 s of closure. The different power consumption values take in account the inertia of the spring inside the valve during the two phases.

9.3 Communication protocol

The autonomous WSAN nodes designed in this project interact with the IoT Gateway over an ultra low power DASH7 wireless communication network. DASH7 [234] is an open source RFID standard that can reach a data rate of 200 kbps and an outdoor range up to 2 km. It is based on the ISO/IEC 18000-7 open air standard that defines the use of 433 MHz band for active RFID applications and can also be extended to non-RFID applications. Node discovery is supported by a beacon mode but this mode is not exploited in our prototype implementation. Even if the protocol is not IP compatible, it can support UDP packets through transport layer adaptation and this allows the development of UDP based standard application protocols such as CoAP [141]. DASH7 supports bursty, light and asynchronous communication. It is attractive in applications where very little power is available on the node-side, and at the same time external node wake-up is not supported.

The coordinator normally works in request-response mode and it expects the nodes to periodically scan the air. It synchronizes with the addressed node in open loop mode through an adjustable message scheduling technique.

After receiving a request from the IoT gateway, the DASH7 coordinator transmits a burst of short packets (BG packets or background frames) that simply notify the listening nodes that a request will be sent in the future with a specified delay. The burst will last T_{FLOOD} (flooding phase) and the payload of the BG packets is the (decreasing) time left before the scheduled request broadcast. The autonomous nodes periodically scan the air looking for

BG packets (standby phase), alternating scans and sleep periods. The scan period T_{SCAN} is programmable.

Nodes have to capture one and only one BG packet per burst and in order to do so it is enough to keep the scan period below T_{FLOOD} ($T_{SCAN} < T_{FLOOD}$). After detecting a BG packet, the node enters its sleep state until the request is planned to be received. During the standby phase the higher T_{SCAN}, the longer is the node sleep time, and the lower is, at the node side, the average standby current I_{SB} required to detect a request. The request latency is equal to T_{FLOOD} because the coordinator sends its request (FG packet) at the end of the flooding phase. Therefore a decrease in latency specification requires a decrease in T_{SCAN} and hence implies an increase in I_{SB}. Normally both sensors and actuators share the same T_{SCAN} value and the same standby consumption.

In our reference test case $T_{FLOOD} = 5s$, $T_{SCAN} = 4s$ and $I_{SB} = 35\mu A$ (measured). Fig 9.7 shows that node current consumption is $5\mu A$ in sleep state and $12mA$ in scan state.

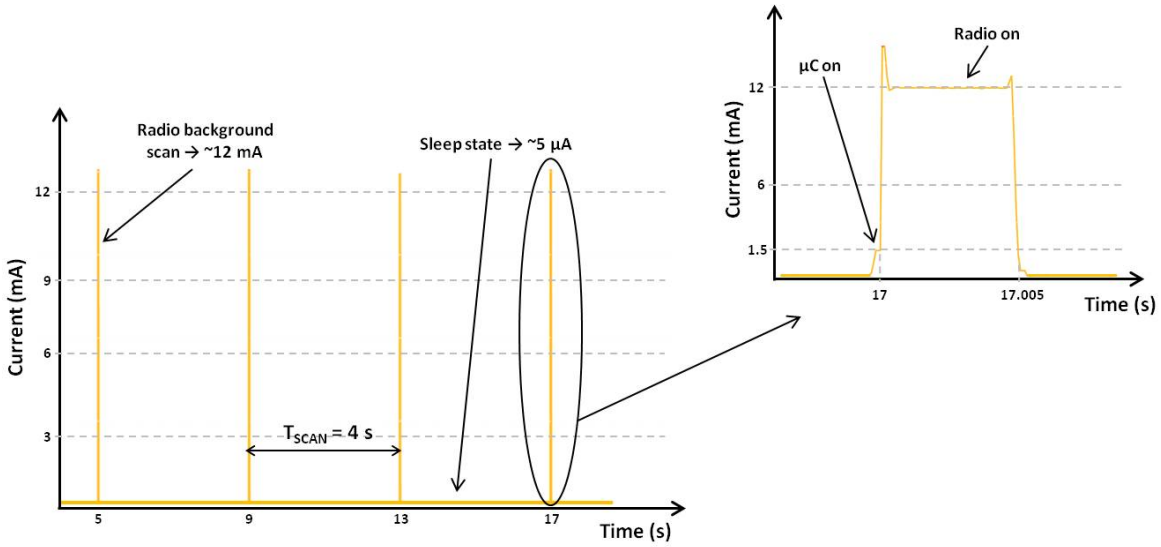


Figure 9.7: Energy consumption in stand-by [232].

Decreasing T_{SCAN} would approximately produce a proportional increase in I_{SB}. For example, with the proposed approach, should a latency lower than $3s$ be required, T_{SCAN} should go down to $2s$, and I_SB would rise up to $70\mu A$. The best tradeoff between I_{SB} and the request latency is application dependent and can be enforced through adjustable message scheduling.

Fig. 9.8 shows that from the DASH7 protocol point of view on top of scanning the air, a node has to go through three processing phases per request handling respectively the request announcement, the received request and the response. In turn, after sending its request

(ForeGround packet) the coordinator enters an FG scan phase waiting for the response from the addressed node. As in the background phase, also in this foreground phase sensor and actuator nodes share approximately the same request-response consumption. With a request per minute the average current needed by each node to entirely handle the DASH7 request-response phase rises up to $I_{REQ} = 200\mu A$ in the specific minute when the request-response phase occurs. On top of this protocol related energy demand, the energy required by the application must be added (e.g. to enforce the required actuating action).

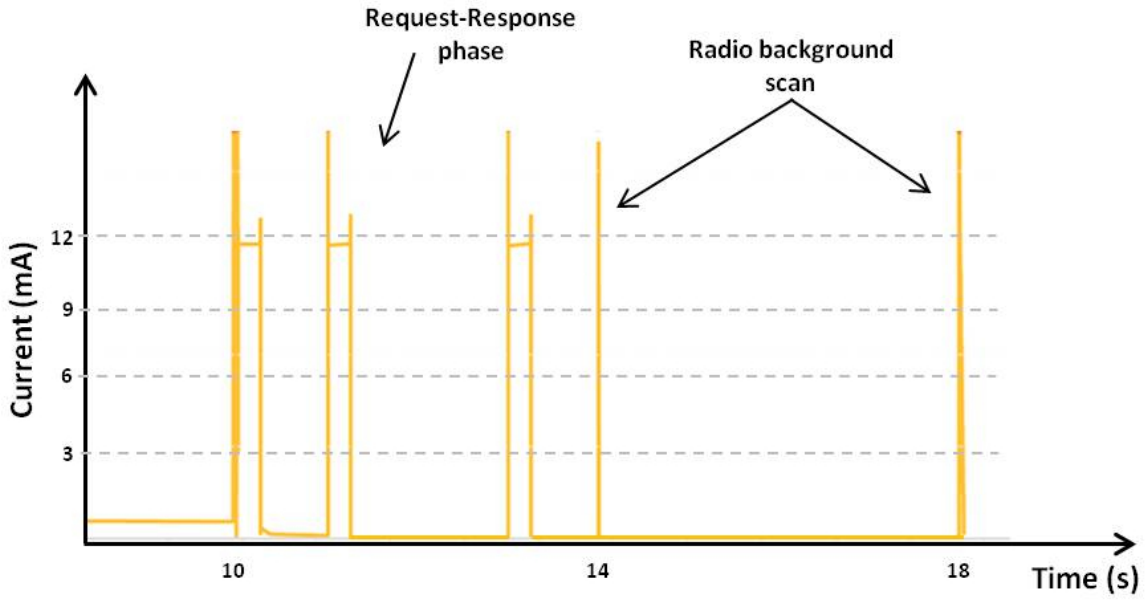


Figure 9.8: Energy consumption during a request [232].

9.4 IoT gateway software modules

The IoT Gateway is composed by three software modules interoperating by means of the Smart-M3 interoperability platform and a custom ontology. A first module, called HTTP Server represents the interface to the external world. Through the server the user sends commands and receives replies using proper high level REST calls. Every request is then analyzed by the Power Management Module (PMM), the second component of our IoT Gateway. This software module satisfies the need for an entity to manage the interactions with the underlying WSAN in order to meet the restrictions caused by the limited amount of energy available.

The PMM then, after a validation phase, decides whether the users request may be accepted or rejected due to a potential risk of overloading the WSAN. In fact, changes to the regulation of the heating system may require hours in order to manifest their effects on the

environment; in this situations the user may feel the need to act on the regulation again and again. So, the PMM tries to accomplish user requests as much as possible with the requirement of preserving the continuity of service. The third software module is then activated by the PMM to forward every request to the WSAN: this module is called Dispatcher. The Dispatcher is responsible for the translation of the received request to proper DASH7 messages. Every information coming from the WSAN and destined to the clients is then routed along the same path in the opposite direction.

9.5 Design considerations for energy efficiency

In order to operate with the right performance level and at the same time be autonomous, thus expanding battery lifetime, the system needs a careful optimization of energy usage. Due to the uncertain amount of energy that can be harvested, this optimization involves two major constraints: tasks (e.g., temperature acquisition, data transmission and actuation) need to be processed with the energy generated by the harvesters, and timing constraints must be satisfied in terms of throughput needed by the application. To ensure node self-sustainability the average energy harvested has to be greater than the average energy used by the node. The battery is only an energy buffer; all the charge supplied by the battery to the node for an operation has to be harvested and provided to the rechargeable battery thereafter, so that the following relation between the average energy E_{mHARV} harvested in a reference period and the average energy E_{mSYS} absorbed by the node is satisfied:

$$E_{mHARV} \geq E_{mSYS}$$

and the battery does not need to be replaced anytime. Considering as integration period an integer number (n) of a reference period $T_{REF} = 60s$, this relation can be expressed as:

$$\sum_{i=1}^{n} V_{DD} I_{mHARV,i} T_{REF} \geq \sum_{i=1}^{n} V_{DD} I_{mSYS,i} T_{REF}$$

where I_{mHARV} and I_{mSYS} are the average currents in the reference period, respectively provided by the harvester and absorbed by the system. Preliminary considerations on self-sustainability can be derived from the characterization of both nodes and harvesters reported before. Assuming that I_{HARV} does not change and only one request-response phase is served during the integration period, the above relation becomes:

$$nI_{HARV} \geq (n-1)I_{SB} + I_{REQ}$$

For example, with $I_{SB} = 35\mu A$ and $I_{REQ} = 200\mu A$, the sensing node with a PV cell under a light intensity of 300 lux, is autonomous if a request-response phase occurs every 7 minutes or more. This value drops to 2 minutes when the light intensity rises up to 700 lux. In the actuating node the energy consumption during thermo-valve opening and closing must be added to the communication contribution (I_{REQ}). The actuating node with a harvester has been characterized at several percentage values of the full range of valve closing and opening because in a real scenario small valve movements are required to control a target temperature. For example with 25 % of full range valve closing, the self-sustainability is guaranteed if the operation is performed every 19 minutes. This value drops down to 8 minutes with 10 % of full range valve closing. In case of full range closing (opening) the node is autonomous if the actuation is performed every 72 (50) minutes.

9.6 Conclusions

This Chapter presented the project of an autonomous WSAN with non-stringent timing constraints where the most important requirement is related to the energy balance of the involved nodes. A power management service at "fog" level is responsible for the self-sustainability of sensors and actuators. The research has been carried out at the joined ARCES - ST Laboratory at the University of Bologna on self powered IoT platforms.

Chapter 10

Semantics-based applications in the sound domain

Contents

This Chapter describes my Research activity carried out in the Centre for Digital Music of the Queen Mary University of London. Objective of this activity has been the application of the results of my research on semantic technologies, and in particular publish-subscribe middlewares (See Chapters 3 and 4), to two new application areas: **Semantic Audio**[1] and

[1]Republished with permission of ACM (Association for Computing Machinery), from Playsound.space: enhancing a live music performance tool with semantic recommendations, Fabio Viola, Ariane Stolfi, Alessia

Internet of Musical Things[2][3]. The work carried out in these two areas is described respectively in Sections 10.1 and 10.2 and is still ongoing.

10.1 Semantic audio

Semantic Audio is an interdisciplinary research area involving Digital Signal Processing, Machine Learning, and various knowledge representation and sharing technologies borrowed from the Semantic Web. **Audio Commons** is a EU funded project framed in the **Horizon 2020** programme and involving universities (the Queen Mary University of London, the University of Surrey and the Universitat Pompeu Fabra) and enterprises like Jamendo SA, AudioGaming and Waves Audio LTD. The aim of the **AudioCommons Initiative** [235] is to make easier for creative industries to access contents available under **Creative Commons** (CC) licenses or belonging to the **Public Domain** (PD). The amount of these artworks grows every day, thanks to the contributions of the creative community or to the expiry of copyright licenses. Accessing CC or PD artworks is not always easy, due to the presence of multiple online repositories and this motivates the need for AudioCommons.

Fig. 10.1 depicts the architecture of the AudioCommons Ecosystem. A central role is played by **Content Providers** (CPs), where artists publish their audio files to share them with the community. In the AudioCommons Ecosystem, there are currently three main CPs: Europeana[4], Freesound[5] and Jamendo[6]. As previously mentioned, the presence of multiple repositories hosting Creative Commons audio contents, each one adopting its own data representation format, is one of the obstacles to the diffusion of these resources: is in fact laborious for the users to have a look on different websites every time they're looking for something. Then, the role of the AudioCommons API (AC API) is to provide a unified way to access this content. AC APIs rely on the CPs APIs, since all of these three online repositories provide a set of web APIs to perform search operations as well as download/upload of audio files. Unfortunately, all these CPs provide different API calls and represent data in a incompatible

Milo, Miguel Ceriani, Mathieu Barthet, Gÿorgÿ Fazekas. SAAM '18 Proceedings of the 1st International Workshop on Semantic Applications for Audio and Music, 2018; permission conveyed through Copyright Clearance Center, Inc.

[2]© IEEE, Reprinted with permission, from Luca Turchet, Fabio Viola, Francesco Antoniazzi, Gÿorgÿ Fazekas, Mathieu Barthet. Towards a Semantic Architecture for the Internet of Musical Things. 2018 Proceedings of the 23rd Conference of FRUCT Association. Nov. 2018.

[3]© IEEE, Reprinted with permission, from Fabio Viola, Luca Turchet, Francesco Antoniazzi, Gÿorgÿ Fazekas. C Minor: a Semantic Publish/Subscribe Broker for the Internet of Musical Things. 2018 Proceedings of the 23rd Conference of FRUCT Association. Nov. 2018.

[4]`http://www.europeana.eu`

[5]`http://www.freesound.org`

[6]http://www.jamendo.org

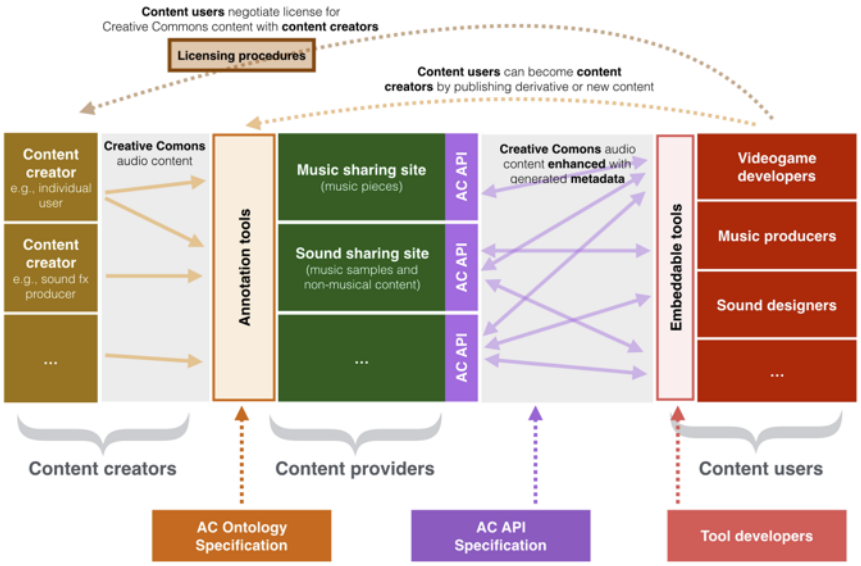


Figure 10.1: The AudioCommons Ecosystem [235]

way.

10.1.1 Semantic technologies in the ACE

The heterogeneity of the formats used by the APIs of CPs, can be faced through the use of Semantic Web technologies. With these protocols, in fact, it is possible to define a shared vocabulary of the concepts and represent all the data available on CPs. However, is not conceivable to force all the content providers to switch from their current data representation and APIs to an entirely new semantic architecture (at least not in a short time interval). For this reason, a solution consists of a middleware in charge of:

1. providing unified semantic APIs;

2. routing of the requests to the CPs;

3. performing real-time translation of messages according to a set of ontologies.

This central role in the ACE is played by the **semantic mediator** (SM) that I designed and implemented during my staying in London and I'm still working on with the team of Centre for Digital Music. The semantic mediator leverages the work performed on another tool belonging to the Audio Commons Ecosystem (ACE): PlaySound (PS) that is described in the following Sections.

10.1.2 Playsound – Semantic recommendation for music composition

PlaySound is collaborative tool for music composition based on the audio repository Freesound. The main aim of Playsound is to allow composing music in a collaborative way using samples retrieved from Freesound. As a first research work carried out on the Playsound platform and involving semantic technologies, I designed and implemented a recommendation system with the aim to experiment the basic functionalities for the mediator. The main idea driving this research has been using the sound samples selected by the users of PS to look for similar contents on all the ACE CPs (with a simple keyword search) and reject bad results according to a subsequent audio analysis.

The main challenge was then to simultaneously contact different CPs to perform a keyword-based search and represent all of their replies according to the AudioCommons Ontology.

The requirements of the desired recommendation system can be summarized as:

- **Re-usability** – the desired recommender should discover relevant contents on Freesound, Europeana and Jamendo. This is a common task in the ACE, so there is the need for a re-usable component, generally accessible by all the entities in the ecosystem.
- **Scalability** – the number of users of Playsound is expected to grow, so it is important to grant scalability to support a high number of simultaneous recommendation requests.
- **Interoperability** – CPs adopt different formats to represent the metadata bound to each audio file. This is an obstacle to interoperability. Moreover, multiple applications in the ACE could benefit from the recommendations provided by Playsound.

The proposed solution: a SWoT architecture

A Semantic Web of Things architecture was envisioned to face all the issues highlighted in the previous Section: a set of web things providing, among others, search functionalities could be run and made available to all the entities in the ecosystem. Moreover, multiple instances of the same Web Thing can run at the same time, providing an effective way to support load balancing in large scenarios. Lastly, the adoption of a semantic model to represent data grants interoperability among CPs as well as among other ACE entities.

Then, a multiagent system (Fig. 10.2) was designed and implemented to absolve this tasks. Despite being a pure software entity, every agent in the system was designed as a Web Thing, acting then as a virtual device. All the implemented Web Things in the ecosystem interoperate through the SPARQL Event Processing Architecture presented in Section 4.4. The developed Web Things are:

- **AudioQuery Server WT** – This Web Thing implements the software agent exposing the semantic description of the PlaySound server and responsible for generating recommendation requests to the proper Web Thing.
- **Europeana WT**, **Freesound WT**, **Jamendo WT** – These are the Web Thing providing search mechanisms for the proper CP. They are software agents acting as semantic bridges to the original API. They provide a search mechanism that returns data represented according to the Audio Commons ontology. As previously mentioned, multiple instances of these web things may be running at the same time to efficiently serve clients in case of high number of requests.
- **Sonic Annotator WT** – A software agent offering Sonic Annotator [236] as a service. This Web Thing is invoked to perform audio analysis on the input (in our ecosystem it is used to calculate the linear centroid of every sample to determine similarity with the sample selected by the user).
- **Recommender WT** – A service that performs orchestration of requests by discovering and invoking CPs (if available) to search for audio files and then discovering and invoking Sonic Annotator's action to compute similarity measures.

To represent CPs output messages in a semantic way, a tool from the École des Mines de Saint-Étienne was used: SPARQL-Generate [237, 238]. This tool is in charge of performing the translation of the input according to a set of mapping rules defined with an extended version of the SPARQL query language.

As depicted in Fig. 10.2, multiple ontologies compose the final system:

- **Semantic Web of Things ontology** – developed during the PhD and described in Section 7.1. It is based on a previous work by Serena et al. [175] and according to the current W3C's Web of Things terminology [95]. This ontology is used to provide a semantic description of all the web things.
- **Audio Commons ontology** [239] – designed to have common data model to search and interact with audio resources, as required by the EU Research Project Audio Commons. It generalizes the Music Ontology [240] and extends the FRBR ontology. All the web things offering a search action exploit the Audio Commons ontology to map the results in a uniform way.
- **Audio Features ontology** – The Audio Features ontology (AF) [165] allows sharing content-derived information about musical recordings; it is used by Sonic Annotator to represent the extracted audio features.

- **Vamp Plugins Ontology** – it describes the Vamp API used to invoke vamp plugin system for audio analysis [241].
- **Recommendation ontology** – it is based on the Similarity ontology, the Dublin Core Metadata Initiative (DCMI) Metadata Terms, the Association ontology and the Ordered List ontology. The aim of this ontology is to provide basic concepts and properties for describing recommendations [242].

The resulting architecture is represented in Fig. 10.2: the ontologies act as a bridge among the Web of Things and the underlying SPARQL Event Processing Architecture. The former are semantically mapped into SEPA, and SEPA also provides a publish/subscribe interface to invoke them and make them interoperate.

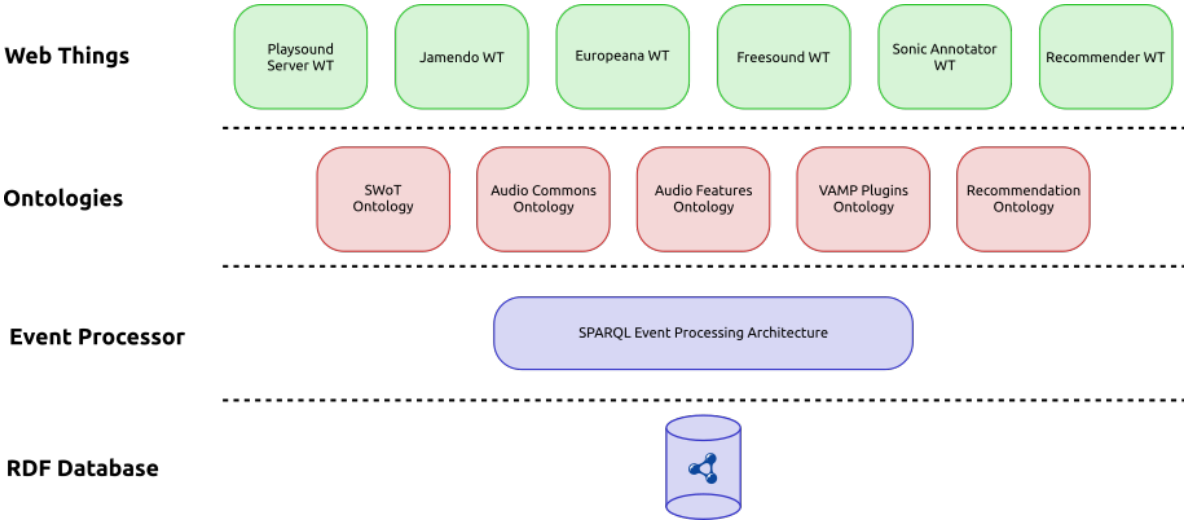


Figure 10.2: Software Architecture of the Playsound recommendation system [243]

The flowchart represented in Fig. 10.3 shows the steps performed by the recommendation system to identify and suggest interesting results to the users of PlaySound.

10.1.3 SPARQL-Generate

The calls to the CPs API return results represented in a custom format, without semantic representation. SPARQL-Generate [237, 238] allows translating non-semantic data into a set of RDF triples by means of rules defined according to a properly extended version of the SPARQL query language.

Let's consider for example the request performed to Jamendo APIs to get all the tracks matching a given keyword. The following listing proposes an excerpt of an example reply message:

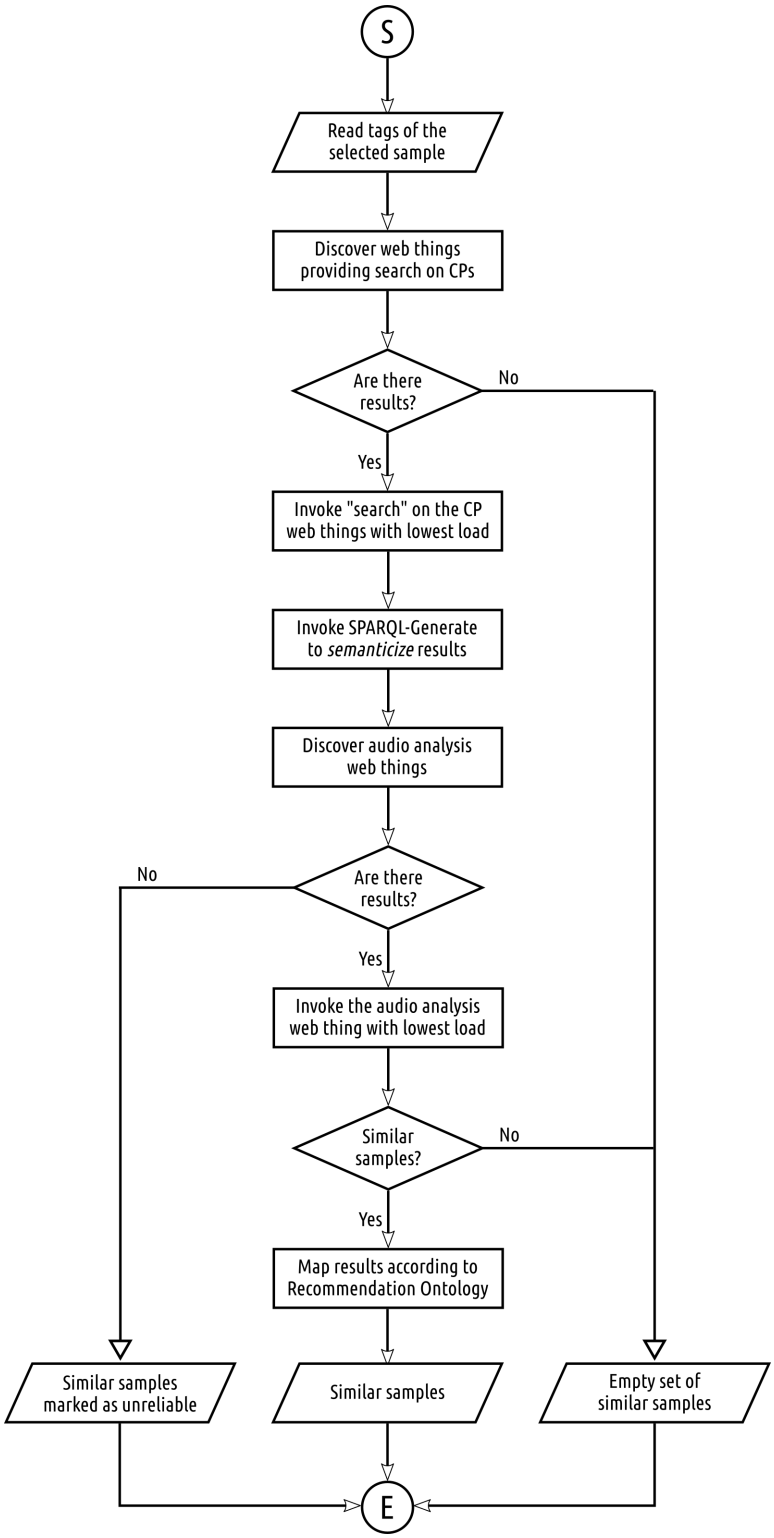


Figure 10.3: Flowchart of the Playsound recommendation system

```
{
   "headers": {
      "status": "success",
      "code": 0,
      "error_message": "",
      "warnings": "",
      "results_count": 1,
      "next": "http://api.jamendo.com/v3.0/tracks?client..."
   },
   "results": [{
      "id": "1342543",
      "name": "Motorway",
      "duration": 423,
      "artist_id": "459669",
      "artist_name": "John Russell",
      "artist_idstr": "John_Russell_(2)",
      "album_name": "Eclectic Electro",
      "album_id": "158505",
      "license_ccurl": "http://creativecommons.org/lic...",
      "position": 7,
      "releasedate": "2016-05-09",
      "album_image": "http://imgjam1.jamendo.com/album...",
      "audio": "https://mp3l.jamendo.com/?trackid=1342...",
      "downloadurl": "https://mp3d.jamendo.com/downloa...",
      "prourl": "https://licensing.jamendo.com/track/1...",
      "shorturl": "http://jamen.do/t/1342543",
      "shareurl": "http://www.jamendo.com/track/1342543",
      "image": "http://imgjam1.jamendo.com/albums/s158..."
   } ... ]
}
```


Results include a track named "Motorway" with ID 1342543, authored by John Russel (ID 459669). This content was released on the 9 May 2016 with a CC license. According to the AudioCommons Ontology, this information could be represented creating:

- an instance of the class `ac:AudioClip` identified by the URI retrieved from the field

`shareurl`. The datatype property `dc:title` (defined by the DC ontology) can be used to bind the title of the track to this resource, while the object properties `ac:compiled` and `cc:license` link the resource to the artist and the license of the content. Tracking the provenience of information is also important, so, leveraging the Prov Ontology [244], a further object property (i.e., `prov:wasAttributedTo`) can be added.

- an instance of the class `ac:AudioFile` identified by the URI retrieved from the key `audiodownload`.

This translation from non-semantic JSON to RDF can be performed through the following SPARQL-Generate query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ac: <http://audiocommons.org/ns/audiocommons#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX cc: <http://creativecommons.org/ns#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX fn: <http://w3id.org/sparql-generate/fn/>
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
GENERATE {
   ?audioClip rdf:type ac:AudioClip ;
        dc:title ?title ;
        cc:license ?license ;
        ac:compiled ?artistURI ;
        ac:available_as ?audioFile ;
        rdf:type prov:Entity ;
        prov:wasAttributedTo <http://jamendo.com> .
   ?audioFile rdf:type ac:AudioFile .
}
SOURCE <http://api.jamendo.com/v3.0/tracks?...> AS ?s
ITERATOR iter:JSONPath(?s,"$.results") AS ?res
WHERE {
   BIND(fn:JSONPath(?res, ".id" ) AS ?id)
   BIND(fn:JSONPath(?res, ".shareurl" ) AS ?audioClip)
   BIND(fn:JSONPath(?res, ".downloadurl" ) AS ?audioFile)
   BIND(fn:JSONPath(?res, ".name" ) AS ?title)
   BIND(iri(fn:JSONPath(?res, "license_ccurl")) AS ?license)
```

```
26      BIND(fn:JSONPath(?res, "artist_id") AS ?artist_id)
27      BIND(iri(CONCAT("http://www.jamendo.com/artist/",
28          ?artist_id)) AS ?artistURI)
29  }
```

Through this listing it is possible to notice the new constructs introduced by SPARQL-Generate to permit the translation:

- `GENERATE`: to specify the template for the final triples;
- `SOURCE`: to declare the URI of the input, in this case the API call;
- `ITERATOR`: to define a way to iterate over the input fields (requires the `iter` namespace).

Through the `BIND` command, the input fields can be retrieved, manipulated and bound to a SPARQL variable to be later used in the `GENERATE` section.

A set of mappings has been defined for every Content Provider in the Audio Commons Ecosystem, all available on the following repositories:

- `https://gitlab.com/desmovalvo/jamendo-to-audiocommons`
- `https://gitlab.com/desmovalvo/europeana-to-audiocommons`
- `https://github.com/miguel76/freesound-to-audiocommons`

Proof of concept

The designed system has been implemented on the branch `semrec` of the GitHub PlaySound repository[7]. This Section proposes two examples of the execution of the system.

Example 1 Searching for "`8 bit`", forty results are presented to the user on the first page (forty is the limit imposed by the pagination system). Clicking on the sample named "`8-Bit explosion`", this file is added to the left bar. Then PlaySound asks for recommendations. First of all a discovery of all the available web things offering a search service is performed. In this example, three web things, one for Europeana, one for Freesound and one for Jamendo were available. The keywords characterizing the selected sample are: `computerized`, `wav`, `bit`, `8` and `8bit`.

Five matches (this is the configured limit) are returned by both Jamendo and Freesound, while only two by Europeana. All the returned samples are processed during the similarity

[7] https://github.com/arianestolfi/audioquery-server/tree/semrec

analysis that calculates the spectral centroid. After this step, all the results coming from Europeana are rejected; respectively two and four samples are removed from the lists of Jamendo and Freesound results.

The following list shows a summary of the results of the first step, reporting the source, the name of the audio file, and an asterisk if the file is filtered out in the second step. A screenshot of this test is visible in Fig. 10.4.

- **Jamendo**
  - Ode to Zork – by: Octabitron*
  - Bouncy Chips – by: Melhadf*
  - The warp repeater theme – by: Dementialcore
  - TELEPORTER – by: DANJYON KIMURA
  - danza terrestre – by: Kamarina SOUND MATHINE
- **Freesound**
  - Zerothru9.wav*
  - boxes.wav*
  - Pattern01.wav*
  - SequenceText1.wav
  - Craxy.wav*
- **Europeana**
  - Viaduct Westrandweg Halfweg*
  - Mikail Ivanovic Glinka: "Russlan e Ludmilla - Ouverture"*

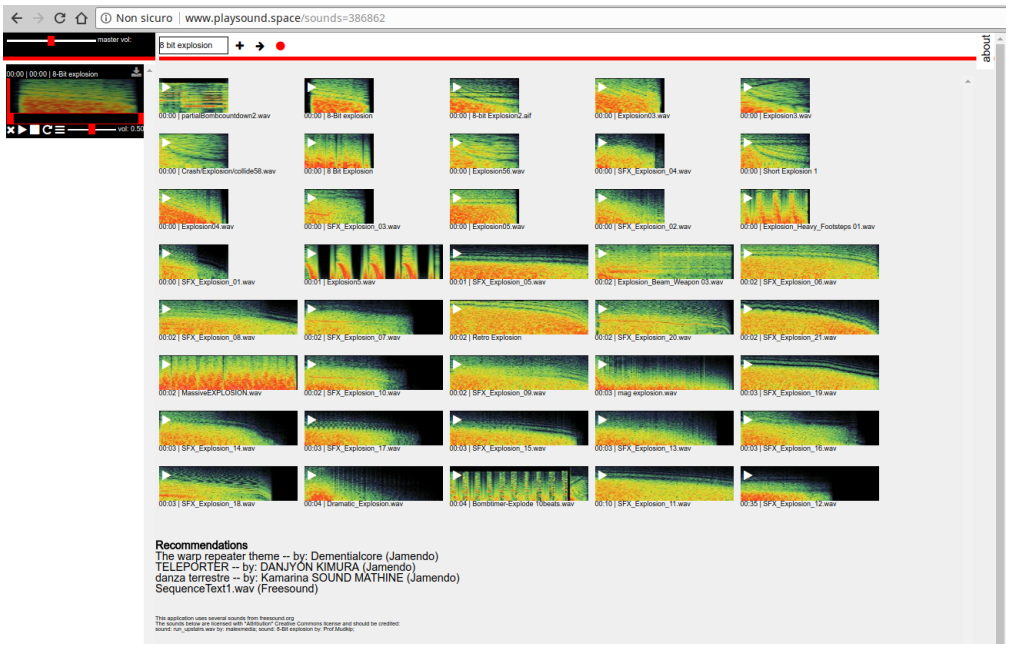


Figure 10.4: An example of recommendations in Playsound

The system is intrinsically able to adapt to the available services. In fact, as expected, shutting down Jamendo web thing, the recommendation service still provides suggestion (i.e., in this case only the file named "`Pattern01.wav`"), while on the other hand closing the Freesound WT only the three songs provided by Jamendo will be suggested to the user. Of course, shutting down Europeana's service nothing changes. This is possible thanks to the discovery phase in which the Recommender WT looks for the available Web Things, before deciding with one to contact.

Example 2 With the same set of Web Things of the previous example, a second test has been performed. Searching for "`husky howl`" and selecting the audio sample "`Igor-13B.wav`" (tagged with `moan`, `growl`, `malamute`, `dog`, `Wolf`, `husky`, `bark` and `howl`) both Freesound and Jamendo returned five results for this set of keywords, two less for Europeana. The subsequent audio analysis carried out by Sonic Annotator, allowed rejecting two suggestions from Freesound and Europeana (considered too different from the original file), and one from Jamendo. The following is the list of samples returned by the three CPs Web Things; again, an asterisk is used to mark all the samples considered too different from the original file, thanks to the audio analysis performed by the sonic annotator web thing.

- **Jamendo**
  - Curious Day With Rufus Hot Sauce, Op 192 – by: Edward Schaffer

  - The night drives the wolf – by: DJ Mircomix*
  - The Wolf (Acoustic) – by: Nemo Wilson
  - Reaction 7 - Cool Dog – by: Reaction 7
  - Alain (instrumental) – by: FilsTool

- **Europeana**
  - Dog barking and birds
  - Grey Wolf', 'Woodland*
  - Love Is A Dog*

- **Freesound**
  - Igor-13C.aif*
  - Igor- I wan't dinner.wav
  - Igor Dinner Anticipation.wav
  - Igor17B-a drink of water.aif*
  - Igor16B-big talk.aif*

Summarizing...

The recommendation system depicted in Fig. 10.2 has been successfully implemented and this proof-of-concept, that will be further extended in the future, constitutes the first step towards the next generation of the semantic mediator of the Audio Commons Ecosystem. The multiagent architecture based on the Semantic Web of Things paradigm is in fact: 1) scalable thanks to independent and multiply instantiable software agents; 2) able to grant interoperability among heterogeneous software agents thanks to a set of ontologies used to represent data in a uniform and agreed format; 3) reusable, since every software component provides a set of actions that are not limited to the recommendation task.

10.1.4 Semantic mediator

My collaboration with the C4DM of the Queen Mary University of London is still ongoing and the current research is aimed at the development of the next generation of the Semantic Mediator, a server providing a simplified REST API through which all the clients in the Audio Commons Ecosystem can simultaneously interact with different Content Providers. Results of every API call are represented according to the Audio Commons Ontology and encoded

using JSON-LD. The research carried out on PlaySound represents an important building block for the development of this new software component, also due to the experiments about the integration of the SPARQL-Generate approach to map non-semantic data as a set of RDF triples.

10.2 Internet of Musical Things

The Internet of Musical Things is an emerging research area binding IoT technologies and approaches to the musical domain. IoMusT can be defined 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"* [245].

Still according to Turchet et al., a Musical Thing is *"a computing device capable of sensing, acquiring, processing, or actuating, and exchanging data serving a musical purpose"*. A key aspect of the IoMusT paradigm is the interoperability among Musical Things. Generally speaking, interoperability involves three levels: network, syntax, and semantics. While network interoperability concerns protocols for exchanging information among heterogeneous devices (regardless of the content of the messages), syntax interoperability level regards the way messages are structured and encoded. The third and most important level (on which my research activity is focused) conveys the meaning of the exchanged messages [14]. To achieve this task and grant interoperability in an IoT scenario, a set of standardized protocols is usually employed: those belonging to the Semantic Web. Up to now, interoperability across musical devices in co-located settings has mostly relied on existing communications standards such as Wi-Fi and Open Sound Control (OSC) protocol [246]. The adoption of semantic technologies has been envisioned [245], but no effort has been conducted yet to apply semantic technologies to IoMusT scenarios. Then, the research activity presented in the rest of the Chapter is aimed at the design and development of a semantic Internet of Musical Things ecosystem.

10.2.1 Semantic IoMusT architecture and ecosystem

A mandatory requirement for a semantic architecture for the IoMusT is to grant a timely and loosely-coupled interaction among heterogeneous entities.

A semantic publish-subscribe message-oriented middleware [38], together with a set of agreed ontologies, enables this vision. The IoMusT semantic architecture that I developed and that I propose in the following, relies on a SEPA instance and a set of domain-specific ontologies. Later on, I designed and developed a second prototype, adopting a novel context broker specifically designed for the IoMusT scenario and presented in Section 4.5.

Based on this architecture, an IoMusT ecosystem may encompass several different Musical Things playing the roles of producers, consumers and aggregators as shown by Figure 10.5.

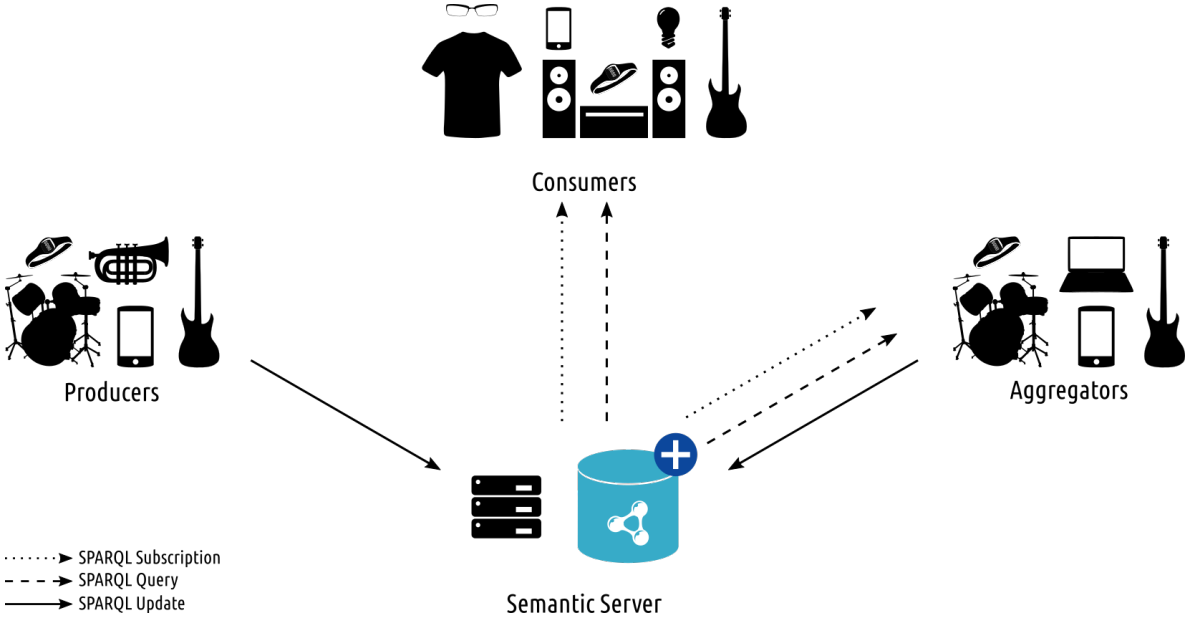


Figure 10.5: IoMusT ecosystem [247].

In this ecosystem at least one producer must be present and this role can be played by devices belonging to different categories (e.g., Musical Things such as SMIs, MHWPAs, MHWPs, or smartphones with musical apps, which publish audio features calculated on board). Notably, these calculations are particularly relevant to the edge computing paradigm as instead to leave the centralized server compute features from the signals generated by the devices, these are computed by the Musical Things themselves. Multiple and heterogeneous consumers can simultaneously exist in the ecosystem (e.g., SMIs, which may modify some of the parameters of their sound engine according to the information read from SEPA or stage equipment like lighting systems, or wearables such as smart glasses, virtual reality headsets

and so on). All these Musical Things change their behavior in response to the information to which they have subscribed from SEPA. Finally, aggregators are not mandatory in this ecosystem. If present, aggregators could be for instance SMIs, MHWPAs, MHWPs, smartphones, or laptops.

10.2.2 Validation of the ecosystem – prototype 1

This Section describes a proof of concept of the described IoMusT ecosystem developed at the Queen Mary University of London.

A Bela board for low-latency audio and sensors processing [248] (both in its normal version and in the pocket version called "Bela-mini") has been employed to build five prototypes of Musical Things (wireless connectivity provided by the NETGEAR A6100-100PES Wi-Fi USB dongle attached to the Bela board supporting the IEEE 802.11.ac Wi-Fi standard), while power supply has been provided by a powerbank. From the point of view of the business logic, the prototypes can be described as:

- A **producer**: a generator of synthesized notes (by means of a basic sinusoidal oscillator) with random density in the range of [1, 200] notes per second. The information put by this component into the RDF graph is not the set of generated notes, but the average of the four parameters (density, frequency, duration, and amplitude) computed every 5 seconds and mapped the requests according to the Audio Features Ontology [165].

- An **aggregator**: analyzes the information sent by the producer using a fuzzy logic [249] where the 81 possible combinations resulting from dividing into 3 parts the range of each of the 4 parameters, were randomly grouped into 4 subsets of 20, 20, 20, and 21 quadruplets. The quadruplets belonging to each subset were then associated to one of the 4 possible statuses: "A major", "E major", "F# minor", "silence". Such statuses were then sent back to the semantic server that dispatched it to the three consumers.

- Three **consumers**: they are notified by SEPA of the current status of the system and generate accompaniment of the melody played by the producer. Their sound engine was configured to produce one of the following chords: A major, E major, F# minor. These chords were selected to achieve a sense of consonance with the played melody (according to the tenets of the classic harmony theory [250]). These chords were rendered by a bank of sinusoidal oscillators.

For both producer and consumers, a small loudspeaker was used to deliver sound. The

sound engine was coded in libpd[8], a porting of the Pure Data[9] computer music environment into a library for embedded systems [251].

The Audio Features Ontology [165] was exploited by the clients to represent or interpret information. The ontology was extended to define a new class for the inferred status and a new object property linking instances of the current performance with the instance of the status. The Semantic Server composing the ecosystem ran on a Dell Alienware 17 R2 laptop supporting the IEEE 802.11ac Wi-Fi standard and running Ubuntu Linux 17.10. The version 0.8.4 of the Java implementation of SEPA was used. To enhance the performance of the application and meet the requirements of the IoMusT domain, the semantic server only hosted the current state of the context. Then, from a semantic point of view, the context of this use case encompassed the following entities:

1. the current performance;
2. the last high-level audio features extracted by the producer;
3. the most recent state inferred by the aggregator.

10.2.3 Validation of the ecosystem – prototype 2

The devices involved in the semantic IoMusT ecosystem described in the previous Section are all battery-powered and all of them are characterized by limited resources. Moreover, this application field has very strict requirements in terms of latency. These considerations caused the start of a new Research project for the development of a SEPA-inspired context broker for constrained environments. The resulting broker, C Minor (see Section 4.5), was then employed in the second prototype of the ecosystem.

This sligthly different proof-of-concept ecosystem was aimed at simulating the interaction between a smart musical instrument performer and audience members in a TMAP context. The ecosystem, illustrated in Fig. 10.7, comprised the following components:

- A smart mandolin [252] (playing the role of a producer) consisting of a conventional acoustic mandolin smartified with a sensors interface, a contact microphone, a loudspeaker, wireless connectivity, embedded battery, and the Bela board for low-latency audio and sensors processing [253]. The data published by the mandolin consists of a set of audio features (i.e., the note onset, its pitch and amplitude).

[8] http://libpd.cc/
[9] https://puredata.info/

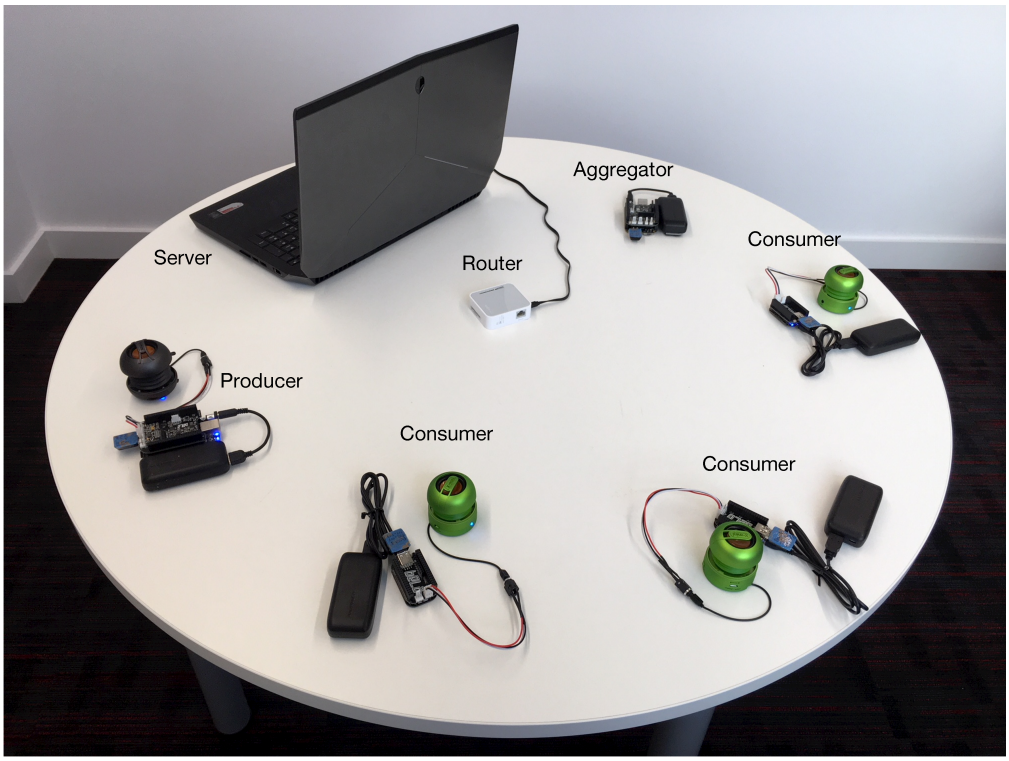


Figure 10.6: The IoMusT ecosystem (prototype 1) [247].

- Six prototypes of Musical Things playing the role of consumer, created to simulate an IoMusT scenario involving audience participation (i.e., where audience members use the prototypes to generate musical sounds). The prototypes were composed by the Bela board, a NETGEAR A6100-100PES Wi-Fi USB dongle, a loudspeaker, and a powerbank. Consumers were given the role of accompaniment of the melody played by the smart mandolin. The sound engine of three prototypes was configured to produce sequences of synthesized notes.

- A semantic server running an instance of C Minor and an aggregator that elaborates data published by the smart mandolin to calculate additional audio features and deliver them to the musical things (ODROID-XU4 board manufactured by Hadkernel, enhanced with the Wi-Fi router TP-Link TL-WR902AC which features the IEEE 802.11ac standard over the 5GHz band). Following the recommendations reported in [254] to optimize the components of a Wi-Fi system for live performance scenarios to reduce latency and increase throughput, the router was configured in access point mode, security was disabled, and only the IEEE 802.11ac standard was supported. As previously mentioned, the context broker of the semantic server only hosts the current context. This allows

one to increase the performance of the application, since removing outdated information allow the engine to timely process a lower amount of data.

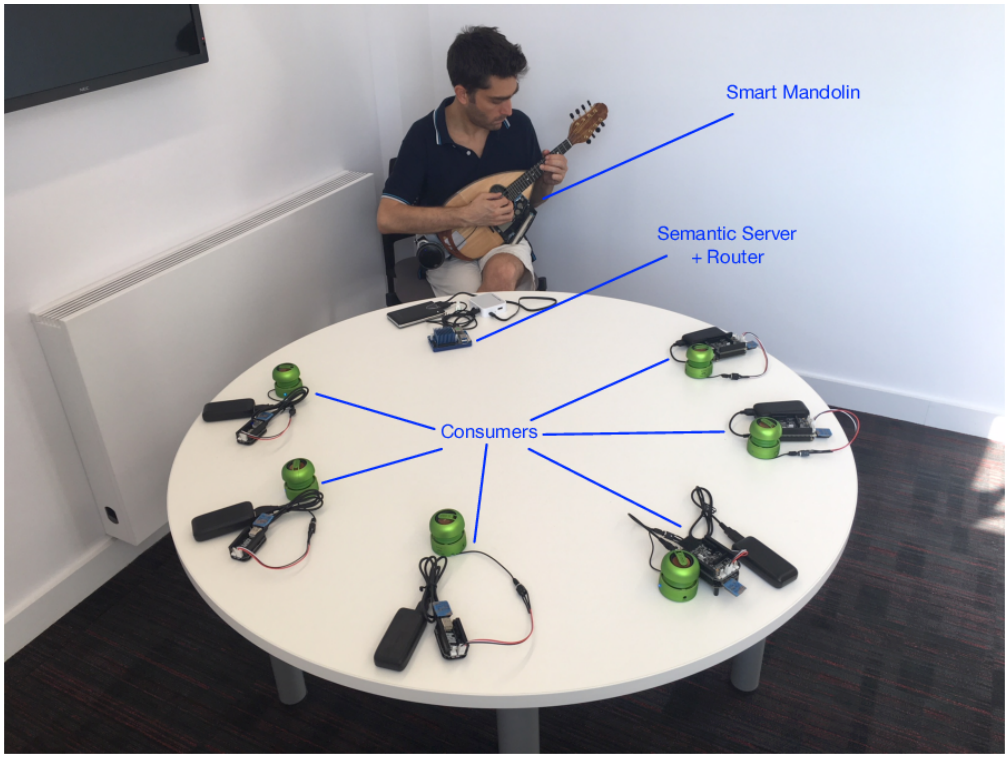


Figure 10.7: Semantic IoMusT ecosystem based on C Minor

10.3 Conclusion

Both the semantic audio and the Internet of Musical Things represented interesting test beds for the semantic platforms developed throughout the whole PhD. As regards the semantic audio, the SEPA platform together with the SWoT ontology represented an effective way to develop small independent services according to the Semantic Web of Things paradigm. Moreover, thanks to the common ontology to handle Web Things input and output, it has been possible to orchestrate these services to perform complex tasks on data coming from different sources.

As regards the IoMusT instead, a minimalist approach taking in consideration constrained devices was adopted. With this approach, devices are not semantically mapped into the SEPA broker, but only interoperate through it with a simple set of subscriptions on the data of interest. Further investigation brought to the development of a lightweight implementation of

the SEPA platform based on the popular CoAP protocol. The research activity on semantic ecosystems is still ongoing: future works on this topic will be aimed at 1) a more detailed assessment of the performance of all the elements in the ecosystem; 2) extension of the work towards the remote interaction among performers and audience.

Part V

Conclusions

Chapter 11

Conclusion and future work

In this Thesis I have presented my PhD research focused on Semantic Web technologies and their application to the Internet of Things, in the area known as Semantic Web of Things.

The main research topic I have addressed is the study of efficient and effective ways to adopt Semantic Web technologies, in order to develop publish/subscribe context brokers for SWoT applications. This activity has been carried out through the analysis and design of algorithms to process subscriptions in context brokers. These algorithms adopt: 1) a filtering mechanism that allows detecting the subscriptions interested by a SPARQL update; 2) a local context store per subscription containing a subset of the KB, aimed at performing faster queries when comes to detect changes on the subscription bindings; 3) grouping of equivalent o even partly equivalent subscriptions. The resulting algorithms presented in Chapter 3, (i.e., LUTT, CLUTT and CHLUTT) proved to enhance the performance of the Smart-M3/SEPA platforms in terms of scalability and memory footprint. These algorithms are now part of the context brokers that I developed during the PhD (activity described in Chapter 4). In fact, this research activity brought also to the design and development of three different semantic context-brokers framed in the Smart-M3 interoperability platform named SPS, OSGi SIB and pySIB. Such brokers represented the playground for the implementation of the different algorithms as well as the ideal platform to propose novel semantic primitives like the Delayed SPARQL Update and the Persistent Update, which allows for in-broker rule-based reasoning mechanisms. The research on the Smart-M3 platform converged into the development of the SPARQL Event Processing Architecture and related tools (i.e., the client-side APIs, the SWoT ontology and the SEPA Dashboard). Aim of SEPA is to support Big Data in the Semantic Web of Things by means of a redefined architecture and the adoption of web standards. This platform was officially presented at the W3C international meeting of the Working and Interest

Groups on the Web of Things in Düsseldorf in July 2017 and later on at the 21st FRUCT Conference in Helsinki (November 2017) and represents the core of the ongoing research. Indeed, further experimental algorithms to detect changes in the graph are currently being explored in this platform. The study of the applicability of the publish-subscribe paradigm for semantic brokers aimed at the Semantic Web of Things has also involved a highly constrained domain, the Internet of Musical Things. A novel context broker based on the lightweight IoT protocol CoAP has been developed, providing in this way a low-latency platform that maps the SPARQL 1.1 protocol, accessible also by constrained devices.

A second Research topic (still ongoing) has been the design of the first benchmark aimed at SWoT applications, with particular focus on the discoverability problem. The benchmark is oriented at semantic publish-subscribe brokers, like the above-mentioned SEPA, but can be easily extended to support other systems. The first experimental version of the benchmark has permitted the characterization of the performances of several subscription algorithms implemented on the target context brokers, so it proved to be effective. Nevertheless, a set of other Key Performance Indicators to be implemented in the future release have been identified and will be integrated in the next version of the benchmark.

From my research activity heavily based on the development of applications pivoting on ontologies and semantic KBs, emerged the need for effective ways to debug RDF graphs. My research has then been extended towards a third topic concerning the design of a new representation method for semantic knowledge bases aimed both at novice users willing to learn Semantic Web technologies, but also at expert users looking for a quick way to identify incoherent data. The novel approach based on a three-dimensional multi-planar view and the concept of Semantic Planes has been implemented in the 3D graph viewer Tarsier. The tool has proved to be effective for both context-aware/SWoT applications and pure SW datasets. Moreover, the evaluation of the user experience highlighted the validity of the approach from the point of view of both novice and experienced users. Future work in this research area will be aimed at implementing a real-time visualization of the evolution of a graph based on semantic publish-subscribe platforms like SEPA.

The assessment on the field of the validity of the developed Smart-M3 and SEPA platforms has instead lasted for the whole PhD period. The first application domain where I had the chance to study the effects of the Smart-M3 platform has been the Electro-Mobility. This research, carried out in collaboration with several departments of the University of Bologna

as well as enterprises like Arrowhead, Bitron and Gewiss has been framed in the European Project Arrowhead. A set of services supporting a co-simulation environment for the EM in the city of Bologna has been developed. A second activity, still related to energy management, but in a home automation scenario, involved the development of an autonomous WSAN centered on the Smart-M3 platform. Finally, a relevant activity related to the application of SEPA in real scenarios has been framed in the sound domain and mainly in two sub-areas: the Semantic Audio (in the context of the European Project AudioCommons) and the Internet of Musical Things. The research in this domain has brought to the definition of the first semantic ecosystem for the Internet of Musical Things. Future work in this area will involve the exploration of remote interaction among performers and audience in the IoMusT domain. Future work concerning the application of the semantic platforms developed during the PhD will continue involving the EU project SWAMP (smart irrigation).

List of Tables

List of Figures

Acronyms

AAL Ambient Assisted Living. 30

AC Audio Commons. 222

ACE Audio Commons Ecosystem. 14, 221, 223, 224, 230, 233

ACO Audio Commons Ontology. 96

ADR Adverse Drug Reaction. 30

AMQP Advanced Message Queuing Protocol. 61, 90, 91

API Application Programming Interface. 31, 39, 63, 85, 121, 190, 200, 201, 222, 225, 226, 230, 233, 243

ARCES Advanced Research Center on Electronic Systems. 24, 37, 62, 122, 210, 220

ARTEMIS Advanced Research and Technologyfor EMbedded Intelligence and Systems. 37

ARTQ Added Removed Triples Queue. 51, 52, 54, 66, 67

BLE Bluetooth Low Energy. 29

BS Booking Service. 199

BSBM Berlin SPARQL BenchMark. 108

CC Creative Commons. 222, 228

CP Content Provider. 222, 223, 224, 225, 226, 230, 232

C-RTT CoAP Round Trip Time. 98

C4DM Centre for Digital Music. 25, 90, 192, 221, 223, 233

CHIRON Cyclic and person-centric Health management: Integrated appRoach for hOme, mobile and clinical eNvironments. 39

CHLUQT Centralized Hierarchical Look-Up Quad Table. 56

CHLUTT Centralized Hierarchical LUTT. 56, 131, 134

CLUTT Centralized LUTT. 56, 131

CoAP Constrained Application Protocol. 61, 90, 91, 92, 93, 95, 96, 216, 239, 243

COBRA COntext BRoker Architecture. 27, 61

CoRE Constrained RESTful Environments. 91

CRF Centro Ricerche Fiat. 25

CSMA Chained Semantic Matching Algorithm. 46

CSV Comma-Separated Values. 117, 118, 120

CTA concurrent think-aloud protocol. 177

CTS context triple store. 52, 53, 56, 66, 67, 69, 129

D2C Device to Cloud. 198

DC Dublin Core. 228

DCMI Dublin Core Metadata Initiative. 226

EC Everyware Cloud. 199, 200

ECSA EC Service Abstraction. 200, 201

EKP Encyclopedic Knowledge Pattern. 151

EM Electro-Mobility. 187, 192, 195, 197, 198, 199, 200, 206, 207, 208, 209, 210, 244

EMMS Electro-Mobility Management System. 200, 201

EU European Union. 173, 192, 222, 225, 244

EV Electric Vehicle. 202, 207, 208

EVSE Electric Vehicle Supply Equipment. 198, 201, 202, 203, 208

FCT Flow Completing Time. 98

FOAF Friend Of A Friend. 13, 139, 161, 171

FRUCT Finnish-Russian University Cooperations in Telecommunications. 62

FSC Food Supply Chain. 30

HABITAT Home Assistance Basata su Internet of Things per l'Autonomia di Tutti. 83

HCR Health-Care Records. 30

HetIoT Heterogeneous Internet of Things. 33

HTTP Hyper-Text Transfer Protocol. 22, 35, 36, 61, 82, 85, 91, 92, 93, 190

HVAC Heating, Ventilation, Air Conditioning. 30, 192

ICT Information and Communication Technologies. 7, 9, 21, 29, 32, 202

IETF Internet Engineering Task Force. 91

IIoT Industrial IoT. 32

IoE Internet of Energy. 39, 173, 195

IoMusT Internet of Musical Things. 15, 24, 60, 90, 92, 95, 96, 99, 187, 192, 221, 234, 235, 236, 237, 239, 243, 244, 253

IoT Internet of Things. 7, 9, 13, 21, 22, 23, 24, 27, 28, 29, 30, 31, 32, 33, 35, 47, 54, 61, 63, 65, 66, 73, 90, 91, 108, 109, 139, 141, 153, 173, 187, 197, 198, 199, 210, 211, 216, 218, 220, 234, 243

IRI International Resource Identifier. 23, 34, 44, 159

JSAP JSON Semantic Application Profile. 85, 87, 193

JSON JavaScript Object Notation. 35, 69, 70, 71, 84, 87, 95, 146, 151, 229

JSON-LD JSON for Linked Data. 233

JSSAP JSON Smart Space Access Protocol. 70, 71, 118, 121

JU Joint Undertaking. 37, 62, 195

KB knowledge base. 23, 25, 44, 48, 50, 52, 61, 65, 77, 78, 80, 95, 108, 119, 120, 140, 141, 142, 144, 151, 153, 155, 156, 161, 162, 163, 171, 173, 179, 243

KP Knowledge Processor. 37, 62, 63, 65, 66, 68, 70, 71, 79, 85, 119, 120, 121, 189

LD Linked Data. 141, 151

LOD level of details. 159, 179

LOV Linked Open Vocabularies. 140

LUBM Leigh University BenchMark. 108, 109, 117, 130

LUTT Look-Up Triples Table. 51, 52, 53, 54, 55, 56, 66, 67, 114, 115, 116, 129, 131, 134

LW Last Will. 80, 81, 82

M2M Machine to Machine. 61, 198, 199, 200

MCU Microcontroller Unit. 214

MOM Message-oriented Middleware. 24, 109, 117, 121, 207

MPPT Maximum Power Point Tracking. 214

MQTT MQ Telemetry Transport. 61, 80, 90, 91, 199, 200

NFC Near Field Communication. 29

OSC Open Sound Control. 234

OSGi Open Services Gateway initiative. 60, 62, 72, 73, 74, 78, 79, 80, 81, 99, 188, 198

OSMA One-Step Matching Algorithm. 46

OTM On-the-Move. 208

OWL Web Ontology Language. 23, 34, 117, 119, 120, 141, 151, 155, 179

PD Public Domain. 222

PES Performance Evaluation Suite. 108, 117, 118, 119, 120, 121, 134, 250

PGV Paged Graph Visualization. 149

PI Performance Indicator. 109, 110, 111, 121

PMM Power Management Module. 218

PNG Portable Network Graphics. 117, 118

POI Point Of Interest. 30

PS PlaySound. 223, 224, 226, 230, 233

PU Persistent Update. 61, 76, 77, 78, 99

PV Photo-Voltaic. 201, 202, 203, 213, 214, 219, 253

QMUL Queen Mary University of London. 25, 90, 192, 221, 222, 233, 236

RDF Resource Description Framework. 23, 24, 34, 35, 37, 43, 44, 46, 47, 48, 52, 54, 61, 62, 66, 67, 68, 69, 70, 76, 82, 83, 92, 109, 111, 116, 117, 119, 140, 141, 142, 145, 149, 153, 154, 155, 156, 161, 165, 172, 179, 189, 226, 229, 233, 236, 244

RDFS RDF Schema. 23, 34, 141, 151, 155, 179

RECOCAPE REinforcing COopeartion CAPacity of Egypt in embedded ubiquitous computing. 39

REST REpresentational State Transfer. 22, 31, 35, 91, 200, 201, 218, 233

RFC Request For Comment. 91

RFID Radio-Frequency Identification. 7, 9, 29, 30, 216

SAP Semantic Application Profile. 83, 87, 121, 192

SAREF Smart Appliances REFerence. 124

SE Secure Event. 84

SENS Semantic Event Notification Service. 46, 61

SEPA SPARQL Event Processing Architecture. 7, 9, 39, 46, 47, 56, 60, 61, 82, 83, 84, 85, 87, 90, 91, 92, 93, 99, 121, 123, 128, 129, 131, 132, 134, 135, 177, 180, 188, 189, 190, 191, 192, 193, 224, 226, 235, 236, 239, 243, 244

SG Smart Grid. 173, 203, 205, 208, 209

SHriMP Simple Hierarchical Multi-Perspective. 146

SIB Semantic Information Broker. 37, 60, 62, 69, 71, 72, 73, 74, 76, 78, 79, 80, 81, 99, 118, 119, 120, 121, 188, 208, 243

SM Semantic Mediator. 223

SOFIA Smart Objects for Intelligent Applications. 37, 62

SP^{2}B SPARQL Performance Benchmark. 108, 117

SPARQL SPARQL Protocol and RDF Query Language. 23, 37, 44, 45, 46, 47, 48, 50, 51, 52, 54, 61, 63, 64, 66, 67, 68, 69, 71, 73, 76, 77, 78, 82, 83, 84, 85, 87, 91, 92, 93, 95, 96, 99, 108, 111, 116, 117, 119, 120, 122, 123, 124, 128, 129, 130, 131, 133, 134, 156, 159, 161, 163, 164, 165, 166, 167, 169, 177, 179, 180, 189, 190, 243

SPS Semantic Publish-Subscribe. 56, 60, 62, 65, 66, 68, 78, 99, 108, 110, 112, 116, 188, 243

SPU Subscription Processing Unit. 45, 49, 50, 52, 54, 66, 67, 69, 86, 111, 116

SRQ Subscription Requests Queue. 66, 69

SS smart space. 63, 65

SSAP Smart Space Access Protocol. 37, 62, 64, 69, 70, 71, 82, 99, 117, 118

SVG Scalable Vector Graphics. 117, 118, 120, 151

SW Semantic Web. 23, 24, 27, 33, 34, 37, 43, 61, 108, 109, 122, 140, 141, 142, 144, 146, 161, 177, 179, 222, 223, 234, 243, 244

SWAMP Smart WAter Management Platform. 83, 244

SWoT Semantic Web of Things. 7, 9, 23, 24, 25, 27, 37, 39, 43, 44, 55, 99, 108, 109, 121, 122, 123, 128, 134, 135, 187, 188, 189, 192, 211, 221, 224, 225, 233, 239, 243, 244, 252

SWRL Semantic Web Rule Language. 61

TD Thing Description. 22, 23, 35, 36, 37, 122, 131, 132, 134, 189, 190, 191, 192

TEG thermo-electric generator. 213, 214, 216, 253

TMAP Technology-Mediated Audience Participation. 237

UI user interface. 146, 154, 156, 159, 161, 163, 164, 177, 179, 251

URI Uniform Resource Identifier. 36, 94, 95, 111, 122, 228, 229, 230

URQ Update Requests Queue. 66, 67, 69

V2G Vehicle to Grid. 207

WoT Web of Things. 9, 22, 23, 35, 36, 37, 83, 122, 123, 189, 190, 225, 226

WS WebSocket. 36, 189, 190

WSAN Wireless Sensor and Actuator Networks. 32, 211, 212, 213, 216, 218, 220, 244

WT Web Thing. 37, 108, 122, 123, 124, 125, 131, 133, 135, 189, 190, 191, 192, 224, 225, 232, 239

XML eXtensible Markup Language. 34, 64, 71

YAML Yet Another Markup Language. 87, 154, 155, 159

YSAP YAML Semantic Application Profile. 87

Bibliography

[1] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. The internet of things—a survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015. (cit. on pp. 21)

[2] Rob Van Kranenburg. *The Internet of Things: A critique of ambient technology and the all-seeing network of RFID*. Institute of Network Cultures, 2008. (cit. on pp. 21)

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010. (cit. on pp. 21)

[4] Alicia Asin and David Gascon. 50 sensor applications for a smarter world. *Libelium Comunicaciones Distribuidas, Tech. Rep*, 2012. (cit. on pp. 21, 30, 141)

[5] Mark Weiser. The computer for the 21 st century. *Scientific american*, 265(3):94–105, 1991. (cit. on pp. 21)

[6] Mahadev Satyanarayanan et al. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001. (cit. on pp. 21)

[7] Bill N Schilit and Marvin M Theimer. Disseminating active map information to mobile hosts. *IEEE network*, 8(5):22–32, 1994. (cit. on pp. 21, 29, 247)

[8] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, 16(1):414–454, 2014. (cit. on pp. 21, 22, 28, 32, 44)

[9] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001. (cit. on pp. 21, 22, 44)

[10] Katsuhiro Naito. A survey on the internet-of-things: Standards, challenges and future prospects. *Journal of Information Processing*, 25:23–31, 2017. (cit. on pp. 22)

[11] P. Desai, A. Sheth, and P. Anantharam. Semantic gateway as a service architecture for iot interoperability. In *IEEE International Conference on Mobile Services*, pages 313–319. IEEE, 2015. (cit. on pp. 22)

[12] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *Systems Engineering Symposium (ISSE), 2017 IEEE International*, pages 1–7. IEEE, 2017. (cit. on pp. 22, 91)

[13] Mohab Aly, Foutse Khomh, Yann-Gaël Guéhéneuc, Hironori Washizaki, and Soumaya Yacout. Is fragmentation a threat to the success of the internet of things? *IEEE Internet of Things Journal*, 2018. (cit. on pp. 22, 32, 33)

[14] 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*, 2(3):94–105, 2011. (cit. on pp. 22, 234)

[15] Soumya Kanti Datta, Christian Bonnet, Hamza Baqa, Mengxuan Zhao, Franck Le Gall, and Easy Global Market. Approach for semantic interoperability testing in internet of things, 2018. (cit. on pp. 22, 23)

[16] Paulo Carvalho, Patrik Hitzelberger, Benoît Otjacques, Fatma Bouali, and Gilles Venturini. Using information visualization to support open data integration. In *International Conference on Data Management Technologies and Applications*, pages 1–15. Springer, 2014. (cit. on pp. 22)

[17] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the Web of Things. *Proc. of 2010 Internet of Things (IOT'10)*, pages 1–8, 2010. (cit. on pp. 22)

[18] Debasis Bandyopadhyay and Jaydip Sen. Internet of things: Applications and challenges in technology and standardization. *Wireless Personal Communications*, 58(1):49–69, 2011. (cit. on pp. 22)

[19] Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, volume 15, 2009. (cit. on pp. 22, 35)

[20] Sebastian Kaebisch and Takuki Kamiya. Web of things (wot) thing description, Jul 2017. (cit. on pp. 23, 122, 189)

[21] World Wide Web Consortium et al. Web of things (wot) architecture (editor's draft, 26 september 2018). 09 2018. (cit. on pp. 23, 37)

[22] Dennis Pfisterer, Kay Romer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth, et al. Spitfire: toward a semantic web of things. *IEEE Communications Magazine*, 49(11):40–48, 2011. (cit. on pp. 23, 37, 61)

[23] Floriano Scioscia and Michele Ruta. Building a semantic web of things: issues and perspectives in information compression. In *Semantic Computing, 2009. ICSC'09. IEEE International Conference on*, pages 589–594. IEEE, 2009. (cit. on pp. 23, 24, 37)

[24] Antonio J Jara, Alex C Olivieri, Yann Bocchi, Markus Jung, Wolfgang Kastner, and Antonio F Skarmeta. Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence. *International Journal of Web and Grid Services*, 10(2-3):244–272, 2014. (cit. on pp. 23, 24, 108, 122)

[25] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI Global, 2011. (cit. on pp. 23)

[26] Guinard Dominique. *A web of things application architecture-Integrating the real-world into the web*. PhD thesis, ETH, 2011. (cit. on pp. 23)

[27] Vlad Mihai Trifa. *Building blocks for a participatory web of things: devices, infrastructures, and programming frameworks*. PhD thesis, ETH, 2011. (cit. on pp. 23)

[28] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001. (cit. on pp. 23, 34)

[29] Ora Lassila, Ralph R Swick, et al. Resource description framework (rdf) model and syntax specification. 1998. (cit. on pp. 23)

[30] Natalya F Noy, Deborah L McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001. (cit. on pp. 23, 34)

[31] Dan Brickley, Ramanathan V Guha, and Brian McBride. Rdf schema 1.1. *W3C recommendation*, 25:2004–2014, 2014. (cit. on pp. 23, 34)

[32] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004. (cit. on pp. 23, 34)

[33] Eric Prud, Andy Seaborne, et al. Sparql query language for rdf. 2006. (cit. on pp. 23, 35, 44)

[34] Andy Seaborne, Geetha Manjunath, Chris Bizer, John Breslin, Souripriya Das, Ian Davis, Steve Harris, Kingsley Idehen, Olivier Corby, Kjetil Kjernsmo, et al. Sparql/update: A language for updating rdf graphs. *W3c member submission*, 15, 2008. (cit. on pp. 23, 35, 44)

[35] Michael Martin, Jörg Unbehauen, and Sören Auer. Improving the performance of semantic web applications with sparql query caching. In *Extended Semantic Web Conference*, pages 304–318. Springer, 2010. (cit. on pp. 24)

[36] Shaun Howell, Yacine Rezgui, and Thomas Beach. Water utility decision support through the semantic web of things. *Environmental Modelling & Software*, 102(C):94–114, 2018. (cit. on pp. 24)

[37] S. Calbimonte, J.P.and Sarni, J. Eberle, and K. Aberer. XGSN: An Open-source Semantic Sensing Middleware for the Web of Things. In *Proceedings of the International Workshop on the Foundations, Technologies and Applications of the Geospatial Web*, pages 51–66, 2014. (cit. on pp. 24)

[38] Michele Albano, Luis Lino Ferreira, Luís Miguel Pinho, and Abdel Rahman Alkhawaja. Message-oriented middleware for smart grids. *Computer Standards & Interfaces*, 38:133–143, 2015. (cit. on pp. 24, 235)

[39] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *International symposium on handheld and ubiquitous computing*, pages 304–307. Springer, 1999. (cit. on pp. 27)

[40] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, 1992. (cit. on pp. 27)

[41] Jakob E Bardram. Applications of context-aware computing in hospital work: examples and design principles. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1574–1579. ACM, 2004. (cit. on pp. 28)

[42] Daqing Zhang, Tao Gu, and Xiaohang Wang. Enabling context-aware smart home with semantic web technologies. *International Journal of Human-friendly Welfare Robotic Systems*, 6(4):12–20, 2005. (cit. on pp. 28)

[43] Jiafu Wan, Daqiang Zhang, Shengjie Zhao, Laurence Yang, and Jaime Lloret. Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions. *IEEE Communications Magazine*, 52(8):106–113, 2014. (cit. on pp. 28)

[44] Daniel Salber, Anind K Dey, and Gregory D Abowd. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 434–441. ACM, 1999. (cit. on pp. 28)

[45] Harry Chen, Tim Finin, Anupam Joshi, Lalana Kagal, Filip Perich, and Dipanjan Chakraborty. Intelligent agents meet the semantic web in smart spaces. *IEEE Internet computing*, 8(6):69–79, 2004. (cit. on pp. 28)

[46] Craig Schlenoff and Michael Uschold, editors. *A Context Broker for Building Smart Meeting Rooms*, Stanford, California, March 2004. AAAI Press, Menlo Park, CA. (cit. on pp. 28)

[47] Vagan Terziyan, Olena Kaykova, and Dmytro Zhovtobryukh. Ubiroad: Semantic middleware for context-aware smart road environments. In *Internet and web applications and services (iciw), 2010 fifth international conference on*, pages 295–302. IEEE, 2010. (cit. on pp. 28)

[48] Fabio Viola, Alfredo D'Elia, Dmitry Korzun, Ivan Galov, Alexey Kashevnik, and Sergey Balandin. The m3 architecture for smart spaces: Overview of semantic information broker implementations. In *Open Innovations Association (FRUCT), 2016 19th Conference of*, pages 264–272. IEEE, 2016. (cit. on pp. 28)

[49] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of ad Hoc and ubiquitous Computing*, 2(4):263–277, 2007. (cit. on pp. 28)

[50] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop on advanced context modelling, reasoning and management, UbiComp*, volume 4, pages 34–41, 2004. (cit. on pp. 28)

[51] Reto Krummenacher and Thomas Strang. Ontology-based context modeling. In *Proceedings Third Workshop on Context-Aware Proactive Systems (CAPS 2007)(June 2007)*, page 22, 2007. (cit. on pp. 28)

[52] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 85–90. IEEE, 1994. (cit. on pp. 28)

[53] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009. (cit. on pp. 29)

[54] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends*, 1(2011):9–52, 2011. (cit. on pp. 29)

[55] Alessandro Bassi and Geir Horn. Internet of things in 2020: A roadmap for the future. *European Commission: Information Society and Media*, 22:97–114, 2008. (cit. on pp. 29)

[56] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012. (cit. on pp. 30)

[57] Feng Gao, Muhammad Intizar Ali, and Alessandra Mileo. Semantic discovery and integration of urban data streams. In *Proceedings of the Fifth International Conference on Semantics for Smarter Cities - Volume 1280*, S4SC'14, pages 15–30, Aachen, Germany, Germany, 2014. CEUR-WS.org. (cit. on pp. 30)

[58] Juan Rico, Juan Sancho, Bruno Cendon, and Miguel Camus. Parking easier by using context information of a smart city: Enabling fast search and management of parking resources. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 1380–1385. IEEE, 2013. (cit. on pp. 30)

[59] Luis Sánchez, Verónica Gutiérrez, José Antonio Galache, Pablo Sotres, Juan Ramón Santana, Javier Casanueva, and Luis Muñoz. Smartsantander: Experimentation and service provision in the smart city. In *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, pages 1–6. IEEE, 2013. (cit. on pp. 30)

[60] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014. (cit. on pp. 30)

[61] Theodoros Anagnostopoulos, Arkady Zaslavsky, Stefanos Georgiou, and Sergey Khoruzhnikov. High capacity trucks serving as mobile depots for waste collection in iot-enabled smart cities. In *Conference on Smart Spaces*, pages 80–94. Springer, 2015. (cit. on pp. 30)

[62] Alexey Medvedev, Petr Fedchenkov, Arkady Zaslavsky, Theodoros Anagnostopoulos, and Sergey Khoruzhnikov. Waste management as an iot-enabled service in smart cities. In *Conference on Smart Spaces*, pages 104–115. Springer, 2015. (cit. on pp. 30)

[63] Alexander Smirnov, Alexey Kashevnik, Sergey I Balandin, and Santa Laizane. Intelligent mobile tourist guide. In *Internet of things, smart spaces, and next generation networking*, pages 94–106. Springer, 2013. (cit. on pp. 30, 39)

[64] Alexander Smirnov, Alexey Kashevnik, Nikolay Shilov, Nikolay Teslya, and Anton Shabaev. Mobile application for guiding tourist activities: tourist assistant-tais. In *Open Innovations Association (FRUCT16), 2014 16th Conference of*, pages 95–100. IEEE, 2014. (cit. on pp. 30)

[65] Nicola Bui and Michele Zorzi. Health care applications: a solution based on the internet of things. In *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies*, page 131. ACM, 2011. (cit. on pp. 30)

[66] Iuliana Chiuchisan, Hariton-Nicolae Costin, and Oana Geman. Adopting the internet of things technologies in health care systems. In *Electrical and Power Engineering (EPE), 2014 International Conference and Exposition on*, pages 532–535. IEEE, 2014. (cit. on pp. 30)

[67] SM Riazul Islam, Daehan Kwak, MD Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. The internet of things for health care: a comprehensive survey. *IEEE Access*, 3:678–708, 2015. (cit. on pp. 30)

[68] A. J. Jara, F. J. Belchi, A. F. Alcolea, J. Santa, M. A. Zamora-Izquierdo, and A. F. Gómez-Skarmeta. A pharmaceutical intelligent information system to detect allergies and adverse drugs reactions based on internet of things. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 809–812, March 2010. (cit. on pp. 30)

[69] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660, 2013. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. (cit. on pp. 30)

[70] Angelika Dohr, R Modre-Opsrian, Mario Drobics, Dieter Hayn, and Günter Schreier. The internet of things for ambient assisted living. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 804–809. Ieee, 2010. (cit. on pp. 30)

[71] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014. (cit. on pp. 30, 33)

[72] L. D. Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov 2014. (cit. on pp. 31)

[73] Qinghai Ou, Yan Zhen, Xiangzhen Li, Yiying Zhang, and Lingkang Zeng. Application of internet of things in smart grid power transmission. In *Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on*, pages 96–100. IEEE, 2012. (cit. on pp. 31)

[74] S Naga Jyothi and K Vijaya Vardhan. Design and implementation of real time security surveillance system using iot. In *Communication and Electronics Systems (ICCES), International Conference on*, pages 1–5. IEEE, 2016. (cit. on pp. 31)

[75] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. Uav-based iot platform: A crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128–134, 2017. (cit. on pp. 31)

[76] Mauro AA da Cruz, Joel José PC Rodrigues, Jalal Al-Muhtadi, Valery V Korotaev, and Victor Hugo C de Albuquerque. A reference model for internet of things middleware. *IEEE Internet of Things Journal*, 5(2):871–883, 2018. (cit. on pp. 32)

[77] Pablo Fernández, José Miguel Santana, Sebastián Ortega, Agustín Trujillo, José Pablo Suárez, Conrado Domínguez, Jaisiel Santana, and Alejandro Sánchez. Smartport: a platform for sensor data monitoring in a seaport based on fiware. *Sensors*, 16(3):417, 2016. (cit. on pp. 32)

[78] Carlos Kamienski, João Kleinschmidt, Juha-Pekka Soininen, Kari Kolehmainen, Luca Roffia, Marcos Visoli, Rodrigo Filev Maia, and Stenio Fernandes. Swamp: Smart water management platform overview and security challenges. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018. (cit. on pp. 32, 83)

[79] Kiran Jot Singh and Divneet Singh Kapoor. Create your own internet of things: A survey of iot platforms. *IEEE Consumer Electronics Magazine*, 6(2):57–68, 2017. (cit. on pp. 32)

[80] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, 17(4):2347–2376, 2015. (cit. on pp. 32)

[81] Zhonggui Ma, Xinsheng Shang, Xinxi Fu, and Feng Luo. The architecture and key technologies of internet of things in logistics. In *International Conference on Cyberspace Technology (CCT 2013)*, pages 464–468, Nov 2013. (cit. on pp. 32)

[82] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the architecture of internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–484. IEEE, 2010. (cit. on pp. 32, 33)

[83] Mari Carmen Domingo. An overview of the internet of things for people with disabilities. *Journal of Network and Computer Applications*, 35(2):584–596, 2012. (cit. on pp. 32)

[84] K. Wang, Y. Wang, Y. Sun, S. Guo, and J. Wu. Green industrial internet of things architecture: An energy-efficient perspective. *IEEE Communications Magazine*, 54(12):48–54, December 2016. (cit. on pp. 32)

[85] Yujun Ma, Yulei Wang, Jun Yang, Yiming Miao, and Wei Li. Big health application system based on health internet of things and big data. *IEEE Access*, 2016. (cit. on pp. 33)

[86] M. Aazam and E. N. Huh. Fog computing and smart gateway based communication for cloud of things. In *2014 International Conference on Future Internet of Things and Cloud*, pages 464–470, Aug 2014. (cit. on pp. 33)

[87] Giancarlo Fortino and Wilma Russo. Towards a cloud-assisted and agent-oriented architecture for the internet of things. In *WOA@ AI* IA*, pages 60–65, 2013. (cit. on pp. 33)

[88] Tie Qiu, Ning Chen, Keqiu Li, Mohammed Atiquzzaman, and Wenbing Zhao. How can heterogeneous internet of things build our future: A survey. *IEEE Communications Surveys & Tutorials*, 2018. (cit. on pp. 33)

[89] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260. IEEE, 2012. (cit. on pp. 33)

[90] Dongcai Shi, Jianwei Yin, Yiyuan Li, Jianfeng Qian, and Jinxiang Dong. An rdf-based publish/subscribe system. In *Semantics, Knowledge and Grid, Third International Conference on*, pages 342–345. IEEE, 2007. (cit. on pp. 34, 61)

[91] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007. (cit. on pp. 35)

[92] Matthias Kovatsch. Coap for the web of things: from tiny resource-constrained devices to the web browser. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 1495–1504. ACM, 2013. (cit. on pp. 35, 61, 62)

[93] Dominique Guinard, Iulia Ion, and Simon Mayer. In search of an internet of things service architecture: Rest or ws-*? a developers' perspective. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 326–337. Springer, 2011. (cit. on pp. 35)

[94] Dominique Guinard and Vlad Trifa. *Building the web of things: with examples in node. js and raspberry pi*. Manning Publications Co., 2016. (cit. on pp. 35, 91, 108, 189)

[95] Victor Charpenay, Sebastian Käbisch, and Harald Kosch. Introducing thing descriptions and interactions: An ontology for the web of things. In *SR+ SWIT@ ISWC*, pages 55–66, 2016. (cit. on pp. 37, 225)

[96] Thing description model. http://w3c.github.io/wot/w3c-wot-td-ontology.owl. (cit. on pp. 37)

[97] Haitham S Hamza, Enas Ashraf, Azza K Nabih, Mahmoud M Abdallah, Ahmed M Gamaleldin, Alfredo D'Elia, Hadeal Ismail, Shourok Alaa, Kamilia Hosny, Aya Khattab, et al. Design and implementation of an interoperable and extendable smart home semantic architecture using smart-m3 and soa. In *The Tenth International Conference on Networking and Services, ICNS*, pages 48–53, 2014. (cit. on pp. 39)

[98] Luca Bedogni, Luciano Bononi, Alfredo D'Elia, Marco Di Felice, Simone Rondelli, and Tullio Salmon Cinotti. A mobile application to assist electric vehicles' drivers with charging services. In *Next Generation Mobile Apps, Services and Technologies (NG-MAST), 2014 Eighth International Conference on*, pages 78–83. IEEE, 2014. (cit. on pp. 39, 208)

[99] Simone Rondelli. *Un Framework di analisi e di servizi innovativi per la mobilità veicolare elettrica*. PhD thesis. (cit. on pp. 39, 208)

[100] Roberta Gazzarata, Fabio Vergari, Tullio Salmon Cinotti, and Mauro Giacomini. A standardized soa for clinical data interchange in a cardiac telemonitoring environment. *IEEE J. Biomedical and Health Informatics*, 18(6):1764–1774, 2014. (cit. on pp. 39)

[101] Alfredo D'Elia, Fabio Viola, Federico Montori, Marco Di Felice, Luca Bedogni, Luciano Bononi, Alberto Borghetti, Paolo Azzoni, Paolo Bellavista, Daniele Tarchi, et al. Impact of interdisciplinary research on planning, running, and managing electromobility as a smart grid extension. *Access, IEEE*, 3:2281–2305, 2015. (cit. on pp. 39, 197, 206, 207, 210, 253)

[102] Jerker Delsing. Application system and services: Design and implementation-a cookbook. In *IoT Automation*, pages 175–196. CRC Press, 2017. (cit. on pp. 39, 210)

[103] Alfredo D'Elia, Fabio Viola, Federico Montori, Paolo Azzoni, and Matteo Maiero. Electro mobility automation through the arrowhead framework. In *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, pages 5246–5252. IEEE, 2016. (cit. on pp. 39, 197, 200, 201, 203, 204, 205, 252, 253)

[104] Dmitry G Korzun, Ivan V Galov, Alexey M Kashevnik, Nikolay G Shilov, Kirill Krinkin, and Yury Korolev. Integration of smart-m3 applications: Blogging in smart conference. In *Smart Spaces and Next Generation Wired/Wireless Networking*, pages 51–62. Springer, 2011. (cit. on pp. 39)

[105] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131, 2003. (cit. on pp. 44, 61, 63)

[106] Martin Murth and Eva Kühn. A semantic event notification service for knowledge-driven coordination. 2008. (cit. on pp. 46, 61)

[107] Haris Abdullah, Mikko Rinne, Seppo Törmä, and Esko Nuutila. Efficient matching of sparql subscriptions using rete. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 372–377. ACM, 2012. (cit. on pp. 46)

[108] Mikko Rinne, Haris Abdullah, Seppo Törmä, and Esko Nuutila. Processing heterogeneous rdf events with standing sparql update rules. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 797–806. Springer, 2012. (cit. on pp. 46)

[109] Mikko Rinne, Esko Nuutila, and Seppo Törmä. Instans: high-performance event processing with standard rdf and sparql. In *Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914*, pages 101–104. Citeseer, 2012. (cit. on pp. 46)

[110] Charles L Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. In *Readings in Artificial Intelligence and Databases*, pages 547–559. Elsevier, 1988. (cit. on pp. 46)

[111] Laurent Pellegrino, Françoise Baude, and Iyad Alshabani. Towards a scalable cloud-based rdf storage offering a pub/sub query service. *CLOUD COMPUTING*, 2012:243–246, 2012. (cit. on pp. 46)

[112] Laurent Pellegrino, Fabrice Huet, Françoise Baude, and Amjad Alshabani. A distributed publish/subscribe system for rdf data. In *International Conference on Data Management in Cloud, Grid and P2P Systems*, pages 39–50. Springer, 2013. (cit. on pp. 46)

[113] Srdjan Komazec, Davide Cerri, and Dieter Fensel. Sparkwave: continuous schema-enhanced pattern matching over rdf data streams. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 58–68. ACM, 2012. (cit. on pp. 47)

[114] Sören Auer and Heinrich Herre. A versioning and evolution framework for rdf knowledge bases. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 55–69. Springer, 2006. (cit. on pp. 47)

[115] Manuel Fiorelli, Maria Teresa Pazienza, Armando Stellato, and Andrea Turbati. Version control and change validation for rdf datasets. In *Research Conference on Metadata and Semantics Research*, pages 3–14. Springer, 2017. (cit. on pp. 47)

[116] Sven Groppe, Jinghua Groppe, Dirk Kukulenz, and Volker Linnemann. A sparql engine for streaming rdf data. In *Signal-Image Technologies and Internet-Based System, 2007. SITIS'07. Third International IEEE Conference on*, pages 167–174. IEEE, 2007. (cit. on pp. 47, 48)

[117] Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming sparql-extending sparql to process data streams. In *European Semantic Web Conference*, pages 448–462. Springer, 2008. (cit. on pp. 47)

[118] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, and Michael Grossniklaus. An execution environment for c-sparql queries. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 441–452. ACM, 2010. (cit. on pp. 47)

[119] Jean-Paul Calbimonte, Oscar Corcho, and Alasdair JG Gray. Enabling ontology-based access to streaming data sources. In *International Semantic Web Conference*, pages 96–111. Springer, 2010. (cit. on pp. 47)

[120] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. Ep-sparql: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, pages 635–644. ACM, 2011. (cit. on pp. 47)

[121] Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *International Semantic Web Conference*, pages 370–388. Springer, 2011. (cit. on pp. 47)

[122] Vicky Papavasiliou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. High-level change detection in RDF(S) KBs. *ACM Transactions on Database Systems*, 38(1):1–42, 2013. (cit. on pp. 48)

[123] Vicky Papavassiliou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. On detecting high-level changes in rdf/s kbs. In *International Semantic Web Conference*, pages 473–488. Springer, 2009. (cit. on pp. 48)

[124] Luca Roffia, Francesco Morandi, Jussi Kiljander, Alfredo D'Elia, Fabio Vergari, Fabio Viola, Luciano Bononi, and Tullio Salmon Cinotti. A semantic publish-subscribe architecture for the internet of things. *IEEE Internet of Things Journal*, 3(6):1274–1296, 2016. (cit. on pp. 54, 55, 83, 99, 134, 180)

[125] Emanuele Della Valle, Stefano Ceri, Frank Van Harmelen, and Dieter Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6), 2009. (cit. on pp. 61)

[126] Danh Le-Phuoc, Josiane Xavier Parreira, and Manfred Hauswirth. Linked stream data processing. In *Reasoning Web International Summer School*, pages 245–289. Springer, 2012. (cit. on pp. 61)

[127] Jinling Wang, Beihong Jin, and Jing Li. An ontology-based publish/subscribe system. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 232–253. Springer-Verlag New York, Inc., 2004. (cit. on pp. 61)

[128] Paul-Alexandru Chirita, Stratos Idreos, Manolis Koubarakis, and Wolfgang Nejdl. Publish/subscribe for rdf-based p2p networks. In *European Semantic Web Symposium*, pages 182–197. Springer, 2004. (cit. on pp. 61)

[129] Kristian Ellebaek Kjaer and Klaus Marius Hansen. Modeling and implementing ontology-based publish/subscribe using semantic web technologies. In *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 63–71. IEEE, 2010. (cit. on pp. 61)

[130] Martin Murth and Eva Kühn. A semantic event notification service for knowledge-driven coordination. 2010. (cit. on pp. 61)

[131] Martin Murth and Eva Kühn. Knowledge-based coordination with a reliable semantic subscription mechanism. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1374–1380. ACM, 2009. (cit. on pp. 61)

[132] Martin Murth, Dietmar Winkler, Stefan Biffl, Eva Kühn, and Thomas Moser. Performance testing of semantic publish/subscribe systems. In Robert Meersman, Tharam Dillon, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2010 Workshops*, pages 45–46, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. (cit. on pp. 61, 109)

[133] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou. Task computing–the semantic web meets pervasive computing. In *International semantic web conference*, pages 866–881. Springer, 2003. (cit. on pp. 61)

[134] Harry Chen, Tim Finin, and Amupam Joshi. Semantic web in the context broker architecture. Technical report, MARYLAND UNIV BALTIMORE DEPT OF COMPUTER SCIENCE AND ELECTRICAL ENGINEERING, 2005. (cit. on pp. 61)

[135] L Lamorte, CA Licciardi, M Marengo, A Salmeri, P Mohr, G Raffa, L Roffia, M Pettinari, and T Salmon Cinotti. A platform for enabling context aware telecommunication services. In *Third workshop on context awareness for proactive systems*, 2007. (cit. on pp. 61)

[136] Zhexuan Song, Alvaro A Cárdenas, and Ryusuke Masuoka. Semantic middleware for the internet of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010. (cit. on pp. 61)

[137] Juan Ignacio Vazquez, Diego López de Ipiña, and Inigo Sedano. Soam: A web-powered architecture for designing and deploying pervasive semantic devices. *International Journal of Web Information Systems*, 2(3/4):212–224, 2007. (cit. on pp. 61)

[138] Graham Thomson, Sébastien Bianco, Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny. Amigo aware services. In *European Conference on Ambient Intelligence*, pages 385–390. Springer, 2007. (cit. on pp. 61)

[139] Jaeho Kim and Jang-Won Lee. Openiot: An open service framework for the internet of things. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 89–93. IEEE, 2014. (cit. on pp. 61)

[140] D Locke. Mq telemetry transport (mqtt) v3. 1 protocol specification. ibm developerworks technical library (2010), 2010. (cit. on pp. 61, 91, 199)

[141] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). Technical report, 2014. (cit. on pp. 61, 62, 90, 91, 216)

[142] Steve Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6), 2006. (cit. on pp. 61, 90)

[143] Carsten Bormann, Mehmet Ersue, and Ari Keranen. Terminology for constrained-node networks. Technical report, 2014. (cit. on pp. 62)

[144] Francesco Morandi, Luca Roffia, Alfredo D'Elia, Fabio Vergari, and Tullio Salmon Cinotti. Redsib: a smart-m3 semantic information broker implementation. In *Proc. 12th Conf. of Open Innovations Association FRUCT and Seminar on e-Tourism*, pages 86–98. SUAI, 2012. (cit. on pp. 62, 66, 73, 78)

[145] Ivan V Galov, Aleksandr A Lomov, and Dmitry G Korzun. Design of semantic information broker for localized computing environments in the internet of things. In *Open*

Innovations Association (FRUCT), 2015 17th Conference of, pages 36–43. IEEE, 2015. (cit. on pp. 62)

[146] Sergey Balandin and Heikki Waris. Key properties in the development of smart spaces. In *International conference on universal access in human-computer interaction*, pages 3–12. Springer, 2009. (cit. on pp. 62)

[147] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006. (cit. on pp. 63)

[148] Alfredo D'Elia, Fabio Viola, Luca Roffia, Paolo Azzoni, and Tullio Salmon Cinotti. Enabling interoperability in the internet of things: A osgi semantic information broker implementation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 13(1):147–167, 2017. (cit. on pp. 64, 78, 79, 80, 81, 99, 249, 250)

[149] Susanna Pantsar-Syväniemi, Eila Ovaska, Susanna Ferrari, Tullio Salmon Cinotti, Guido Zamagni, Luca Roffia, Sandra Mattarozzi, and Valerio Nannini. Case study: Context-aware supervision of a smart maintenance process. In *2011 IEEE/IPSJ International Symposium on Applications and the Internet*, pages 309–314. IEEE, 2011. (cit. on pp. 68)

[150] Fabio Viola, Alfredo D'Elia, Luca Roffia, and Tullio Salmon Cinotti. A modular lightweight implementation of the smart-m3 semantic information broker. In *18th FRUCT*, pages 370–377, 2016. (cit. on pp. 70, 77, 99, 118, 121, 249)

[151] Christian Bizer and Andreas Schultz. The berlin sparql benchmark, 2009. (cit. on pp. 78, 109)

[152] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005. (cit. on pp. 78, 109, 118, 130)

[153] R Garcìa-Castro et al. Web semantics: Science, services and agents on the world wide web, special issue on evaluation of semantic technologies (vol. 21), 2013. (cit. on pp. 78)

[154] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. Spˆ 2bench: a sparql performance benchmark. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 222–233. IEEE, 2009. (cit. on pp. 78, 108, 118, 119)

[155] Alfredo D'Elia, Paolo Azzoni, Fabio Viola, Cristiano Aguzzi, Luca Roffia, and Tullio Salmon Cinotti. The osgi sib: A resilient semantic solution for the internet of things. In *Semantic Web Science and Real-World Applications*, pages 48–74. IGI Global, 2019. (cit. on pp. 80, 99)

[156] M Saifur Rahman, Md Yusuf Sarwar Uddin, Tahmid Hasan, M Sohel Rahman, and M Kaykobad. Using adaptive heartbeat rate on long-lived tcp connections. *IEEE/ACM Transactions on Networking (TON)*, 26(1):203–216, 2018. (cit. on pp. 82)

[157] Francesco Antoniazzi, Giacomo Paolini, Luca Roffia, Diego Masotti, Alessandra Costanzo, and Tullio Salmon Cinotti. A web of things approach for indoor position monitoring of elderly and impaired people. In *Open Innovations Association (FRUCT), 2017 21st Conference of*, pages 51–56. IEEE, 2017. (cit. on pp. 83)

[158] Frederic Font, Tim Brookes, George Fazekas, Martin Guerber, Amaury La Burthe, David Plans, Mark D Plumbley, Meir Shaashua, Wenwu Wang, and Xavier 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. (cit. on pp. 83)

[159] Luca Roffia, Paolo Azzoni, Cristiano Aguzzi, Fabio Viola, Francesco Antoniazzi, and Tullio Salmon Cinotti. Dynamic linked data: A sparql event processing architecture. *Future Internet*, 10(4):36, 2018. (cit. on pp. 86, 87, 99, 250)

[160] Isam Ishaq, David Carels, Girum K Teklemariam, Jeroen Hoebeke, Floris Van den Abeele, Eli De Poorter, Ingrid Moerman, and Piet Demeester. Ietf standardization in the field of the internet of things (iot): a survey. *Journal of Sensor and Actuator Networks*, 2(2):235–287, 2013. (cit. on pp. 91)

[161] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, and Elias Torres. Sparql 1.1 protocol. *Recommendation, W3C, March*, 2013. (cit. on pp. 91)

[162] Fabio Viola, Luca Turchet, Francesco Antoniazzi, and György Fazekas. C minor: a semantic publish/subscribe broker for the internet of musical things. In *Open Innovations Association (FRUCT), 23rd Conference of*. IEEE, 2018. (cit. on pp. 92, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 247, 250)

[163] V. Charpenay, S. Käbisch, and H. Kosch. μrdf store: Towards extending the semantic web to embedded devices. In *European Semantic Web Conference*, pages 76–80. Springer, 2017. (cit. on pp. 96)

[164] Jim Gray. Database and transaction processing performance handbook., 1993. (cit. on pp. 96)

[165] Alo Allik, György Fazekas, and Mark B Sandler. An ontology for audio features. In *ISMIR*, pages 73–79, 2016. (cit. on pp. 96, 225, 236, 237)

[166] Zheng Fei, Fu Baicheng, and Cao Zhen. Coap latency evaluation. (cit. on pp. 98)

[167] Damien Graux, Pierre Genevès, and Nabil Layaïda. SPARUB: SPARQL UPDATE Benchmark. working paper or preprint, May 2017. (cit. on pp. 109)

[168] Kai Sachs, Stefan Appel, Samuel Kounev, and Alejandro Buchmann. Benchmarking publish/subscribe-based messaging systems. In *Database Systems for Advanced Applications*, pages 203–214. Springer, 2010. (cit. on pp. 109)

[169] Martin Arlitt, Manish Marwah, Gowtham Bellala, Amip Shah, Jeff Healey, and Ben Vandiver. Iotabench: an internet of things analytics benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 133–144. ACM, 2015. (cit. on pp. 110)

[170] Anshu Shukla and Yogesh Simmhan. Benchmarking distributed stream processing platforms for iot applications. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 90–106. Springer, 2016. (cit. on pp. 110)

[171] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. Riotbench: A real-time iot benchmark for distributed stream processing platforms. corr abs/1701.08530 (2017). *arxiv. org/abs/1701.08530*, 2017. (cit. on pp. 110)

[172] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets. In *International Semantic Web Conference*, pages 374–389. Springer, 2015. (cit. on pp. 110)

[173] David Crocker. Standard for the format of arpa internet text messages. Technical report, 1982. (cit. on pp. 118)

[174] Jukka Honkola, Hannu Laine, Ronald Brown, and Olli Tyrkkö. Smart-m3 information sharing platform. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 1041–1046. IEEE, 2010. (cit. on pp. 118)

[175] Fernando Serena, María Poveda-Villalón, and Raúl García-Castro. Semantic discovery in the web of things. In *International Conference on Web Engineering*, pages 19–31. Springer, 2017. (cit. on pp. 122, 123, 124, 189, 225, 250)

[176] Laura Daniele, Frank den Hartog, and Jasper Roes. Created in close interaction with the industry: the smart appliances reference (saref) ontology. In *International Workshop Formal Ontologies Meet Industries*, pages 100–112. Springer, 2015. (cit. on pp. 124)

[177] Fabio Viola, Alfredo D'Elia, Luca Roffia, and Tullio Salmon Cinotti. Performance evaluation suite for semantic publish-subscribe message-oriented middlewares. In *UBICOMM 2016, The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 190–196. IARIA, 2016. (cit. on pp. 135)

[178] Enrico Motta, Paul Mulholland, Silvio Peroni, Mathieu d'Aquin, Jose Manuel Gomez-Perez, Victor Mendez, and Fouad Zablith. A novel approach to visualizing and navigating ontologies. In *International Semantic Web Conference*, pages 470–486. Springer, 2011. (cit. on pp. 140, 143)

[179] Pierre-Yves Vandenbussche, Ghislain A Atemezing, María Poveda-Villalón, and Bernard Vatant. Linked open vocabularies (lov): a gateway to reusable semantic vocabularies on the web. *Semantic Web*, 8(3):437–452, 2017. (cit. on pp. 140)

[180] Artem Chebotko, Shiyong Lu, Hasan M Jamil, and Farshad Fotouhi. Semantics preserving sparql-to-sql query translation for optional graph patterns. *Wayne State University, Tech. Rep. TR-DB-052006-CLJF*, 2006. (cit. on pp. 140)

[181] Lihua Zhao and Ryutaro Ichise. Ontology integration for linked data. *Journal on Data Semantics*, 3(4):237–254, 2014. (cit. on pp. 141)

[182] Mario Arias Gallego, Javier D Fernández, Miguel A Martínez-prieto, and Pablo De Fuente. RDF Visualization using a Three-Dimensional Adjacency Matrix. 2011. (cit. on pp. 142)

[183] E R Gansner, E Koutsofios, S C North, and K P a Vo K P Vo. A technique for drawing directed graphs\nA technique for drawing directed graphs. *Software Engineering, IEEE Transactions on*, 19(3):214–230, 1993. (cit. on pp. 143)

[184] Emden R Gansner and Stephen C North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, 2000. (cit. on pp. 143)

[185] ER Gansner and Yehuda Koren. Improved circular layouts. *Graph Drawing*, pages 386–398, 2007. (cit. on pp. 143)

[186] Emden R. Gansner and Yifan Hu. Efficient, proximity-preserving node overlap removal. *J. Graph Algorithms Appl.*, 14(1):53–74, 2010. (cit. on pp. 143)

[187] Carla Binucci, Markus Chimani, Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. Placing Arrows in Directed Graph Drawings. pages 1–19, 2016. (cit. on pp. 143)

[188] Ben Shneiderman and Aleks Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733–740, 2006. (cit. on pp. 143)

[189] Emden R Gansner, Yehuda Koren, and Stephen North. Graph Drawing by Stress Majorization. *Proc. 12th Int. Symp. Graph Drawing (GD 2004)*, LNCS 3383(2004):239–250, 2005. (cit. on pp. 143)

[190] J Ellson, E R Gansner, E Koutsofios, S C North, and G Woodhull. Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools. *Graph Drawing Software*, pages 127–148, 2004. (cit. on pp. 143)

[191] Charles D Stolper, Minsuk Kahng, Zhiyuan Lin, Florian Foerster, Aakash Goel, John Stasko, and Duen Horng Chau. GLO-STIX : Graph-Level Operations for Specifying Techniques and Interactive eXploration. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2320–2328, 2014. (cit. on pp. 143)

[192] Eugene Wu, Fotis Psallidas, Zhengjie Miao, Haoci Zhang, Laura Rettig, Yifan Wu, and Thibault Sellam. Combining design and performance in a data visualization management system. In *CIDR*, 2017. (cit. on pp. 143)

[193] Franz-Josef Brandenburg, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, Giuseppe Liotta, and Petra Mutzel. Selected open problems in graph drawing. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 515–539. Springer, 2003. (cit. on pp. 143)

[194] Emden R. Gansner, Yifan Hu, and Stephen G. Kobourov. GMap: Drawing Graphs as Maps. 2009. (cit. on pp. 143)

[195] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsm*, 8(2009):361–362, 2009. (cit. on pp. 143, 144, 145, 250)

[196] Francesco Antoniazzi and Fabio Viola. Rdf graph visualization tools: a survey. In *Open Innovations Association (FRUCT), 23rd Conference of*. IEEE, 2018. (cit. on pp. 144, 145, 147, 148, 150, 152, 250, 251)

[197] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003. (cit. on pp. 144)

[198] Tuukka Hastrup, Richard Cyganiak, and Uldis Bojars. Browsing linked data with fenfire. 2008. (cit. on pp. 145)

[199] Walter Hop, Sven de Ridder, Flavius Frasincar, and Frederik Hogenboom. Using hierarchical edge bundles to visualize complex ontologies in glow. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 304–311. ACM, 2012. (cit. on pp. 146)

[200] Emmanuel Pietriga. Isaviz: A visual authoring tool for rdf. *World Wide Web Consortium.[Online]. Available: http://www. w3. org/2001/11/IsaViz*, 2003. (cit. on pp. 146)

[201] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001. (cit. on pp. 146)

[202] Margaret-Anne Storey, Natasha F Noy, Mark Musen, Casey Best, Ray Fergerson, and Neil Ernst. Jambalaya: an interactive environment for exploring ontologies. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 239–239. ACM, 2002. (cit. on pp. 146)

[203] M-A Storey, Casey Best, and Jeff Michand. Shrimp views: An interactive environment for exploring java programs. In *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pages 111–112. IEEE, 2001. (cit. on pp. 146)

[204] Diego Valerio Camarda, Silvia Mazzini, and Alessandro Antonuccio. Lodlive, exploring the web of data. In *Proceedings of the 8th International Conference on Semantic Systems*, pages 197–200. ACM, 2012. (cit. on pp. 146, 147, 251)

[205] Sean Falconer. Ontograf protege plugin. (cit. on pp. 147, 148, 251)

[206] Alessio Bosca, Dario Bonino, and Paolo Pellegrino. Ontosphere: more than a 3d ontology visualization tool. In *Swap*. Citeseer, 2005. (cit. on pp. 148)

[207] Matthew Horridge. Owlviz. *Available on: http://protegewiki.stanford.edu/wiki/OWLViz*, 2010. (cit. on pp. 149)

[208] Leonidas Deligiannidis, Krys J Kochut, and Amit P Sheth. Rdf data exploration and visualization. In *Proceedings of the ACM first workshop on CyberInfrastructure: information management in eScience*, pages 39–46. ACM, 2007. (cit. on pp. 149)

[209] Maciej Janik and Krys Kochut. Brahms: a workbench rdf store and high performance memory system for semantic association discovery. In *International Semantic Web Conference*, pages 431–445. Springer, 2005. (cit. on pp. 149)

[210] Philipp Heim, Sebastian Hellmann, Jens Lehmann, Steffen Lohmann, and Timo Stegemann. Relfinder: Revealing relationships in rdf knowledge bases. In *International Conference on Semantic and Digital Media Technologies*, pages 182–187. Springer, 2009. (cit. on pp. 149, 150, 251)

[211] Harith Alani. Tgviztab: an ontology visualisation extension for protégé. 2003. (cit. on pp. 150)

[212] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. Visualizing ontologies with vowl. *Semantic Web*, 7(4):399–419, 2016. (cit. on pp. 151)

[213] Steffen Lohmann, Vincent Link, Eduard Marbach, and Stefan Negru. WebVOWL: Web-based visualization of ontologies. In *Proceedings of EKAW 2014 Satellite Events*, volume 8982 of *LNAI*, pages 154–158. Springer, 2015. (cit. on pp. 151, 152, 251)

[214] Steffen Lohmann, Stefan Negru, and David Bold. The ProtégéVOWL plugin: Ontology visualization for everyone. In *Proceedings of ESWC 2014 Satellite Events*, volume 8798 of *LNCS*, pages 395–400. Springer, 2014. (cit. on pp. 151)

[215] Marc Weise, Steffen Lohmann, and Florian Haag. Ld-vowl: Extracting and visualizing schema information for linked data. In *2nd International Workshop on Visualization and Interaction for Ontologies and Linked Data, Kobe, Japón*, pages 120–127, 2016. (cit. on pp. 151)

[216] Florian Haag, Steffen Lohmann, Stephan Siek, and Thomas Ertl. Queryvowl: A visual query notation for linked data. In *International Semantic Web Conference*, pages 387–402. Springer, 2015. (cit. on pp. 151)

[217] Pavel Lomov and Maxim Shishaev. Creating cognitive frames based on ontology design patterns for ontology visualization. In Pavel Klinov and Dmitry Mouromtsev, editors, *Knowledge Engineering and the Semantic Web*, pages 90–104, Cham, 2014. Springer International Publishing. (cit. on pp. 151)

[218] Philipp Heim and Steffen Lohmann. Semlens: Visual analysis of semantic data with scatter plots and semantic lenses. *Proceedings of the 7th International Conference on Semantic Systems - I-Semantics '11*, pages 175–178, 2011. (cit. on pp. 151)

[219] Eric A Bier, Maureen C Stone, Ken Pier, William Buxton, and Tony D DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80. ACM, 1993. (cit. on pp. 151)

[220] Aba-Sah Dadzie and Emmanuel Pietriga. Visualisation of linked data - reprise. *Semantic Web*, 8(1):1–21, 2017. (cit. on pp. 151)

[221] Andrea Giovanni Nuzzolese, Valentina Presutti, Aldo Gangemi, Silvio Peroni, and Paolo Ciancarini. Aemoo: Linked data exploration based on knowledge patterns. *Semantic Web*, 8(1):87–112, 2017. (cit. on pp. 151)

[222] Fabio Viola, Luca Roffia, Francesco Antoniazzi, Alfredo D'Elia, Cristiano Aguzzi, and Tullio Salmon Cinotti. Interactive 3d exploration of rdf graphs through semantic planes. *Future Internet*, 10(8), 2018. (cit. on pp. 154, 155, 157, 160, 162, 164, 165, 166, 169, 171, 172, 174, 175, 178, 180, 181, 182, 183, 247, 251, 252)

[223] Daniel Hernández, Aidan Hogan, and Markus Krötzsch. Reifying rdf: What works well with wikidata? In Thorsten Liebig and Achille Fokoue, editors, *SSWS@ISWC*, volume 1457 of *CEUR Workshop Proceedings*, pages 32–47. CEUR-WS.org, 2015. (cit. on pp. 171)

[224] Luca Roffia, Paolo Azzoni, Cristiano Aguzzi, Fabio Viola, Francesco Antoniazzi, and Tullio Salmon Cinotti. Dynamic Linked Data: A SPARQL Event Processing Architecture. *Future Internet*, 10(4):36, apr 2018. (cit. on pp. 177, 180)

[225] Mikko Rinne and Esko Nuutila. Constructing Event Processing Systems of Layered and Heterogeneous Events with SPARQL. *Journal on Data Semantics*, 6(2):57–69, 2017. (cit. on pp. 180)

[226] Zoltan Kis, Kazuaki Nimura, Daniel Peintner, and Johannes Hund. Web of things (wot) scripting api, Oct 2018. (cit. on pp. 190)

[227] Fredrik Blomstedt, Luis Lino Ferreira, Markus Klisics, Christos Chrysoulas, Iker Martinez de Soria, Brice Morin, Anatolijs Zabasta, Jens Eliasson, Mats Johansson, and Pal Varga. The arrowhead approach for soa application development and documentation. In *Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE*, pages 2631–2637. IEEE, 2014. (cit. on pp. 196, 197, 252)

[228] OASIS Standard. Mqtt version 3.1. 1. *URL http://docs.oasis-open.org/mqtt/mqtt/v3*, 1, 2014. (cit. on pp. 199)

[229] Luca Bedogni, Luciano Bononi, Marco Di Felice, Alfredo D'Elia, Randolf Mock, Francesco Morandi, Simone Rondelli, Tullio Salmon Cinotti, and Fabio Vergari. An integrated simulation framework to model electric vehicle operations and services. *IEEE Transactions on Vehicular Technology*, 65(8):5900–5917, 2016. (cit. on pp. 208)

[230] Michele Ornato, Tullio Salmon Cinotti, Alberto Borghetti, Paolo Azzoni, Alfredo D'Elia, Fabio Viola, Federico Montori, and Riccardo Venanzi. Application system design: Complex systems management and automation. In *IoT Automation*, pages 317–352. CRC Press, 2017. (cit. on pp. 210)

[231] Sujesha Sudevalayam and Purushottam Kulkarni. Energy harvesting sensor nodes: Survey and implications. *IEEE Communications Surveys & Tutorials*, 13(3):443–461, 2011. (cit. on pp. 211)

[232] Alfredo D'Elia, Luca Perilli, Fabio Viola, Luca Roffia, Francesco Antoniazzi, Roberto Canegallo, and Tullio Salmon Cinotti. A self-powered wsan for energy efficient heat distribution. In *Sensors Applications Symposium (SAS), 2016 IEEE*, pages 1–6. IEEE, 2016. (cit. on pp. 212, 213, 214, 215, 217, 218, 253)

[233] Bidyadhar Subudhi and Raseswari Pradhan. A comparative study on maximum power point tracking techniques for photovoltaic power systems. *IEEE Transactions on sustainable energy*, 4(1):89–98, 2013. (cit. on pp. 214)

[234] JP Norair. Introduction to dash7 technologies. *Dash7 Alliance Low Power RF Technical Overview*, pages 1–22, 2009. (cit. on pp. 216)

[235] Frederic Font, Tim Brookes, George Fazekas, Martin Guerber, Amaury La Burthe, David Plans, Mark D Plumbley, Meir Shaashua, Wenwu Wang, and Xavier Serra. Audio Commons: Bringing Creative Commons Audio Content to the Creative Industries. In *Audio Engineering Society Conference: 61st International Conference: Audio for Games*, feb 2016. (cit. on pp. 222, 223, 253)

[236] Chris Cannam, Michael O. Jewell, Mark Sandler, Christophe Rhodes, and Mark d'Inverno. Linked data and you: Bringing music research software into the semantic web. *Journal of New Music Research*, 39(4):313–325, 2010. (cit. on pp. 225)

[237] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A sparql extension for generating rdf from heterogeneous formats. In *European Semantic Web Conference*, pages 35–50. Springer, 2017. (cit. on pp. 225, 226)

[238] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. Flexible RDF generation from RDF and heterogeneous data sources with SPARQL-Generate. In *Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW'16)*, Bologna, Italy, November 2016. (cit. on pp. 225, 226)

[239] Miguel Ceriani, Gyorgy Fazekas, Johan Pauwels, Mathieu Barthet, and Mark Sandler. Deliverable d2.3 final ontology specification. H2020 Project "AudioCommons" research and innovation grant 688382. (cit. on pp. 225)

[240] G. Fazekas and M. Sandler. Knowledge representation issues in audio-related metadata model design. In *Proc. of the 133rd Convention of the Audio Engineering Society*, 2012. (cit. on pp. 225)

[241] C Cannam. The vamp plugin ontology, 2009. (cit. on pp. 226)

[242] The recommendation ontology 0.3, 2010. (cit. on pp. 226)

[243] Fabio Viola, Ariane Stolfi, Alessia Milo, Miguel Ceriani, Mathieu Barthet, and Gÿorgy Fazekas. Playsound.space: enhancing a live performance tool with semantic recommendations. In *SAAM '18 Proceedings of the 1st International Workshop on Semantic Applications for Audio and Music*, pages 46–53. ACM, 2018. (cit. on pp. 226, 253)

[244] Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. Prov-o: The prov ontology. *W3C recommendation*, 30, 2013. (cit. on pp. 229)

[245] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet. Internet of Musical Things: Vision and Challenges. *IEEE Access*, 2018 (submitted). (cit. on pp. 234)

[246] 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*, pages 153–160, 2003. (cit. on pp. 234)

[247] Luca Turchet, Fabio Viola, György Fazekas, and Mathieu Barthet. Towards a semantic architecture for the internet of musical things. In *Open Innovations Association (FRUCT), 23rd Conference of*. IEEE, 2018. (cit. on pp. 235, 238, 253)

[248] A. P McPherson, Robert H Jack, and Giulio Moro. Action-sound latency: Are our tools fast enough? In *Proceedings of the Conference on New Interfaces for Musical Expression*, 2016. (cit. on pp. 236)

[249] P. Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013. (cit. on pp. 236)

[250] W. Piston. *Harmony*. WW Norton, 1948. (cit. on pp. 236)

[251] Peter Brinkmann, Peter Kirn, Richard Lawler, Chris McCormick, Martin Roth, and Hans-Christoph Steiner. Embedding pure data with libpd. In *Proceedings of the Pure Data Convention*, volume 291, 2011. (cit. on pp. 237)

[252] Luca Turchet. Smart mandolin: autobiographical design, implementation, use cases, and lessons learned. In *Proceedings of Audio Mostly Conference*, 2018. (cit. on pp. 237)

[253] A. McPherson and V. Zappi. An environment for Submillisecond-Latency audio and sensor processing on BeagleBone black. In *Audio Engineering Society Convention 138*. Audio Engineering Society, 2015. (cit. on pp. 237)

[254] T. Mitchell, S. Madgwick, S. Rankine, G.S. Hilton, A. Freed, and A.R Nix. Making the most of wi-fi: Optimisations for robust wireless live music performance. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 251–256, 2014. (cit. on pp. 238)

Acknowledgements/Ringraziamenti

Music has always played an importan role in my life and I've been so lucky for the chance to apply my research to this beautiful field. So, I'd like to conclude my Thesis with my *musical* acknowledgements...

La musica ha sempre rivestito un ruolo importante nella mia vita e fortunatamente ho avuto la possibilità di applicare i miei studi in materia di Semantic Web of Things a questo campo affascinante. Quindi quale miglior modo di chiuder la tesi di...ringraziamenti in musica?

... Why not think about times to come?
And not about the things that you've done
If your life was bad to you
Just think what tomorrow will do
Fleetwood Mac, Don't stop

I'd like to thank **Prof. Tullio Salmon Cinotti** for giving me the chance to carry out my PhD at the ARCES department of the University of Bologna as well as the chance to organise the 23rd IEEE FRUCT conference. I also thank him for the esteem he has for me and for the precious advice to always look at a brighter tomorrow. It is also important for me to thank **Luca Roffia** for the passion and the enthusiasm he instilled in the group, as well the will to make our group better.

Ringrazio il **Prof. Tullio Salmon Cinotti** per avermi dato la possibilità di svolgere il mio dottorato in ARCES, per avermi permesso di vivere la straordinaria esperienza di organizzare la conferenza 23rd IEEE FRUCT, ma soprattutto per la stima che nutre nei miei confronti e per l'importante insegnamento di esser sempre col pensiero rivolto ad un futuro più brillante. Ci tengo a ringraziare **Luca Roffia** per la passione e gli stimoli che ha saputo trasmettere e per la ventata di novità che ha portato al gruppo.

Mama she has taught me well
Told me when I was young
"Son your life's an open book
Don't close it 'fore its done"
Metallica, Mama said

A huge thank goes to **my mother** who has supported me during the many blue moments of the latest years, who has always believed in me (more than I have done), but even more important I have to thank her for tolerating me despite by caustic nature.

Un grazie enorme va a **mia madre** che mi ha sostenuto durante i tantissimi momenti bui di questi ultimi anni, che ha creduto in me sempre più di quanto io stesso abbia fatto, ma soprattutto che mi ha sopportato nonostante il mio brutto carattere.

. . . . . .

. . . And so today, my world it smiles
Your hand in mine, we walk the miles
Thanks to you it will be done
For you to me are the only one
Happiness, no more be sad
Happiness, I'm glad. . .
Led Zeppelin, Thank You

Serena, it would take a whole book to say thank you as you would deserve. An you know, you have been my model during the whole PhD duration. I owe you everything I've learnt during this adventure; I owe you the growth and awareness I've achieved; I owe you the whole PhD. My gratitude goes beyond the pure academic career. This 4-year adventure has been unparalleled thank to you: you gave me tons of happiness and I cannot imagine such an adventure without you. Thank you, I'll never say it enough.

Serena ci vorrebbe un libro intero per dirti grazie come meriteresti. Come sai, sei stata il mio modello per tutto il dottorato. Devo a te molto di quel che ho imparato in questo percorso, devo a te la maturazione e la consapevolezza che ho acquisito, devo a te il raggiungimento di questo traguardo. Ma i miei ringraziamenti vanno ben oltre ciò che tu hai fatto per il mio percorso accademico. Quest'esperienza di quasi quattro anni è stata unica grazie a te, e non potrei mai immaginare di ripeter tutto questo senza di te che mi hai donato felicità a palate. Grazie, non te lo dirò mai abbastanza.

. . . . . .

. . . We fought for good,
stood side by side,
Our friendship never died.
On stranger waves,
the lows and highs,
Our vision touched the sky. . .
Joy Division - A means to an end

Francesco has been a brother, not just a colleague. I shared with him my sorrow and my fears, but we've also had a lot of fun travelling and playing ping pong. It has been an honour to work together and, even if our ways will probably split, I bet that our friendship is so strong that will hardly fade away.

Francesco è stato un fratello prima che un collega. Con lui ho condiviso le mie ansie e frustrazioni, ma fortunatamente anche tanti momenti fantastici come divertenti trasferte ed innumerevoli partite di ping pong. È stato un onore lavorare insieme e anche se, probabilmente, le nostre strade lavorative si separeranno, son certo che in questi anni è nato un legame che difficilmente si scioglierà.

. . . . . .

. . . Everyone around, love them, love them
Put it in your hands, take it, take it
There's no time to cry, happy, happy
Put it in your heart where tomorrow shines. . .
R.E.M., Shiny happy people

A huge thank goes to the Centre For Digital Music of the Queen Mary University of London where I've found way more than an stimulating place to work.

Un ringraziamento speciale va al Centre For Digital Music della Queen Mary University of London dove ho trovato ben più di un posto di lavoro stimolante.

The AudioCommons family embraced in a warm environment and gave me the chance to have fun while learning a lot of exciting stuff. This amazing experience changed me profundly and would not have been possible without the help of **Prof. György Fazekas** that I warmly thank for his affection, esteem and his professionalism that made me a better person as well as a better researcher. During this period I had the chance to work with wonderful collegues that I here mention in alphabetical order: **Mathieu Barthet**, **Miguel Ceriani**, **Alessia Milo**, **Johan Pauwels**, **Sasha Rudan**, **Ariane Stolfi**, **Anna Xambò**.

At the C4DM I have also met **Luca Turchet** who introduced me to the amazing world of Internet of Musical Things. This friend, a smart and determined guy, taught me a lot during the short period we worked together.

La famiglia AudioCommons mi ha accolto in un ambiente caloroso e mi ha dato la possibilità di divertirmi confrontandomi con tante nuove eccitanti tematiche. Questa esperienza esaltante mi ha cambiato profondamente e non sarebbe stata possibile senza l'aiuto del **Prof. György Fazekas** che ringrazio enormemente per l'affetto, la stima e la professionalità con cui mi ha seguito rendondomi una persona ed un ricercatore migliore. Durante questo periodo ho avuto modo di collaborare con persone splendide a cui va un ringraziamento di cuore e che qui riporto in ordine alfabetico: **Mathieu Barthet**, **Miguel Ceriani**, **Alessia Milo**, **Johan Pauwels**, **Sasha Rudan**, **Ariane Stolfi**, **Anna Xambò**.

Al C4DM ho conosciuto anche **Luca Turchet**, che mi ha introdotto all'affascinante mondo dell'Internet of Musical Things. Questo amico, un ragazzo brillante e determinato, mi ha insegnato tantissimo durante il breve periodo in cui abbiamo lavorato insieme.

. . . . . .

. . . Trust yourself
Trust yourself to know the way that will prove true in the end
Trust yourself
Trust yourself to find the path where there is no if and when. . .
Bob Dylan, Trust yourself

A person who strongly contributed to my personal and professional growth is **Sergey Balandin**. I thank him and the whole FRUCT community for the chances they have gave me and the trust they have always put in me. They have always helped me believing in myself.

Una persona che ha contribuito particolarmente alla mia formazione durante questo dottorato è **Sergey Balandin**. Ringrazio lui e l'intera comunità FRUCT per le occasioni di crescita che mi hanno offerto e la fiducia che han riposto in me spingendomi a credere sempre di più in me stesso.

THESIS END

Fabio Viola

Computer Science Engineer
PhD in Computer Science and Engineering (2015-2018)

E-mail: fabio.viola@unibo.it; desmovalvo@gmail.com
Personal website: https://www.unibo.it/sitoweb/fabio.viola/
(Curriculum Vitae available on the personal website)

Office address
Via Toffano 2/2, IT-40125, Bologna (BO), Italy

Home address
Via Marsala 20, IT-40126, Bologna (BO), Italy