In the past decade the Internet-of-Things concept has overwhelmingly entered all of the fields where data are produced and processed, thus, resulting in a plethora of IoT platforms, typically cloud-based, that centralize data and services management. In this scenario, the development of IoT services in domains such as smart cities, smart industry, e-health, automotive, are possible only for the owner of the IoT deployments or for ad-hoc business one-to-one collaboration agreements. The realization of "smarter" IoT services or even services that are not viable today envisions a complete data sharing with the usage of multiple data sources from multiple parties and the interconnection with other IoT services.

In this context, this work studies several aspects of data sharing focusing on Internet-of-Things. We work towards the hyperconnection of IoT services to analyze data that goes beyond the boundaries of a single IoT system. This thesis presents a data analytics platform that: i) treats data analytics processes as services and decouples their management from the data analytics development; ii) decentralizes the data management and the execution of data analytics services between fog, edge and cloud; iii) federates peers of data analytics platforms managed by multiple parties allowing the design to scale into federation of federations; iv) encompasses intelligent handling of security and data usage control across the federation of decentralized platforms instances to reduce data and service management complexity.

The proposed solution is experimentally evaluated in terms of performances and validated against use cases. Further, this work adopts and extends available standards and open sources, after an analysis of their capabilities, fostering an easier acceptance of the proposed framework. We also report efforts to initiate an IoT services ecosystem among 27 cities in Europe and Korea based on a novel methodology.

We believe that this thesis open a viable path towards a hyperconnection of IoT data and services, minimizing the human effort to manage it, but leaving the full control of the data and service management to the users' will.

Flavio Cirillo

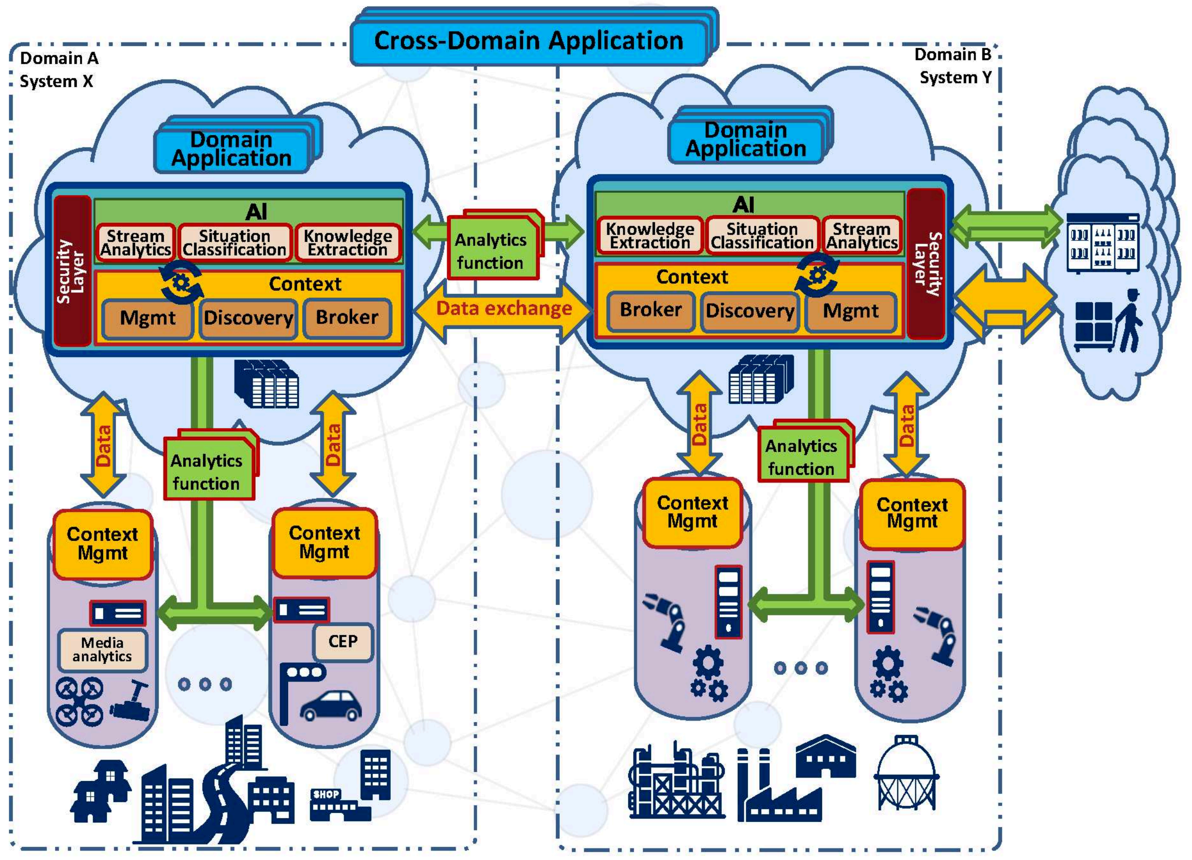
Towards Data Sharing across Decentralized and Federated IoT Data Analytics Platforms

2020 - XXXIII

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Ph.D itee INFORMATION TECHNOLOGY ELECTRICAL ENGINEERING

TOWARDS DATA SHARING ACROSS DECENTRALIZED AND FEDERATED IOT DATA ANALYTICS PLATFORMS



FLAVIO CIRILLO

PH.D. IN INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

PH.D. THESIS

IN

INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

TOWARDS DATA SHARING ACROSS DECENTRALIZED AND FEDERATED IOT DATA ANALYTICS PLATFORMS

FLAVIO CIRILLO

TUTOR: PROF. SIMON PIETRO ROMANO

COORDINATOR: PROF. DANIELE RICCIO

XXXIII CICLO

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base

Ph.D. Thesis in
Information Technology and Electrical Engineering (ITEE)

Towards Data Sharing across Decentralized and Federated IoT Data Analytics Platforms

XXXIII Cycle

Author:
Flavio Cirillo

Tutor:
Prof. Simon Pietro Romano

Abstract

In the past decade the Internet-of-Things concept has overwhelmingly entered all of the fields where data are produced and processed, thus, resulting in a plethora of IoT platforms, typically cloud-based, that centralize data and services management. In this scenario, the development of IoT services in domains such as smart cities, smart industry, e-health, automotive, are possible only for the owner of the IoT deployments or for ad-hoc business one-to-one collaboration agreements. The realization of ”smarter” IoT services or even services that are not viable today envisions a complete data sharing with the usage of multiple data sources from multiple parties and the interconnection with other IoT services.

In this context, this work studies several aspects of data sharing focusing on Internet-of-Things. We work towards the hyperconnection of IoT services to analyze data that goes beyond the boundaries of a single IoT system. This thesis presents a data analytics platform that: *i)* treats data analytics processes as services and decouples their management from the data analytics development; *ii)* decentralizes the data management and the execution of data analytics services between fog, edge and cloud; *iii)* federates peers of data analytics platforms managed by multiple parties allowing the design to scale into federation of federations; *iv)* encompasses intelligent handling of security and data usage control across the federation of decentralized platforms instances to reduce data and service management complexity.

The proposed solution is experimentally evaluated in terms of performances and validated against use cases. Further, this work adopts and extends available standards and open sources, after an analysis of their capabilities, fostering an easier acceptance of the proposed framework. We also report efforts to initiate an IoT services ecosystem among 27 cities in Europe and Korea based on a novel methodology.

We believe that this thesis open a viable path towards a hyperconnection of IoT data and services, minimizing the human effort to manage it, but leaving the full control of the data and service management to the users’ will.

Contents

List of Publications

In the following we list the key publications directly supporting this PhD thesis, indicating the corresponding chapters within the thesis.

[1] **Cirillo, Flavio**, Fang-Jing Wu, Gürkan Solmaz, and Ernö Kovacs; Embracing the future internet of things. MDPI Sensors 19, no. 2 (2019).

The material of this publication can be found in Chapters 1 and 7.

[2] Sciancalepore, Vincenzo; **Cirillo, Flavio**; Costa-Perez, Xavier; Slice as a service (SlaaS) optimal IoT slice resources orchestration 2017 IEEE Global Communications Conference, 2017

The material of this publication can be found in Chapter 2.

[3] Cheng, Bin; Solmaz, Gürkan; **Cirillo, Flavio**; Kovacs, Ernö; Terasawa, Kazuyuki; Kitazawa, Atsushi; Fogflow: Easy programming of iot services over cloud and edges for smart cities, IEEE Internet of Things Journal, 5, 2, 696-707, 2017, IEEE

The material of this publication can be found in Chapter 2.

[4] Zanzi, Lanfranco; **Cirillo, Flavio**; Sciancalepore, Vincenzo; Giust, Fabio; Costa-Perez, Xavier; Mangiante, Simone; Klas, Guenter; Evolving Multi-Access Edge Computing to Support Enhanced IoT Deployments, IEEE Communications Standards Magazine, 2019

The material of this publication can be found in Chapter 3.

[5] **Cirillo, Flavio**; Solmaz, Gurkan; Berz, Everton Luís; Bauer, Martin; Cheng, Bin; Kovacs, Ernoe; A standard-based open source iot platform: Fiware, IEEE Internet of Things Magazine, 2019

The material of this publication can be found in Chapter 3.

[6] **Cirillo, Flavio**; Capuano, Nicola; Romano, Simon Pietro; Kovacs, Ernö; LIoTS: League of IoT Sovereignties. A Scalable approach for a Transparent Privacy-safe Federation of Secured IoT Platforms 2019 IEEE 44th Conference on Local Computer Networks (LCN) 2019

The material of this publication can be found in Chapter 4.

[7] **Cirillo, Flavio**; Cheng, Bin; Porcellana, Raffaele; Russo, Marco; Solmaz, Gurkan; Sakamoto, Hisashi; Romano, Simon Pietro; IntentKeeper: Intent-oriented Data Usage Control for Federated Data Analytics, IEEE Local Computer Networks 2020.

The material of this publication can be found in Chapter 5.

[8] **Cirillo, Flavio**; Straeten, Detlef; Gomez, David; Gato, Jose; Diez, Luis; Maestro, Ignacio Elicegui; Akhavan, Reza; Atomic Services: sustainable ecosystem of smart city services through pan-European collaboration 2019 Global IoT Summit (GIoTS)

The material of this publication can be found in Chapter 6.

[9] **Cirillo, Flavio**; Gómez, David; Diez, Luis; Maestro, Ignacio Elicegui; Gilbert, Thomas Barrie Juel; Akhavan, Reza; Smart City IoT Services Creation through Large Scale Collaboration, IEEE Internet of Things Journal 2020

The material of this publication can be found in Chapter 6.

[10] Solmaz, Gurkan; Wu, Fang-Jing; **Cirillo, Flavio**; Kovacs, Ernoe; Santana, Juan Ramón; Sánchez, Luis; Sotres, Pablo; Munoz, Luis; Toward understanding crowd mobility in smart cities through the Internet of Things, IEEE Communications Magazine, 57, 4, 40-46, 2019, IEEE

The material of this publication can be found in Chapter 7.

Chapter 1

Introduction

The Internet-of-Things (IoT) concept has been broadly adopted by heterogeneous communities due to its potential benefits. Progresses in technologies enabled the realization of today's IoT services. Nevertheless, new technical capabilities are needed to realize "smarter" IoT services or even open possibilities not viable today.

In the current IoT, multiple data sources contribute sensing information and the sensed data is gathered into a single cloud service. Let us consider the example use case of environmental monitoring for noise pollution measurement in a city. Data from sources such as noise sensors and people's comments from social media are gathered in for the particular cloud service and these sources are used only for noise monitoring purpose aiming at supporting city policies decision-making [11]. In the future, on the other hand, produced data need to be shared among multiple applications (see Fig. 1.1). For instance, the gathered noise information might be utilized for other services such as a human-centered mobility application to plan pedestrian or bike routes avoiding busy areas [12]. Further, the same noise information might be used to detect criminal or dangerous situations (e.g., car accidents, shootings) by the police [13]. Therefore, the linkage between objects, devices, edge devices, actuators, agencies, and services needs to evolve to many-to-many instead of the current many-to-one or one-to-one linkages. This unique characteristic requires transparent discovery and information exchanges opening to more opportunities for information mash-ups among interdependent and symbiotic cloud services. Data sharing is a central focus for the European Commission strategy as demonstrated by the launch of the GAIA-X project in 2020 in collaboration with academia and big global industries. GAIA-X project aims to "jointly enhance the development of federated, trusted and user-friendly digital ecosystems based on fundamental European values" [14].

This thesis work towards a *hyperconnected IoT framework* to connect in synergy multiple interdependent IoT systems and services. The first technical capability lacking in the present systems is the ability to orchestrate data analytics among distributed and federated IoT systems [3], respecting data sovereignty [6, 16] and privacy policies [7], geographical IoT components topology [17], real-time constraints and Quality-of-Service. A second aspect that hampers the global IoT interconnection is the scarcity of infor-

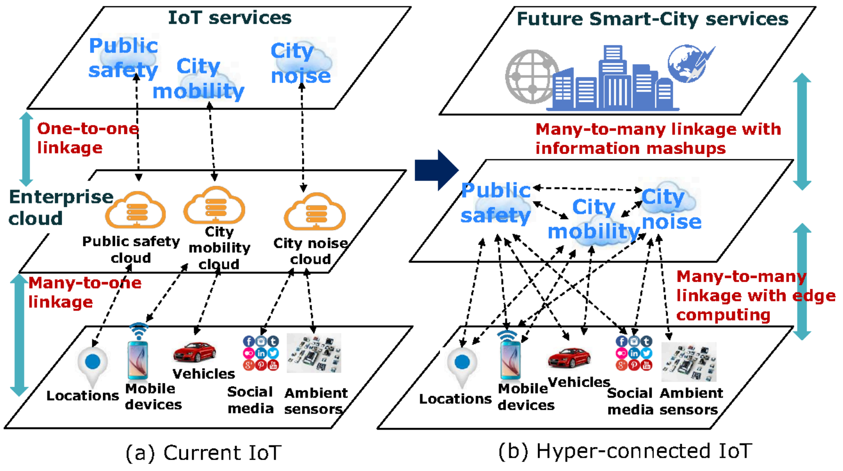


Figure 1.1: Evolution towards hyperconnected IoT [15]

mation transparency that encompasses the semantic interoperability capabilities, which ensures consistent data discovery and data exchange among heterogeneous systems, and the semantic mediation between IoT systems that enables interaction between different system interfaces. Finally, a third aspect is the management of IoT resources in a contextualized[1] manner, in order to overcome the isolation of vertical IoT silos. It is necessary to seamlessly manage diversified information, from physical sensing data (such as temperature observations measured by a sensor placed on a street lamp post) to high-level contextual information (e.g., disaster detection and monitoring within a geographic area) inferred from the status of multiple things (both physical, and virtual such as social media data). Thus, to realize this vision IoT advancements on data analytics, standardization, and resource and context management are required. The future IoT framework is envisioned for many different applications of IoT such as smart cities, smart transportation, health-care, public services, smart commercial buildings, smart homes, and smart industry.

1.1 Future HyperConnected IoT Framework

A *hyperconnected IoT framework* [15, 20] envisions the collaboration of multiple services. These services utilize and produce resources (such as data sources, objects, and data analytics components) and context with each other following an omnidirectional topology, differentiating from the vertical approach of today's IoT instances. The information in the future IoT framework is represented as a knowledge graph shared among multiple IoT domains with both nodes (representing things) and links between them discovered by automated services. Figure 1.2 summarizes the needed technical capabilities with some of them highlighted (in orange in the figure) that are discussed in this thesis.

In this work we advance towards the future IoT framework with capabilities horizon-

[1]IoT *context* [18, 19] is a core concept defined as the actual status of the real world.

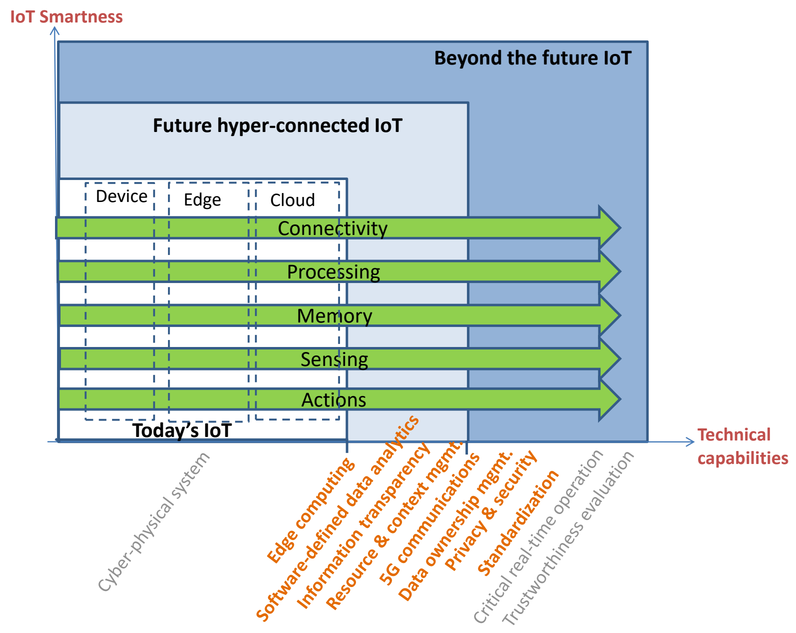
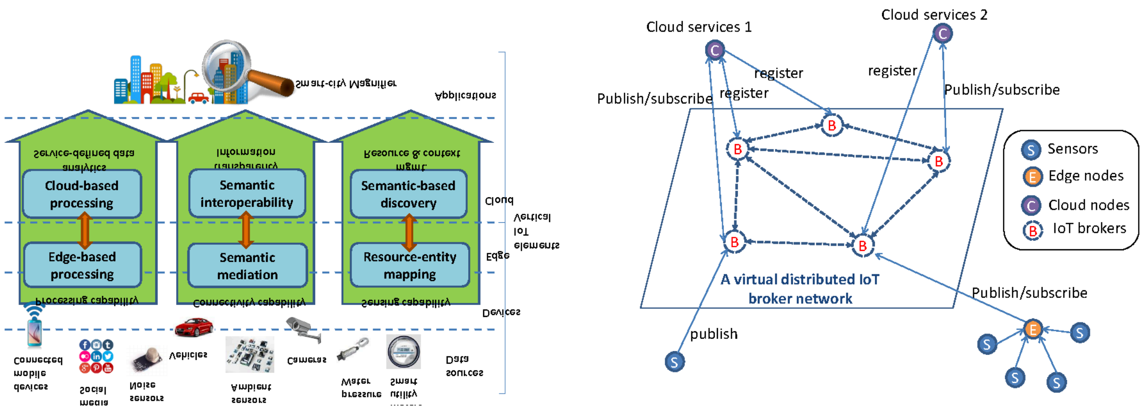


Figure 1.2: Key technical tipping points towards the future hyperconnected Internet-of-Things.

tally placed on vertical IoT elements such as device, edge and cloud. Data is produced by sources through the observation of the real-world such as physical phenomena measurements. The capabilities are then fulfilled by functions that are executed either in the cloud or in the edge depending on the hardware resource requirements (such as storage, communication and computation), real-time constraints, and/or optimization of network bandwidth [3]. Moreover, due to the geographically distributed nature of the IoT services, edge computing has become very significant [17]. Some of the data sources might be devices comprehensive of computation and communication capability (e.g., smartphones, cars) hence able to actively participate at the execution of the IoT framework capabilities as dew computing [21].

Figure 1.3a depicts the horizontal placement of the IoT platform capabilities addressed by this thesis on the vertical IoT elements. The first is *service-defined data analytics* that configures data analytics as a topology of functions, each of which are placed on the cloud or edge (or even device) depending on the requirements of the service. For example, at the edge, some stream analytics are executed in order to reduce the amount of data or to apply privacy-preservation algorithm (such as data anonymization). The cloud, which is orders of magnitude richer in terms of physical resources, is then responsible for performing high-demanding computation for data analytics (e.g., MapReduce tasks). The second capability is the *information transparency* that aims to make the data understandable and usable by heterogeneous IoT systems and applications by hiding the complexity of the IoT elements topology. The translation of raw data to a common form is made by the semantic mediation, usually left to the edge of the IoT (e.g., IoT gateways), whereas the semantic interoperability is performed as cloud services. Finally, the third capability is the *resource and context management* that allows the handling of data not only as isolated pieces of data or isolated datasets but as a global virtual graph of things, having attributes representing the things' status, linked with each other. The edge is left to associate the incoming real-world observations to a thing uniquely identified by a knowledge base. The data is then semantically discoverable at the cloud level, having the latter a broader view of the IoT elements, making possible a mash-up of heterogeneous information coming from heterogeneous data sources.



(**a**) Future IoT framework.

(**b**) Deployment with a distributed IoT broker network.

Figure 1.3: Future Hyperconnected IoT.

A generic future IoT framework deployment is depicted in Figure 1.3b where each one of the cloud services, often indicated as IoT domains, are handling sensors, edge nodes and cloud nodes. Those cloud services are connected with each other through a network of IoT Brokers that dispatch IoT messages (requests) and provision data from a cloud service to another. In order to announce their availability, each of the cloud services announces their capabilities through a registration to its IoT Broker.

1.2 Service-Defined Data Analytics

Data analytics in the future IoT needs to adapt to the intrinsic distributed nature of IoT. Thus, the data is processed at every layer of IoT (i.e., device, edge and cloud). This is not only for scalability reasons (such as minimizing network bandwidth by filtering or aggregate useless and redundant data) but also for taking into consideration the high dynamism of IoT infrastructure topology (e.g., new devices joining the IoT system, mobile devices changing their physical location, battery powered devices temporary unavailable because of recharging, intermittent communication channel to devices). Moreover, privacy requirements steer the decision about the allocation of processing tasks since it is always wiser to execute privacy preserving data processing as closest as possible to the generation of the data. The needed analytics are leveraged by IoT services with heterogeneous requirements such as geographical scope of data analytics [17] (see section 2.2) or Quality-of-Service [2] (see section 2.1). The data analytics framework of the future IoT transforms data analytics requests to multiple atomic tasks to be instantiated and orchestrated among the edge and cloud layers depending on the service requirements.

As an example of a service-defined data analytics, we consider crowd mobility analytics [22] that estimates the crowd levels within an area and the flow of people moving between areas. Data sources (e.g., Wi-Fi sniffers, bluetooth beacons, ambient sensors such as temperature sensors) generate observation of the real world. Edge nodes act as gateways, collecting the observations and then sending them to cloud nodes. With the concept of service-defined data analytics, the processing is decomposed in multiple tasks. Thus, the needed analytics can be divided into two categories, the first for lightweight tasks that can run on the edge nodes and the second for power-consuming tasks to be executed on cloud nodes. In the first category, for our example, we can include the *stream processing* part which comprehends a filtering task that filters redundant or unnecessary data, and a privacy preserving task that anonymizes the observations (important for Wi-Fi monitoring). In the second category, continuing the example, we can include the *batch processing* part where historical data is aggregated and patterns are detected.

In this thesis we present examples of service-defined data analytics. A first one is a contextualized smart city monitoring service, namely Smart City Magnifier (SCM), of smart city data (see section 7.2). The service is meant to compute and visualize city indicators at different layers of abstraction, from sensors level till abstract contextualized levels such as buildings, streets, neighbourhoods etc.. This smart city service takes advantage from the decoupling of service design to the service management by replicating the single analytics tasks throughout the federated IoT systems, each of the replicas focusing on different subset of the available data. A second example is an automotive scenario (see section 5) where electric car owners share car data (e.g., battery status, GPS location) with the manufacturer of their car. An external data analytics company develops a service to analyze the trends and geographic distribution of electric cars. The original data is covered by a privacy policy to protect the identity of the car owners through anonymization procedures that must happen before the data is actually shared with third parties.

1.3 Information Transparency

The capability of having transparent exchange is fundamental for the future hyperconnected IoT systems. This capability consists of semantic interoperability and semantic mediation. Interoperability is the ability of multiple systems to exchange data. *Semantic interoperability* ensures consistency of data across interacting systems regardless of their individual data formats as these systems attribute the same meaning to the exchanged data. The semantics of data can be defined in a way that different data formats use a shared vocabulary and the vocabulary is specified in an ontology. The concept of semantic interoperability is applicable to all elements of the future IoT framework from data sources to cloud and applications.

Semantic mediation is the capability of transforming data coming from one system to make it useful to another system. To achieve interoperability cloud needs to have certain standards so that it can provide contextualized access to the applications. Conversion of data that has certain standards such as oneM2M [23] to other standards such as FIWARE OMA NGSI (Open Mobile Alliance Next Generation Service Interface) [24] can be done on both edge and cloud [25]. Considering that there are many existing standards and communication protocols, semantic mediation is necessary for the realization of communications interoperability between different IoT elements and devices with heterogeneous communication protocols (e.g., Wi-Fi, Zigbee, Bluetooth, 3GPP).

Although in this thesis we do not focus on the development of new ontologies, we extensively adopt existing ones, such as FIESTA-IoT [26] and FIWARE data models [27], map ontologies to standard IoT data formats, and discuss how to use them to hide the heterogeneity of interconnected systems. For example, we successfully use this approach to seamlessly execute the SCM service across 16 IoT deployments, and, also, as common ground to steer the collaboration for a creation of a services ecosystem between 27 cities in Europe and Korea.

1.4 Resource and Context Management

In the future IoT framework, the resource-context management refers to the *contextualization process* from data to services which requires the capabilities of *resource-entity mapping* and *semantic-based discovery*. In the traditional IoT framework, multiple entities which are objects in the real world (e.g., sensors, actuators, and cloud services) are considered together to provide resources (e.g., such as data and context) for a single purpose. For example, sensing readings from the temperature and humidity sensors in a smart home are only used to trigger the heating system. However, the future IoT platforms consist of multiple interdependent systems that collaborate with each other in a symbiotic manner to share all available resources. As shown in Figure 1.4, multiple single-purpose IoT deployments (e.g., a smart home, video-surveillance) owned by different parties (e.g., private home owner, homeland security), depicted as silos, have their own sensors and devices, computation capabilities to analyse data locally (e.g., Complex Event Processing), and storage capabilities with a context management that handles data. Silos are then handled by a domain administration in the cloud that allows interconnectivity with other domains and therefore other silos. Sharing resources allows, for instance, temperature and humidity readings in smart homes to contribute a city-scale monitoring system. Thus, the capability of resource-entity mapping enables omnidirectional information flows across devices, edge, cloud, and systems to collaboratively leverage these resources. Here, “resources” are not limited to physical sensing data, but they can be high-level contextual information shared among multiple entities in the real world.

The whole interconnection of resources creates a next generation of context, resembling a global distributed graph of information, that can be seamlessly accessed by any actors of the Internet-of-Things. Data requests are brokered by specific components, namely Next Generation Brokers (NG Brokers), that provision data from data providers

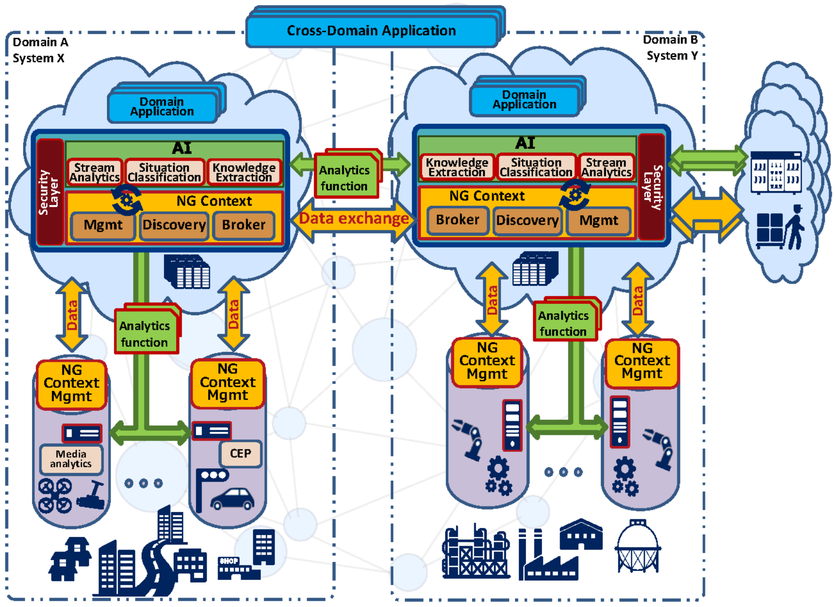


Figure 1.4: Resources and context management federation.

to data consumers. This achieves a transparent global IoT, keeping the IoT topology hidden from data producers and data consumers perspectives.

Semantic-based discovery removes the need for human involvement and assistance and allows worldwide IoT applications to have fully automated data flows reconfiguration and information mash-ups. High-level contextual information are computed by processes of *knowledge extractions* that combine sensed resources mapping them to common entity models and information model. This capability generates new metadata and attributes necessary for linking information into a context mash-ups. For instance, crowdsensed measurements of noise through smartphones [28] are aggregated to assess the noise level of a city neighbourhood level. The produced data and knowledge might be not of much value if those are not interpreted for understanding a situation of the real world. For example, some indexes of high values of temperature and CO2 together with detected hectic crowd patterns, if correctly interpreted, would permit the identification of an ongoing fire break. This interpretation is done by *situation classification* analytics components, of which output is either signalled to a human or automatically triggers actions. In addition to this specific contextualization computation, other *stream analytics* processing are executed for various purposes such as privacy preservation, data polishing, sensor fusion. We envision the service-defined data analytics as topologies of atomic *analytics functions* (or tasks) dynamically instantiated within the *Artificial Intelligence (AI)* layer federated among systems (see Figure 1.4).

Due to the fact that computation and storage capabilities are distributed, data analytics routines, modelled as topologies of analytics functions (e.g., lambda functions),

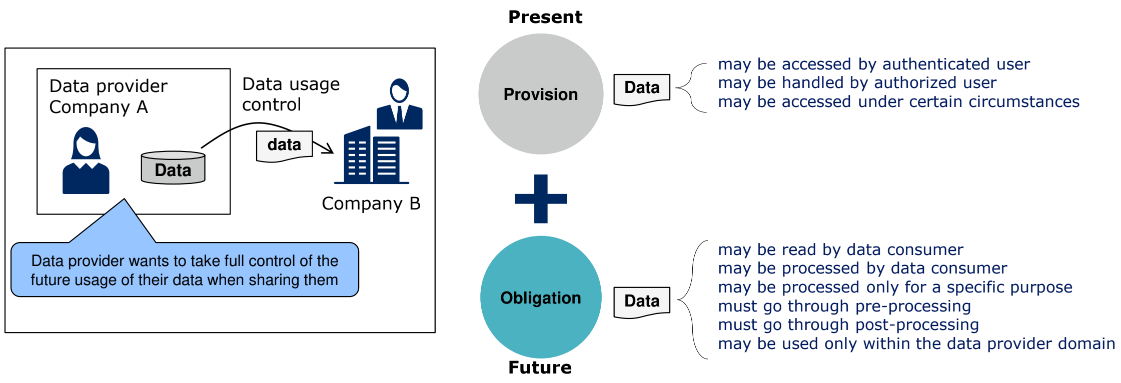

Data provider
Company A
Data
Data usage
control
data
Company B
Data provider wants to take full control of the future usage of their data when sharing them
Present
Provision
Data
may be accessed by authenticated user
may be handled by authorized user
may be accessed under certain circumstances
Obligation
Future
Data
may be read by data consumer
may be processed by data consumer
may be processed only for a specific purpose
must go through pre-processing
must go through post-processing
may be used only within the data provider domain

Figure 1.5: Data usage control concept

might be physically distributed according to several optimization directives with different perspectives [3] orchestrated by a stream analytics system. The synergistic work of analytics and data flowing towards the analytics functions is handled by the next generation context management exploiting the common meaning given to the information among the hyperconnected IoT.

Security layer, horizontally placed on all the components handling and exposing data, is managed by local administrations that allow the owner of the IoT deployments to have full control of the data, and consequently regulates analytics access and computation over the data.

1.5 Data Usage Control

Connecting services and data resources for holistic services development needs to take into consideration the will of the data owners to have continuous control over their data. Common enterprise security, such as authentication and access control, limits the access to data, but, once the data is shared, there is no possibility to control how the data are used (see Fig. 1.5). Licenses, regulations (e.g., GDPR), and ad-hoc agreements legally might impose limitations to the data consumers. However those do not avoid the misuse of data but only apply penalties, with costly legal procedures, after the misbehaviour happens and it is discovered. This approach is undesired by the data owners (since when the misbehaviour is discovered, it is too late to remedy) but it is even harmful for the data consumers that unwittingly violate policies. Indeed, design data consuming services considering data usage policies might be a very complex task. In the fully connected IoT vision where the data provisioning happens automatically and transparently from the data owners and data consumers perspectives, design policies compliant services might not be even feasible due, also, to the highly dynamic scenario of the global IoT.

In this thesis we propose an intent-oriented approach to cope with such complex scenarios. The idea is to decouple the service design to the data usage policies. Data owners express their intent on how their data must be used and the data consumers express their intent on how they want to use data. The service orchestration and the context management, then, cooperate to execute the data processing service by proactively changing the service configuration to comply with the policies in order to prevent any data misuse. The proposed solution is designed to work among federation of peers data analytics platforms that are also decentralized between fog and cloud. We present an automotive scenario as an example for data usage control among data sharing domains.

1.6 Challenges and Open Issues

The number of objects generating data as IoT elements is of a very large scale with continuous growth over time. We have performed a trend analysis based on the reports from International Data Corporation (IDC) [29], Gartner [30] and our estimations [15]. 11.4 billion "things" were installed in 2014 and 13.7 billion in 2015 with a 20.8% increase. Even if the numbers are forecast to continue increasing, the rate will gradually decrease to 11.5% in 2020, with an estimation of 28.1 billion "things". Consequently, the generated data volume will grow by an order of magnitude (10×) from 2015 to 2020 [31]. This exceptional growth of IoT is due to the promised financial benefits. Smart cities and smart homes are, amongst the others, the most advanced field where many businesses compete to earn a share of the market. In the smart cities field, a big push was made by the governments that gradually increased investments, whereas the change in the way of people thinking about everyday life, always more digitalized, opens great business opportunities in the smart homes area (an example is the wide spread of home assistant devices). According to [29, 30, 15], the market share of smart homes and smart cities is forecast to be the 25% over the total IoT market, by 2020.

These vast numbers of connected "things" and large data volumes are only possible with new technologies and standards. The unprecedented increase in "things" coming together with big data brings many new challenges and problems in connectivity, processing, memory, sensing, and actions that require enabling of the future IoT platform capabilities and 5G in order to lead the aforementioned expectations into a reality. Based on this observation, we point out some open issues in the future IoT as follows:

(1) *Data ownership management*: For a future IoT where data is globally accessible and discoverable, special attention should be paid in order to assure that the producer of the data (or the owner of the observed things) keeps ownership of the data, especially for privacy-sensitive data. A study of the International Data Spaces Association (IDSA)[2] [32], where more than 200 companies have been interviewed regarding data exchanged with other companies, states that one of the major concerns that blocks a company from sharing data with another peer is the uncertainty of losing control over the data once the data has been released, and thus losing the "sovereignty" of the data. A first issue is to state "who is the owner of the data": for instance, we are keen to think that the owner of the IoT deployment is the owner of the data; for example, a public transportation company deploying

[2]International Data Spaces Association (IDSA). Available online: `https://www.internationaldataspaces.org/` (Accessed on the 15th of December 2020).

sensors on its buses is the owner of such data. However, in other situations, the owner of the data is the observed thing; this is the case of health sensors deployed by the health care system at home of a patient where the patient is the "thing" observed and the owner of the data. In addition, another open issue is "how to control the data migration to other users and services". Often, users are requested to sign agreements on processing their data, as specified on common data regulations (e.g., General Data Protection Regulation-GDPR[3], but, afterwards, there is not an easy way to control if those agreements are respected. In addition, the data owners should be capable to visualize where, how, by whom and why their data are accessed. Moreover, usage terms might dynamically change over time due to new regulations, changing of the mind of the data owner, or other factors (e.g., expiration of a time period). An automatic system of managing these data access rights' dynamism is a clear challenge.

(2) *Privacy and security*: With the realization of the presented capabilities, the future IoT will encounter new security and privacy threats. Every IoT layer, from applications to devices, has peculiarities on the security risks and possible attacks. Considering the vertical elements in the bottom-up architecture, each level (i.e., devices, edge, cloud and applications) has its own security requirements. Each level is exposed to various types of security threats and possible attacks. Currently, there is a lack of and a certain need for a dynamic IoT security model for enabling mission-critical applications (e.g., autonomous vehicle control) and expected advancements in the IoT systems. Furthermore, for building trust and secure relationships between the IoT components, proper identification and authentication capabilities, and cooperation among these techniques in the IoT platform are currently missing. On the other hand, preserving privacy of data in IoT is an open challenge. The existing privacy protection policies for today's IoT include *encryption*, *anonymization* and *obfuscation* techniques, which are mainly for single services. However, new privacy preservation techniques in these interdependent services (e.g., searchable encryption, usage control, end-to-end encryption [33] with homomorphic encryption) by design principle for objects, devices, users, subsystems, and services are required.

(3) *Critical real-time operation*: The IoT of the future should be flexible and adaptable to sudden changes of the status and conditions of the infrastructures. This is due in order to have fast response to critical situations such as the increasing frequency of natural disasters due to the global climate change [34]. Infrastructureless alternatives for communication in networks [35] or easy-to-deploy infrastructures [36, 37] can help on the communication infrastructure. Whereas, autonomous orchestration of mission critical services among available resources might address the performance requirements.

(4) *Standardization*: Different layers of IoT are studied within many standardization

[3]General Data Protection Regulation-GDPR. Available online: `https://gdpr.eu/` (Accessed on the 15th of December 2020).

activities. However, there is little consensus regarding which layers and relevant techniques should be standardized and which layers should remain open to be designed. In addition, governments showed their interest in standardization and their involvement implies innovation restrictions due to ever stricter regulations. New requirements for IoT are defined by IoT organizations such as OpenFog, the Industry 4.0, Made-in-China 2025 [38], and the Industrial Internet Consortium. New activities are expected to come from ETSI, IEEE, IEC, ISO, FIWARE and oneM2M, to name a few. The advancements in standards should cover every ICT field such as connectivity (e.g., 5G and satellite connections), data format and models (e.g., semantic interoperability and data contextualization), sensing, actuations and security at all levels.

This thesis moves towards the path of addressing the four research challenges presented above.

1.7 Outline of this thesis

This thesis provides a set of novel approaches to deal with two aspects of the hyperconnected IoT: i) multiple stakeholders scenario, and ii) decentralization. The decentralization of IoT platforms happens with the deployment of heterogeneous computing nodes placed into a range that varies from close to the devices (i.e., for or edge) or in the cloud. In addition, we consider a multi-party scenario where the IoT systems are not managed and owned by a single authority but there are diverse stakeholders that, depending on the use case, play one of the different roles of IoT data producer, data consumer or service developer. Our research study starts with a preliminary work proposing a solution to cope with multiple IoT network slices, owned by different parties, on a shared 5G network infrastructure. In addition, we propose a solution to decouple the IoT service development from the IoT service orchestration among fog/edge computing (happening at the edge) and cloud computing. Further, we investigate the standardization and open source background focusing on the 5G edge computing with ETSI MEC (Multi-access Edge Computing), proposing an enhancement to support IoT deployments, and the context information management analyzing FIWARE, one of the most prominent open source IoT framework. Once chosen the base IoT framework to start from, we address the trust problem that arise into multi-parties scenario. Therefore, we propose an architecture to federate peers of decentralized IoT platforms that can be scaled by design, taking into consideration enterprise security aspects. We, then, present a new data usage control approach called IntentKeeper to enable preventive data usage enforcement for federated data analytics in a proactive manner. Finally, we demonstrate the actual building of an IoT services ecosystem among 27 cities and the collaboration of diverse stakeholders among industry, governance and academia achieving their goals. We also present examples of hyperconnected IoT services addressing crowd mobility in urban environments with experiences in Australia and Spain, and a portable smart city monitoring system applied on 16 IoT deployments in Europe and Korea.

The remainder of this thesis is structured as follow:

- In chapter 2 we target the needs to efficiently tailor network operator infrastructures to IoT diverse requirements through network slicing. Differently to the long-term agreements approach between network operators and tenants as MVNOs, in this chapter we focus on a new business model where network operators offer network slices as a service (SlaaS). We propose a novel system comprising an IoT Broker managing massive IoT network slices services and a Network Slice Broker that through bi-directional negotiations are able to efficiently allocate and orchestrate network resources. We also analyze the major challenge for developers to program their services to leverage benefits of fog computing. We propose FogFlow that decouples the IoT service implementation to the IoT service orchestration among decentralized computing nodes. We assess FogFlow's performances based on microbenchmarking.

- In chapter 3 we analyze the Multi-access Edge Computing (MEC) for 5G to address the stringent requirements of critical applications (such as eHealth, automotive). We propose an architecture compliant with the ETSI MEC standard for seamless integration of IoT platforms. Further, we analyze the background of the widely used open source FIWARE framework and its application on three real scenarios. We compare the potentiality of FIWARE against both commercial and other open sources frameworks.

- In chapter 4 we introduce a secured platform where a federation overlay is distributed among parties and the control over the data is delegated to data owners. We show that our approach is scalable by design, since it allows iterative formation of multiple levels of domains thanks to the transparent nature of its federation approach. Experiments show that the overhead introduced is minimal when considering wide IoT deployments and, in some scenarios, our platform performs even better than centralized approaches.

- In chapter 5 we show IntentKeeper, a solution to enforce data usage control on a federation of decentralized IoT platforms. IntentKeeper allows data producers to specify their intents through data usage policies and data consumers with their service usage without considering the complexity of the underlying system. Moreover, IntentKeeper enforces preventive and proactive data usage control for better security and efficiency through joint decisions based on policy enforcement and service orchestration. We validate the system with an automotive scenario and we perform microbenchmarking to assess the performances.

- In chapter 6 we show the advantage of the hyperconnected IoT also in terms of sustainability. We take the unique opportunity to build a real smart city IoT services open ecosystem given by the EU project SynchroniCity. This is due to SynchroniCity's concrete implementation of a hyperconnected IoT overlay platform (as described in chapter 3) and the involvement of tens of heterogeneous stakeholders, among industries, academia, small and medium enterprises (SMEs) and cities' governance. Each IoT service in the ecosystem is not bound to a specific city IoT deployment and reused for different smart city solutions. The results of this activity comes from the implementation of 35 city solutions in 27 cities between Europe and South Korea in a time span of two years (2018-2019).

- In chapter 7 we discuss the challenges and the recent advancements to build crowd mobility analytics in urban environment benefiting from the information sharing across crowd management stakeholders. In particular, this chapter discusses real world pilots in Gold Cost, Australia and Santander, Spain. Further, we present a smart city monitoring system, namely Smart City Magnifier, that semantically transforms the data, link data to entities and entities to entities, and augment data by computing indicators. The service is run over 16 IoT deployments in Europe and South Korea.

- In chapter 8 we reports our conclusions of this thesis and open challenges to be tackles in future research.

Chapter 2

Background on IoT: 5G and Fog/Edge Computing

The increasing deployment of smart devices using mobile networks is pushing operators to consider efficient ways to tailor their infrastructure to the Internet of Things (IoT) diverse requirements and traffic characteristics. A promising approach to address this need is the concept of *network slicing*, which aims at allocating portions of network resources to specific tenants, such as enhanced mobile broadband (eMBB), IoT, e-health, connected vehicles, etc. While this has been traditionally done with long-term agreements between network operators and tenants as MVNOs, in this work we focus on a new business model where network operators offer network slices as a service (SlaaS). In particular, we propose a novel system comprising an *IoT Broker* managing massive IoT network slices services and a *Network Slice Broker* that through bi-directional negotiations are able to efficiently allocate and orchestrate network resources.

The 5G benefits to boost the performances in terms of latency, throughput and scale (number of handled devices) might be jeopardized if the services are relying only on the cloud. For example, a smart city infrastructure forms a large scale Internet of Things (IoT) system with widely deployed IoT devices, such as sensors and actuators that generate a huge volume of data. Given this large scale and geo-distributed nature of such IoT systems, fog computing has been considered as an affordable and sustainable computing paradigm to enable smart city IoT services. However, it is still a major challenge for developers to program their services to leverage benefits of fog computing. Developers have to figure out many details, such as how to dynamically configure and manage data processing tasks over cloud and edges and how to optimize task allocation for minimal latency and bandwidth consumption. In addition, most of the existing fog computing frameworks either lack service programming models or define a programming model only based on their own private data model and interfaces; therefore, as a smart city platform, they are quite limited in terms of openness and interoperability. To tackle these problems, we propose a standard-based approach to design and implement a new fog computing-based framework, namely FogFlow, for IoT smart city platforms. FogFlow's programming model allows IoT service developers to program elastic IoT services easily over cloud and edges. Moreover, it supports standard interfaces to share and reuse contextual data across services. To showcase how smart city use cases can be realized with FogFlow, we describe three use cases and implement an example application for anomaly detection of energy consumption in smart cities. We also analyze FogFlow's performance based on microbenchmarking results for message propagation latency, throughput, and scalability.

2.1 Internet-of-Things network slicing on 5G

With the continuously increasing number of connected Internet-of-Things (IoT) devices, foreseen as tenths of Billion by 2020, as well as with even more IoT use-case enablers, the IoT world has got its momentum in the small and medium-sized enterprises (SMEs) market. To this aim, the evolution of new network technologies is enlarging its horizon by exhibiting as one of their strengths a bigger flexibility on network definition and network virtualization. The context of the 5th generation of mobile network, namely 5G, is further enriched when the *network slicing* concept comes into play.

The Next Generation Mobile Network Alliance (NGMN) defines the 5G network slice as a driver paradigm to run multiple self-contained logical networks as independent business operations on a shared physical infrastructure [39]. Therefore, each network slice represents a virtualized independent end-to-end network allowing infrastructure providers to deploy different architectures in parallel. Leveraging on this concept, the infrastructure providers may customize their own networks by opening their facilities to novel business players, such as virtual mobile network operators (VMNOs), third-parties as well as Over-The-Top (OTT) applications, as shown in Fig. 2.1. Such entities behave as tenants of the same physical infrastructure. This brings new challenges in designing a network resources allocation policy, which must guarantee the resource isolation principle and, at the same time, it improves the multiplexing gain resulting in a more cost-effective resource allocation for the infrastructure provider.

While the flexibility introduced with network slicing dynamics fosters a network virtualization evolution, infrastructure providers do not quantify yet the real benefit brought to their current business cases. There is a real need of assessing and brokering the network slicing operations between infrastructure providers and different tenants. Recently, a novel logically centralized entity was defined, namely a *capacity broker* [40], considering its mapping into current 3GPP architectures and in charge of network slicing admission control operations. This functional block has been extensively improved in [41] to provide a means for optimally allocating and configuring Radio Access Network (RAN) slices based on on-demand network slice requests. Therefore, the network operators efficiently face with the network paradigm change by providing network slicing capabilities as a service, namely Slice as a Service (SlaaS).

The main benefit introduced by a multi-tenant-enabled network is the ability for heterogeneous industrial segments to acquire and use the same network infrastructure. The IoT world can be envisioned as the most suitable customer exploiting a self-managed and isolated slice of network resources, given the high heterogeneity level of its traffic

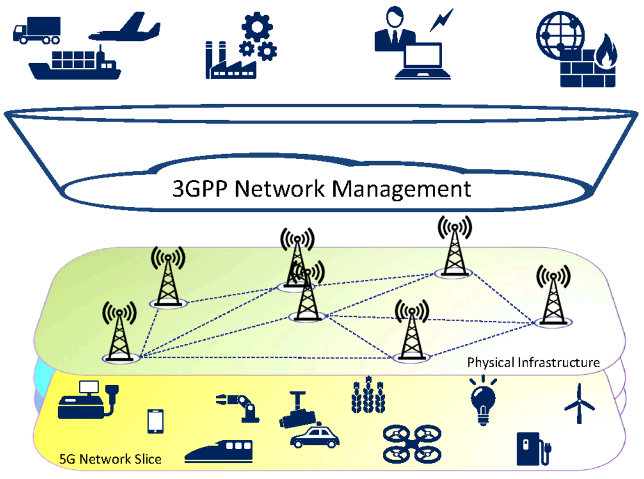


Figure 2.1: 5G Network Slicing concept

requirements [42]. The advantage of engaging IoT traffic is two-fold: (*i*) it is flexible enough to be reshaped based on the network conditions, (*ii*) it may require advanced Service Level Agreements (SLAs) for very short periods, which, in turn, translate into an additional profitable gain for the network operator. Fig. 2.2 introduces an IoT system built on top of a 5G network slice, wherein sensors communicate wirelessly (e.g., Bluetooth, ZigBee, LoRa) with an IoT Gateway (GW), which is capable of handling, storing and exposing data through 5G facilities. The IoT platform, then, enables the communications between IoT applications and “things” in a service-oriented fashion.

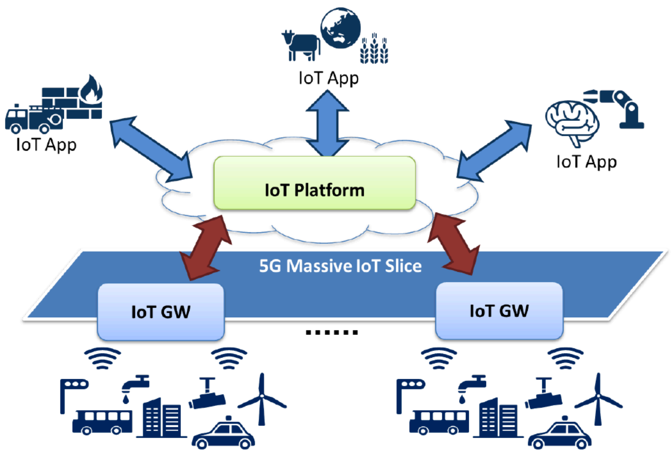


Figure 2.2: IoT System deployed on 5G Network Slice

A number of research works proposed to optimize IoT traffic flows based on different Quality of Service (QoS). An example is provided by [43], where an advanced resource allocation scheme is suggested to optimally select IoT data streams to be processed in a cloud environment and to drastically reduce the upload bandwidth by means of prediction algorithms. Therefore, the idea of an *IoT Broker* in charge of interacting with external application while optimally delivering different data information is taking off, as suggested in [44] where the authors propose a shortest processing time algorithm for scheduling web-based IoT messages. In addition, a central IoT Broker is best suited to shape IoT traffic using service-aware QoS at application level. Some research has been done in order to analyze QoS requirements for the IoT domain [45] including specific IoT services. [46] goes a step forward and proposes an optimization approach for a specific class of IoT traffic, such as periodic reports of sensor networks, within given network resources. This scheme permits to minimize the size of the network capacity without incurring in traffic congestions for occasional bursts of traffic.

None of those solutions considers an IoT Broker with a slicing traffic-aware feature. To the best of our knowledge we pioneer the idea of blending together the IoT traffic reshaping features with a 5G network slice broker, pursuing the network utilization maximization and QoS satisfaction. Differently from well-known game theory approaches, coalition schemes and sealed-bid systems with resource efficiency or profit objectives ([47, 48]), this chapter targets solving the provider-customer problem of efficiently allocating/maintaining/configuring 5G network slices for IoT tenants.

In particular, this work provides the message flow between the IoT platform and the 5G network management responsible for properly configuring network slices to (*i*) limit (e.g., by changing granularity, quality and frequency [49] of requested information) the IoT messages load when 5G network congestions occur, (*ii*) rescheduling IoT messages for underloaded periods of time, (*iii*) prepare 5G network facilities (e.g., by rescaling other network slices, offloading, denying) when mission-critical messages must be exchanged between IoT players in safety contexts.

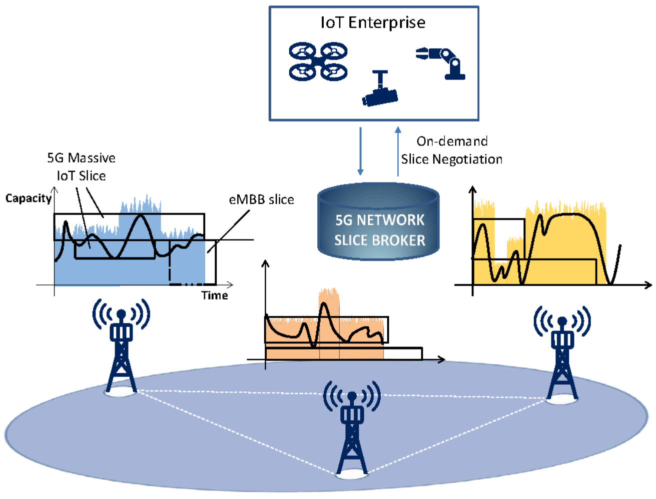


Figure 2.3: 5G Network Slicing Brokering operations

2.1.1 Problem Statement

Along the direct communication between infrastructure provider and tenant, several challenges are identified. On the one hand, the infrastructure provider applies and defines optimization algorithms for maximizing the allocation of virtual network slices aiming at the maximization of the revenues. On the other hand, the tenants (network slice customers) aim at minimizing the slice parameters with the dual target of minimizing the costs while keeping affordable the Quality of Service required. This usually results in a sub-optimal slicing configuration and limited resource monetization.

In the following, we provide a detailed analysis of IoT traffic reshaping features and 5G network slice capabilities, separately. This paves the road towards our optimized joint solution, presented in Section 2.1.2.

IoT Broker

IoT communications are enabled by means of an *IoT Broker component* [50]. This entity serves as a middleware that exposes to IoT applications a northbound interface for IoT data requests, which can be conveyed either through a conventional network means, which does not involve resources limitation issues (e.g., fixed-line technology means), or through massive IoT network slices, as shown in Fig. 2.2. The IoT application developers seamlessly interact with the IoT Broker that takes care of interfacing with different IoT Gateways (GWs) to retrieve all needed information.

The interaction between IoT applications and IoT GWs can be synchronous and asynchronous. The former is a query-response interaction, where applications inquire for data, and expect a single response. The latter follows the subscription-notify paradigm: once subscribed to an IoT data service, the IoT application is notified with the corresponding data, when an update occurs. In general, the IoT Broker maps the northbound requests onto a set of southbound requests to IoT GWs. Conveniently, the subscription-notify paradigm activates data flows of an IoT service between IoT platform and IoT GWs upon the reception of the subscription request, differently from a publish-subscribe scheme where data is continuously pushed to the IoT platform regardless of the IoT applications' interests. The IoT Broker is meant here to be a single logical component, which can be scaled if necessary. How to reach this scalability is out of the scope of this work.

5G Network Slice Broker

The *5G Network Slice Broker* is the network component in charge of (*i*) interfacing with external network tenants in order to accept/reject new network slice requests, (*ii*) managing the slice instantiation/resize/maintenance/deletion operations, (*iii*) monitoring slice traffic [51]. Fig. 2.3 shows a self-explained example, where the 5G Network Slice Broker receives a new slice request from an IoT Enterprise asking for a certain amount of network resources. If network capabilities are enough to accommodate the new slice request, the 5G Network Slice Broker instantiates a new slice by instructing the RAN elements to dedicate a given portion of resources. This example mostly focuses on the network slicing management on RAN premises. However, it can be readily extended to the transport and core network elements.

Decoupled optimizations applied on both concepts drive the system toward suboptimal states, as no direct installed communication prevents the system from reacting dynamically.

2.1.2 Joint IoT Slice Traffic-aware Solution

When the 5G network slice customer-provider relationship is broken down, the network slice brokering process acquires more knowledge about the real tenant application needs thereby optimally instantiating/configuring/scaling network slicing resource requests. Conversely, the IoT traffic might also be optimally reshaped to account for unexpected 5G network congestions.

Fig. 2.4 depicts a system architecture where the IoT Broker and 5G Network Slice management communicate and efficiently orchestrate IoT traffic and/or network slicing operations. IoT-related messages reach the IoT platform (data-plane) or the IoT GWs (control-plane) through the IoT massive 5G Slice. Control-plane messages are meant to dynamically adjust the amount of data exchanged between the IoT GWs and the IoT Broker. The IoT Broker might decide to enhance the quality of the data traffic (e.g., more fine-grained, smaller sampling period) when 5G network premises are underutilized or, conversely, it might ask to worsen the data quality (or delay after collecting a larger set of them), when network condition degradation or network resources preemption occur. Network resources assigned to the 5G massive IoT slice are dynamically managed by the 5G Network Slice Broker through a dedicated control-plane channel. This communication might trigger a slice resources update to cope with unexpected and rapid network changes (such as network congestions, additional network slice instantiations, IoT SLAs re-negotiations).

Interestingly, the IoT Broker provides different features: (*i*) shaping the IoT traffic (transmitted through the IoT massive 5G slice) by properly choosing the set of IoT GWs for satisfying the data request, or the QoS parameters of the southbound subscriptions (e.g., data granularity, notification frequency), (*ii*) measuring the data traffic in order to monitor traffic fluctuations, changes of notification frequency (from the IoT GWs), changes of QoS parameters in query/subscriptions messages, (*iii*) optimizing the data traffic in order to maximize the utilization efficiency of the 5G network slice resources while still satisfying IoT applications QoS requirements.

Interface and Data exchange

A functional view of our system solution is shown in Fig. 2.4. When requests arrive from the application layer, minimum QoS requirements are evaluated against the status of the IoT traffic load. The *QoS Aggregator* functional block aggregates overlapping QoS requirements coming from different applications. For example, road traffic condition of the same area with a certain granularity might be requested for traffic light coordination and for bus scheduling optimization. The output of the *QoS Aggregator* is a unique

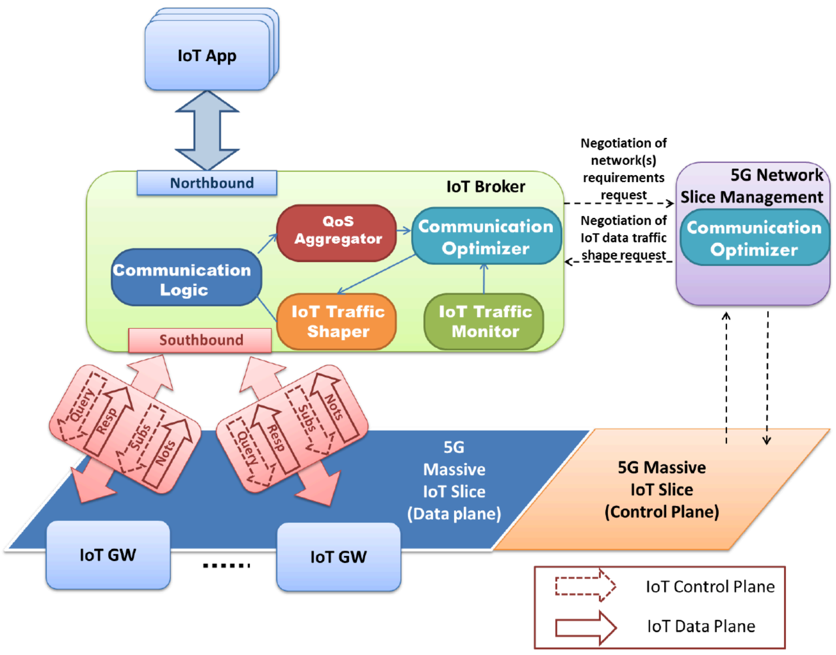


Figure 2.4: Joint IoT/Slice Broker Orchestration Architecture

set of QoS requirements per area by considering the more stringent ones. In addition, QoS models all the synchronous requests into aggregated minimum QoS requirements for each service, similarly to asynchronous traffic. These new requirements are then further aggregated to the QoS requirements of the asynchronous communication.

Aggregated QoS parameters are fed to the *Communication Optimizer* (CO) functional block, which finds a good trade-off between minimizing the SLA of the 5G massive IoT slice and satisfying the QoS requested. In case of IoT slice resource over-provisioning, it might trigger the 5G Network Slice Broker to *dynamically re-negotiate the slice size.* As output, the CO issues the actual QoS parameters (quality of data, frequency of data sampling and granularity of data aggregation) for each service.

The *IoT Traffic Shaper* (TS) functional block is in charge of shaping the southbound traffic, given as input the QoS parameters. TS chooses the IoT GWs, providing requested services (e.g., crowd behaviour, video surveillance, traffic situation) within a given geographical scope, and, for each of them decides specific QoS parameters carried on the southbound request. Then the *Communication Logic* (CL) functional block (*i*) binds the northbound request to the southbound requests, (*ii*) issues the southbound requests to the IoT GWs and, (*iii*) once the IoT data is flowing from the southbound interface, it returns messages to the IoT applications.

The southbound IoT traffic is continuously monitored by the *IoT Traffic Monitor* (TM) functional block, which feeds back CO with the current status of the IoT network slice. The traffic load might increase or decrease due, for example, to sensors availability or changes on the data produced for critical events (e.g., a big crowded event).

When current network slice resources are not enough to accommodate reshaped IoT

Table 2.1: Examples of QoS parameters for different IoT Services

	Data Quality	Geographical Granularity	Notifications Frequency
Video Surveillance	1.3, 3, 5, 8, 10 Megapixels	Single Camera, Grid 8x8 for: Building, Streets, Neighbourhood, City	10 fps, 1 fps, 0.5 fps, 0.1 fps, 0.05 fps, 0.015 fps ($\sim$ 1 per minute)
Road Situation	Historical Record, Averaged last 10m, Only Car Count & Speed Avg., Norm. Traffic Situation	Sensors, Averaged by: Building, Streets, Neighbourhood, City	10s, 30s, 1m, 5m, 10m, 30m, 1 hour
Air Quality	All Sensed Particles, Pollutant levels, Norm. Air Quality	Sensors, Averaged by: Building, Streets, Neighbourhood, City	10s, 30s, 1m, 5m, 10m, 30m, 1 hour
Crowd Behaviour	WiFi Sniffer Record, Detected Devices, Crowd Pattern, Crowd Estimation	Sensors, Averaged by: Building, Streets, Neighbourhood, City	30s, 1m, 5m, 10m, 30m,1 hour

data traffic requests without infringing QoS requirements, CO might request the 5G Slice Broker to scale up the network slice size. The 5G Network Slice Broker is constantly monitoring the traffic of instantiated 5G network slices by checking that the traffic does not exceed the network slice boundaries. When a re-negotiation is required, the 5G Network Slice Broker checks internally whether it is feasible to satisfy such a request without breaking other slice SLAs. If the network conditions allow the 5G Network Broker to accommodate this network slice resources upgrade, the 5G Network Slice Broker provides the new amount of granted resources to the CO.

Similarly, the 5G Network Slice Broker may detect an over-provisioning of available 5G resources in one of already running network slices and, it might request CO to reshape the IoT data traffic. Automatically, the *Communication Optimizer* calculates a feasible solution for the IoT resources allocation. If a successful solution is found, the IoT Broker responds back to the 5G Network Slice Broker with a scaled amount of network slice resources.

To further validate our architecture, in the next sections we show in details two use-cases wherein this joint orchestration is strongly recommended. We then propose a problem, mathematical analysis and practical algorithm for one of those use-cases, in Section 2.1.3.

Use case: City Council Network Slice

We envision a city council owning a virtualized infrastructure (or network slice prior granted) and offering (part of) it as a slice service to different municipal domains, e.g., police, homeland security, public transportation companies, domestic energy providers, and markets association (shops or shopping malls). Both IoT Broker and Network Slice Broker reside within the same administrative premises pursuing the same objective: maximization of network utilization and reduction of QoS violations. In this case, IoT Broker, as an infrastructure tenant, manages the massive IoT network slice. Other tenants can be envisaged as the other municipal domains sharing the same physical infrastructure.

Different IoT services might have different QoS parameter values (see Table 2.1). For example, *Road Situation* service might be implemented as a set of compound sensors (comprehensive of road occupancy sensor, car counting and speed sensor together with an embedded algorithm for computing a normalized value of traffic situation) installed on the streets. A low data quality level could be the retrieval of only a summarized value of the traffic situation whilst a higher data quality could allow the transmission of all the aggregated sensed information (i.e., car counting, speed average, road occupancy in the last hour) till the highest quality level, which configures the transmission of all recorded data, such the speeds history of cars using the road. An orthogonal QoS parameter is the geographic granularity data. Using again the example of the Road Situation service, the data transmitted over the network slice might be with the highest granularity (retrieving all real compound sensors data) or with a lower granularity (assuming a data aggregation performed by the IoT GW per geographical scope, such as urban blocks, streets or neighborhoods). Also the timing of the messages sent through the network slice can be controlled. In this case, the IoT GW might send messages every 10s, 30s, 1 min, 2 min, 5 min, 10 min, 30 min or 1 hours.

As another example of IoT services, we might account for the *Video Surveillance*. The quality of the data is the resolution of the recorded images. Changing the granularity automatically configures the IoT GWs to send a grid of frames coming from different cameras surrounding the same geographical object, with a total amount of pixel indicated by the quality of data. In case the number of cameras is larger than the allowed number of cells of the grid, the IoT GW chooses images to be sent through certain selection algorithms, not analyzed here. The frequency of the messages affect the bitrate of the data flow, such as 30 fps, 10 fps, 1 fps, 0.5 fps, 0.1 fps, 0.05 fps, 0.015 fps ($\sim$ 1 per minute). In Table 2.1, we have summarized QoS configurations also for: *Air Quality* service, in charge of monitoring the pollution of the air and based on weather compound sensors; *Crowd Estimation and Behaviour* service, which infers the crowd estimation in public spaces and their mobility pattern based on Wi-Fi packages sniffers.

Use-case: Cooperation between different domains

We also target a different network use-case: network operator owning the Network Slice Broker [52], whereas a massive IoT network slice is already instantiated (with an IoT Broker). 5G network Slice Broker and the IoT Broker belong to different administrative domains so that they selfishly aims to increase own network performance. An interaction may be established aiming at: improving the resource utilization efficiency (from the network operator perspective); reducing the network slice cost (from the IoT tenant's perspective) by offering a better utilization fee for a flexible and dynamic adaptable slice. Pricing models used for such a relationship are out of the scope of this work.

While those user scenarios lay the basis for this novel cooperation between 5G Network slice management and IoT world, they are not intended to be exclusive or exhaustive. However, they provide a solid basis for evaluating and fostering the adoption of such jointly orchestration. In the next section, we present an insightful analysis of such a problem shedding the light on practical algorithmic solutions.

2.1.3 Analysis and Practical Solution

We rely on the city council network slice use case, wherein the compound effect of an IoT Broker and a 5G network slice management benefits the same administrative domain, as explained in Section 2.1.2.

The IoT application subscribes a particular service a asking for sensor information updates within a given geographical scope s. We assume that our system area $\mathcal{S}$ is a grid divided into multiple areas $s \in \mathcal{S}$ not overlapping [1]. Services a might be, and not limited to, the examples listed in Table 2.1. Each of these services results in a certain amount of data delivered by the IoT Broker through the 5G network facilities. The subscription request will be issued for any single area s and is defined as $r_s = (a_s, \gamma_s, \omega_s, \mu_s, \rho_s)$. $\gamma_s \in \Gamma$ index specifies the granularity of such information, such as individual sensor, building, street, neighbourhood or urban context. Moving from fine-grained granularity (sensors level) to data aggregation may require IoT GWs to perform data analytics operations by considering average, peak or other statistical values. While this drastically reduces network slice congestions, it may prevent the IoT application from acquiring detailed information. Furthermore, we also denote the frequency of data updates with index $\omega_s \in \mathcal{W}$, which may range from 10s to 5 min. $\mu_s \in \mathcal{Q}$ specifies the quality of delivered information, whereas $\rho_s \in \mathcal{P}$ defines the priority of the subscription request. Please note that a higher priority index corresponds to a higher cost for being guaranteed with required parameters [2]. All these sets are discrete sets of values that drive the overall system utilization, as diverse service configurations require different amounts of data exchanged. Our joint system leverages on this powerful trade-off to accommodate first high-priority service requests into the available network slice capacity while differing low-priority traffic flows.

Service requests for different geographical areas come periodically and are processed by the IoT Broker. As shown in Fig. 2.4, the IoT Broker forwards such request parameters to a *Communication Optimizer* functional block, in charge of optimally scheduling service requests into the slice capacity following a priority-based policy. *Communication Optimizer* block is defined as a dual functional entity: it aims to properly reshape the service configuration in order to fulfil the network slice capacity limits. As soon as the application service requests exceed the network slice capacity, it may trigger its counterpart on the 5G network management to promptly adjust the network slice boundaries if no network congestions occur. Analytically, we can express the optimization problem as follows

[1] This assumption makes tractable the problem analysis. However, it can be easily extended for advanced cases with overlapping spatial areas.

[2] We assume that the IoT entity aims at minimizing the overall cost while getting an acceptable level of quality. A detailed discussion about this mechanisms is out of the scope of this dissertation. However, advanced mechanisms for optimally adjusting such priority/cost levels might be considered without affecting the problem analysis presented in this chapter.

Problem `IoT-Optimizer`:

$$\begin{aligned} \text{maximize} \quad & \sum_{s\in\mathcal{S}} (x_s - p_s) \\ \text{subject to} \quad & \sum_{s\in\mathcal{S}} g_a(\gamma_s) q_a(\mu_s) f_a(\omega_s)(x_s - p_s) \leq \eta_{\text{o}}; \\ & x_s \geq \mathbb{1}_{\{\rho_s \geq \bar{\rho}\}}, \quad \forall s \in \mathcal{S}; \\ & x_s, p_s \in \{0;1\}, \quad \forall s \in \mathcal{S}; \end{aligned}$$

where $f_a(\cdot), g_a(\cdot), q_a(\cdot)$ are discrete functions providing the amount of datarate given certain frequency, granularity and quality values, respectively, for a particular service a, whereas x_s is a binary value indicating whether the service request for area s can be scheduled. The datarate might change over time due to a new IoT resources availability (e.g., new sensors installed under an IoT service) or because of critical situations (e.g., more data is generated outside a stadium right after a crowded event). The *IoT Traffic Monitor* measures the IoT traffic and updates the $f_a(\cdot), g_a(\cdot), q_a(\cdot)$ functions. η_{o} is the capacity assigned to the IoT network slice and p_s is a binary value (penalty) indicating that the service request for area s cannot be admitted within the current amount of assigned network slice resources. $\bar{\rho}$ is a threshold defined by the *Communication Optimizer* to force the IoT Broker to accept service requests above a certain priority level (e.g., security issues). The main constraint defines the total amount of data exchanged $(g_a(\gamma_s)q_a(\mu_s)f_a(\omega_s))$ per area s by the IoT devices, which must be scheduled into the network slice traffic capacity. The tuple $(\gamma_s, \mu_s, \omega_s)$ is sent to the *IoT Traffic Shaper*, which adjusts the service configurations on the IoT GWs, accordingly. When $\sum_{s\in\mathcal{S}} p_s \geq 0$, some IoT service requests cannot be accommodated and an intervention is required by the network slice management. Therefore, the *Communication Optimizer* sends an upgraded network slice request, which comprises $\lambda_{\text{o}} = \eta_{\text{o}} + p_{\text{o}}$, where $p_{\text{o}} = \sum_{s\in\mathcal{S}} p_s \left(g_a(\gamma_s)q_a(\mu_s)f_a(\omega_s)\right)$ is the exceeding capacity required by the IoT slice. On the other side, the *Communication Optimizer* installed on the 5G network slice management deals with the following optimization problem to optimally scheduled its network resources between different network services $i \in \mathcal{I}$

Problem `Network-Optimizer`:

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in\mathbb{I}} \eta_i \\ \text{subject to} \quad & \sum_{i\in\mathbb{I}} \eta_i \leq C_{\text{slice}}; \\ & \eta_i \geq \lambda_i, \quad \forall i \in \mathcal{I}; \\ & \eta_i \in \mathbb{R}_+, \quad \forall i \in \mathcal{I}; \end{aligned}$$

where λ_i is the amount of network resources in terms of datarate required by the tenant i, whereas C_{slice} is the slice capacity managed by the 5G controller [3]. The output directly

[3] We assume that the slice capacity is properly designed from the network management perspective. If network congestions occur, Problem `Network-Optimizer` might be unfeasible so that the 5G network slice management needs to discard some network slice request updates, based on a priority basis. For further information, we refer the reader to advanced 5G network slicing brokering mechanisms as explained in [41].

Algorithm 1: IoT Slice-aware Traffic Optimizer

```
1. Initialise sets $\mathcal{K} \leftarrow 0$ and $\mathcal{J} \leftarrow 0$.
2. Update high-priority set $\mathcal{K} \leftarrow s : \rho_s \geq \bar{\rho}$ and low-priority set
   $\mathcal{J} \leftarrow s : \rho_s < \bar{\rho}, \forall s \in \mathcal{S}$.
3. Sort $\mathcal{K}$ in a decreasing order and $\mathcal{J}$ in an increasing order based on
   $g_a(\gamma_s)q_a(\mu_s)f_a(\omega_s)$.
4. Place $s \in \mathcal{K}$ into $\mathcal{O}$ while fulfilling the capacity constraint
   $\sum_{s\in\mathcal{O}} g_a(\gamma_s)q_a(\mu_s)f_a(\omega_s) \leq \eta_{\text{o}}$ following the order.
5. Place $s \in \mathcal{J}$ into $\mathcal{O}$ while fulfilling the capacity constraint
   $\sum_{s\in\mathcal{O}} g_a(\gamma_s)q_a(\mu_s)f_a(\omega_s) \leq \eta_{\text{o}}$ following the order.
6. Place into $\mathcal{O}$ remaining $s \in \mathcal{K}$ and update penalties $p_s$.
7. $s \in \mathcal{O}$ will be notified as service requests accepted and
   $p_{\text{o}} = \sum_{s\in\mathcal{O}} p_s \left(g_a(\gamma_s)q_a(\mu_s)f_a(\omega_s)\right)$ will be notified to the 5G network
   management as $\lambda_{\text{o}} = \eta_{\text{o}} + p_{\text{o}}$.
```

feeds back the IoT Broker (through the *Communication Optimizer* block) by specifying the updated amount of resources assigned to the IoT slice (i.e., $\eta_{i=\text{o}}$).

Interestingly, Problem `IoT-Optimizer` and Problem `Network-Optimizer` are tightly connected, as the output of one provides the input for the other one and vice-versa. This implies that in case of emergency or security threats, if the slice size is not enough (η_{o}), the IoT Broker may trigger an update request to the 5G network management (λ_{o}) and promptly get augmented network resources to accommodate the traffic burst. Conversely, when the 5G network experiences congestions, it might ask the IoT Broker to reduce the resources utilization (η_{o}) so as to make room for other network services i.

Algorithm Description

Problem `IoT-Optimizer` above-described is an Integer Linea Programming (ILP) problem. Such class of problems is known to be NP-Hard [53]. For the sake of brevity, we skip the formal proof of NP-Hardness and NP-Completeness. However, small instances of the problem with few levels of granularity, frequency and quality may help in finding an optimal solution within a polynomial time by means of an exhaustive search.

Here we present a heuristic solution for solving Problem `IoT-Optimizer`, as most of the ready-to-use algorithms in the literature properly address Problem `Network-Optimizer`. The pseudocode is listed in Algorithm 1. The IoT Slice-aware Traffic Optimizer splits the service requests into two subsets: in the former there are only high-priority service requests, i.e., where $\rho_s \geq \bar{\rho}$ whereas in the latter all the other services requests s. After placing high-priority requests within the slice capacity η_{o} following a decreasing order (the rationale is based on first fit decreasing schemes proposed in the literature for well-known knapsack problems), our algorithm tries to place as many low-priority requests as possible that still fit into the available slice capacity. If some high-priority service requests are not scheduled yet, the algorithm increases the capacity limit at expenses of a penalty value p_s. In this case, the system realizes that the current slice configuration is not enough to satisfy high-priority instances requirements, and it automatically triggers the 5G network management for updating the slice capacity boundaries.

It is clear that the priority threshold $\bar{\rho}$ is a key-parameter for driving the system towards optimal solutions. However, it can be a configurable parameter chosen by the network provider for guaranteeing different levels of QoS and, in other cases, identifying emergency situations. It can be also chosen based on different pricing models to foster external administrative domains to increase the pay-off for ensuring mission critical communications.

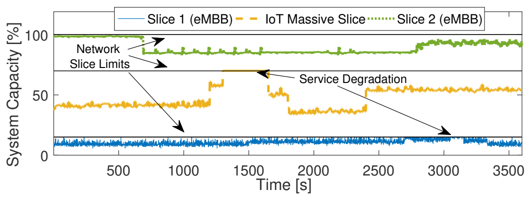
2.1.4 Performance evaluation

We carried out a preliminary simulation campaign to evaluate the compound effect of an IoT traffic-aware slice by means of an ad-hoc simulator, written in MATLAB®. In particular, we simulate a 5G network environment with 50 eNBs covering a 30 km^2 area. In this area, we deploy several IoT gateways collecting data from 50000 different sensors, including traffic sensors, air quality sensors, crowd control sensors and HD cameras, and transmitting data by means of 5G network facilities. The IoT deployment is structured onto $|\mathcal{S}| = 100$ not overlapping areas and, the service request per area might come to the IoT Broker at regular time interval equal to $n = 10$ min, with QoS configuration parameters randomly chosen. All simulation parameters are listed in Table 2.2.

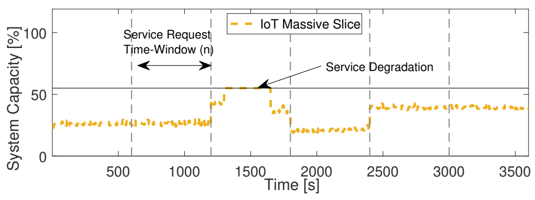
Table 2.2: Simulation parameters

Area	30 km^2	**Capacity (UL)**	75 Mb/s
$\lvert\mathcal{S}\rvert$	100	**Capacity (DL)**	150 Mb/s
Sim. Duration	1 hour	**IoT service req. (n)**	10 min.
5G eNBs	50	**Quality levels** $\lvert\mathcal{Q}\rvert$	5
Data Frequencies $\lvert\mathcal{W}\rvert$	6	**Granularities** $\lvert\Gamma\rvert$	4
Service Priorities $\lvert\mathcal{P}\rvert$	5	**Priority Threshold** $\bar{\rho}$	3

We show the impact of implementing a joint IoT traffic brokering and 5G network slice traffic-aware mechanism compared with the legacy solution, where the 5G network management takes slicing decisions, independently. We assume that the 5G network management deals with three different slices, such as *Slice 1* characterized by enhanced Mobile BroadBand (eMBB) traffic, *Slice 2* and *IoT Massive Slice*, asking for 15%, 30% and 55% of the overall system capacity, respectively. In Fig. 2.5, we show the system behavior when no joint mechanisms are devised. In Fig. 2.5b, we focus on the IoT Massive Slice traffic by marking different time windows wherein IoT service requests vary. After the third service requests set, the IoT traffic dramatically increases and the IoT Massive Slice limit degrades the quality of service, as some service requests cannot

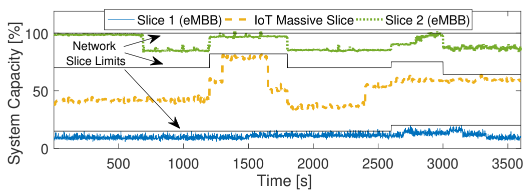


(a) Network Slices Traffic

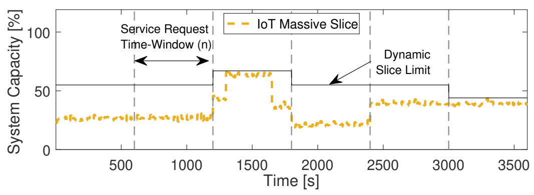


(b) IoT Massive Slice Traffic

Figure 2.5: IoT Traffic adaptation without joint mechanisms



(a) Network Slices Traffic



(b) IoT Massive Slice Traffic

Figure 2.6: IoT Network Slice Traffic-aware adaptation

be satisfied. Note that, also *Slice 1* experiences QoS degradation after 2800 seconds, as slice boundaries are fixed and not flexible.

In Fig. 2.6a, we show the same system configuration after running our joint mechanism, where we implement the IoT Traffic-aware Slice Optimizer algorithm. In this example, the 5G network management can dynamically adjust the network slice limits based on the traffic dynamics. In particular, the IoT Massive Slice requires a slice resources upgrade at time 1200s because of some high-priority service requests. Since the other network slices are underutilized, the 5G network management can promptly accommodate this request, as shown in Fig. 2.6b. When *Slice 1* requires more resources, the 5G network management triggers the IoT Broker thereby asking for a traffic reduction. This automatically leads the system to efficiently utilize available resources while avoiding service degradation.

2.2 Fog computing for Smart City

Nowadays cities are becoming more and more digitalized and connected as numerous sensors are widely deployed for various purposes. For example, deployments in smart cities include CO_2 sensors for measuring air pollution, vibration sensors for monitoring bridges, and cameras for watching out potential crimes. Those connected devices form a large scale of IoT system with geographically distributed endpoints, which generate a huge volume of data streams over time. Potentially, the generated data can help us increase the efficiency of our city management in various domains such as transportation, safety, and environment (e.g., garbage management). However, to utilize the data efficiently we need to have an elastic IoT platform that allows developers to easily program various services on top of a shared and geo-distributed smart city IoT infrastructure.

In the past, most of the existing city IoT platforms are built only based on cloud, such as CityPulse [54] and our previous City Data and Analytics Platform (CiDAP) [55]. However, this is no longer a sustainable and economical model for the next generation of IoT smart city platforms, given that many city services (e.g., car accident detection) require ultra-low latency and fast response time. Moreover, bandwidth and storage costs can be substantially high if we send all sensor data such as video frames to the cloud. Recently there is a new trend to offload more computation from the cloud and device layer to the middle layer components which are IoT gateways and edge/core networks, called *fog computing* [56]. While fog computing perfectly fits the geo-distributed nature of smart city infrastructure, it is still challenging for smart city IoT platforms to adapt to this new computing paradigm. The heterogeneity, openness, and geo-distribution of the new cloud-edge environment raise much more complexity on the management of data and processing tasks than the centralized cluster or cloud environments. Therefore, we need a sufficient and flexible programming model with open interfaces that allow developers to implement various IoT services on top of the cloud-edge environment without dealing such complexities.

The current state of art on fog computing, such as Foglets [57], has been mainly focused on how to optimize task deployment over distributed edges in terms of saving bandwidth and reducing latency. However, there is not much work that has been done to explore the programming model for fog computing. The existing studies either just reuse the programming models from existing frameworks (e.g., Apache Storm and Spark) or come up with their own programming models with non-standardized interfaces. In this chapter we argue that both solutions are not suitable for smart city IoT platforms to adopt fog computing in terms of *openness, interoperability, and programmability.*

To tackle these problems, we take a standards-based approach and propose a NGSI-based programming model to enable easy programming of IoT services over cloud and edges. In this chapter we introduce the overall architecture of our new fog computing framework, namely *FogFlow*, and also report its core technologies for supporting the proposed programming model. Furthermore, we introduce some concrete application example to showcase how IoT services can be easily realized on top of our NGSI-based programming model. The main contributions of this chapter are highlighted as follows.

- **Standard-based programming model for fog computing**: we extend dataflow programming model with *declarative hints* based on the widely used standard *NGSI*, which leads to two benefits for service developers: 1) *fast and easy development* of fog computing applications, this is because the proposed hints hide lots of task configuration and deployment complexity from service developers; 2) *good openness and interoperablity* for information sharing and data source integration, this is because NGSI is a standardised open data model and API and it has been widely adopted by more than 30 cities all over the world.

- **Scalable context management**: to overcome the limit of centralized context management, we introduce a distributed context management approach and our measurement results show that we can achieve much better performance than existing solutions in terms of throughput, response time, and scalability.

2.2.1 Smart City Use Cases

Use Case 1: Anomaly Detection of Energy Consumption: The first use case study is for retail stores to detect abnormal energy consumption in real-time. As illustrated in Figure 2.7, a retail company has a large number of shops distributed in different locations. For each shop, a Raspberry Pi device (edge node) monitors the power consumption from all PowerPanels in the shop. Once an abnormal power usage is detected on the edge, the alarm mechanism in the shop is triggered to inform the shop owner. Moreover, the detected event is reported to the cloud for information aggregation. The aggregated information is then presented to the system operator via a dashboard service. In addition, the system operator can dynamically update the rule for anomaly detection.

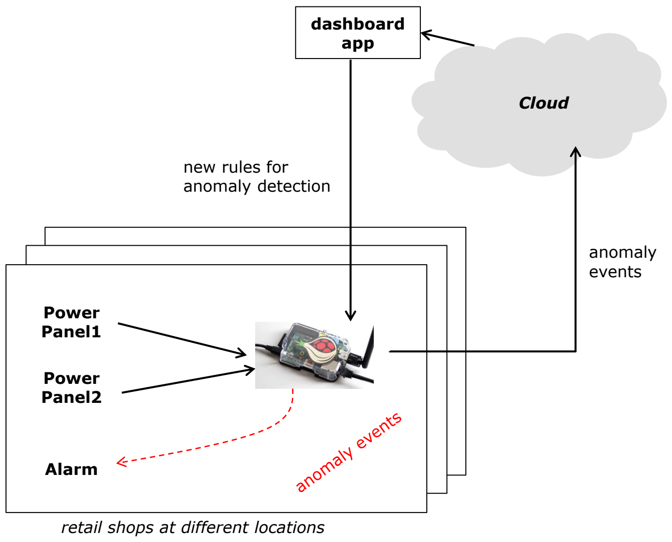


Figure 2.7: Detecting abnormal electricity usage in retail stores.

Use Case 2: Video Surveillance in Stadiums: The second use case is for providing stadium security with video survelliance and real-time analytics. Figure 2.8 illustrates this use case based on three layers: terminal gateway, IoT gateway, and cloud. In the lower layer, terminal gateway devices are deployed to process the video streams captured by cameras. In the upper layer, each stadium has an IoT gateway to perform further data processing. Terminal gateways and the IoT gateway are connected to the local area network of the stadium. In the top layer, all IoT gateway devices are connected to the cloud via the Internet. The following services are expected to be enabled.

- *Crowd counting:* Aggregation of the number of people extracted from the captured video streams and the total number of people at each area to show the stadium crowdedness in real-time.
- *Finding lost child:* When a child gets lost in a stadium, their parents ask the staff for help. Based on the picture provided by the parent, video analytics tasks are launched dynamically on demand at the edge nodes to identify the lost child in

real-time. Once the child is found, the staff is notified and a digital signage close to the child is actuated in order to ensure the safety of the child.

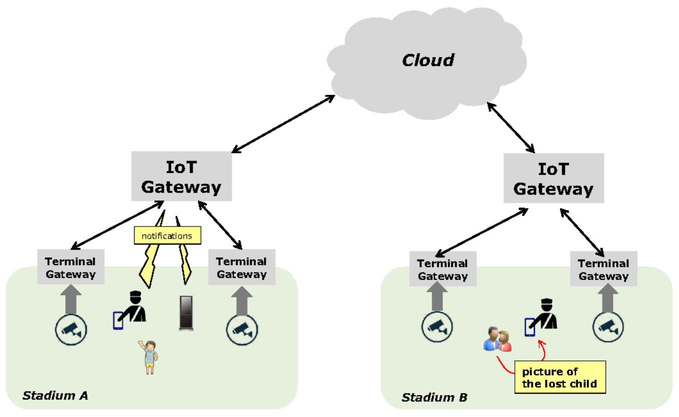


Figure 2.8: Video surveillance in stadiums.

Use Case 3: Smart City Magnifier: The last use case is an application for visualization of smart cities which we named as $\mathcal{S}$mart City Magnifier (SCM). SCM provides a user interface for displaying the results from environmental monitoring, critical situations such as safety alerts, as well as the view of city-wide deployments. Figure 2.9 illustrates SCM for environmental monitoring. Environmental monitoring results include traffic, air pollution, and crowd levels. Overall condition of the situations for each of them are shown with green, yellow, or red lights (left). Furthermore, these levels are also displayed with graphs (right). The dashboard includes the data analytics results from past measurements (historical data), current (real-time) measurements, as well as future predictions based on the current trends.

As shown in Figure 2.9, the visualization tool can be set based on three parameters: 1) *space* for specifying the geographic scope (the map view); 2) *time* for the evaluation time window or the forecasting horizon; 3) *abstraction* for defining the level of detail of the view. The level of detail varies based on the analytics results. For instance, for air pollution results, lower abstraction can be CO_2 levels, while higher abstraction can be overall air quality. Furthermore, abstraction depends on the geographic scope such that it varies if the scope is a building, a street, or a city. We will described the full-fledged design and implementation of the Smart City Magnifier in section 7.2.

2.2.2 High Level Requirements

Let us briefly discuss the main requirements of the three use cases. The general requirements for such systems include *system interoperability and openness* since different IoT system components such as things, edge nodes, middlewares (e.g., context broker), and developers need to connect through interfaces. Considering the variety of "things" in the IoT, this is a challenging task. Other than those there exist certain performance requirements. The IoT services for smart cities require certain data processing capabilities

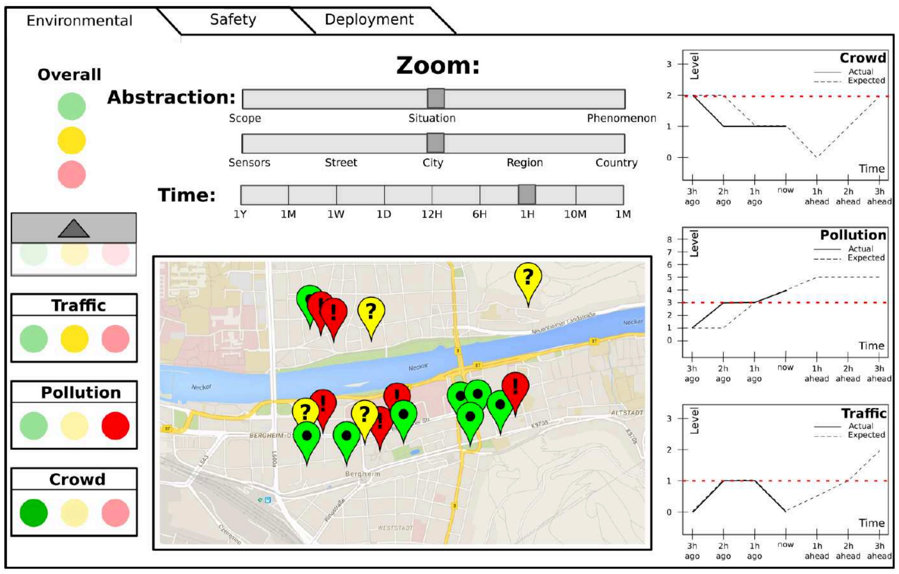


Figure 2.9: Illustration of Smart City Magnifier.

including offline big data analytics through frameworks such as Apache MapReduce as well as real-time stream processing through frameworks such Apache Spark. Therefore, a major performance requirement is *dynamic orchestration of the data processing tasks.* Various data processing tasks (e.g., video analytics, air quality measurements) need to be performed on the shared cloud and edge resources.

2.2.3 Fogflow: Programming IoT Services over Cloud and Edges

To meet the openness and interoperability requirements of IoT smart city platforms, we propose a standards-based approach for designing and implementation of the FogFlow programming model based on the two standards: *Dataflow* (de facto standard) and *NGSI* (official standard).

Dataflow is a popular programming model to decompose applications which are widely used by cloud service developers for big data processing in cloud environments. Google Cloud Dataflow and Amazon Data Pipeline are among the data processing services that are built based on the dataflow programming model. As a unified programming model for both batch and stream processing, Dataflow is still suitable for defining fog services; however, it is missing certain features or extensions to adapt to the challenges introduced by fog computing. For example, from the underlying infrastructure perspective, fog computing requires more geo-distributed, dynamic, and heterogeneous infrastructure than cloud computing. From the application perspective, fog services usually require low latency and location awareness. This is especially true for smart city applications. Thus, we introduce *declarative hints* and extend the traditional dataflow programming model for enabling efficient fog computing.

In the dataflow programming model, the data processing logic for a service is usually decomposed as multiple *operators*. Operators form a directed acyclic graph (DAG) called *topology* through the linked inputs and outputs among different operators. Traditionally operators are defined as functions with certain APIs, but this is no longer a suitable model to define operators for fog computing due to its bad isolation and limited flexibility and interoperability. Conversely, FogFlow requires service developers to define operators as dockerized applications based on NGSI [58] (details to be discussed in Sec. 2.2.3).

Let us introduce the overall system achitecture of FogFlow first with some background information on the NGSI standard. We then present the detailed design of our NGSI-based programming model and how such a programming model is supported by scalable context management and dynamic service orchestration in FogFlow.

Next Generation Service Interface (NGSI)

As an open standard interface from Europe, Open Mobile Alliance (OMA) NGSI [59] is currently used in industry and academia as well as in large scale research projects such as FIWARE [60] and Wise-IoT [61]. In 2015 an Open & Agile Smart Cities (OASC) initiative [62] has been signed by 31 cities from Finland, Denmark, Belgium, Portugal, Italy, Spain and Brazil, for adopting the NGSI open standard in their smart city platforms. It is strategically important to design our programming model based on NGSI in order to achieve openness and interoperability in the areas of IoT and smart cities.

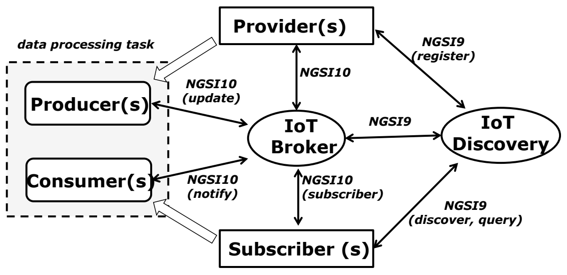


Figure 2.10: Typical NGSI-based interactions and system diagram.

From the technical perspective, NGSI defines both the *data model* and *communication interface* to exchange contextual information between different applications via context brokers. The NGSI data model characterizes all contextual information as context entities where each entity must have an ID and a type. Entities also optionally have a set of attributes and metadata related to domains and attributes. Typically metadata includes the source of information, observation areas, and the location of the IoT device. The NGSI communication interface defines a lightweight and flexible mean to publish, query, and subscribe to context entities. *NGSI10* and *NGSI9* are respectively designed for managing the data values of context entities and their availability (e.g., discovery of entities). As opposed to the existing message brokers (e.g., MQTT), NGSI not only defines a unified data model to express contextual data (both raw sensor data and derived intermediate results) but also provides missing features that are highly demanded by geo-distributed fog computing. For instance, geoscope-based resource discovery and

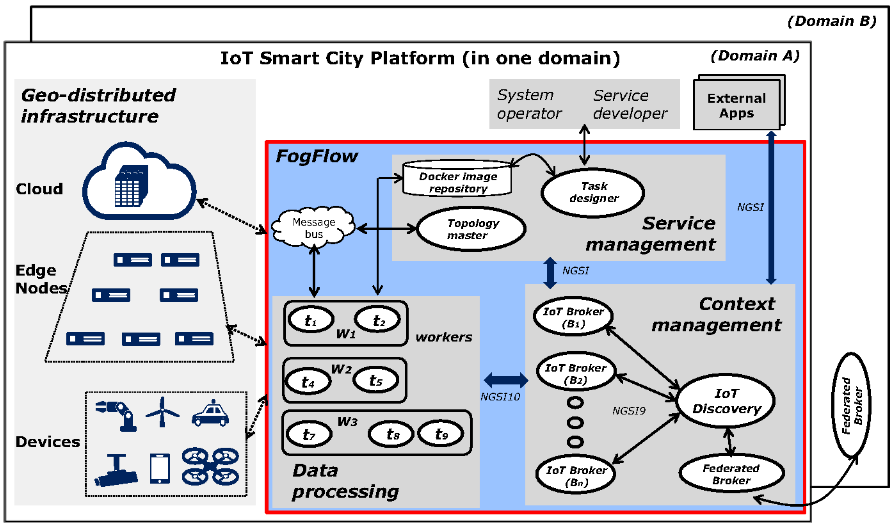


Figure 2.11: System architecture of FogFlow.

subscription are needed by our service orchestrator for dynamic configuration and management of the data processing tasks.

Figure 2.10 illustrates the usage of NGSI in different scenarios. Typically an IoT Broker (e.g., Aeron [63], Orion [64]) middleware is deployed between context provider(s) and context consumer(s) to allow them to exchange NGSI-based context entities. Meanwhile, an IoT Discovery component creates an index for the availability of all registered context entities. Providers register the availability of their context data via NGSI9 to make them discoverable. To subscribe or query any context entities via NGSI10, consumers must find out which provider offers the requested context entities via NGSI9 request to IoT Discovery. In FogFlow, each data processing task acts as a provider to publish its outputs, while it also acts as a consumer to receive its input streams. However, for easy programming of operators, the FogFlow framework dynamically registers the generated outputs via NGSI9 and subscribes the inputs via NGSI10 on behalf of data processing tasks. In the end, developers only need to deal with NGSI10 update and notify when they implement operators. More details can be seen in Sec. 2.2.3)

Architecture Overview

The system architecture of FogFlow is illustrated in Fig. 2.11. The figure includes the FogFlow framework, *geo-distributed infrastructure resources*, and FogFlow's connection with the users (system operator and service developers) and external applications through its API and interfaces. Infrastructure resources are vertically divided as *cloud*, *edge nodes*, and *devices*. Computationally intensive tasks such as big data analytics can be performed on the cloud servers, while some tasks such as stream processing can be effectively moved to the edge nodes (e.g., IoT gateways or endpoint devices with computation capabilities). Devices may include both computation and communication capabilities (e.g., tablet computer) or only one of them (e.g., beacon nodes advertising

Bluetooth signals). The FogFlow framework operates on these geo-distributed, hierarchical, and heterogeneous resources, with three logically separated divisions: *service management*, *data processing*, and *context management*.

The service management division includes task designer, topology master (TM), and docker image repository, which are typically deployed in the cloud. Task designer provides the web-based interfaces for the system operators to monitor and manage all deployed IoT services and for the developers to design and submit their specific services. Docker image repository manages the docker images of all dockerized operators submitted by the developers. TM is responsible for service orchestration, meaning that it can translate a service requirement and the processing topology into a concrete task deployment plan that determines which task to place at which worker.

The data processing division consists of a set of workers $(w_1, w_2, \cdots, w_m)$ to perform data processing tasks assigned by TM. A worker is associated with a computation resource in the cloud or on an edge node. Each worker can launch multiple tasks based on the underlying docker engine and the operator images fetched from the remote docker image repository. The number of supported tasks is limited by the computation capability of the compute node. The internal communication between TM and the workers is handled via a Advanced Message Queuing Protocol (AMQP)-based message bus such as RabbitMQ to achieve high throughput and low latency.

The context management division includes a set of *IoT Brokers*, a centralized *IoT Discovery*, and a *Federated Broker*. These components establish the data flow across the tasks via NGSI and also manage the system contextual data, such as the availability information of the workers, topologies, tasks, and generated data streams. IoT Discovery handles registration of context entities and discovery of them. This component is usually deployed in the cloud. IoT Brokers are responsible for caching the latest view of all entity objects and also serving context updates, queries, and subscriptions. In terms of deployment, IoT Brokers are distributed on the different nodes in the cloud and on the edges. They are also connected to the other two divisions (workers, task designer, TM, external applications) via NGSI. Federated Broker is an extended IoT Broker used as a bridge to exchange context information with all other Federated Brokers in different domains. For instance, Federation Broker enables communication from one deployment in a European smart city (e.g., Domain A: Heidelberg) to another in a Japanese smart city (e.g., Domain B: Tokyo). These deployments are considered as two different domains. Within the same domain, all IoT Brokers and Federated Broker are connected to the same IoT Discovery.

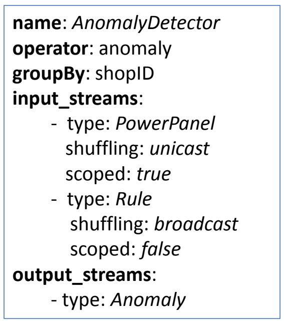
NGSI-based Programming Model

In FogFlow, an IoT service is represented by a *service topology* which consists of multiple *operators*. Each operator receives certain types of input streams, performs data processing, and then publishes the generated results as output streams. The FogFlow programming model defines the way of how to specify a service topology using declarative hints and how to implement operators based on NGSI.

Declarative Hints Developers decompose an IoT service into multiple operators and then define its service topology as a DAG in JSON format to express the data dependencies between different operators. This is similar to the traditional dataflow programming model. On the other hand, the FogFlow programming model provides *declarative hints* for developers to guide service orchestration without introducing much complexity. Currently, it requires developers to specify two types of hints in the service topology: *Granularity* and *Stream Shuffling*.

Granularity hint is associated with each operator in the service topology and represented by the "groupBy' property, as shown by a task specification example (written in YAML language) in Figure 2.12. The granularity hint is defined using the name of one stream attributes. In FogFlow, every data stream is represented as a unique NGSI context entity generated and updated by either an endpoint device or a data processing task. Different types of metadata are created by FogFlow on the fly to describe the data stream. For instance, metadata includes which device or task is producing the data stream, the location of the producer, which IoT Broker is providing the stream, and so on. Regarding the geo-distributed nature of the underlying infrastructure, some common granularity hints are geo-location related attributes such as "Section", "District", "City", or "ProducerID". These granularity hints are later used as an input by TM to decide the number of task instances to be created and configured for each operator during system runtime.

Stream Shuffling hint is associated with each type of input stream for an operator in the service topology, represented by the "shuffling" property. TM uses this hint as additional information to decide how to assign matched input streams to task instances. Based on the granularity hint, multiple task instances could be instantiated from the same operator, but they can be configured with different set of input streams. For each operator, the type of its input streams determines which type of streams should be selected to configure its task instances, but its "shuffling" property can further decide how the selected streams should be assigned to the task instances as their inputs. The value of the shuffling property can be either "broadcast" or "unicast". The "broadcast" value means the selected input streams should be repeatedly assigned to every task instance of this operator, while the "unicast" value means each of the selected input streams should be assigned to a specific task instance only once.



```
name: AnomalyDetector
operator: anomaly
groupBy: shopID
input_streams:
     - type: PowerPanel
       shuffling: unicast
       scoped: true
     - type: Rule
       shuffling: broadcast
       scoped: false
output_streams:
     - type: Anomaly
```

Figure 2.12: An example of task specification written in YAML.

Figure 2.13 shows a concrete example of how these two types of hints are used by TM to create and configure data processing tasks for service orchestration. The left side illustrates a service topology with two simple operators, A and B, which is designed for Use Case 1 in Sec. 2.2.1. The right side illustrates the execution plan generated by TM. Operator B is named as "AnomalyDetector" and its detailed specification is listed in Figure 2.12. The goal of Operator B is to detect the abnomal usage of eletricity for each shop based on a given rule. Based on this requirement, the granularity of Operator B is defined by "shopID", meaning that TM needs to create a dedicated "AnomalyDetector" task instance for each shop. Operator B has two types of input streams: the anomaly detection rule and the measurement from power panel. Assume that three power panel devices are from three different shops S1, S2, and S3. In this case three task instances (TB0, TB1, TB2) must be created because its operator granularity is based on "shopID". Also, each task instance is assigned with the stream from a specific power panel but they all share the same detection rule as another input. This is because the shuffling property of the rule input stream is "broadcast" while the shuffling property of the PowerPanel devices is "unicast".

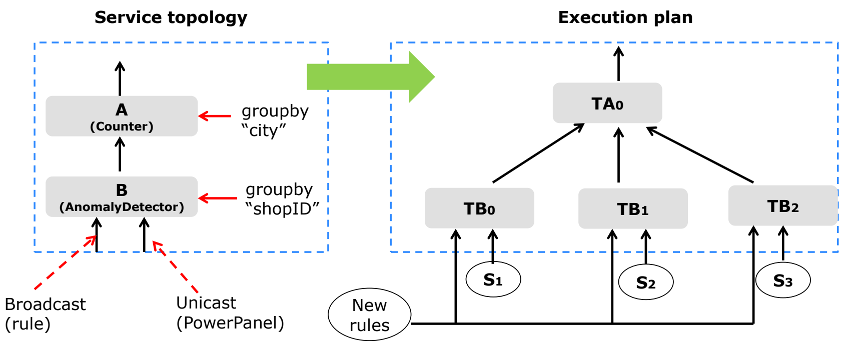


Figure 2.13: Left: A service topology example with declarative hints, right: the execution plan generated by TM.

NGSI-based Operators Developers need to implement each operator in a service topology as a dockerized application. As illustrated in Fig. 2.14, once a worker instantiates a dockerized operator application (a task instance running in a docker container), the task instance interacts with the FogFlow framework via the following steps.

First, before starting its internal processing logic, the task instance receives a JSON-based configuration from the worker through environment variables. The initial configuration includes which IoT Broker the task instance should talk to and also the metadata of its input and output streams. Later on, if there is any configuration change, those changes can be sent to the task instance via a listening port (input port). In FogFlow, the important stream metadata required by the task instance include: 1) the entity type and entity ID of the associated stream that can be used by the task instance to know which entity to subscribe as inputs and which entity to update as outputs; 2) the way of how the stream data is provided from the producer to consumers, which can be PUSH-based or PULL-based. More specifically, PUSH-based means that the stream entity will be updated by its context producer actively and context consumers can receive the updates from IoT Broker via subscriptions, while PULL-based means that the stream entity only represents the information of the context producer and the actual stream data must be pulled by the task instance from a service URL, which is part of the stream entity. For example, a temperature sensor that actively sends temperature observations periodically can be represented as a PUSH-based stream entity; a webcam that sends captured images or video streams on request can be represented as a PULL-based stream entity.

Second, after the task instance is launched and configured, it will start to get its input streams and process the received data. If the stream is PUSH-based, the task instance can receive all input stream data as NGSI10 notify via the input port without sending any subscritpion, because the worker issues NGSI10 subscriptions to the IoT Broker on behalf of the task; If the stream is PULL-based, e.g., video streams from an IP camera, the task instance needs to fetch the input stream data from a provided URL in the stream metadata.

Lastly, once some results are generated from the received stream data, the task instance publishes or announces them as output streams. If the output stream is PUSH-based, the task instance sends the generated outputs to the IoT Broker as NGSI10 update under the specified entity type and ID; if the output stream is PULL-based, the worker can register the output stream on behalf of the task. With this design we allow the worker to handle more management complexity in order to dynamically configure and establish the data flows cross different task instances. Therefore, we can try to reduce the complexity of the implementation of dockerized operators and reduce the required effort from developers; on the other hand, we can provide enough flexibility for various application use cases and also comply with the NGSI standard.

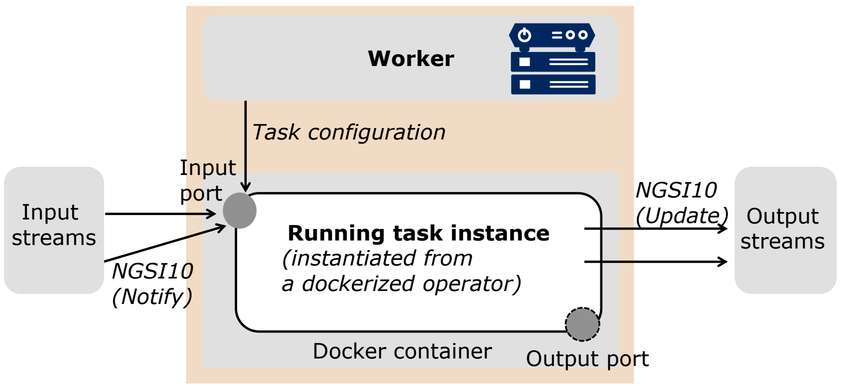


Figure 2.14: Interactions of the task instance with FogFlow.

Scalable Context Management

The context management system is designed to provide a global view for all system components and running task instances to query, subscribe, and update context entities via the unified NGSI. Our NGSI-based context management has the following additional features. These features are different from the ones provided by traditional pub-sub message brokers such as MQTT-based Mosquitto or Apache Kafka, but they play an important role to support FogFlow's NGSI-based programming model.

- It provides separate interfaces to manage both context data (via NGSI10) and context availability (via NGSI9). This feature enables flexible and efficient data management with standardized open APIs.
- It supports not only ID-based and topic-based query and subscription but also geoscope-based query and subscription. This feature enables FogFlow to efficiently manage all geo-distributed resources such as workers, tasks, and generated streams.
- It allows a third-party to issue subscriptions on behalf of the subscribers. This feature provides the chance to achieve the minimized complexity within the operators and the maximized flexibility of the operators.

Context availability represents the outline of context data. Usually context availability changes less frequently than context data over time. For example, the following availability information is used to register context entities: context type, attribute list, and domain metadata (e.g., provider information). The FogFlow programming model benefits from these separated interfaces because of two reasons. First, FogFlow can automatically manage context availability information on behalf of tasks so that we reduce the complexity of operator implementation for developers. Second, context availability and context data updates are forwarded to task instances via separate channels; therefore, we do not have to feed the unchanged context availability information to the tasks repeatedly. This can significantly reduce the bandwidth consumption of cross-task communication.

In FogFlow, whenever a service topology is triggered, a large number of geo-distributed task instances are created, configured, and instantiated on the fly in a very short time. This introduces two challenges to the context management system: 1) it must be fast enough to discover available resources in a specified scope; 2) it must provide high throughput to forward context data from one task to another. In addition, we assume that data processing tasks can only be instantiated from a service topology within a single FogFlow-enabled smart city IoT platform. However, they should also be able to share and reuse context data from other smart city IoT platforms as long as these platforms are compatible with NGSI. In terms of terminology, each smart city IoT platform is represented by a domain and the FogFlow framework can be duplicated to realize other smart city platforms for different domains. Different domains can be different cities or business domains such as transportation and e-health in the same city.

Currently, the Orion Context Broker developed by Telefonica [64] is the most popular message broker supporting NGSI; however, it is not scalable due to the lack of distributed solutions and federation support. To achieve a scalable context management system, we apply the following two approaches.

Scaling light-weight IoT Broker up with shared IoT Discovery: As illustrated in Fig. 2.11, within each smart city platform domain a large number of IoT Brokers work together in parallel with a shared IoT Discovery. The centralized IoT Discovery provides a global view of context availability of context data and provides NGSI9 interfaces for registration, discovery, and subscription of context availability. Each IoT Broker manages a portion of the context data and registers data to the shared IoT Discovery. However, all IoT Brokers can equally provide any requested context entity via NGSI10 since they can find out which IoT Broker provides the entity through the shared IoT Discovery and then fetch the entity from that remote IoT Broker.

Connecting different domains via Federated Broker: In each domain there is one Federated Broker responsible for announcing what the current domain provides and fetching any context data from the other domains via NGSI10. Within the domain, Federated Broker informs IoT Discovery that it can provide any context data out of the current domain. Federated Broker needs to coordinate with the other Federated Brokers in different domains. The coordination can be done using different approaches, such as table-based, tree-based, or mesh-based. In the table-based approach, all Federated Brokers can know which Federated Broker is responsible for which domain via a shared and consistent table that is maintained and updated by a bootstrap service. In the tree-based (hierarchical) approach, a hierarchical relationship between different domains is configured manually or maintained automatically by a root node. In the mesh-based approach, each Federated Broker maintains a routing table based on its partial and local view and relies on a next hop from the routing table to locate its targeted domain. In practice, which approach to take is up to the actual scale of domains. Due to a limited number of domains in our current setup, FogFlow takes the table-based approach and looks up the Federated Broker for a target domain directly from the shared table.

Dynamic Service Orchestration

Once developers submit a specified service topology and the implemented operator docker images, the service data processing logic can be triggered on demand by a high level processing requirement. The processing requirement is sent (as NGSI10 update) to the submitted service topology entity. It is issued either by the system operator via Task Designer or by a subscriber via an external application. The following three inputs are necessary for TM to carry out service orchestration.

- *Expected Output* represents the output stream type expected by external subscribers. Based on this input parameter, TM decides which part of service topology should be triggered. This allows FogFlow to launch only part of the data processing logic defined in the service topology.
- *Scope* is a defined geoscope for the area where input streams should be selected. This allows FogFlow to carry out the selected data processing logic for the selected area such as a specific city or a polygon area.

- *Scheduler* decides which type of scheduling method should be chosen by TM for task assignment. Different task assignment methods lead to different service level agreements (SLAs) because they aim for different optimization objectives. For instance, we provide two methods in FogFlow: one for optimizing the latency of producing output results and the other for optimizing the internal data traffic across tasks and workers.

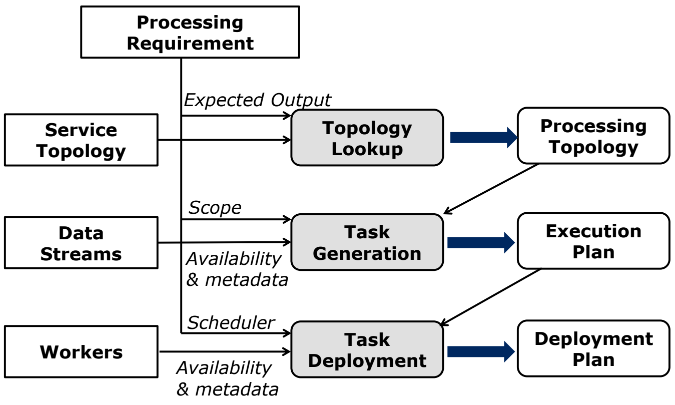


Figure 2.15: Major steps of service orchestration.

For a given processing requirement, TM performs the following steps (illustrated in Fig. 2.15) to dynamically orchestrate tasks over cloud and edges.

- **Topology Lookup:** Iterating over the requested service topology to find out the processing tree in order to produce the expected output. This extracted processing tree represents the requested processing topology which is further used for task generation.

- **Task Generation:** First querying IoT Discovery to discover all available input streams and then deriving an execution plan based on this discovery and the declarative hints in the service topology. The execution plan includes all generated tasks that are properly configured with right input and output streams and also the parameters for the workers to instantiate the tasks.

- **Task Deployment:** Performing the specified scheduling method to assign the generated tasks to geo-distributed workers according to their available computation capabilities. The derived assignment result represents the deployment plan. To carry out the deployment plan, TM sends each task to the task's assigned worker and then monitors the status of the task. Each worker receives its assigned tasks and then instantiates them in docker containers. Meanwhile, worker communicates with the nearby IoT Broker to assist the launched task instances for establishing their input and output streams.

Since the focus of this chapter is on the NGSI-based programming model and its supporting system framework, we skip the algorithms for task generation and task assignment. More details can be found in our previous GeeLytics platform [65] study.

Virtual Sensor

In FogFlow we dynamically composite multiple data processing tasks to form the data processing flow of each IoT service based on the standardized NGSI data model and service interface. However, to interact with sensors and actuators, we still need to deal with the diversity of various IoT devices. For example, some devices might not be able to talk with the FogFlow system via NGSI due to their limited upload bandwidth; some existing devices might only support other protocols such as MQTT or COAP; or some devices need to be turned into a sleep mode from time to time in order to save their battery lifetime. To handle these issues, we introduce *virtual device* to unify the communication between FogFlow and IoT devices.

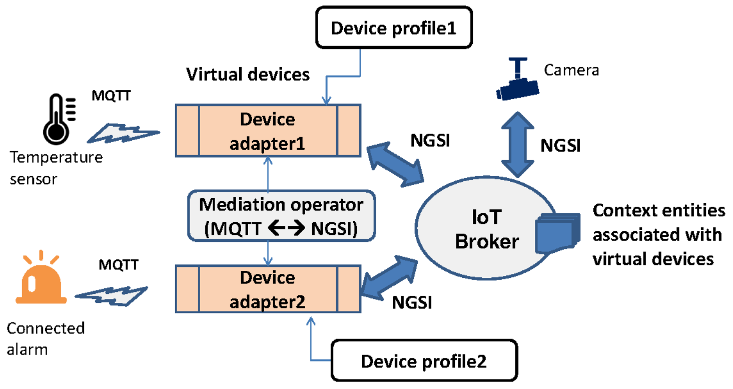


Figure 2.16: An example to illustrate the concept of virtual devices.

Figure 2.16 illustrates this concept. Any physical device that already supports NGSI can be integrated to the FogFlow system directly by interacting with a nearby IoT Broker. On the other hand, a physical device that does not support NGSI must be integrated into the FogFlow system as a virtual device via a *device adapter*. As shown in Figure 2.16, the device adapter is a proxy that is triggered and initialized from a device profile in order to mediate the communication between the device and the FogFlow system. The proxy is a task instance instantiated from a specific *mediation operator*, which handles the detailed mediation procedure for different types of devices. For conversion between different interfaces, different mediation operators must be developed.

By adding a device profile, we can trigger an adapter task to integrate a non-NGSI device into the FogFlow system. The *device profile* provides which operator should be used and the necessary configuration information to interact with the device. In the end, the physical device is presented as a NGSI context entity via its device adapter. All FogFlow services just need to interact with the NGSI context entity associated with the physical device, such as sending context updates or subscribing to some attribute changes. Using this virtual device approach, we handle the availability and relability issues of IoT devices within the device adapters.

2.2.4 Use Case Validation

In this section, we discuss our implementation of an example application which realizes the first use case (described in Sec. 2.2.1): anomaly detection of energy consumption in retail stores. The service topology (illustrated in Fig. 2.17) is defined to meet the requirements of the use case. Two data processing operators are defined as follows:

- **Anomaly Detector:** This operator is to detect anomaly events based on the collected data from power panels in a retail store. It has two types of inputs: (1) **detection rules** which are provided and updated by the operator; (2) **sensor data** from power panel. The detection rules input stream type is associated with "broadcast", meaning that the rules are needed by all task instances of this operator. The granularity of this operator is based on "shopID", meaning that a dedicated task instance will be created and configured for each shop.
- **Counter:** This operator is to count the total number of anomaly events for all shops in each city. Therefore, its task granularity is by "city". Its input stream type is the output stream type of the previous operator (Anomaly Detector).

There are two types of result consumers: (1) a dashboard service in the cloud, which subscribes to the final aggregation results generated by the counter operator for the global scope; (2) the alarm in each shop, which subscribes to the anomaly events generated by the Anomaly Detector task on the local edge node in the retail store.

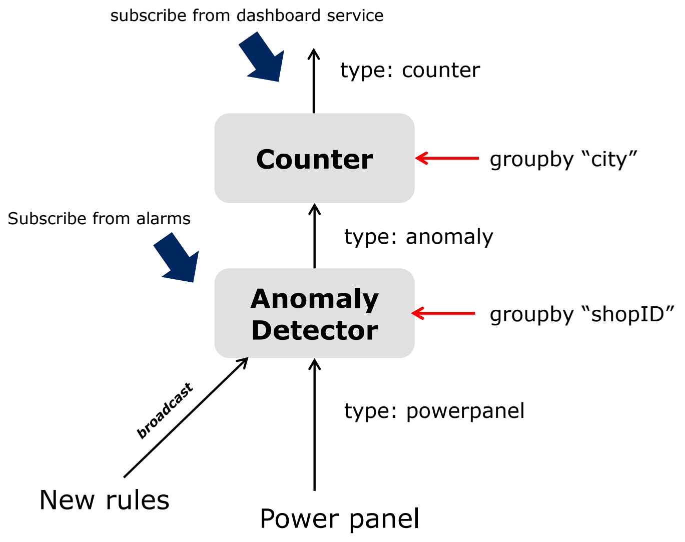


Figure 2.17: Illustration of how intermediate results are shared at various levels of granularity across application topologies.

The second and third use cases can be realized in a similar way by defining a processing topology based on their specific requirements.

2.2.5 Performance Evaluation

This section includes our experimental evaluation of NGSI-based context management systems of the FogFlow framework. Our analyses include the efficiency of context availability discoveries and context transfers in the smart city scale. Moreover, we analyze the scalability of FogFlow using multiple IoT Brokers. Our metrics are throughput (number of messages per second) and response time/message propagation latency. The results show the performance of the IoT Brokers (our FogFlow-Broker and Orion-Broker) as well as the IoT Discoveries (our FogFlow-Discovery and Orion-Discovery).

The query for discovery and update requests are generated using Apache JMeter performance testing tool. We analyze 3 types of queries for context discovery: ID-based, topic-based, and geoscope-based. In ID-based queries, a match (discovery) occurs when the queried entity ID is already registered as available. In topic-based (pattern-based) queries, a match happens when there is a registered entity ID of the similar pattern defined by a regex (e.g., searching pattern "Room.*" for the registered entity ID "Room12"). Geoscope-based queries match when the entity is registered with the location that is inside the defined geographical area (defined by latitude/longitude and radius values). IoT Brokers mainly forward updates from the context producers to the context consumers. We analyze the throughput and propagation latency of updates under various cases.

Considering the shared discovery in the FogFlow architecture, we conduct lab experiments for IoT Discovery using single server instance, which has 12 CPUs, 128GB memory, and 256GB disk storage. For IoT Brokers, we conduct experiments on AWS cloud using multiple server instances for context updates. Since IoT Brokers in FogFlow can be widely deployed on edge nodes such as IoT Gateways, we use only micro instances in our tests, where each micro instance has 1 CPU and 1GB memory. We consider various number of threads (clients) in a smart city (1, 10, 100, 200 entities) accessing the context management system at the same time. The threads may represent devices such as sensor nodes in a smart city or applications accessing the system.

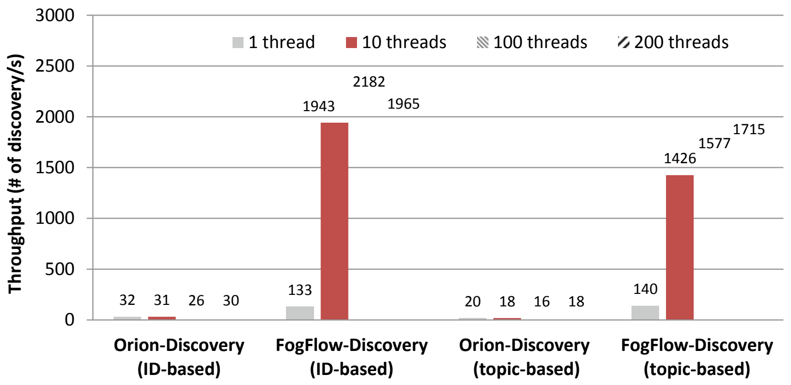


Figure 2.18: FogFlow-Discovery and Orion-Discovery throughputs for a matched ID- and topic-based query among 10000 entities.

Let us first start with the ID- and topic-based query performance for discovery.

Figure 2.18 shows the average throughput of query for FogFlow-Discovery and Orion-Discovery. The query always returns 1 match out of 10000 registered entities. For 1 client, Orion-Discovery has throughput of less than 50 discoveries per second, while FogFlow-Discovery serves more than 100 discoveries. With the increased number of threads, throughput of FogFlow-Discovery significantly increases to more than 1900 in ID-based and more than 1400 for topic-based queries. On the other hand, when the number of threads increase, Orion-Discovery has even worse performance, showing that using Orion-Discovery can cause a bottleneck in certain scenarios where multiple IoT Brokers query at the same time.

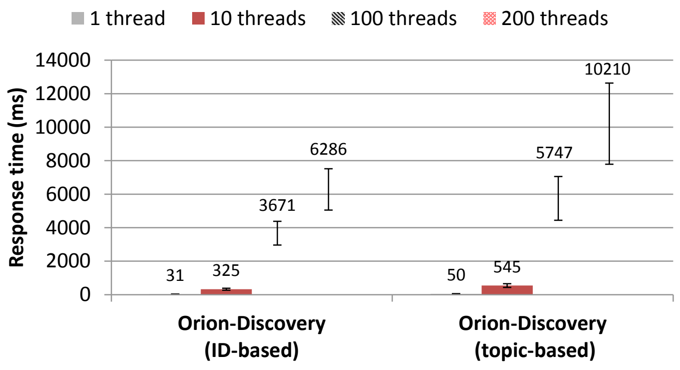


Figure 2.19: Orion-Discovery response times for ID- and topic-based queries.

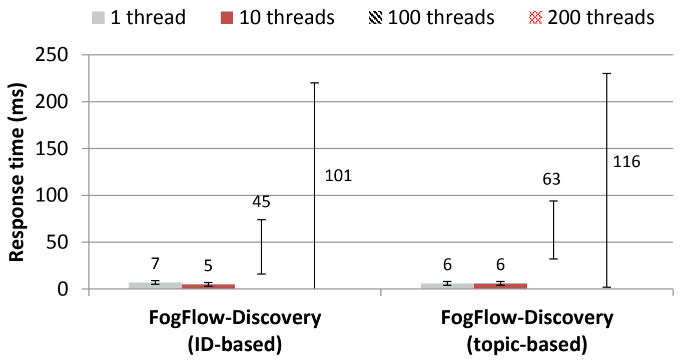


Figure 2.20: FogFlow-Discovery response times for ID- and topic-based queries.

The average response times of Orion-Discovery are shown Fig. 2.19. We observe that Orion-Discovery returns fast responses in the case of 1 thread and 10 threads. However, the response times dramatically increase in the case of 100 or 200 threads. In the case of 200 threads in topic-based queries, the average response time is more than 10s. On the other hand, as can be seen in Fig. 2.20, the response times of FogFlow-Discovery are shorter in all cases. Moreover, for more than 100 threads, FogFlow-Discovery still provides high performance with an average response time around 100ms per query. Overall, considering a smart city with multiple IoT Brokers querying for ID- and topic-based discoveries, we find FogFlow-Discovery clearly a more reliable component to handle such loads. While Orion-Discovery provides the same functionalities, its performance is not sufficient for such scenarios.

Let us now discuss the geoscope-based query performances of the discovery components. We compare the response times for different numbers of matched entities in Fig. 2.20 among 10000 registered entities using 200 threads. For 0-match case where IoT Brokers query for an entity which is located outside of any registered areas, Orion-Discovery has significantly better performance compared to FogFlow-Discovery. Same performance difference exists for 1-match case. On the other hand, this performance gap diminishes with the increased number of matches where the volume of transferred data increases. FogFlow-Discovery produces slightly higher throughput for more than 100 matches. Furthermore, both components provide a reliable service for queries up to 1000 matches. Figure 2.22 shows the response times of the geoscope-based queries. Both discovery components achieve short response times in most cases. Only exception is seen in the case of 1000 matches, which produces a certain load in the network. In that case, Orion-Discovery performs slightly better (≈800ms) than FogFlow-Discovery (≈1000ms).

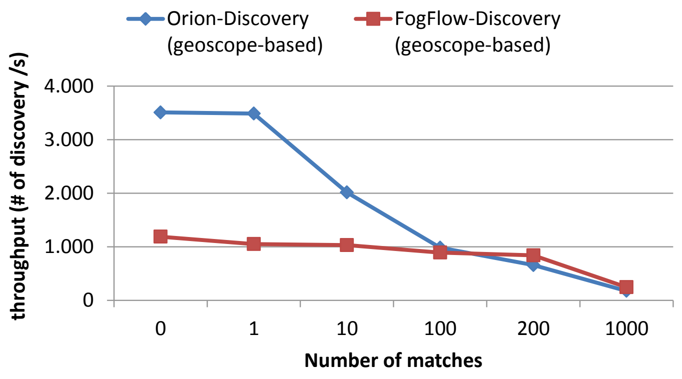


Figure 2.21: FogFlow-Discovery and Orion-Discovery throughputs in geoscope-based queries.

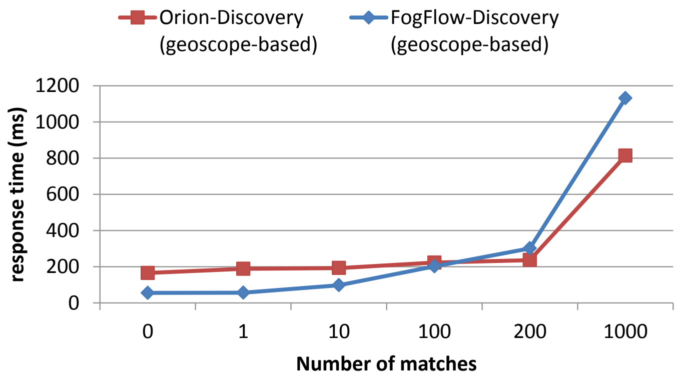


Figure 2.22: FogFlow-Discovery and Orion-Discovery response times in geoscope-based queries.

Orion-Discovery is a built-in feature of Orion Context Broker while in FogFlow-

Discovery is a stand-alone component separated from FogFlow-Brokers. With this design, FogFlow is able to scale up brokers to handle data transfer between different tasks in parallel. We now look at the performance of FogFlow-Brokers. Figure 2.23 shows the throughput of one FogFlow-Broker as the number of subscribers increases. When there is no subscriber, FogFlow-Broker's throughput reaches 6500 updates per second while Orion-Broker can only achieve 2200 updates per second. We observe that FogFlow-Broker performs much better than Orion-Broker in terms of update throughput. This is mainly because FogFlow-Broker keeps the latest updates and all subscriptions in memory while Orion-Broker has to save them into the database (MongoDB). Furthermore, we find that the update throughput decreases as the number of subscribers increases as FogFlow-Broker becomes busy with forwarding received updates to all subscribers.

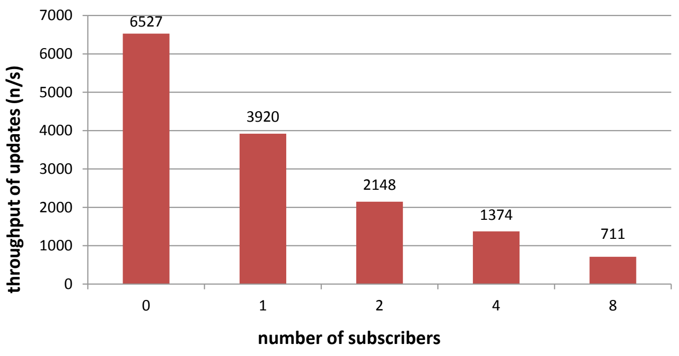


Figure 2.23: FogFlow-Broker and Orion-Broker throughput of updates between publishers and subscribers.

We also test the propagation latency of updates from publishers to subscribers when FogFlow-Brokers are not overloaded. To calculate the latency, we run a program to simulate both the publisher and the subscriber on the same cloud instance. The latency is defined as the time difference between when a update message is sent out by the publisher and when the update is received by the subscriber. Table 2.3 lists the propagation latency of updates in three different situations: 1) both the publisher and the subscriber contact with the same broker, 2) they communicate with two different brokers located at the same data center, 3) they communicate with two different brokers located at two different data centers.

The results shows that the propagation latency via the same broker is very low (less than 1 millisecond on average). On the other hand, if the data flow between publishers and subscribers is established via two different brokers, the propagation latency increases. In this case the latency depends on where these two brokers are located. If they are located at the same data center, the propagation latency can still be less than 50 millisecond on average; however, it becomes unpredictable since we do not know whether our cloud instances are at the same rack. If the two brokers are located at different data centers, the average propagation latency significantly increases ($\approx$500ms). This result indicates that we need to carefully select and configure a proper broker for each running task in order to minimize data propagation latency for any time critical services.

Table 2.3: Propagation latency of update

	Propagation latency	
Different test cases	Avg. (ms)	Std. (σ)
Same broker	0.7	0.7
Different broker, same data center	<50	<50
Different broker, different data center	430.8	194.2

We do further experiments for the scalability of FogFlow-Brokers when they work on different topics in parallel, but still share the same discovery component. As can be seen in Fig. 2.24, the aggregated throughput of updates increases linearly with the increased number of brokers. Note that the number of brokers increase two times for each result. This result shows that FogFlow-Brokers can scale up very well without overloading the shared discovery component. This is due to the fact that the coordination with FogFlow-Discovery is only needed for subscriptions and initial updates to decide which stream should be provided to which subscribers. After that, the workload triggered by frequent value updates can be easily handled by brokers in parallel. Hence, by separating broker and discovery components, FogFlow is able to achieve scalability of forwarding context data between publishers and subscribers.

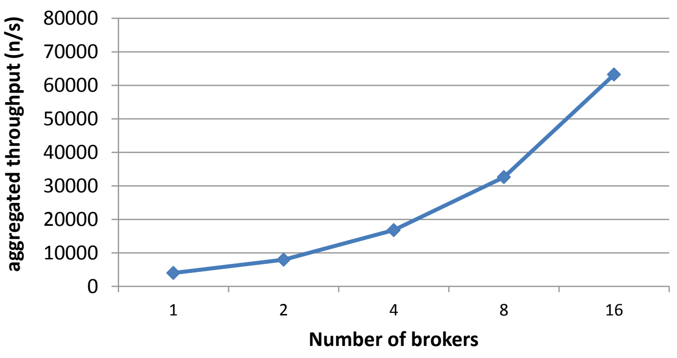


Figure 2.24: Aggregated throughput of updates for different number of parallel brokers.

2.2.6 Related Work

There are many studies related to the IoT smart city platforms. However, most of the efforts focus only on the cloud environment. For example, as an open software platform, FIWARE is helping service providers to quickly and cost-effectively build their cloud-based applications and services by providing various open-source generic enablers (GEs). Nevertheless, none of the GEs offered by FIWARE enable flexible fog computing. In the FIWARE community, Orion Context Broker has been extensively used to enable the interoperability between different GEs, whereas Orion provides centralized context management. This is not a scalable solution considering large scale scenarios; in particular, when we consider exchanging real-time context information at edges or across various domains.

Most of the existing programming models focus on supporting batch and real-time data processing efficiently in a cluster or cloud environment. For instance, MapReduce has become the de facto standard for batch data processing in Apache Hadoop framework. Apache Spark is a distributed batch processing framework, while it also supports stream processing based on micro-batching. Other frameworks involve Apache Storm which supports event-based stream processing and Apache Flink which enables both batch and stream processing with its unified APIs. Recently, due to the requirement of having a unified programming model for both batch processing and stream processing, the generic dataflow programming model is mostly preferred over MapReduce to support cloud-based data processing in many new frameworks such as Apache Beam, MillWheel, and Google Cloud Dataflow. All of these frameworks are only tailored to the cloud environment and they are unsuitable for fog computing due to their limited considerations on the heterogeneity, geo-distribution, openness, and interoperability requirements of future fog computing infrastructures.

In 2015 Cisco formed the OpenFog Consortium [66] together with partners from industry and academia, trying to accelerate the adoption of open fog computing in various domains. OpenFog Consortium emphasize the importance of openness and interoperability of fog computing infrastructure in their blueprint architecture document, while they do not provide any concrete proposal to achieve these two goals. The existing studies on fog computing mainly focus on optimization of resource and task allocations. For instance, Foglets [57] and MCEP [67] support live task migrations with their own APIs in a cloud-edge environment for location-aware applications. Mobilefog [68] provides a programming model for fog computing applications based on its own APIs. FogHorn is a commercial edge computing infrastructure with the focus on complex event processing at edge devices. Different from those existing fog computing frameworks, FogFlow is not designed to invent a completely new programming model for fog computing with private APIs. FogFlow, on the other hand, extends cloud-based dataflow programming model with standard-based APIs and make it suitable for the cloud-edge environment. In this way, our programming model can be quickly adopted by cloud service providers to build their fog computing services without much learning effort.

2.3 Research Directions

In this chapter we have analyzed a novel system architecture in charge of efficiently creating and efficiently adjusting self-contained and isolated network slices in massive IoT scenarios building on IoT Brokers features. With the proposed solution, network operators and IoT tenants can interact to trade-off resources among slices to avoid service degradation. The novelty relies on a new system architecture where 5G Network Slice Brokers and IoT Brokers are interconnected. The results show that a joint IoT Broker-Network Slice Broker Orchestration might provide benefits to all parties. By means of simulations, we have shown that the *IoT Slice-aware Traffic Optimizer* algorithm can efficiently drive the system towards fully-utilization states while fulfilling the required SLAs.

Further in this chapter, we propose the FogFlow framework which provides a standards-based programming model for IoT services for smart cities that run over cloud and edges. The FogFlow framework enables easy programming of elastic IoT services and it supports standard interfaces for contextual data transfers across services. We showcase three use cases and implement an example application for smart cities. Furthermore, we analyze the performance of context management using NGSI interfaces to see feasibility of the standard-based approach in the smart city scale.

The preliminary study presented in this chapter gives us directions to open challenges that we will address throughout the remainder of this work. With the developments of supporting technologies on 5G and Fog/Edge computing and the speedily connection of IoT devices, it is necessary to frame a visionary scenario of an hyperconnected Internet-of-Things to be expected in the next 10 years. This will give the scoping context for several technical research problems such as:

- standardization and open source,
- federation of decentralized (fog and cloud) IoT analytics platform,
- data usage control across federation of IoT analytics platforms,
- creation of an IoT data analytics services ecosystem

Chapter 3

Standardization and Open Source

The Internet of Things (IoT) ecosystem is getting its momentum in urban environments where several devices, originally deployed for a specific purpose and belonging to different technologies, are now linked together to harmonize heterogeneous and holistic scenarios. Early deployments have proven their value, but novel use cases such as automotive, public safety, e-health are being considered in the context of Smart Cities. Such use cases demand for new and stringent requirements that cannot be supported by current solutions both in terms of latency and computing power. In order to meet these requirements in a cost-efficient manner the *Multi-access Edge Computing* (MEC) paradigm is considered here as currently being defined by the ETSI MEC Industry Specification Group. In this chapter, we propose an ETSI MEC-compliant architectural solution that allows for seamlessly integrating existing and future IoT platforms. In addition, an *IoT gateway middleware* is presented as a novel component that enables running low-latency and computationally intensive applications on generalized MEC-based systems.

Further, this chapter introduces the capabilities of the open source *FIWARE* framework, based on the NGSI standard, that is transitioning from a research to a commercial level. We base our exposition on the analysis of three real-world use cases (global IoT market, analytics in smart cities, and IoT augmented autonomous driving) and their requirements that are addressed with the usage of FIWARE. We highlight the lessons learnt during the design, implementation and deployment phases for each of the use cases and their critical issues. We also perform a functional evaluation of the framework compared with both open source and commercial platforms. We demonstrate that the framework is going to become commercially ready but still maintaining its openness to innovation, by, for instance, introducing two aspects to be addressed in the FIWARE agenda: semantics and privacy.

3.1 Evolving Multi-Access Edge Computing (MEC) to Support Enhanced IoT Deployments

The penetration of *Internet of Things* (IoT) deployments is advancing at an increasing pace as the technology matures and the cost of the required equipment benefits from economies of scale. Early deployments focused on basic monitoring and sensing applications, e.g., in smart cities. Based on the proven value of these solutions more advanced use cases are being considered, e.g., in the automotive, public safety, and e-health fields (see Fig. 3.1) with more stringent requirements in terms of latency, computing power and coupling to the network infrastructure.

Nowadays, the raw data produced by distributed networks of sensors is usually centrally processed and analyzed in data centers to derive added value information and, eventually, trigger the corresponding actions. In these deployments, the so-called *IoT Gateway* is a key entity acting as a mediator between field sensors and cloud data centers. IoT gateways act as a bridge for narrow-band communication protocols of (typically) energy-constrained networks, e.g., Bluetooth and ZigBee, and broadband (wireless and wired) systems, e.g., optical fibers or LTE, used as transport channel toward cloud facilities. But, they offer very limited processing capabilities for cost reasons given the large number of gateways required in real deployments.

In order to meet both the latency and/or computing power requirements of the upcoming advanced IoT-based use cases as well as to leverage the upgrades in mobile networking, in this chapter we consider *Multi-access Edge Computing* (MEC) as the technology able to boost IoT to more sophisticated deployments. MEC is envisioned as a key technology to transition to the fifth generation (5G) mobile networks. The European Telecommunications Standards Institute (ETSI) has chartered the MEC Industry Specification Group[1] (ISG) in order to define a multi-vendor edge environment toward which IT and Telco stakeholders can converge.

MEC allows to dynamically install the applications of IoT services on top of cloud facilities at the edge of the network, thus with low communication latency. This way, IoT gateway functions like data pre-processing and *things* management can be lifted to the MEC platform, allowing to install cheaper hardware with simpler functionalities. Moreover, MEC resources can be efficiently shared among different IoT networks (providing isolation guarantees) leading to unexplored business opportunities based on the novel concept of *Network Slicing* [2].

In the light of the above considerations, this chapter brings the following contributions:

- Detailed review of a state-of-the-art solution for a smart city deployment, indicating its limitations and technical challenges when addressing advanced use cases.
- A *taxonomy of future IoT use cases* for next generation networks along with the corresponding derived requirements.
- A proposal for an *ETSI-compliant MEC architectural solution* to facilitate the integration of IoT networks and service deployments.
- An *IoT gateway middleware* enabling high-computational applications with low-latency requirements running on generalized MEC-based systems.

[1] http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing (Accessed the 15th of December 2020)

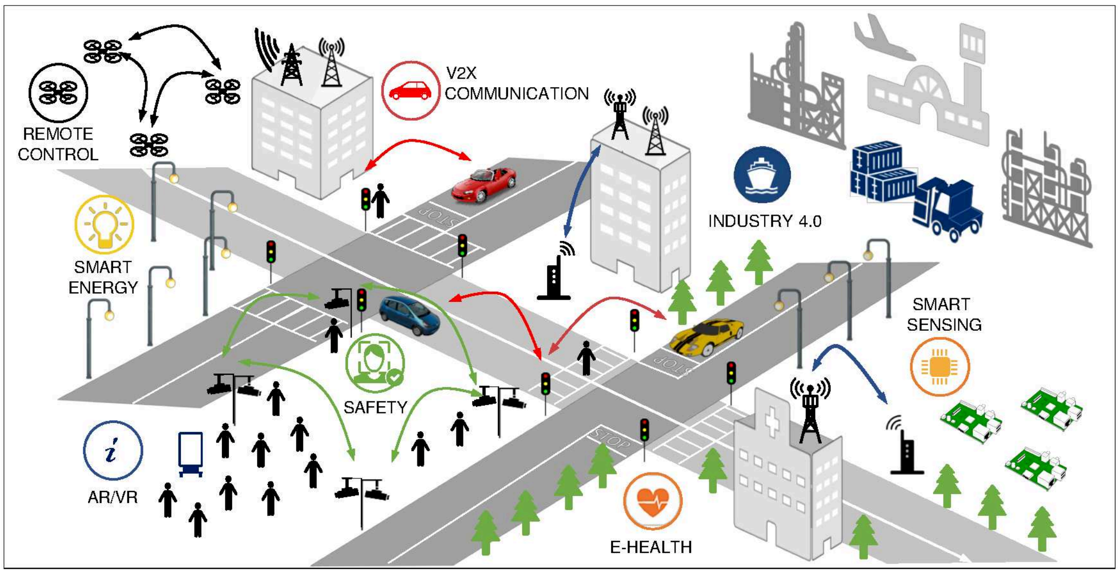


Figure 3.1: IoT ecosystem in Smart City scenario.

3.1.1 Smart cities as the key use case for large scale IoT deployments

Smart cities represent the first step toward a multi-domain and inter-connected IoT world [69], aiming at the digital transformation and interconnection of urban processes, like vehicular traffic control, utilities supply, health care, manufacturing, entertainment, etc.

Sensed data from IoT networks are used to detect safety threats or to measure and guide human behaviors for different purposes (e.g., pollution reduction, energy savings, public safety in crowded events). Such data are transmitted by means of several technologies and might need to be quickly delivered (low-latency) or fully processed (high computational power) based on different use cases. In the following we examine the existing smart city case-study in the context of the European project SmartSantander[2], showing deployed IoT use case features, current limitations and potential enhanced use cases.

The SmartSantander case-study

The deployment comprises 20000 static and mobile sensors installed within 35 square kilometers area in the city of Santander, Spain. The solution scope covers different purposes, such as environmental monitoring (for e.g., temperature and humidity checks for irrigation control of public garden) or vehicular monitoring (for e.g., traffic intensity and public parking management), as summarized in Table 3.1. Recently to further

[2]Information and public deliverables at `http://www.smartsantander.eu` (Accessed on the 15th of December 2020)

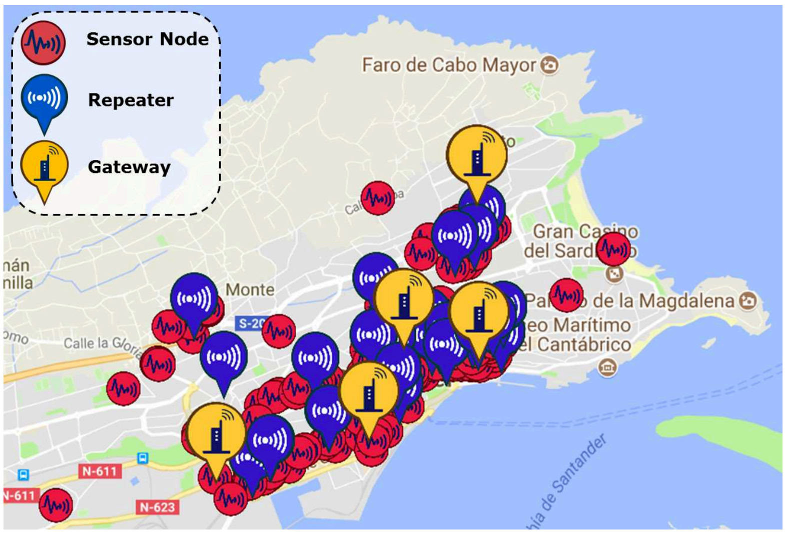


Figure 3.2: Location of the IoT elements in the SmartSantander project.

deliver augmented reality services, 2500 RFID tags have been spread among the tourist attractions of the city.

The IoT network is built as a 3-tier architecture, composed of *i*) wireless sensors, *ii*) repeaters and *iii*) IoT gateways, as depicted in Fig. 3.2. The first two tiers communicate with each other via 802.15.4 interfaces whereas the third tier forwards the data coming from the low-rate and limited-power interfaces to the computational servers located within the city premises, by means of Wi-Fi, GPRS/UMTS or wired interfaces.

An interesting approach to create a big data analytic platform able to gather all data generated by the sensors of the SmartSantander deployment is represented by CiDAP [55], which requires computational features in cloud premises to expose collected information to external applications. However, the same authors point out that most of the sensor data are processed and analyzed in more than 60 seconds, making real-time applications (e.g., dynamic route calculation for ambulance based on real-time traffic conditions) unsuitable for this environment. Therefore, such applications are allowed to retrieve data directly from the IoT gateways requiring intelligence (and additional power) on such devices. Nonetheless, this approach does not satisfy low-latency requirements, mainly due to the absence of cross-layer optimization between transport and processing facilities typical of today's legacy solutions.

SmartSantander is an attempt, together with many others in Europe and in the rest of the world, to grow smart city capabilities and enable digital transformation. Nevertheless, smart cities are continuously evolving, aiming at incorporating future-looking use cases as those showcased in the next paragraphs.

Table 3.1: SmartSantander deployment summary.

Node Type		**Amount of Nodes**	**Sensors**	**Radio I/F**
Gateway	General Purpose	26	N/A	IEEE 802.15.4, IEEE 802.11, Digimesh, GPRS/UMTS
	Irrigation	3	N/A	
	Traffic	2	N/A	
Repeater	Temperature	74	Temperature, Acceleration	IEEE 802.15.4, Digimesh
	Light	553	Light, Temperature, Acceleration	
	Noise	58	Noise, Acceleration	
	Gases	13	Temperature, CO, Acceleration	
	Traffic	9	N/A	IEEE 802.15.4
	Weather	3	Temperature, Relative Humidity, Soil Moisture, Solar Radiation, Rainfall, Windspeed, Atmospheric Pressure, Acceleration	
	Irrigation	23	Pluviometer and Anemometer sensing temperature, relative humidity, soil moisture, soil temperature,	
	Water Flow	2	Water Flow, Acceleration	
	Agriculture	19	Temperature, Relative Humidity, Acceleration	
Parking Sensors and Tags		723	Ferromagnetic sensors buried under the asphalt for occupancy and authorization	Proprietary
Traffic Sensor		59	Road Occupancy, Vehicle Counting, Vehicle Speed Monitoring	IEEE 802.15.4
Mobile Node	Bus	95	CO, Particles, NO$_2$, Ozone, Temperature, Relative Humidity, Speed, Odometer, Location	IEEE 802.15.4, IEEE 802.11, GPRS
	Car	80		GPRS
Augmented Reality Tag		2500	Presence (+ metadata)	NFC
Participatory Sensing Smartphone		6500	Multiple	IEEE 802.11, GPRS/UMTS
Augmented Reality Smartphone		~14000	Presence (+ metadata)	
Total:		**31 Gateways** **1516 Fixed Nodes** **175 Mobile Nodes** **2500 Tags**	**3029 Fixed Sensors** **1750+ Mobile Sensors** **20000+ Smartphone Sensors**	

A glimpse on future evolution of smart city use cases

The impelling need of high processing capabilities on the edge premises is supported by a number of advanced IoT use cases, which are envisioned to underpin future smart cities [70]. In particular, we explore *i*) autonomous (and remote) driving and advanced traffic monitoring, *ii*) public safety and assistance of large crowds, *iii*) industrial automation scenarios, shedding the light on feasibility aspects and future requirements.

Augmented context awareness for autonomous driving and road safety A huge number of sensor networks are already deployed along pedestrian and vehicular roads, including monitoring cameras, traffic and visibility sensors. These systems are usually single-purpose platforms deployed in different time periods and belonging to few verticals. Due to different business orientations and vendor-specific features, every platform represents an independent environment that hardly combines with the others to build a single and homogeneous solution. Unfortunately, this is a recurrent obstacle in the smart city market because to address novel and advanced use-cases measurements from sensors must be treated seamlessly together with information coming from other platforms in the same context, thus demanding for additional levels of interoperability that cannot be reached by current solutions. Two applications of this data melting-pot are augmenting the context-awareness of autonomous driving systems and enhancing emergency assistance for manned driven vehicles.

A real deployment of such applications would require external sensors with a holistic view of the environment to achieve state awareness and perform traffic balancing and routing optimization, a joint combination of local and remote video analysis for identifying unexpected obstacles (e.g., pedestrians or animals crossing the road), sensor fusion algorithms for inferring a myriad of diverse situations in a complex scenario like the urban environment, together with a reliable alerting system able to reach the driver even in unfavorable conditions.

Another use-case, with even more challenging requirements, is the introduction of automated driving solutions on highways. Once again, this kind of solution must be able to collect and analyze data coming from heterogeneous sources, starting from global traffic video streams and ending with the sensor co-located with the vehicles.

An example of piloting those kind of scenarios in the real world is the European project AUTOPILOT[3] that aims to enhance the safety of automated driving with the means of surrounding smart objects.

Obtaining real-time information about the overall context state is crucial to handle automated vehicles moving along with human driven vehicles and other vulnerable road users such as cyclists and pedestrians. Automated vehicles are requested to identify and possibly predict complex situations and quickly react without any human input. This would require high computation resources but, at the same time, below 10ms communication latency [71] in order to timely react to the detection of the issue and deliver the corresponding alarm. Matching both requirements in nowadays systems is tough given the limited computational power of programmable gateways that necessarily delegate heavy processing tasks to remote data centers, further increasing the experienced traffic delay.

Public safety in multi-domain smart cities Typical applications of IoT in smart cities are related to public safety. Piloting projects, such as MONICA[4], aim to launch a series of security applications during big public events. An example is the usage of digital signage (e.g., advertisement display or projectors) to steer crowds in case of emergency situations like fires or flash floods. Computing expensive sensors fusion algorithms would be used to identify the exact location of the danger and infer its future development. At the same time, estimating the crowd distribution and its mobility behavior is crucial in order to derive an optimal rescue strategy considering both location information gathered from personal devices and global monitoring systems. Finally, the computation output needs to promptly arrive at the distributed network of actuators in order to orchestrate the displays and steer the crowd to safe places.

Another application is the support to security services during big events to handle potential threats. Also in this case, the computation and latency demands are unlikely to be met when the raw data traffic is characterized by very localized and unpredictable traffic peaks, like in mission critical situations or occasional public events.

Many public entities support the *open data* concept, which consists of exposing sensor data to anyone who wants to leverage them for smart applications as agreed by more

[3]`http://autopilot-project.eu/` (Accessed on the 19th of May 2019).

[4]http://www.monica-project.eu/ (Accessed on the 15th of December 2020)

than hundred cities worldwide in the Open&Agile Smart Cities initiative (OASC).[5] While this would help to raise a positive IoT ecosystem, the information collected from sensors networks could be very sensitive and can not be shared openly because it might endanger citizens' willingness to use smart city services [72] or even break government regulations[6]. On the one side, the setup of IoT networks involves complex bureaucracy which forces specific data typologies to be stored in trusted physical locations (e.g., locally in the municipality premises), rather than on remote uncontrolled data centers. On the other side, a fully distributed approach based on powerful devices deployed in the urban context locally running applications opens new safety issues. For example, in facial recognition applications, software and complementary datasets, e.g., target databases, are often protected by secret. Running this kind of applications directly on top of cameras which might be directly exposed to vandalism attacks may not be a good solution.

System integration for large scale industrial automation Industrial IoT (IIoT) and the related term "Industry 4.0" refer to the automation of modern factories operating a large number of connected smart resources like robots and sensors. The focus of IIoT is on the exchange and real-time control of mission critical information: in fields like energy, oil and gas, health care, reliability and accuracy are not optional. Machines must monitor physical processes and react often through decentralised and autonomous decisions.

Factory operators have already started to use analytics and machine-learning algorithms to predict consumption of raw materials and optimize their process control and supply chains in real time. In order to achieve proactive maintenance, many industrial settings seek technologies that enable real-time monitoring, anomaly detection and alerts, failure prediction, and predictive servicing of critical equipment. In an IIoT system a delayed response, or even a network interruption, causes data loss leading to unsynchronised, un-optimised processes and loss of money. With a high volume of sensitive data to be processed and analysed in real-time, tactical decisions must be made locally where they matter and data security and reliability emerge as keystones of a robust solution.

While a self-contained IIoT may be easily designed and tailored for a single, isolated factory, meeting all its specific requirements, the same cannot be said in larger industrial environments where multiple IIoT systems with different requirements must be interconnected and orchestrated. A harbour is an example of such a heterogeneous ecosystem:

- multiple tasks must be performed and synchronized over a geographically wide area: ship mooring, containers management, control tower operations, freight transport logistic, etc.
- each task needs specific sensors and requirements, which are often conflicting: precision and reliability for detecting and moving containers, bandwidth and compu-

[5]OASC background, http://oascities.org/wp-content/uploads/2016/02/Open-and-Agile-Smart-Cities-Background-Document-3rd-Wave.pdf (Accessed on the 15th of December 2020)

[6]EC, "Data Protection in EU", https://ec.europa.eu/info/law/law-topic/data-protection_en, (Accessed on the 15th of December 2020)

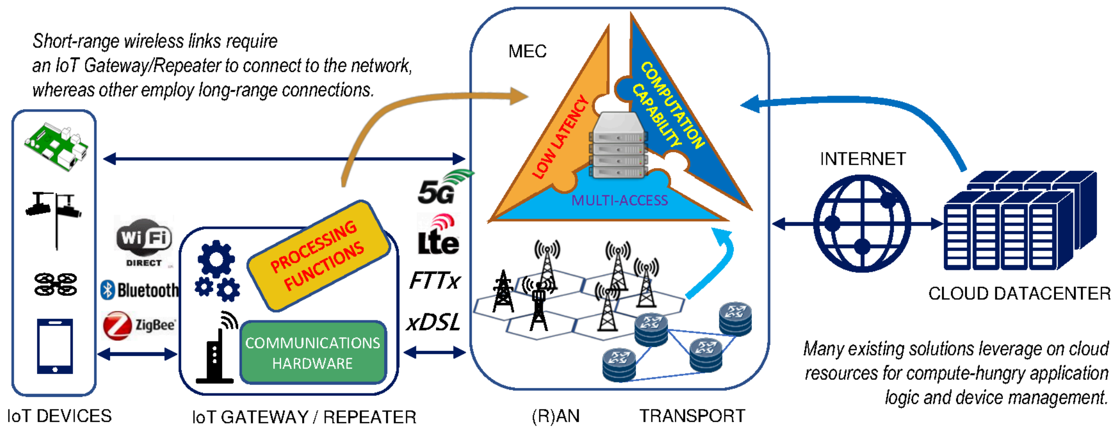

Short-range wireless links require
an IoT Gateway/Repeater to connect to the network,
whereas other employ long-range connections.
MEC
LOW LATENCY
COMPUTATION CAPABILITY
MULTI-ACCESS
INTERNET
CLOUD DATACENTER
Wi Fi DIRECT
Bluetooth
ZigBee
PROCESSING FUNCTIONS
COMMUNICATIONS HARDWARE
5G
lte
FTTx
xDSL
Many existing solutions leverage on cloud
resources for compute-hungry application
logic and device management.
IoT DEVICES
IoT GATEWAY / REPEATER
(R)AN
TRANSPORT

Figure 3.3: A MEC-based IoT solution: Building blocks overview.

tation to process content from video surveillance cameras, real-time monitoring [71] of sea and ship conditions, etc.

- a single connectivity technology cannot guarantee coverage or performance for all tasks, therefore different technologies (e.g. Bluetooth, WiFi, LoRa, LTE) must be in place.

Most big and medium-sized ports (e.g. Amsterdam, Valencia) are running "Smart Port" initiatives like the H2020 project Inter-IoT[7] facing similar problems: data and assets belong to different owners (shipowners, terminals, port authorities, etc.) with diverse technical/business requirements and collecting technologies; resulting information must be securely shared and processed in real time in order to improve overall mission critical operations. A Smart Port architecture relies on an IIoT platform to manage data from sensors and devices, and distribute that data to other stakeholders' components. Such platform cannot cover all sensors and business needs in a single element, due to complexity and cost, but as a set of systems integrated through a central common system.

3.1.2 The advantages of deploying IoT at the edge

As discussed in previous sections, future smart cities are meant to exhibit evolved features that leverage three technology pillars: *i*) low latency communications, *ii*) high computing capabilities and virtualization, and *iii*) heterogeneous access technologies and devices.

Because of such aspects, the network's edge is widely recognized as a favorable location to deploy computing capabilities, and the industry and scientific community are already working on different solutions [73]. Low latency and high computing capabilities are necessary for real-time automated control systems wherein the event-decision-enforcement-feedback loop needs to be executed in a short, constrained time budget.

[7]http://www.inter-iot-project.eu/ (Accessed on the 15th of December 2020)

In these regards, *fog computing* enables data processing and application logic to run at fog nodes scattered in the network, including end devices, edge and cloud resources. Many solution providers have gathered in the OpenFog Consortium [8] to define an open architecture for fog computing [74]. OpenFog is not meant to be a standard developing organization, but still its contribution is relevant to understand use cases and to foster their implementation. The airport scenario in [74] is an interesting example, which, similarly to what previously introduced about seaports, shows how interconnected heterogeneous systems represent an increasing trend in the IIoT community.

Cloud computing is essential to overcome the power and form factor limitation of IoT devices, by offloading data analysis and processing tasks to remote application servers. By virtualizing such application servers, it is possible to flexibly deploy them over different platforms and even to relocate them if necessary. Major cloud computing providers have expanded their offer with a custom IoT solution, as for instance in the case of Azure IoT Suite [9] and AWS IoT [10]. Similarly to remote clouds, edge computing platforms lend themselves to run a variety of IoT applications, but avoid the tromboning effects when transferring data meant to be generated and consumed locally. This aspect is tackled, among others, by Cloudlet Applications [11], AWS GreenGrass [12] and Azure IoT Edge[13].

The technologies above belong to a broad set of proprietary solutions that tend to focus on application level and service hosting rather than on the communication technologies underneath. In fact, the IoT ecosystem nowadays comprises a plethora of communication technologies, spanning from short range wireless like Bluetooth, Zigbee, WiFi to Low Power Wide Area Network (LPWAN) links using NB-IoT, LTE-MTC, EC-GSM-IoT and others. Some of them are based on industrial standards by IEEE and/or 3GPP whereas others are proprietary solutions, e.g., Sigfox and LoRa.

Nevertheless, in many situations it is paramount to leverage different deployments, either to aggregate data from distinct domains, or to merge legacy setups with newer ones into a single logical service. Therefore, it is necessary to abstract from the access technology underneath. The authors of [75] suggest employing Software Define Networking (SDN) to route data packets between the IoT device and the most appropriate fog node running the data filtering algorithms and the application's logic. Bringing IoT software components to the edge is the conceptual basis of EdgeX Foundry [14], which is an open source project purposed to implement an open framework for IoT edge computing. Although targeting a broader scope than IoT, a similar endeavor is attempted by the Open Edge Computing initiative [15] and by the Edge Computing Working Group within the Telecom Infra Project [16].

[8]https://www.openfogconsortium.org/ (Accessed on the 13th of July 2018)
[9]https://azure.microsoft.com/en-us/suites/iot-suite/ (Accessed on the 15th of December 2020)
[10]https://aws.amazon.com/iot/ (Accessed on the 15th of December 2020)
[11]https://cloudlets.akamai.com/ (Accessed on the 15th of December 2020)
[12]https://aws.amazon.com/greengrass/ (Accessed on the 15th of December 2020)
[13]https://azure.microsoft.com/it-it/campaigns/iot-edge/ (Accessed on the 15th of December 2020)
[14]https://www.edgexfoundry.org/ (Accessed on the 15th of December 2020)
[15]http://openedgecomputing.org/index.html (Accessed on the 15th of December 2020)
[16]http://telecominfraproject.com/project/access-projects/edge-computing/ (Accessed on the 15th of December 2020)

Because of its position in the industry, the ETSI MEC Industry Specifications Group is regarded as a key entity to produce enabling technologies for network operators to evolve their IoT service offering [76]. With the recent expansion to cover heterogeneous access, ETSI MEC not only provides means to support the above-mentioned technology pillars and depicted in Fig. 3.3, but also outlines an open and standardized path for telco operators, vendors and IT players. For this reason, in the next we propose an IoT platform for ETSI MEC, which aims to consolidate different IoT technologies into a single body able to expose a homogeneous IoT service to customers in a multi-tenant fashion (highlighted in the central block of Fig. 3.3) regardless the presence of different IoT and access technologies, by means of softwarized IoT gateways deployed within the MEC premises.

3.1.3 ETSI MEC Enhancements to support multi-domain IoT deployments

In order to make edge clouds a standardized computing environment, ETSI has recently re-chartered the ISG formerly known as Mobile Edge Computing: the scope of the new Multi-access Edge Computing ISG is enlarged to embrace a variety of access technologies beyond cellular. The most relevant outcome of ETSI MEC is the definition of a framework and reference architecture [77], as well as a number of specifications for application enablement and API design [78].

However, despite IoT being deemed a pivotal use case for MEC, the ETSI solution still lacks the due level of details when it comes to the components that are supposed to support IoT use cases. We hence propose a new architectural element for extending MEC capabilities to support IoT deployments, namely the MEC IoT platform described in the next section.

The MEC IoT platform

The MEC IoT platform is a software artifact meant to create a substrate (or *middleware*) where multiple (virtualized) IoT gateway instances, from different access link types and implementations, can run. This can be achieved in two steps. First, IoT gateways are split into lower (hardware) and upper (software) layers and the latter is migrated to the MEC facilities, hosted as software instances within the MEC IoT platform. In Section 3.1.2, we have already observed few (commercial) implementations of such softwarized IoT gateway approach. The second step would be to implement the middleware functionalities of the MEC IoT platform as an environment where the IoT gateway instances are interconnected, by means of an appropriate messaging service, a gateway instance registration and discovery mechanism, and a semantic extraction and protocol translation function. The purpose is to unify into a single logical entity the operations that are typically executed by IoT Gateways, such as identification and management of IoT devices and their secure communication. By hiding the complexity of the underneath IoT networks, the MEC IoT platform can expose a homogeneous method to control a variety of IoT devices grouped into a single logical set, comprising diverse deployments.

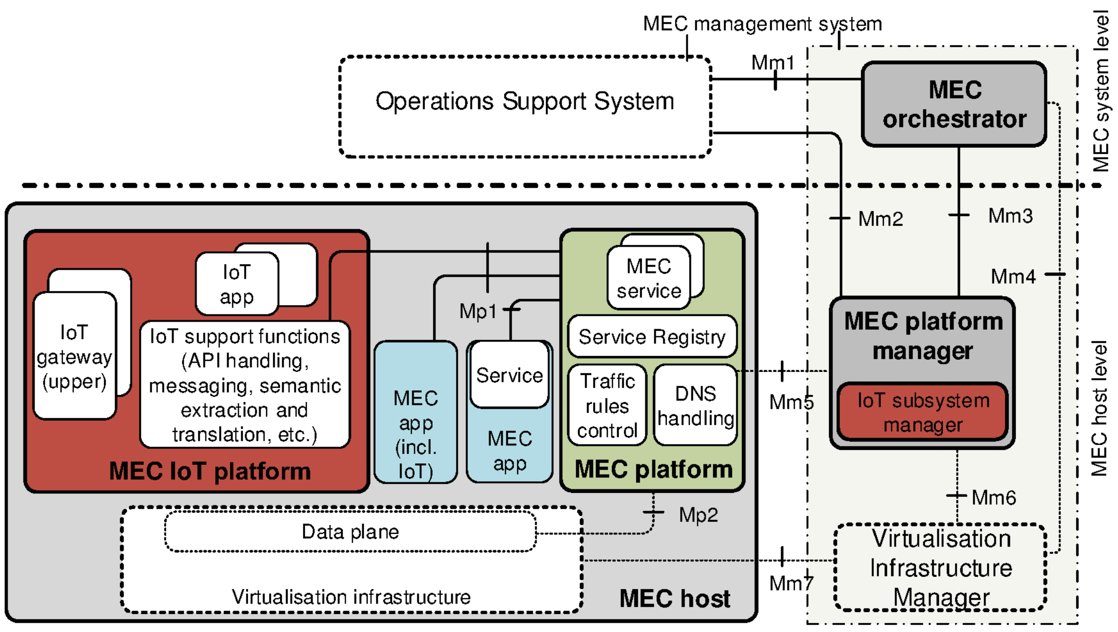


Figure 3.4: The MEC IoT platform within the ETSI MEC architecture.

From the perspective of the ETSI MEC architecture, despite its internal complex logic, the MEC IoT platform would appear as a service provider MEC application, installed in a MEC host and enabled by the MEC platform through Mp1 interface [79]. In other words, MEC applications can discover the MEC IoT platform by querying the MEC platform's service registry, and interact through a defined IoT API exposed by the MEC IoT platform. This abstraction and the associated API would enable MEC applications to interact with deployed IoT devices with little or no knowledge of the actual deployment, thus hiding the complexity of the IoT network from end applications.

The ultimate task of the MEC IoT platform is to enable multiple and different IoT services by exposing the capabilities of a set of IoT devices like sensors and actuators to either built-in or MEC applications. Therefore, the MEC IoT platform needs to be populated with the appropriate IoT gateway instances and the set of associated devices. In addition, it should be configured with the traffic filters and policies to allow shared usage of the framework. An IoT subsystem manager is devised to perform these jobs. This logical entity may be integrated in the same MEC platform manager, which is already providing element management functions, as depicted in Figure 3.4. Such a deployment is desirable for instance to enable the MEC management system to orchestrate an IoT service when distributed across multiple MEC hosts. However the management interfaces defined by ETSI MEC, namely Mm1, Mm2 and Mm3, do not support IoT-specific functionalities and should be extended for such purpose.

An API for IoT in Edge Computing

The MEC IoT platform can decouple the set of IoT devices from the application logic that leverages the capabilities of such devices (sensors and/or actuators). This enables to partition (or share) the IoT resources thereby allowing a simultaneous usage following the IoT-as-a-service paradigm. Therefore, the multiple services running within the MEC premises might utilize a higher level of data abstraction together with a common API and data format representation. Running services also need to interact with the deployed nodes and sensors for performing IoT tasks, such as device management or actuation. In order to realize a seamless interaction among service-to-service and service-to-device, it is desirable to create a single interface, i.e., the IoT API for MEC. In addition, this API being open and standardized would lead to a broad and well supported ecosystem. All these characteristics might be fulfilled by the Open Mobile Alliance Next Generation Service Interface for Context Management (OMA NGSI [58]) and the FIWARE foundation, which adopted it and have developed components for several aspects in the field of IoT, cloud and edge analytics, and privacy and security. In addition, the work of the NGSI standard is continuing within the ETSI ISG Context Information Management (CIM) in the direction of semantics and linked data in order to achieve high interoperability with the many data models present in the IoT world.

Prototyping edge computing in a real scenario

Real-time detection of vulnerable road users (VRUs) for steering autonomous driving decisions has been already implemented in the context of AUTOPILOT project[17]. Devices installed on vehicles collect Wi-Fi probe packets captured on the road and transmit them to a Road Site Unit (RSU) with sensitive information (such as MAC addresses) obfuscated and data tagged with GPS coordinates. The RSU collects all these data from several cars and push to a cloud platform that computes an estimation about the crowd on the roads as well as people mobility patterns. The analysis outcome is fed back to the cars that adapt the speed accordingly. The IoT communications is based on NGSI for dispatching VRUs analysis results to the RSUs, and on oneM2M for the communication to the vehicle. Applying the proposed prototype on an ETSI MEC-enabled infrastructure would blend together the RSUs and the cloud platform at the edge and allow to perform VRUs analysis closer to vehicles with advantages in latency and bandwidth consumption. This also fosters more advanced analytic processes, for example based on computer vision.

3.1.4 Value proposition of the MEC-based IoT Platform

The MEC platform offers a complete IT environment close to the IoT deployments. The advantage is two-fold: the former is that IoT service providers might easily migrate their own IoT applications to more powerful (and close to the edge) IT environments without

[17]We refer the reader to the project AUTOPILOT deliverable "D1.3-Initial IoT Self-organizing Platform for Self-driving Vehicles."

Table 3.2: Qualitative analysis against commercial solutions.

	Orchestration	**Communication**	**Application interoperability**	**Open Specification**	**Multi-tenancy**	**QoS at the edge**
MEC-based IoT Platform	Standard	MQTT+JSON, HTTP, LoRa, MQTT+UL2.0, AMQP+UL2.0, LWM2M	OMA NGSI + Smart data models	✓	✓	Network Slicing
Microsoft Azure IoT Edge	Proprietary	MQTT, AMQP, HTTP, Modbus, OPC-UA, BLE, BACnet	Topic-based + AMQP Type System (only data interop.)	✗	✗	Limited MQTT message reliability
AWS Greengrass	Proprietary	FreeRTOS, MQTT, OPC-UA	Topic-based	✗	✗	✗
IBM Watson IoT Platform Edge	Proprietary	MQTT, HTTP	Topic-based	✗	✗	Limited MQTT message reliability
GE Predix Machine	Proprietary	MQTT, AMQP, TCP+OPC UA, Websocket+OPC UA, Modbus	Topic-based + PDataValue (only attribute interop.)	✗	✗	✗

losing direct control over their private (and reserved) applications data; the latter is that the edge cloud facilities (e.g., MEC hosts) can be directly deployed and managed by the IoT service provider. This last case might be generally envisioned as the business scenario wherein the IoT service provider acts as an IoT infrastructure provider (e.g., building on top of public or private infrastructures, such as airport areas, shopping malls or building blocks) able to open its own premises to other IoT service providers. However, this would require a multi-tenancy-enabled IoT platform where different *platform tenants* are willing to rent a subset of resources of the same shared IoT infrastructure, properly tailored to their own provided services.

Therefore the MEC system brings into play the network slicing concept as the main enabler for a multi-tenancy platform. Heterogeneous IoT application requirements might be handled on the same platform while ensuring service-level-agreement (SLA) guarantees (e.g., public safety deployments for monitoring crowded events), and data isolation property among competing IoT service providers. Additionally, the MEC environment allows the execution of softwarized IoT gateways. This enables to build a seamless and multi-domain IoT platform where different specific-purpose technology sources are interconnected by means of standardized interfaces and can interact by means of *mediation gateways*.

In the following, we compare the proposed MEC solution for IoT against generic commercial IoT platforms, namely AWS Greengrass, Microsoft Azure IoT Edge, and IIoT solutions such as IBM Watson IoT Platform Edge and General Electric Predix Machine, summarizing the main advantages and limitations in Table 3.2. Multiple IoT protocol implementations might be adopted in our scenario. For the sake of concreteness, we account for NGSI as main IoT application protocol, but other implementations, e.g., ETSI oneM2M, can also be considered equivalently or together with NGSI [25].

Our approach presents a full-fledged open orchestration solution for an IoT platform by means of ETSI MEC-compliant interfaces. In particular, the NGSI application protocol can be used for both device-to-device and device-to-gateway communication in combination with different lightweight and energy-efficient protocols, such as LoRa and OMA LWM2M. NGSI can be also combined with open available Smart Data Models [80] (formerly known as FIWARE data models) bringing the remarkable advantage to support native application interoperability for a variety of IoT scenarios. This eases the deployment of new IoT services escaping integration activities such as interface adaptation (e.g., query definition, data subscription parameters) or data models translation. Such a desirable feature is already deployed in real context, e.g., the harmonization of eight smart cities among Europe within the SynchroniCity EU project framework[18], fostering the co-creation of the services offered to the citizens and the sharing of a global ecosystem.

Differently, commercial platforms provide a limited application interoperability either through proprietary data format (e.g., GE Predix Machine) or AMQP-based data types (e.g., Microsoft Azure IoT Edge), leaving the developers to enforce interface and data models interoperation. Interestingly, the open specification of the MEC-based IoT Platform does not only encompass the application interoperability, but it also fosters the definition of architecture and components' functionalities to further support federation of both MEC instances and IoT services, where the same edge infrastructure can be shared between different tenants. It is worth mentioning that while the multi-tenancy and slicing concepts are specifically addressed by the MEC standardization group, there is a lack of such concepts in existing edge platforms resulting in no support for the Quality-of-Service (QoS) provisioning.

To the best of our knowledge, this is the first, non-proprietary ETSI MEC-compliant solution integrating the IoT facilities into edge computational nodes.

3.2 FIWARE: A Standard-based Open Source IoT Platform

The evolution in all fields of technology is accelerating the shift towards the Internet-of-Things (IoT) vision in any area where data are produced and analyzed.

The market of IoT platforms is comprised of a huge number of solutions. From a business perspective, we can cluster them in two classes of platforms: commercial and open source. The commercial platforms are more stable and more appealing for industrial and business-oriented scenarios due to the contractual support of the providers. The open-source platforms are usually implementing standards and maintained by communities of researchers and innovators, enhancing them with cutting-edge trends, driven by use case and practical requirements. Here we discover the transition of FIWARE[60] from research to commercial.

FIWARE and its evolution The FIWARE framework is supported by the European Commission since almost a decade, through funding several projects either for developing and enhancing it (e.g., FI-WARE, FI-Core, FI-Next, SmartSDK) or for using it in pilots

[18]https://synchronicity-iot.eu

(e.g., City Platform as a Service - CPaaS.io[19], frontierCities, Fiware4Water). Lately, the FIWARE framework is going through a transition phase towards business readiness. This is demonstrated by the European large scale pilots projects (each with 15-20 Million Euro of funds), such as SynchroniCity[20], Autopilot[21], Internet of Food and Farm - IoF2020, and Activage focusing on smart cities, autonomous driving, agriculture & food and e-health, respectively.

3.2.1 Public governance and growth: global IoT market

Public governance is attentive to new technologies that have a strong potential for economic growth, public safety, and citizens' well-being. IoT is certainly promising all these benefits. This has triggered in the past decade a widespread adoption of a plethora of IoT solutions in many urban scenarios. However, closed commercial APIs and implementations hamper an open market development (vendor lock-in). For this reason, an open approach is preferred by public institutions, as commonly agreed, for instance, by the 140+ cities of the Open & Agile Smart Cities (OASC) network[22]. Horizontal harmonization is provided locally in cities such as Milan[23] and Helsinki[24] achieving good results but remaining regionally isolated, thus resulting in city lock-in. Both vendor lock-in and city lock-in discourage small and medium-sized enterprises (SMEs).

With this rationale, SynchroniCity project proposes to synchronize existing smart city solutions by overcoming their misalignments. More specifically, the synchronization targets enabling five Minimal Interoperability Mechanisms (MIMs):

1. Context Management
2. Data Models
3. Ecosystem Transaction Management ("Marketplace")
4. Security
5. Storage

In 2019 the first three MIMs have been officially adopted by the OASC consortium [81].

The SynchroniCity project demonstrates the feasibility of such an approach by harmonizing eight cities (Antwerp, Carouge, Eindhoven, Helsinki, Manchester, Milan, Porto, and Santander). The final goal is the co-design of IoT services [8] and IoT applications, and the establishment of a shared market fostering economic growth.

[19]`https://cpaas.bfh.ch/`
[20]`https://synchronicity-iot.eu/`
[21]`https://autopilot-project.eu/`
[22]`http://oascities.org/`
[23]`http://www.milanosmartcity.org/joomla/`
[24]`https://www.helsinkismart.fi/`

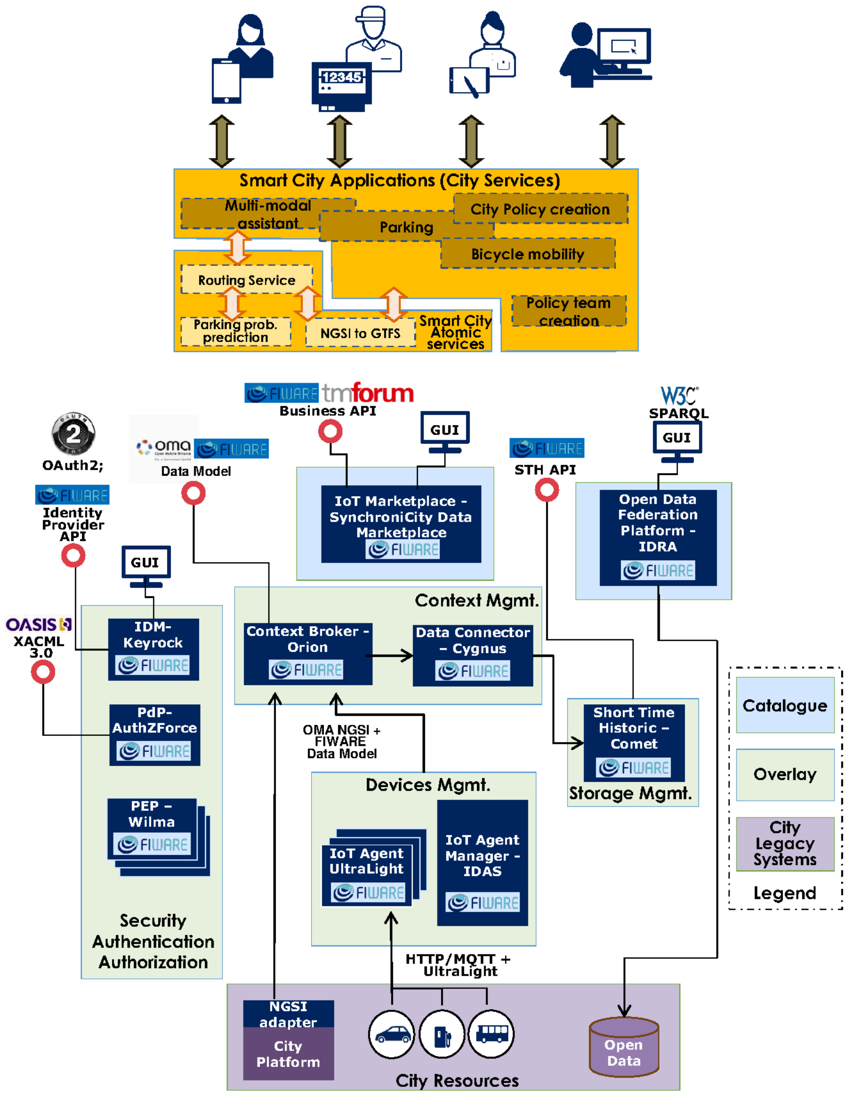


Figure 3.5: Federation of pilot sites through the FIWARE IoT platform.

Legacy IoT platforms are not neglected

In today's smart cities it is common to have platforms that are the outcome of past projects and pilots of city governments. Often these solutions are ad-hoc and only used for the local smart city ecosystem and market. City governments are not keen on throwing away the obtained achievements, but they are interested in expanding the IoT business. Thus, instead of re-designing systems and replacing the already operational ones, in SynchroniCity a generic overlay platform over existing infrastructures is realized. This overlay finds in the FIWARE framework many building blocks to implement the

Table 3.3: Device interfaces supported by FIWARE

IoT Devices	HTTP, MQTT, AMQP
Low-power wireless sensors	LoRaWAN, SigFox
Smart Industry	OPC-UA, ROS2

five foreseen MIMs (see Fig. 3.5). Starting from the available city resources (the purple box in Fig. 3.5) data is integrated and exposed in several manners: sensor streams, open data, and historical time series.

Harmonizing smart cities data

The sensor streams are integrated with different approaches depending on the requirements and constraints given by the different city platforms. For most of the involved cities, ad-hoc integration modules are implemented and deployed. These obtain data from the running platforms and translate them to the Next Generation Service Interface (NGSI) [58] standard data format. Only in a few cases, such as Santander and Porto, sensor data are exposed through typical IoT interfaces, such as MQTT and HTTP. For native IoT device interfaces, the FIWARE framework offers IoT Agents (see Table 3.3) that translate device interfaces to NGSI. Having all data formatted using the same data structure is not enough to ensure harmonization since data might be modeled very differently by IoT developers. For instance, a location might be referred to as ”position”, ”geolocation” or a term in another language. Therefore, the used adapters, both ad-hoc modules and IoT Agents, generate context data following the Smart Data Models [80] (formerly known as FIWARE data models[27]). These data models are defined by the FIWARE and TMForum community and adopted by OASC, and they harmonize the description of data for several application areas such as parks and gardens, points of interest, parking, waste management, transportation, and weather.

The homogeneously modeled data is then handled by the context management layer that exposes context with a standard interface, which is NGSI, fulfilling the Context Management MIM. NGSI specifies both a context management interface with an HTTP binding and a context data format using JSON. SynchroniCity adopted the Orion Context Broker (CB) as the context manager implementation, which holds the latest attribute values for each entity (or ”thing”) and exposes NGSI query and subscription methods. Orion is conceived to work with high-frequency messages and to respond to queries with minimal latency. For stream-based applications, Orion offers an NGSI subscription interface, notifying with atomic notifications as soon as a matching attribute of a matching entity becomes available. A sensor generating a stream of data with small intervals between observations, such as an accelerometer, might create a flood of notifications in the network. Thus, a “throttling” parameter can be set in the subscription. This regulates data notification streams, instructing Orion to issue two notifications for the same subscription apart in time for at least the throttling period. The drawback is that it might result in data loss in case more than one value for a matching attribute is generated within the throttling period. The missing value is not an option for some applications. For this reason, SynchroniCity adopted the Comet Short Time Historic (STH)[25] component. The Cygnus connector receives the NGSI data stream from Orion and forwards it to a persistent data sink, such as STH Comet (in the SynchroniCity case), but also to other commonly used data storage systems such as MongoDB, Hadoop Distribute File Systems (HDFS) for big data processing, or CKAN for Open Data publication. STH and Cygnus together address the Storage MIM.

IoT Marketplace

Often, in smart cities, there are already many services that produce and use data for their purposes, such as urban facilities, public transportation, tourism operators. Unfortunately, these IoT providers have no interest in spending effort on sharing their data if there is no payback for it. On the other side, companies that might want to create smart city services need data and they are keen on paying a fee for datasets otherwise impossible to get. Here the necessity of having an IoT marketplace arises. Available data needs to be cataloged: a) as valuable assets in case of private companies, b) as open data in case of public institutions.

For the first scenario, a gap is identified, and, therefore, the project worked closely with the FIWARE community to close it. FIWARE officially adopted the TM Forum business API standard which is implemented within the SynchroniCity project. This software component, named SynchroniCity IoT Data Marketplace[26], exposes a catalog of available data (either in Orion or in STH Comet) describing endpoints to access them, license, and price. Users can buy data items and the marketplace sets access rules in the security layer to allow the data exchange. To smooth the usage of the rather complex business API, scripts are available to automatically integrate the FIWARE data management by crawling Orion and publish the found data items in the marketplace.

In the case of open data, smart city governments usually have already published datasets, most of the time on their ad-hoc platforms or simply on their institutional website. That is not handy for a data consumer that is completely unaware of the city governance structure and, therefore, of the different institutional web pages. The situation is even worse when the open data website is only available in the local language. This situation is addressed by the Open Data Federation Platform IDRA[27] that was developed in the context of the FESTIVAL EU project that faced a similar scenario. Available datasets are federated in IDRA either via typical open dataset interfaces (such as CKAN) or simply via the web scraping functionality that crawls the generic website for datasets. All the federated datasets are then exposed by a graphical user interface.

[25]`https://github.com/telefonicaid/fiware-sth-comet`
[26]`https://iot-data-marketplace.com`
[27]`https://opendata.synchronicity-iot.eu/`

Security

Identity management and access control requirements of SynchroniCity are not specific to smart cities and IoT. Therefore, common standards are used, such as OAuth2 protocol for authorization, and Oasis eXtensible Access Control Markup Language (XACML) 3.0 protocol for access control. The FIWARE framework already offers software components for both, namely KeyRock and AuthZForce respectively, that are already well integrated with the other FIWARE components (such as the IoT marketplace).

Lessons learned for the creation of a Smart Cities global market

Forgoing from a local to a global market, SynchroniCity tackles three aspects: the IoT service interface, the data models, and an IoT marketplace where data can be bought and sold.

The first barrier faced by SynchroniCity is that putting aside running legacy systems is not a solution for city governments, even if this enables a business ecosystem and, perhaps, it boosts local IoT economic growth. Thus, an overlay on top of the running IoT infrastructure is designed and deployed in every pilot city. The overlay harmonizes live sensors data and open data. FIWARE offers off-the-shelf solutions in case of data exposed through common IoT interfaces (see Table 3.3), and for open data (i.e., IDRA). In the case of data exposed through proprietary platforms, the burden of creating ad-hoc adapters cannot be avoided.

A technical issue occurred when data streams from sensors with very frequent observations are not optimally handled by the Orion CB component. With the latter, either too many notifications flood the network, or data may be lost. The solution for SynchroniCity is to use a time-series database as a data sink. A different approach to this problem is the usage of the Aeron IoT Broker component, as in the CPaaS.io scenario (see below).

The fact that FIWARE is supported by a community directly helps in making such a framework useful for IoT scenarios since it is driven by practical problems. Indeed, SynchroniCity could leverage the outcome of other projects to solve open data integration. SynchroniCity itself contributes to the IoT marketplace, releasing it as free to use[28], thus, closing a gap in FIWARE.

3.2.2 Data analytics on IoT federation: a smart city scenario

IoT services require IoT data. As seen above, in a smart city data may come from different public providers, such as public transportation or traffic management, and it may be centrally handled by the city governance. But what happens when data comes from private entities, such as a company, or even private citizens? Such private providers like to keep their own IoT infrastructure and their data, and not give data away to a centralized unknown platform. Providers are willing to share their datasets [32], if they are licensed and protected by access control, or even to earn money considering data

[28]https://github.com/capossele/SynchroniCityDataMarketplace

as valuable assets. Such a scenario of fragmented IoT platforms is a nightmare for IoT service providers since they need to look for IoT providers and make a great effort on integrating heterogeneous data sources. Furthermore, an IoT service is typically composed of multiple data analytics routines, each exploiting a different set of IoT data and depending on each other. And what would happen if the IoT service provider wants to port its service in another environment? Simply additional effort on data providers discovery, data integration, and analytics orchestration.

City Platform as a Service

The CPaaS.io [82] EU-Japan project (City Platform as a Service-Integrated and Open) faced these problems in a smart city scenario. To overcome such a situation, CPaaS.io defined the following fundamental requirements to be addressed:

i) allow easy integration of data sources into the platform (e.g., sensors operated by private citizens or established providers),

ii) offer services and federation capabilities among IoT platform instances,

iii) support the deployment in many cities with distinct requirements (flexibility and elasticity),

iv) self-orchestration of data analytics processing routines, each part of the same IoT service, among multiple IoT data providers,

v) provide security mechanisms for the desired privacy and data protection,

vi) demonstrate that the use cases developed in one city can be transferred to other cities.

To test the feasibility and effectiveness of the solution, CPaaS.io targets three smart city use cases spread among Europe (Amsterdam) and Japan (Sapporo, Yokosuka, Tokyo), such as water management, public event management, and public transportation.

The CPaaS.io platform exploits different technology to address the identified requirements, such as IoT platform federation, IoT data access control, security & privacy and data analytics tasks orchestration. Also, in this case, the FIWARE framework provides support on the European side, while the u2 platform [83] is used on the Japanese side. The solution of adopting two heterogeneous platforms validates the viability of a loosely coupled federation in the real world [82].

The European platform is FIWARE-based where FIWARE components are deployed and, in some cases, enhanced to provide the basis for a smart city data infrastructure. Table 3.4 presents a list of the FIWARE framework components in use by the CPaaS.io project and the mapping of each component onto the CPaaS.io functional architecture.

Table 3.4: FIWARE-based components present in the CPaaS.io project

CPaaS.io architecture layer	Component
Security and Privacy	KeyRock PEP-Proxy
Data Analytics Routines Management and Operation	FogFlow
Semantic Data & Integration	IoT Knowledge Server NGSI to RDF Mapper STH Comet
Virtual Entity	IoT Discovery Context Broker
IoT Data and Ingestion	IoT Broker
IoT Resource	IoT Agent: LoRaWAN to NGSI

Urban water management scenario

As a relevant application scenario, Waterproof Amsterdam uses the CPaaS.io platform for water management. In the urban context, periods of drought and sudden heavy flashes of rainfall occur more and more often due to the urbanization trend and global warming. In both weather conditions, smart water management is required to ensure water availability, and to handle high pressure in the sewerage infrastructure to prevent street floods. The solution to this problem is water supply peak shaving. This is done by using smart and integrated rain buffers, such as rain barrels, retention rooftops, and retention buffers. These smart devices are centrally controlled by the IoT water management service that makes use of weather information, sewerage capacity, and environment data to calculate the optimal water filling degree for each buffer.

The Waterproof application scenario necessitates several functional layers:

- *IoT Resources layer* for handling the communication with devices and sensors.
- *IoT Data and Ingestion layer* to persist and index, and to expose data to IoT data consumers.
- *Virtual Entity layer* that holds representations in the virtual world of the observed things.
- *Data Analytics Routines Management and Operation layer* to orchestrate the different data processing tasks for computing the operations for the smart devices.

The overall system architecture of the Waterproof solution is presented in Figure 3.6. Similar to the SynchroniCity case, the IoT resources are handled through IoT Agents, but in this case, using the Long Range Wide Area Network (LoRaWAN) protocol. Even though a LoRaWAN IoT Agent is already available in the FIWARE framework, a new one has been implemented in order to optimize it for the specific Waterproof use case, starting from the FIWARE IoT Agent library[29] for accelerating the software development. For handling data, instead of the Orion Context Broker, as in SynchroniCity, the Aeron IoT Broker is used, which has additional features such as federation readiness both for query and subscription [6], and throttling-based aggregation of notifications. The latter permits to receive data notifications not more often than the throttling but encompassing all sensor observations pushed within a period through aggregation such as to avoid data loss. The aggregation can be appending all the data values in a set, or applying an actual function (e.g., averaging). The drawback is a bit higher latency compared to Orion, which is acceptable in the Waterproof use case as it is a smaller IoT scenario compared to a complete city in the case of SynchroniCity. Afterward, all the data are stored in the STH historical component, which is extended to handle also the historical storage of metadata.

The virtual world representation is kept within the FIWARE IoT Discovery which acts as a registry of the available data and resources with associated metadata. This layer abstracts real-world objects (things) from the observation points (sensors). The IoT Discovery allows us to query and subscribe for resources by requesting the relevant context, such as a street name.

The Waterproof data analytics is broken down into atomic analytics tasks that are linked in a chain or a generic topology. This brings the advantage of flexibility. With such an approach, the plugging of the right data streams to the tasks' inputs is not hardcoded, and, therefore, an external process can automatize it. For this purpose, CPaaS.io uses the FIWARE FogFlow [3] framework. The latter leverages the virtual world representation to discover the needed resources and plugs them to task inputs. The resources can be physical (e.g., a sensor stream), or generated, such as intermediate results of one of the analytics tasks (e.g., geographical data aggregation). With such an approach, porting the service topology into another environment is smooth, since FogFlow works at the higher abstraction level of the virtual world. When porting the Waterproof application, the only configuration needed is the description of the available computing nodes. In this application, analytics tasks are responsible for identifying heavy precipitations and for computing how to set up the water network to avoid damage. The analytics service subscribes to inbound data streams to constantly monitor the state of the water buffer and trigger actions when needed.

FogFlow is conceived to orchestrate tasks also among cloud and edge [3], which is a feature to be used in the future for exploiting the computing resources of the deployed devices.

Lessons learned from smart city services

Federating IoT systems is often not an option [6]. Therefore, CPaaS.io sets federation as a target aiming at integrating private and heterogeneous IoT systems towards a global Internet-of-Things. Luckily, the NGSI protocol has been designed to support federation, and the FIWARE framework already offers implemented solutions with the Aeron IoT Broker and NEConfMan IoT Discovery components [6].

[29]`https://iotagent-node-lib.readthedocs.io`

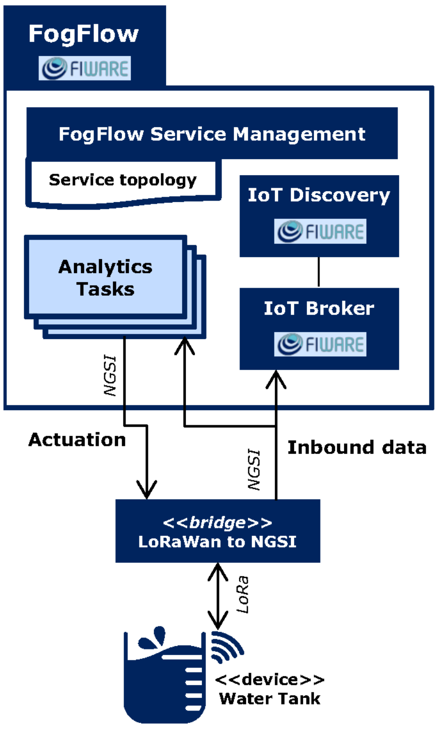


Figure 3.6: Waterproof architecture in the CPaaS.io platform

The water management use case helped to identify other technical gaps not foreseen beforehand. For instance, as the LoRaWAN to NGSI bridge component requires high throughput and low latency, a new lightweight component has been developed by The Things Network[30] to address those performance needs. Another gap was the support of metadata in time series storage that is addressed by extending the STH component. In NGSI, metadata means additional data about attribute values. For instance, in the Waterproof application, the time of measurement and measurement unit are metadata used by the water management system and the metadata need to be historically persisted together with attribute values. The implementation of the new LoRaWAN IoT Agents and the extension of the STH component have been made as part of the CPaaS.io project thanks to the open-source nature of the FIWARE framework. In fact, in the first case, the available library has been used in the implementation and in the second case the source code of the component has been extended.

3.2.3 Research and innovation: an automated driving scenario

In CPaaS.io and SynchroniCity, we have seen requirements for smart cities. But how can we handle extreme scenarios where applications need very low latency, high computation power, ubiquitous computing, and different administrative IoT domains? This

[30]`https://www.thethingsnetwork.org`

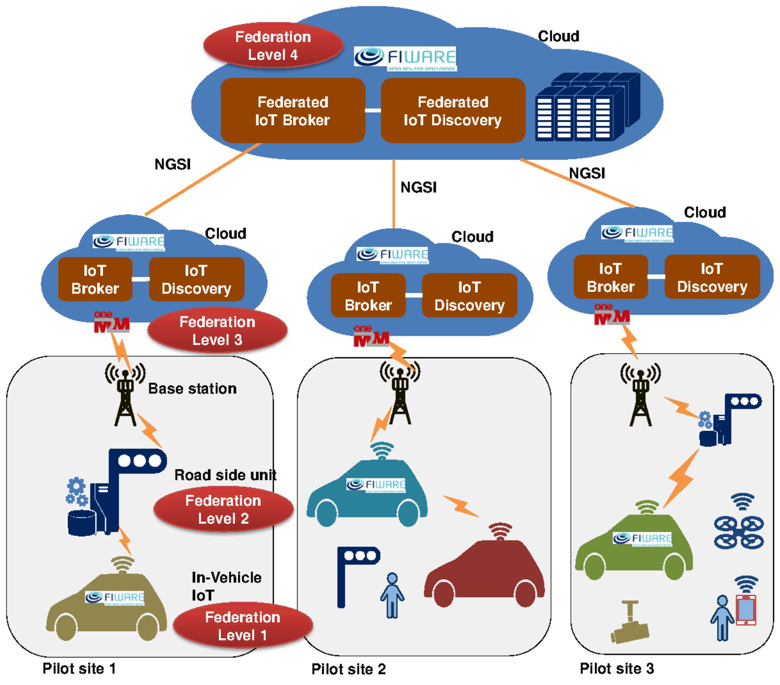


Figure 3.7: Federation of large scale pilot sites through a FIWARE-based IoT platform.

is the boundary situation of the smart mobility domain which is the main focus of the AUTOPILOT project.

AUTOPILOT aims to leverage IoT technologies for enhancing automated driving. The smart mobility use cases include applications of car sharing, car re-balancing, autonomous platooning, and automated valet parking. One key aspect of the future smart mobility scene is providing interoperability between various components which are in-vehicle (e.g., autonomous cars), on-site (e.g., roadside units - RSUs), or in the cloud (e.g., traffic operation center). Hence, the project aims to support the flow of IoT information from the vehicle to the cloud and back to the vehicle passing through on-site units and network infrastructures. Besides, such applications need to be able to work when a car passes from one city to another, and, therefore, it passes road administration boundaries typically handled by different institutions.

Federation of large scale pilots

Together with common data models, the decentralized IoT architecture aims to enable long-distance automated travels, as described in [1]. For instance, when an autonomous vehicle which is registered in Spain travels to the Netherlands, it should be able to operate using information coming from the city through which it is currently driving, while still pushing its key measurements to its home city. Information such as braking distance in different environments, which may indicate the status of its tires or the brake pads, may still be stored in the Spanish pilot site whereas the current location of the vehicle is stored in real-time on servers in the Netherlands. As another example, traffic situation information (e.g., congestion, traffic lights) from different cities are normally served by different vendors that do not federate data with each other. Enabling the sharing of data allows an autonomous car traveling from one country to another to learn the traffic situation in its current country.

For realizing the federation of pilots, the combination of open-source FIWARE Aeron IoT Broker and NEConfMan IoT Discovery[31] components is used to transparently route IoT datasets and data streams from providers to consumers [6]. The transparency is given by the hidden brokering of the requests among IoT actors, where the IoT providers declare the available data and the IoT consumers specify the data of interest. An actor can play both roles, for instance, a car might be a data provider with geographic localization or a consumer of traffic information. The exchange might go from one pilot site to another as illustrated in Fig. 3.7. Federation starts from the vehicle (Level 1) with the in-vehicle IoT platform, where our lightweight version of the IoT Broker, called *ThinBroker* [32], can run on a server in the car. This component can run even in small devices such as a Raspberry Pi.

The autonomous vehicles in the same region are connected (vehicle-to-vehicle communications) or as shown in pilot site 1, cars can connect to a roadside unit. In this case, RSUs are considered as the "edges" with processing and storage capabilities where IoT platform components are deployed as the Level 2 federation. In automotive scenarios, the IoT devices involved are not simply sensors but rather compound devices exposing many different interfaces. For that reason, instead of implementing several brand new IoT Agents, the existing interworking between oneM2M and other IoT technologies is leveraged [25]. As illustrated on the cloud side of Fig. 3.7, each pilot site has a oneM2M implementation for device-to-device communication as well as interworking between several IoT platforms. The cloud-side IoT platform is considered as Level 3 of the federation. This is formed again by IoT Brokers that (besides routing requests to/from oneM2M) also handle data locally which are managed by the pilot site administration. Finally, there exists a federated IoT platform that operates across all pilot sites. Level 3 and 4 of the federation operate with common information models (data models). AUTOPILOT defines a new data model supporting the IoT information for autonomous driving vehicles, considering the AUTOPILOT use cases and the existing legacy models such as SENSORIS[33] and DATEX II[34]. This data model covers IoT information such as vehicles, road infrastructure, road conditions, traffic situations (e.g., congestion), accidents, pedestrians, cyclists, vulnerable road user detection, events, road obstacles, potholes. The data is then mapped into the standard NGSI data format.

[31]`https://github.com/Aeronbroker`

[32]`https://fogflow.readthedocs.io/en/stable/broker.html`

[33]`https://sensor-is.org/`

[34]`https://www.datex2.eu/`

Semantic Interoperability

Interworking between oneM2M and NGSI is achieved through semantic annotations [25]. In AUTOPILOT the interoperability features are provided by the Semantic Mediation Gateway (SMG) technology. SMG offers bidirectional context translations between oneM2M's *Mca* reference point and the NGSI interface. While the syntactic representations in oneM2M and NGSI are different, interoperability is enabled through the agreement on semantics, i.e. the same meaning, mapping homologous underlying concepts, which enables an SMG to do the translation. Figure 3.8 shows the basic setup for AUTOPILOT where the bottom layer consists of autonomous vehicles, RSUs, and pedestrians (e.g., people with smartphones) interacting with the top layer consisting of autonomous driving applications or traffic operation centers (top of Fig. 3.8). The interaction is provided through the FIWARE-based IoT platform. Some devices such as mobile devices of pedestrians connect and push data to the oneM2M platform. In some cases data is directly pushed to the FIWARE platform, or, otherwise, acquired by the SMG from oneM2M. Nevertheless, other third-party platforms or components can receive information directly through oneM2M's Mca interface.

The real-time system, consisting of in-vehicle components, oneM2M and FIWARE IoT platform, is tested at a major pilot site in the Netherlands. For instance, data coming from WiFi scanners of the autonomous vehicle and RSUs are used by analytics modules to estimate crowdedness information in the regions of interest. The contextualized crowdedness information is provided using the standardized data models in NGSI format. This information is then shared with the automated driving application to decide the travel route of the autonomous vehicles. As a result of having an agreement on common semantic concepts, the sharing of information is possible across all levels of the federation.

The experience of IoT-augmented automated driving

The AUTOPILOT project showcases that through a standard-based interworking approach different IoT platforms can run in harmony without causing significant extra costs or overhead. The main benefits of the proposed interworking approach can be summarized as follows.

- Although there are many different and site-specific solutions, automated driving can be successful at the European scale, considering long travels.
- Different pilot sites do not have to adapt their specific devices or software to connect to different IoT platforms.
- Without much effort to adapt, autonomous vehicles can leverage a combination of various IoT services supported by different IoT platforms (e.g., crowd estimation service using FIWARE IoT platform, geofencing service by the Huawei OceanConnect platform).
- The federation at different layers can be achieved considering multiple pilot sites across Europe.

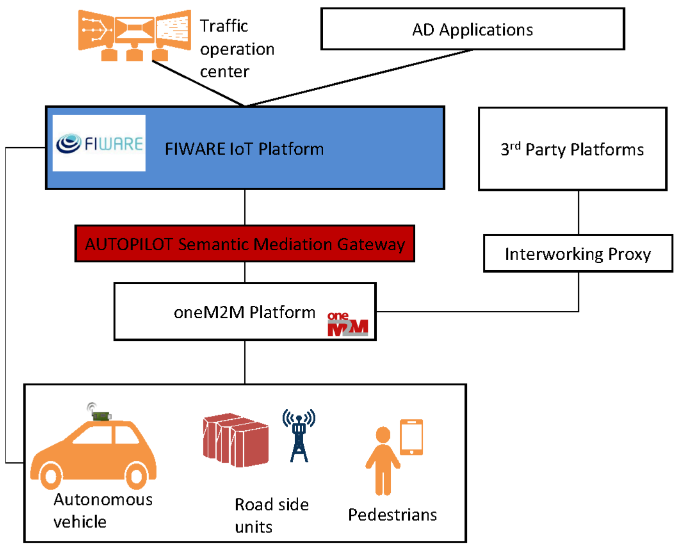


Figure 3.8: Vehicle, RSUs, or pedestrians connecting to the FIWARE platform directly or through the oneM2M platform.

One drawback of the interworking approach is to guarantee the adaptation of heterogeneous data from a vast number of possible services. To prevent service-specific efforts for conversion between different formats, common and standardized information models are a must. Hence, the AUTOPILOT has a specialized activity group having members from different stakeholders such as IoT architecture developers, pilot site leaders and use case leaders to agree on the common data models.

3.2.4 Evaluating FIWARE framework

We perform a functional analysis of different IoT platforms and compare them with an ideal platform fully based on the FIWARE framework. The parameters chosen for this evaluation are strictly related to the requirements identified in the three presented real use-cases. Two analyzed platforms are open source, OpenIoT [87], an outcome of a European research project, and Mobius, a oneM2M implementation, and two commercial and well-known platforms, Microsoft (MS) Azure and Amazon Web Services (AWS) IoT. Table 3.5 summarizes the comparison. We base our evaluation on available published documentation: published papers and project deliverables for OpenIoT, online documentation[35] and oneM2M standard specification for Mobius, and online documentation for MS Azure[36] and AWS[37]. It is worth noting that for the AWS IoT and MS Azure IoT, the features and capabilities are usually at a production level, whereas for FIWARE, OpenIoT, and Mobius those are mainly between research and piloting level.

In terms of sensor connectivity, all five platforms offer a variety of options in addition

[35]http://developers.iotocean.org/ (Accessed 8th of May 2019)
[36]https://docs.microsoft.com/en-us/azure/ (accessed the 3rd of May 2019)
[37]https://docs.aws.amazon.com/iot/ (accessed the 3rd of May 2019)

Table 3.5: Qualitative analysis against other open source or commercial IoT platforms

		FIWARE-based	**OpenIoT**	**OCEAN Mobius**	**MS Azure IoT**	**AWS IoT**
Business Model	Open Source	✓	✓	✓	✗	✗
	Standard-based	✓	✓	✓	✗	✗
	Platform-as-a-Service	✓	✗	✗	✓	✓
Device Support	Protocols	MQTT, HTTP, LoRa, UL2.0, OPC-UA, AMQP, LWM2M, SigFox, ROS2	HTTP, UDP, File System CoAP, Xively, USB Camera Serial Port, JDBC, TinyOS	HTTP, CoAP, MQTT, LWM2M WebSocket, IoTivity (OCF) AllSeenAlliance ALLjoyn	HTTP, MQTT AMQP	HTTP, MQTT, OPC-UA, LoRa, Modbus, FreeRTOS, Serial Port, Raspberry Pi GPIO
	Security	Authentication, Authorization, Remote Attestation [84]	Authentication	Authentication	Authentication, Authorization, Remote Attestation	Authentication, Authorization
	Device management	✓	✓	✓	✓	✓
Data analytics	Cloud	Business Intelligence, Event Processing, Apache Flink	Filtering, Comparing, Aggregation	✗	Spark, Hadoop, Integration, Data Lake, Stream Processing, Machine Learning, Event Processing, Business Intelligence	SQL, Hadoop, Machine Learning, Spark, Data Lake, Business Intelligence, Apache Hive/Pig/Flink, Media processing
	Edge	Event Processing, Media processing, Programmable (containerized) [3]	Aggregation, Event Processing, Data filtering, Semantic Annotation, Programmable (Java)	✗	Machine Learning, Event Processing, Cognitive Service, Programmable (containerized)	ML, Programmable (containerized)
	Edge-Cloud self-orchestration	✓ [3]	Partially	✗	✗	✗
Inter-operability	Linked Data	✓	✓	Partially	✗	✗
	Data Formats	✓	✓	✓	✗	✗
	Cross Platforms	CKAN, DKAN, Socrata API, OneM2M	Xively	FIWARE, OCF IoTivity, AllSeen Alliance AllJoyn	✗	(data replication only) Salesforce IoT
	Federation	✓	✗	✓	✗	(data replication only)
Security and Privacy	Local storage	✓	✗	✓	Device Level (only visualization)	Edge Level
	Third User Access	Authentication, Authorization, Data Access Control, Data Usage Control [85, 7]	Authentication, Authorization Traceability	Authentication Authorization [86]	Authentication, Authorization and Data Access Control	Authentication, Authorization, Resource Access Control
IoT Marketplace	Catalogue of applications, service and data	Resources and data catalog	Partially data catalog (only sensors type)	Resource catalog	Application and services catalog	Application and services catalog
	Accounting	✓	✗	✗	✓	✓

to libraries for extending the available connectors. Security against malicious devices is enforced by all the platforms with identity control (authentication) and functionalities access control (authorization). Remote attestation capabilities are already available in MS Azure and at the research level in FIWARE [84]. A single point for managing devices is supported by the majority of platforms (Mobius has a capability of managing a single device through Lightweight M2M (LWM2M) but a management system has been proposed in literature [88]). As shown in the table, data analytics is well supported by most of the platforms in both cloud and edge. On the other hand, only FIWARE offers a full-fledged framework for automatically orchestrating cloud and edge analytics tasks [3] based on IoT application requirements [17] such as latency, resources availability, and data availability. OpenIoT offers the Apache Storm stream processing envisioning topology of functions to be executed in distributed nodes; the platform orchestrates the functions based on resource availability. The core platform is not part of the Storm stream processing and the configuration of nodes is up to the data providers.

In terms of interoperability, the open-source standard-based platforms are the most advanced as they are designed and developed mostly for research purposes usually in collaboration with standardization institutes. For instance, concepts of linked data and data formats are characteristics of those platforms since they are based on community-developed ontologies. While oneM2M does not have an underlying data model such as NGSI, it is possible to use semantic annotations of resources for semantic operations through SPARQL queries. Amongst all the platforms under analysis, only FIWARE and Mobius are capable of performing true federation which allows complete separation of domains and control but still empowers cross-platform data exchange. The FIWARE federation is by design more advanced, due to the adoption of NGSI, since the possibility of having both peer and hierarchical federation. This allows a minimum amount of shared information between federated domains. Cross-platform interoperability is meant for the capability of bi-directional and transparent API mediation or inclusion. Amongst the platforms under analysis, FIWARE and oneM2M are shown to be interoperable [25]. AWS IoT permits data replication with other AWS IoT instances or other platforms limited to import data from resources and export to data sinks.

An exponential explosion of IoT solutions is often, legitimately, stumbling w.r.t. security and privacy concerns. The first important parameter is the possibility to keep the data locally within the premises of the data owners/producers. FIWARE and Mobius permit that without compromising any functionalities. MS Azure IoT and AWS IoT permit to keep data locally in each edge device, but analysis based on complete data overview can be performed only if data is stored in the cloud. All platforms allow some sort of access control, but only three platforms offer data access control, where FIWARE and MS Azure support a more fine granularity of data access control (such as attribute or metadata) whereas AWS has a granularity of resource level. Only FIWARE perform some steps for enforcing data usage control after the data is supplied to third parties [85, 7] (see also Chapter 5).

In order to build a sustainable IoT ecosystem, means for establishing a marketplace are crucial. All the platforms are exposing catalogs of either IoT applications, IoT services, IoT resources, IoT data or a combination of the above. Nevertheless, FIWARE, MS Azure and AWS IoT have layers for endorsing an accounting system, but only the commercial platforms having marketplaces already alive.

As a conclusion, we can state that FIWARE is the only platform that is, on one hand functionally comparable to the commercial platforms, and on the other able to leverage its nature of open-source community for being ahead in terms of research and innovation[84, 3, 89, 25]. In addition, FIWARE is already available as Platform-as-a-Service for experimenting[38] while it is also ready for a real commercial path[39]. The FIWARE platform is behind compared with AWS and MS Azure in terms of integrated external services (such as machine learning framework, multiple data analytics platforms) and product readiness.

[38]https://cloud.lab.fiware.org/

[39]https://www.nec.com/en/press/201804/global_20180427_03.html

3.2.5 The Road Ahead

FIWARE has a very lively and active community that is continuously expanding. In this section, we include two key areas where FIWARE has the potential to improve.

Semantics: NGSI-LD

NGSI-LD is the evolution of the NGSI Context Interfaces [58] that has originally been standardized by the Open Mobile Alliance (OMA) and further developed in FIWARE. NGSI-LD is standardized as a group specification by the ETSI Industry Specification Group for cross-cutting Context Information Management (ISG CIM). The specification has been published [89] in early 2019 and NGSI-LD is expected to become the new core interface for FIWARE GEs in the course of 2019[40,41,42].

The main new feature is that NGSI-LD is now based on JSON-LD, which enables a semantic grounding. The *LD* stands for *linked data* which practically means that all elements are represented as URIs. Thus, the relevant concepts such as entity types can be explicitly defined in an ontology. Thus, the agreement between different applications and sources is on the level of explicitly specified semantics as provided in an ontology. This enables supporting a level of semantic interoperability on top of an otherwise heterogeneous IoT landscape, which is especially relevant in cross-cutting and large-scale application areas as can be found in smart cities. With the semantic modeling, existing semantic tools for ontology and rule-based reasoning can be used on information retrieved through many NGSI-LD requests.

The new NGSI-LD specification also foresees native support to historical data. It permits applications to directly request complete time series from the data context management together with complex IoT data query capabilities.

Privacy: Data Usage Control

Soon, many advancements are expected in the field of distributed privacy and data usage control for securely sharing local domain data spaces. As presented in the previous sections, these features are a serious blocking point for moving from experimental IoT to a real global IoT. In the latter, sensitive data (e.g. healthcare data, industrial IoT data, crowdsourced data) share the same infrastructure with the less sensitive data. Topics such as continuous control on the usage of the data, also after data have been shared, shall be addressed. The collaboration between FIWARE and International Data Spaces Association (IDSA) is a strategic alliance since the requirements and reference architecture of the latter are the inputs for the evolution of the framework in the hands of the former. The initial results are already available [90], showing the feasibility and the compatibility of the visions of FIWARE and IDSA. This topic is addressed in this thesis in Chapter 5 using a FIWARE-based solution.

[40] `https://github.com/ScorpioBroker/ScorpioBroker`
[41] `https://djane.io`
[42] `https://github.com/FIWARE/context.Orion-LD`

3.3 Conclusions

The IoT market segment represents a huge opportunity for the involved stakeholders as the number of use cases expand both in terms of application areas and complexity. In this chapter, we have analyzed the main limitations of an existing smart city deployment (SmartSantander case study) and derived future requirements for advanced use cases, such as autonomous driving, public safety and industrial IoT. Based on such requirements, we have proposed an ETSI MEC-based architecture to seamlessly integrate existing and future IoT platforms along with the required interfaces and protocols to enable communication between multi-technology sensors and IoT gateways through an IoT gateway middleware.

Further, we introduce the FIWARE platform and highlight its advantages. The open source and standard-based platform inherits the cutting-edge features and innovation brought by the open source community to be ready for commercial usage and business scenarios. We go through three real world use cases characterized by different requirements: IoT federation in smart cities, global IoT market realization by public governance, and research and innovation on autonomous driving scenario. We have, then, described the actual FIWARE-based solutions for the chosen use cases, explaining at the same time, the capabilities of the FIWARE framework. The functional comparison of FIWARE with other commercial and open source platforms shows that FIWARE is the only open source framework which is ready for commercial exploitation but also a forerunner of IoT features such as distributed stream analytics, marketplace, interoperability, privacy and security. The chapter concludes with an analysis of the FIWARE agenda with the next challenges to be addressed, demonstrating also the dynamism of the community.

Chapter 4

Federation of IoT Platform and IoT Data Sovereignties

In the past decade the Internet-of-Things concept has overwhelmingly entered all of the fields where data are produced and processed (e.g., health care, industry), resulting in a plethora of IoT platforms, typically cloud-based, which centralize data and services management. This has brought to a multitude of disjoint vertical IoT silos. Significant efforts have been devoted to making interfaces and data models interoperable, recurrently resulting into bigger centralized infrastructures. Such an approach often stumbles upon the reluctance of IoT system owners to loose control over their data. This chapter introduces a secured platform where a federation overlay is distributed among parties and the control over the data is delegated to data owners. In addition, privacy schemes are adopted to protect data exposure. We show that our architecture is scalable by design, since it allows iterative formation of multiple levels of domains thanks to the transparent nature of its federation approach, which hides the federation overlay from both data providers and data consumers. Experiments show that the overhead introduced is minimal when considering wide IoT deployments and in some scenarios our platform performs even better than centralized approaches.
The suggested work enables a readily global hyper-connected Internet of Things by linking together many of the currently deployed IoT systems.

4.1 LIoTS: League of IoT Sovereignties. A Scalable approach for a Transparent Privacy-safe Federation of Secured IoT Platforms

The Internet-of-Things (IoT) paradigm has lately gained more and more momentum in all the fields where data are produced and processed (e.g., automotive, health care, smart cities, industry) justifying the emergence of a plethora of IoT platforms. A common approach for IoT systems deployment is to leverage the scalability and performance of a cloud-based infrastructure for storing and analyzing data, in order to implement all kinds of IoT services and applications [91]. However, this brought to an abundance of single-scope, disjoint systems generally defined as “vertical IoT silos”.

A lot of efforts have been devoted to define standards and ontologies to enable, on the one hand, interoperability among platforms [92, 26, 87] and, on the other hand, inter-domain interaction [93] [94]. However, the harmonization of IoT systems often results in a further centralization towards another cloud instance [92], which leads to scalability issues when handling billions of devices. In addition, IoT systems owners are reluctant to loose control over the generated data [32] which is a serious obstacle to a global and connected Internet of Things.

As presented in chapter 1, the linkage between objects, devices, edge devices, actuators, agencies and services are becoming many-to-many, instead of the vertical silos of one-to-one connections. Therefore, there is a need to support information exchanges across multiple IoT platforms, formed by both cloud and edge, for more opportunities of holistic IoT services. Furthermore, the information discovery and exchange shall be ”transparent” to the IoT services, aiming at an automation of data provisioning taking into consideration the multitude of platforms administration domains.

This chapter presents the Leauge of IoT Sovereignties (LIoTS), a distributed IoT infrastructure that federates IoT systems while leaving the data ownership and sovereignty in the hands of data owners by offering means for security and privacy control. LIoTS enables data and services brokerage of both data queries and streams among peers of IoT platforms, hence overcoming the known barrier of fragmented IoT vertical silos. The proposed system takes advantage of the standardized protocol NGSI (Next Generation Service Interface), and of open-source software components part of a well established worldwide community, namely FIWARE Foundation, as described into chapter 3. LIoTS is, hence, directly benefiting from continuous advances in both standardization and implementation, as their recent embracing of semantics and linked data concepts [89], moving towards a global hyperconnected Internet-of-Things. The proposed approach enacts security and privacy-preserving schemes to enforce the will of the data owners. LIoTs design is scalable and iteratively replicable, since it allows for the construction of super-domains of domains thanks to the transparent nature of its federation that hides the federation overlay from both the perspectives of data providers and data consumers. Moreover, LIoTS design smooths the integration of new sensors and devices by adopting a plug-and-play approach to alleviates the federation burden in terms of configuration at the time of device integration, and on resource consumption during operation time.

Experimentation demonstrates a slight loss in terms of latency for traversing the federated and secured layer, but shows even better performance figures compared to centralized approach in large scale scenarios with thousands of things comparable to the city IoT deployments, such as SmartSantander [95]. The main contribution by this chapter are highlighted as follow:

- *Scalable federation by design*: The proposed architecture enables the transparent existence of multiple level of federations. This is achieved with the usage of a brokering layer for each of the federation level, and with multiple levels of security systems.

- *Sovereignty of data providers*: IoT providers keep data locally on their premises, thus maintaining their power over the owned data.
- *Privacy preserved*: The privacy of intra-domain users (e.g. applications, providers, persons) is prevented to be exposed externally in the federation. This is achieved with the multiple levels of security systems (intra-domain identities and policies are kept only within the domain), and with the IoT Registrar, a component to automatically generate data availability based on privacy directives given by the IoT providers.
- *High level of abstraction for IoT exchange*: A powerful domain-specific language for IoT data exchange is inherited by the usage of the NGSI protocol.
- *Plug-and-Play approach*: IoT providers and legacy systems are relieved from the burden for joining and maintaining the federation (such as declaring or updating data availability).

4.2 Background

In this section we analyze three IoT fields, centered around the concept of *IoT context*: federation, security, and standards and OSS. IoT context [18, 19] is a core concept associated with the 'status' of the real world. A context refers to an entity representing a thing (e.g., a sensor, a car, a building, a power line) together with its status. The context can be physically measured by a sensor or device, or derived by analytics functions (e.g., crowd estimation [96]).

4.2.1 Federation

Enabling the flowing of data among IoT platform instances and IoT services in a way that is transparent to IoT actors is a key point for a global Internet-of-Things overlay. A centralized approach where everything (viz. data and services) is handled by a remote authority is often not a solution for multiple reasons. For instance, critical and strict real-time applications need to run as close as possible to the location where data are generated. Another reason is the exploding amount of unnecessary data flowing into the Internet backbone, which happens, for instance, with camera-based applications which should actually run as close as possible to the camera. Finally, data confidentiality can be better preserved only if the data do not leave the owner's premises. In such cases, potential analytics from third-parties might be executed directly on local resources [3] (see chapter 2).

A first approach is to have methods for the discovery of services that provide IoT context [97, 87, 94]. Discovery is a pivotal element for seamless interoperability among systems, since not only datasets but also generic IoT services are exposed. This also enables an IoT marketplace, which is a gap identified in [98]. Still, discovery alone is not enough to automatize communications dispatching to the right actors. This last feature is the trait of a broker component that hides the discovery process and provisions data flows [3]. The mentioned approach actually empowers a transparent federation progressing towards the intent-based programming in the IoT [99] field. Nevertheless, the simplicity of cloud-based IoT system deployment is not neglected in this work, hence a hybrid approach is blended for a full-fledged IoT infrastructure.

4.2.2 Privacy and Security

One of the main disincentives of sharing data is the fear of loosing control over the produced data [32, 100]. Thus there is a clear need to include efficient and reliable privacy and security mechanisms. The study conducted in [101] assesses that most of the cloud-based IoT platforms often addresses typical web and network security attacks (such as DoS, eavesdropping), whereas [98] depicts privacy and data access control as open challenges. In this chapter we address those points by offering means for IoT owners to smooth the federation of their systems while, at the same time, preserving privacy.

4.2.3 Standards and Open Source

The usage of standards and open source software enables a faster integration among IoT systems [25]. Furthermore, adhesion to a well established open source community, such as FIWARE [60], permits to take advantage of the current ecosystem, as well as of its future developments (see chapter 3). For instance, disparate available protocol adapters, namely IoT Agents, together with the proven fully functional interoperability with oneM2M [25] and its wide set of supported device protocols, ease the IoT integeration. Furthermore, FIWARE already offers processes to federate static datasets [102] (e.g., open data).

FIWARE relies on a HTTP binding to the OMA NGSI protocol which is currently going to be integrated in the new version of the ETSI standard [89] together with an advanced query language. Moreover, the community has specified data models for many IoT domains. Thus, the LIoTS system inherits from the FIWARE community an openly standardized domain-specific language (DSL) for IoT data retrieval and services usage. Finally, efforts in the community are devoted for the introduction to the data usage control [90]. This thesis further contribute towards the data usage control (see chapter 5).

4.3 Related Works

Aspects of this chapter have been the focus of previous works. CityHub [93] introduces the creation of a centralized access point for IoT data (such as real-time and static datasets). The approach is to blend hybrid cloud for including the already established IoT systems and leverage a green-field cloud for the rest of deployment. Data are exposed through HyperCat [94], which envisages the possibility of doing federated semantic queries across multiple semantic data hubs. CityHub lacks the brokering of data requests to data sources, since a data requestor needs to specify the endpoints within the query, thus resulting into a non-transparent federation. The work [87] proposes a brokering of data through semantic requests, either as synchronous queries or as data streams, to applications. Their middleware automatically discovers semantically annotated sensor streams and associates them with application requests. A cloud data storage is handling the registrations of available sensors. The authors of [103] proposes a context-based query language with a centralized architecture that foresees the IoT data providers to their data with the central authority. Differently to [87, 103], LIoTS adopts a decentralized federation, enabling scalability by design, on top of a distributed security layer that preserves privacy of data and service owners. An approach that is similar to ours is described in [90]. The authors present a FIWARE-based system implementing the specifications and directives of the IDSA [16], namely the secured exchange of data between IoT domains. They propose an architecture for exchanging data among two FIWARE-based systems both protected by an Identity and Access Management (IAM). Each data provider is considered as a domain with centralized domain data management. Brokering context requests is possible for queries but not for subscriptions, due to the limitation of the used component Orion mainly designed as centralized context management. LIoTS overcomes these limitations with the introduction of other components specifically for IoT communication brokering that enables multiple levels of federation. Our proposal also uses multiple levels of security systems together with privacy schemes for protecting the privacy of intra-domain users and providers from the federation.

The interesting survey [104] proposes several solutions, realized at the network layer, such as multicastDNS (mDNS) and DNS Service Discovery (DNS-SD), for interconnecting data sources to consumers. Although the solutions are lightweight and natively offered by the Internet infrastructure, they are lacking the high level abstraction of complex data search. Further, the privacy and security aspects are not easily addressable.

4.4 Use Cases and Requirements

In this section we analyse some use-cases for identifying requirements to take into account into federation scenarios.

4.4.1 Use-Cases

A typical IoT use-case is *healthcare* where data is very sensitive but of the highest importance for saving lives. In present days, many medical applications rely on heterogeneous sensors deployed, for instance, at the patient's home [105], for monitoring and delivering promptly hints to allow timely alerts or for having offline data analysis. Deployed sensors push measurements to a gateway which is delegated to handle security. Gateways expose data or send reports to remote platforms related to diversified domains (e.g. cardiovascular, nephrology, nutrition) controlled by diverse administration. Big data analytics holistic medical applications need data input from multiple domains for computing accurate output. Moreover Quality of Data (QoD) must be very high, thus only allowed and not tampered sensors/gateways should be permitted to push measurements.

In *Smart Cities* many sensors are deployed anywhere in the public space and managed by diverse parties. For instance, there might be bus tracking sensors (measuring, e.g., speed, location) owned by transport companies, surveillance cameras controlled by homeland security, or crowdsensed data collected by tourist operator from smartphone app. A new smart service for citizens might offer to dynamically optimize the public transportation by scheduling bus arrival time and route on the needs, so as serving crowd moving within the city. For this purpose camera-based crowd detection, crowd mobility patterns and real-time fleet capacity utilization would be input for an online analysis. A third party organization offering such service might be not aware on who is providing such data.

In *Smart Industry*, machinery production companies are building apparatuses packed with sensors that monitors the instrumental life-cycle and operational behaviours. Normally industrial plants utilize equipments built by different machinery companies that are specialized on distinct niches offering their own data services[90]. Some applications aim at optimizing the production or strengthening factory security, perhaps offered by third parties, and are realizable only if data from both the producers are consumed. Obviously machinery companies are jealous of their data and want to keep the control over them.

4.4.2 Federation Requirements

The first two requirements from the use-case pertain to the way the data is exchanged. Both *synchronous queries* and *data streams* are important for analytics. In addition data must be *discoverable* by data applications without the need of human intervention especially in case dynamic changes happen in the offered portfolio, e.g. *data marketplace.* An added feature is to When possible, the data discovery should happen *transparently*: the process of finding the right data providers among a myriad of connected subsystems should not be visible. The requester would form the data requests containing the description of the wanted data and a middleware, such as a broker, is in charge of procuring them. To decouple even more the IoT infrastructure from the application developer, the data requests should be content-based (i.e., data messages matches constraints related to attributes) rather than topic-based (i.e., query or subscription to a channel known or discovered in advance).

Furthermore data is often highly confidential and regulated by restrictive laws (e.g. GDPR) defining how information must be handled. Hence in some cases the sensors records are locally stored close to where those are produced (e.g. into edge devices), or stored into clouds under full control of domain administration (e.g. hospitals). This brings the necessity of coping with *diverse IoT platform architectures*, e.g. cloud-based system for energy-constrained devices or edge-based system supported by gateways, and with *multiple levels of federation.* In addition things handled are of *enormous volume.*

4.4.3 Security and Privacy Requirements

Data producers necessitate to hold the *data ownership*, i.e. keep the data under their control and manage *data access control.* In addition a correct *authentication* of users avoids uncontrolled utilization of the platform, and together with *authorization* control, allows only selected actors to push data thus preventing data corruption. When providers want to expose their data they need to register the data availability within a

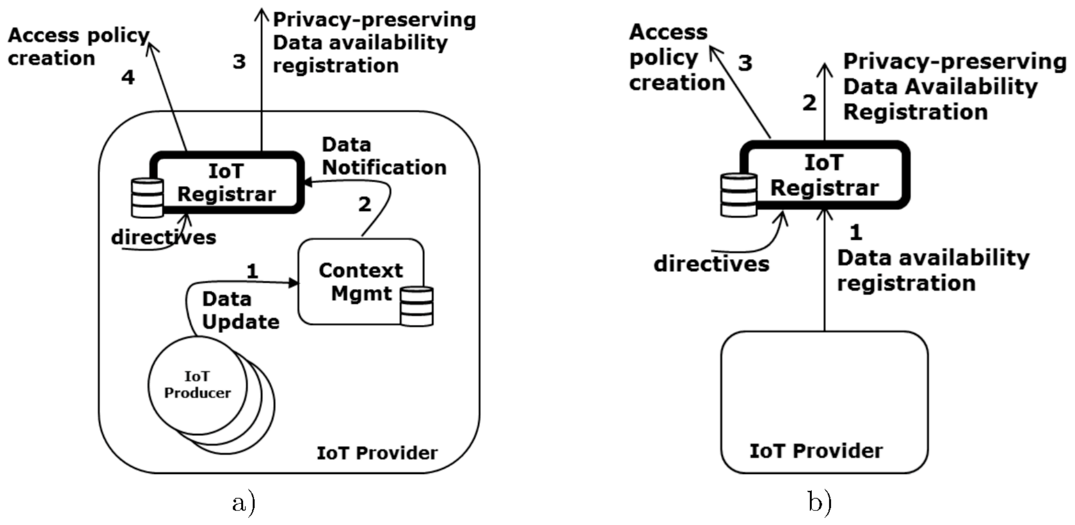


Figure 4.1: Privacy-preserving IoT Registrar: a) based on context; b) based on registrations

discovery component. Having registrations too generic (for example only the addresses of providers) might cause routing requests to providers even if they do not provide the pursued data. On the other hand, having a too detailed description (e.g. sensorId, exact location, name of maintainer) may disclose information which are sensitive, perhaps as much as measurements. For that reason an automatic *privacy-preserving registration system* is required.

4.5 System Design

The system we are going to describe is making usage of disparate components that play different roles divided in: *context layer*, *brokering layer* and *security layer*.

A first role in the context layer, the simplest, is the *IoT Producer* which produces data periodically or following an event (e.g. a wake up signal, a change in the environment). A typical example of IoT producer are sensors. A *Context Management* is capable of storing and indexing data and offers an interface for retrieving data. The interface offers methods for queries and subscriptions. An *IoT Provider* is a system composed of one or more IoT producers and a context management exposing the collected data.

Within the brokering layer, a *Broker* is a mediator that given a data request, dispatches it, transparently, to one or more IoT providers, depending on the request parameters. The broker exposes methods for executing brokered queries and subscriptions. A broker needs to be assisted by a *Discovery* component that serves as a registry of available IoT providers and their data. The discovery can be queried for matching providers and is capable of handling subscriptions, generating notifications when a new matching provider is declared.

In the security layer, the *Identity Manager (IdM)* holds and controls identities within the system. Each component in the system has a registered identity within the IdM, and the latter is offering a token-based protocol which allows components to securely identify each other. A *Policy Decision Point (PDP)* is a decision engine that responds to request with a decision based on policies stored within a *Policy Information Point (PIP)* and managed through a *Policy Management Point (PMP)*. For simplicity, in the rest of the chapter, when referring to PDP we assume a system formed by at least the PDP, PIP and PMP. A *Policy Enforcement Point (PEP)* secures a component by intercepting communications and imposing the policies stated in the system. For this purpose it is assisted by the IdM for authenticating requesters and by the PDP for deciding upon policies.

4.5.1 IoT Registrar

Table 4.1: IoT Registrar input and output

	Input: data	
x	data	Piece of data arriving to the IoT Registrar i
	Input: directive	
q	condition	If data x matches condition then registration and policy generation functions are applied.
$\phi_{reg}(x)$	registration generation functions	Set of functions to be applied on input data x for generating the registrations
$\phi_{pol}(x)$	policy generation functions	Set of functions to be applied on input data x for generating the policies
ρ	priority	Identifies the directive to be applied if more conditions from different directives are matching
	Output	
reg	registration	$reg = \phi_{reg}(x)$ Generated registration(s) by applying the transformation functions to data x.
pol	policy	$pol = \phi_{pol}(x)$ Generated policy(-ies) by applying the transformation functions to data x.

As a sort of glue between the described layer and components acts the *IoT Registrar*, that makes availability registrations to discovery component on behalf of human administrators (see Fig. 4.1). As discussed before, a registration to the discovery is required in order to allow automatic brokering of messages. A registration can be as generic as possible (raising unnecessary traffic to providers) or much detailed but disclosing sensitive information. If such registrations are made by human administrators, making detailed but privacy-preserving registration might require a lot of expertise for each deployment amend but also would induce latency on an effective usage of the sources: a plug-and-play approach would be much more advisable. The IoT registrar solves these problems by subscribing for information and synthesizing registrations (or updates old ones) when necessary. The IoT registrar bases the correct preserving privacy aggregation of information on given directives. For example, it could be instructed to register sensors only by their type (hence, if a second sensor appears of the same type and behind the same provider address, no changes are made to the registrations set) and by loose geographical area (for example, the location of a registration refers to the nearest municipality name instead to the exact coordinates). Also the creation of access policies is also a challenge for a correct behaviour of a secured IoT ecosystem [106]. Thus, we make usage of the registrar to dynamically populate with policies the PDP.

A directive is formed as follow:

$$\delta_i = (q, \phi_{reg}(x), \phi_{pol}(x), \rho)$$

where q is the condition the information x must match, $\phi_{reg}(x)$ and $\phi_{pol}(x)$ are, respectively, the registration and policy generation functions to be applied on input data x, and ρ is the priority to discern in case of conflicts . ϕ_{reg} is a set of functions for generating the registrations (Fig. 4.1.a step 3, Fig. 4.1.b step 2). Each function might be specified, as examples but not limited to, with a mathematical function or code snippet. Similarly policy generation functions is a set of functions for generating the policies (Fig. 4.1.a step 4, Fig. 4.1.b step 3). The functions of the directive are applied when the condition matches against the data message. The priority unambiguously identifies the directive to apply if conditions from different directives are matching.

The directives storage of the IoT Registrar manages the following set:

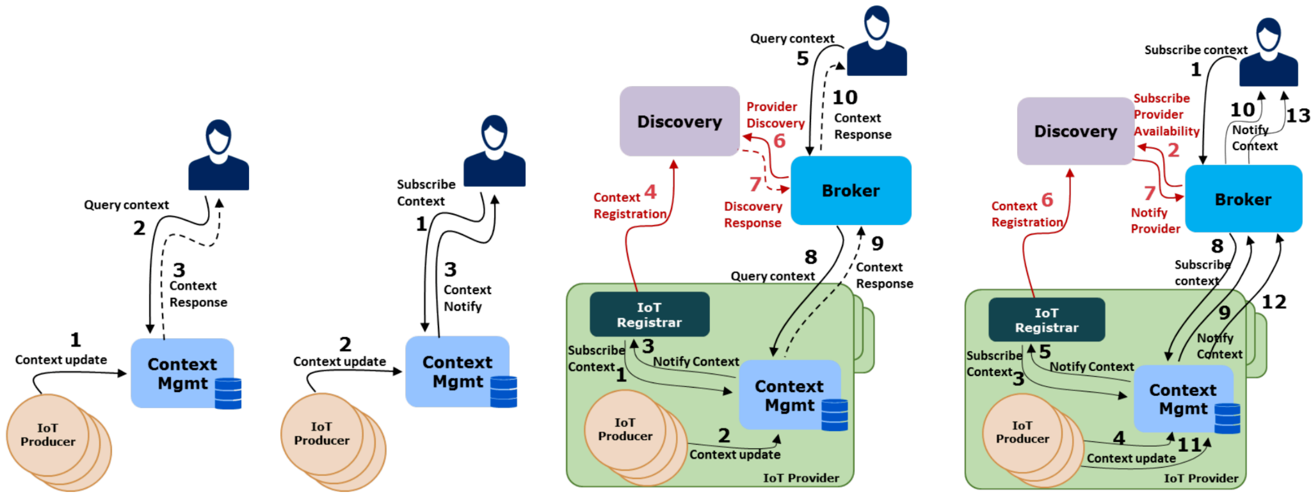
$$D = \delta_1, \delta_2, ..., \delta_n$$

4.5.2 Message Flows

The IoT data traffic is typically following 4 paradigms: *Publish-Query*, *Publish-Subscribe*, *Distributed Query*, *Publish-Notify*. In the first paradigm (Fig. 2a) a context producer, such as a sensor, publishes context updates to a CM which stores them. When the latter is queried it responds with the status of the requested contexts. Listing 4.1 shows an sample of IoT context. Another example of context producers pushing data is an IoT gateway, locally storing data, aggregating messages and periodically sending a report to the cloud.

Listing 4.1: Sample of IoT context in NGSI

```
{ "contextElements": [{
    "entityId": {"id":"bus18", "isPattern":false, "type":"bus"},
    "attributes": [
      {"name":"speed", "type":"float", "contextValue":"25"}],
    "domainMetadata": [
       {"name":"SimpleGeolocation", "type":"point",
        "value": {"latitude":43.4628, "longitude":-3.80031}}]
}]}
```



(a) Publish-Query (b) Publish-Subscribe (c) Distributed Query (d) Subscribe-Notify

Figure 4.2: The four IoT data traffic flow paradigms. Solid lines are requests, dashed lines are responses. Black lines indicate context data messages, red lines for providers information.

In the publish-subscribe (Fig. 2b), an application first declares its interest in context data to the CM (see Listing 4.2). Then context producers publish context updates and, when matching with the subscription, the CM asynchronously notifies subscriber at the given reference.

Listing 4.2: Sample of subscription in NGSI DSL

```
{ "reference":"http://172.17.0.1:8201/ngsi10/notify",
  "entities":[{"id":".*", "isPattern": true, "type":"car"},
              {"id":".*", "isPattern": true, "type":"bus"}],
  "attributes": ["speed"]}
```

The distributed query paradigm is a bit more complex (see Fig. 4.2c). Data is not residing in a single body but need to be procured from different providers. In this case, context producers, a CM and an IoT registrar build together a complete IoT provider. As initialization the IoT registrar subscribes for all context to the CM (step 1). Then, when a context is pushed (step 2) and notified (step 3), the IoT registrar checks if such context has been already treated before. If not, then the IoT registrar announces the availability of context with a registration (step 4) identifying the CM as provider. At this point, when a user or an application queries for context to the broker, the latter first discovers available provider offering such context (step 6 and 7), resulting in a message similar to Listing 4.3, and then it proceeds to contact all the discovered providers (step 8). The responses are then aggregated and the result forwarded to requestor. The responses aggregation might be simply to collect them into a set of context information or might be a function (such as averaging homologous contexts). In the case depicted in Fig. 4.2c, the CM within the IoT provider responds with the data generated by producers. In other scenarios the data might be generated by a services, coming from a storage, or measured by a sensor upon the stimulation of step 8. All this is completely transparent from the broker perspective as long as the services expose the same interface of the CM.

Listing 4.3: Sample of data availability in NGSI

```
{ "contextRegistrationResponses": [{
    "contextRegistration": {
      "providingApplication": "http://172.18.2.70:8060/ngsi10",
      "entities": [{"id":".*", "type":"bus", "isPattern":true}]}
}]}
```

The subscribe-notify paradigm (see Fig. 4.2d) is initiated by a context subscription made by a user or application to the broker (step 1) which, as a consequence, subscribes for the availability of context providers (step 2). When a new provider availability is notified (step 7), due to already available registration or a fresh one (step 6), the broker instantiates a subscription to the notified IoT provider, front-ended by the CM. If an IoT context is already available then it is forwarded to the user (step 9,10). Every new context matching the subscription follow the established channel (12,13) till an amendment of data availability or subscription happens.

Typically the cloud IoT platforms implement one or both the first two paradigms implying that data is continuously flowing to the cloud regardless to actual interest of users and/or applications. The distributed query and subscribe-notify paradigms, instead, foresee ”lazy” data flows as the data is pulled from remote provider on demand. In order to keep the simplicity of the centralized paradigm also for the distributed paradigms, we have introduced the IoT registrar component to be deployed locally to the provider together with the CM. In case that the IoT system administrators do not have the possibility (or do not want) to maintain any system locally, the CM can be offered in the cloud as a service. With this approach the motivation of minimizing traffic to cloud falls apart, but still other advantages hold, such as the full power over the data by administrators and the federation of cloud providers overcoming interoperability issues due to vendor lock-in silos.

4.5.3 Multi-party exchange platform system architecture

The system architecture of a secured multi-party IoT data exchange platform together with an illustrative data exchange is shown in Fig. 4.3. Each provider silo manages its own IoT deployment and handles data within its premises or its cloud of choice, exposing them through a context management (see Fig. 4.2c and 4.2d).

One or more IoT providers are clustered in domains. Every domain has two independent security systems for two scopes of action: the *intra-domain security system* and the *federation security system*. The intra-domain security, coloured in red in Fig. 4.3, is formed by an IdM (idIDM) and a PDP (idPDP) and various PEPs, one for each component to secure against internal access. When a message arrives to a secured component carrying an access token, the PEP first checks the authenticity of the sender with the assistance of the idIdM and afterwards requests an access right check to the idPDP. If any of the verification does not succeed the access is denied otherwise the message is forwarded, transparently, to the secured component. The federation security system is

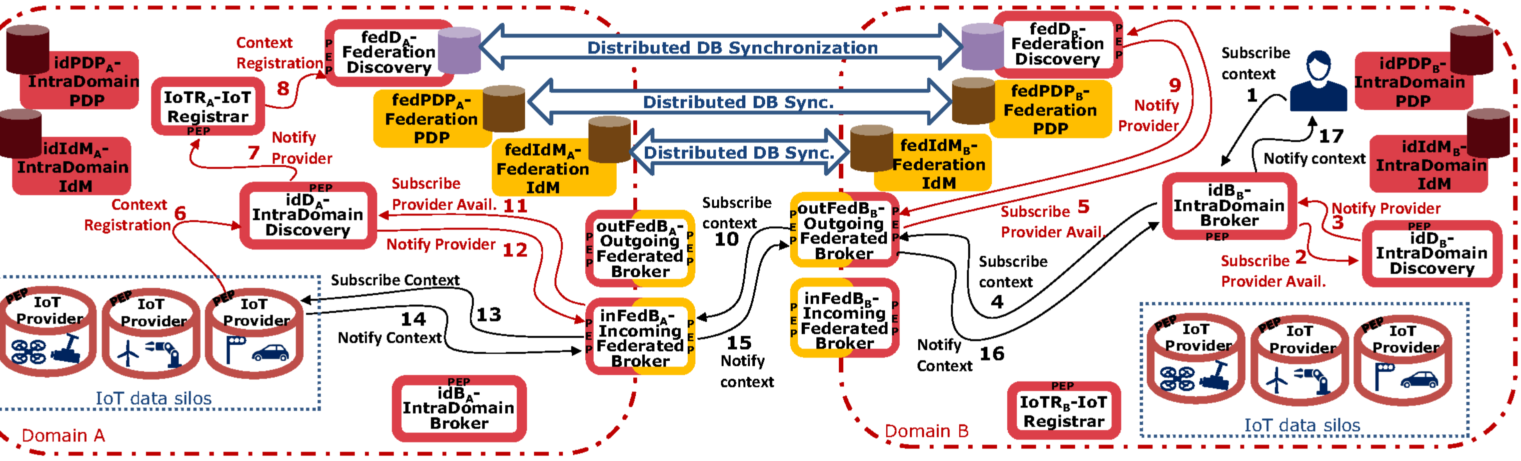


Figure 4.3: General architecture of a multi-party secured exchange platform with subscribe. Messages with no data (e.g., ACKs) are omitted. The tick edges of components indicate the protection by a PEP; the colors of the edges indicate the security tier: red for intra-domain, yellow for federation.

formed by a federation IdM (fedIdM) and PDP (fedPDP) for each domain and a PEP for every component exposed externally. The fedIdMs and fedPDPs have synchronized databases such that the same request done to any of them results in the same response. The synchronization between the databases might be achieved via state-of-the-art distributed databases (e.g. Cassandra, Couchbase) or blockchain-based technology. The scenario we envision for the LIoTS is not fully trusted since the domains are controlled by different administrations that might be embodied by different companies. Every domain acts as a data provider or a data consumer, depending on the particular use case. This scenario is well fitting into a marketplace model where data is offered by data provider and acquired by data consumers, strangers between them. In addition, the data produced by data consumers through analytics might as well be exposed on the marketplace [107] (we will develop further this concept on the data usage control chapter 5). A trend in the IoT field is to adopt blockchain in this kind of scenarios [108]. Therefore, we adopt blockchain technologies to propagate information among the federated discoveries (fedDs) and the federated PDPs (fedPDPs). This brings several advantages such as the convergence to a deterministic state into the federation, enable a transactions approach for policy negotiations (i.e., smart contracts), and allow traceability of data flows.

Every domain possesses an identity for the authentication and authorization between domains. The federation access policies are visible by all parties. At setup time, components needs to make configurations for enabling the federated communication as summarized in Tab. 4.2.

For intra-domain data exchange the behaviours and message flows are similar to the one shown in Fig. 4.2c and 4.2d, with the brokerage of an intra-domain Broker (idB) assisted by an intra-domain Discovery (idD). When the data requested by a user is available in a different domain, other components enter the game. A federation discovery (fedD) is run in every domain with the database synchronized similarly to the fedPDPs and fedIdMs. Thus registrations stored in a fedD are visible by anybody in the federation and therefore it is of utmost importance to control which registrations are made. Each domain has an IoT Registrar (IoTR) that accepts registrations and synthesize them following directives. For brokering messages between domains each of the parties deploy two instances of the broker: one for handling incoming request from outside the domain, namely incoming federation Broker (inFedB), and another one to handle outgoing request to the other domains of the federation, namely outgoing federation Broker (outFedB). The two federation brokers are associated to two different discoveries. In fact, when a request comes from inside the domain, the outFedB needs to inquiry the fedD for discovering other domains providing the data of interest. When a request comes from the outside of the domain, instead, the inFedB discovers data providers within the domain against the idD (see Tab.4.2). Therefore all the registrations done by the IoTR to the fedD must carry the exposed address of the inFedB (or its exposed PEP address). Each of these boundary brokers are protected by two different PEPs, one assisted by the intra-domain security system with the aim of regulating who in the domains may do federated requests, and one assisted by the federation security system for moderating requests from known federation parties.

Having two disjoint security layers allows domain administrators to decide which section of data can be exposed. In other words, a request coming by the inFedB might be treated differently by an IoT provider PEP than a request coming from within the domain. In addition if domain$_B$ has different federation access policies among domain$_B$ users, the outFedB$_B$ will receive unfiltered data, but the PEP$_{idB_B}$ will refine data as prescribed in PDP$_B$. This approach permits to hide a user$_B$ existence and associated policies from external.

Table 4.2: Domain infrastructure settings

	Access Control	**Registration**	**Subscription**	**Discovery**
idB	domain	none	none	idD
idD	domain	none	none	n/a
IoTR	domain	none	provider avail. to idD	n/a
outFedB	domain for query and subscribe; fed for notify	as provider for everything to idD	none	fedD
inFedB	domain for notify; fed for query and subscribe	as provider for every registrations in fedD	none	idD
fedD	domain	none	none	n/a

Fig. 4.3 illustrates also an actual scenario: a user$_B$ within domain$_B$ subscribes for data provided by an IoT provider residing within domain$_A$. User$_B$, hence, sends a request (1) to the PEP of the idB$_B$ carrying the user$_B$ access token. The PEP$_{idB_B}$ checks the user identity, authorization and access control against the idPDP$_B$ and idIdM$_B$ and if

successful, the request is passed to the idB_B which subscribes for provider availability (2) passing through a PEP. For simplicity we omit the next intra-domain hereinafter. The only resulting notification (3) contains the $outFedB_B$, as initialization settings, which is contacted (4) and triggers an availability subscription to the $fedD_B$ (5). Once a new matching registrations appear (step 6-8) the provider is notified (9) and an inter-domain channel is initialized. The $outFedB_B$ performs a subscription (10) to the $inFedB_A$ using the federation identity and token of the $domain_B$. The PEP of the receiver is controlling against the $fedIdM_A$ and $fedPDP_A$ the authorizations. In case of success, a subscription arrives to the idD_A (11) and because of (6) the real provider is finally contacted (13). At this point the inter-domain channel is established and data flowing from $IoTP_A$ to $user_B$ (step 14-17) for each new context update.

The query scenario can be derived from the complex scenario just described and the distributed query paradigm of §4.5.2.

4.5.4 How to scale the federation

The proposed architecture can be scaled for achieving super-domains of domains iteratively. For this purpose, each domain of Fig. 4.3 is seen as domain IoT provider by the super-domain infrastructure (see Fig. 4.4). The latter manages a super-domain set of components such as: a super-domain PDP (sdPDP) and IdM (sdIdM) synchronized, respectively, with the domains federation PDP (dFedPDP) and IdM (dFedIdM), super-domain Discovery (sdD) synchronized with the domains Federation Discoveries (dFedDs), a super-domain IoT Registrar (sdIoTR) subscribed to the sdD, a super-domain Broker (sdB) that references to the sdD, a super-domains federation Discovery (sdFedD) synchronized with federation Discoveries of other super-domains, a super-domains federation PDP (sdFedPDP) and IdM (sdFedIdM) synchronized with other super-domains, and an outgoing and incoming super-domains federation Broker (outsdFedB and insdFedB). In this architecture there are three separated security layers: the intra-domain, super-domain, and super-domains federation.

The system is quite flexible to different scenarios and configurations. It is possible to have hybrid solutions where IoT providers are directly managed by the super-domain, as shown in Fig. 4.4. In the shown example there are 3 levels of federation (i.e. IoT providers, domains and super-domains) but, theoretically, the numbers of levels is unbounded.

4.6 System Implementation

4.6.1 Standards and Open Source software adopted

LIoTS implementation is based on open source software components published within the FIWARE framework [5]. The FIWARE components are in some cases adopted "as is" and sometimes with modifications. Additionally, we implemented the IoT registrar, subscription proxy, blockchain wrapper and adapter. Finally, we used the open source

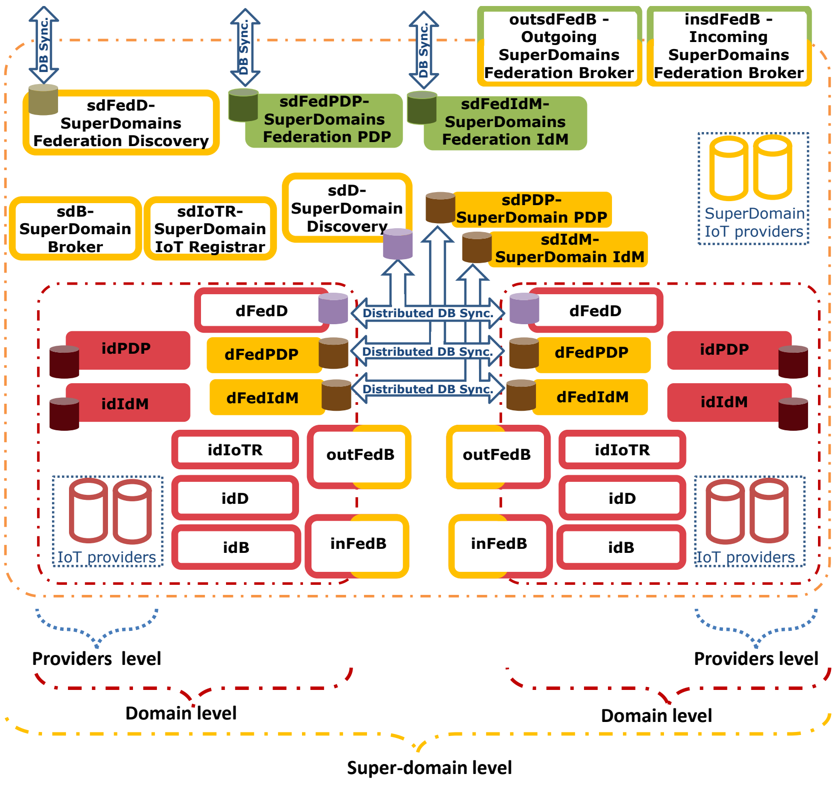


Figure 4.4: Scaling the proposed architecture iteratively.

Hyperledger Fabric software as blockchain technology. Table 4.3 summarize the list of components.

As data protocol we used the OMA/FIWARE NGSI protocol [58]. This permits the content-based data exchange system given by the highly flexibility of its query model. Furthermore, FIWARE community is very active on the definition of data models [80, 27] for several application fields (e.g., smart cities, transportation, weather).

4.6.2 Domain IoT Registrar

We implemented the IoT Registrar (Fig. 4.1) component using NodeJS and relying on the NGSI protocol. The purpose of this component is twofold: i) to register context availability into the fedD, in order to make data available from IoT providers to other domain's data consumers; ii) to store policies into the fedPDP to regulate the data access from other domains. The condition q of a directive (see §4.5.1) is modelled through the NGSI subscription parameters, e.g.,: pattern of entities (things), attributes list, meta-data (e.g., geographic scope). For each new directive, the IoT registrar issues an *NGSI-9 subscribeContextAvailability* request to the idD component and awaits a notification for every new context information available. In case of the notification, the IoT registrar prepare a context availability registration following the registration federation functions.

Table 4.3: Ad-hoc components implementation and changelog of the adopted open source components for our prototype

Component	**Software**	**Modifications**
idIdM, fedIdM	FIWARE Keyrock[1]	as it is
idPdP, fedPdP	FIWARE AuthZForce[2]	Attribute-based rules
PEP	FIWARE Wilma[3]	Forward subscription to subscription proxy
outFedB, inFedB, idB	FIWARE Aeron Broker[4]	Configurability of exposed addresses for forwarded subscriptions Interoperability with Orion (i.e., throttling, datamodel) Integration with IdM (to request access tokens)
idD, fedD	FIWARE NeConfMan[5]	Geographic discovery Integration with IdM (to request access tokens)
CM	FIWARE Orion[6];	as it is
	FIWARE Aeron Broker	(see above)
Subscription Proxy	Our implementation	-
IoT Registrar	Our implementation	-
Blockchain	Hyperledge Fabric[7]	as it is
Discovery wrapper	Our implementation	-
PDP wrapper	Our implementation	-
Blockchain adapter	Our implementation	-

Context availability registrations on the fedD should be privacy-preserving, since they are visible outside the domain into the federation. Thus, we implement different $\phi_{reg}(x)$ (registration generation functions) to produce registrations at various level of details:

- *Entity Level*: every context availability registration refers to a single context entity (i.e., thing, sensor), thus containing ID, type and geographic bounding box of the entity.
- *Type Level*: every context availability registration refers to a type of context entities. Thus, given a type T1, there is only one registration for all type T1 entities within the domain, containing an aggregated geographic bounding box covering all the entities with geographic metadata. The bounding box of the registration is updated whenever a new entity of the same type becomes available and is not covered by the actual bounding box. No entities IDs are published outside the domain into the fedD.
- *Provider Level*: this is the most privacy-preserving configuration, where only one registration is issued for all the context information containing the bounding box for all the geographic annotated entities. The bounding box is updated accordingly to new entities. No types and no IDs are published into the fedD.

The default access policy into the security system is to deny access if not differently stated with a policy. The IoTR updates access policies on the fedPDP, in order to allow access to the registered context information from outside the domain. This task rely on the "providerClass" annotation specified by within the intra-domain registration of the IoT provider (in the next section we will see this being set through a $\phi_{reg}(x)$

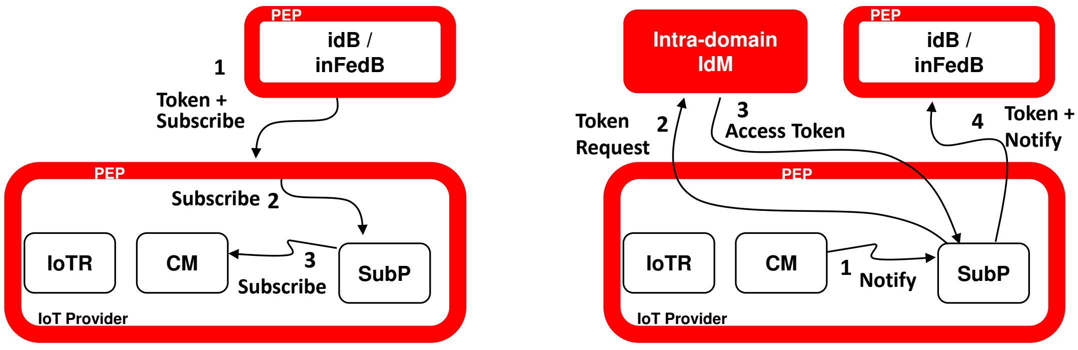


Figure 4.5: Integration of the IoT provider with the intra-domain security layer using a Subscription Proxy (SubP).

of the IoT provider's IoTR). If the intra-domain registration annotates the available information as "openData", the IoT Registrar stores an access policy into the fedPDP, adding the corresponding entity type to the list of types accessible from outside the domain. When the context information is labelled as "protectedData", a registration is made into the fedD but not policy added into the fedPDP, thus, it becomes available but not accessible from outside the domain. In this case, an external domain may still discover the available protected data and, then, negotiate the access to them, for example, through a blockchain-based marketplace as presented in §4.6.5. When data is marked as "confidentialData" no action is taken from the IoT Registrar and, therefore, it becomes not accessible nor even discoverable from outside the domain.

4.6.3 IoT Provider implementation: privacy, security and discoverability

The conceptual paradigms of an IoT provider depicted in Fig. 4.2.c-d do not consider the security aspect. For protecting the Context Manager it suffices to insulate it with a PEP component that intercepts all the incoming requests and enforces authentication, authorization and access control. This approach is enough for synchronous communication (i.e., query) but not for asynchronous paradigm such as subscribe-notify (4.2.d). In the latter case, the CM is initializing the communication for a notification which must carry an access token to get through the PEP that protects the idB from tampered or false data by unauthorized data producers. For that purpose, we introduce a Subscription Proxy that subscribes to the CM on behalf of the data producer, and forward notifications to subscribers enhanced with the access token retrieved from the idIdM (see Fig. 4.5). The IoT provider PEP proxy is configured to forward all the subscription to the Subscription Proxy (SubP).

We used an IoT registrar to build up a full IoT provider (see Fig. 4.1.a). It registers the availability of context information on the idD when needed. The condition q is again modelled through the NGSI subscription parameters, that are carried within a NGSI-10 subscribeContext used for context data messages (differently from the availability messages as in the domain IoT registrar, since a discovery component is not available within the IoT provider). For every directive, the IoT registrar issues an NGSI-10 subscribeContext request to the CM. The IoTR awaits for notifications (i.e., NGSI-10 notifications) regarding new context information published into the CM. The registrar then applies the registration generation functions to the notified context data and issues a registration (i.e., NGSI-9 registerContext) to the idD with the availability of new entities. For the IoT provider case, we configured the registration to be always at the entity level, thus, specifying both the type and the entity id. The implemented $\phi_{reg}(x)$ (with a NodeJS code snippet) aggregates the geographical location of the pushed entity with a bounding box. We used three different directives with conditions matching to three different data types: open data, protected data and confidential data. When data matches one of those directive, the $\phi_{reg}(x)$ adds a "providerClass" annotation to the registration with the appropriate values among "openData" (openly accessible from the federation), "protectedData" (data available to the federation but accessible through further negotiation), and "confidentialData" (data only available from within the local domain). This context metadata will then arrive, through the idD, to the domain IoT Registrar (section 4.6.2), and is used to update the access policies on the fedPDP component.

4.6.4 Federation through Blockchain

We integrate the Hyperledger-Fabric framework [109] with the rest of the system. Each federation domain has a blockchain peer that is connected to the LIoTS through a blockchain adapter (BCa). The BCa provides an API with two methods:

- **/validatePolicy**, used to save a new policy into the ledger;
- **/validateRegistration**, for saving a new registration into the ledger.

This API is used by two more components: a *PDP Wrapper* and a *Discovery Wrapper*. These components intercept every write request to the fedPDP or fedD components and interact with the blockchain for storing and updating policies and registrations. The wrappers provide a method invoked by the BCa to notify the consensus reached on the blockchain network for the new policy or registration:

- **/notifyPolicy,** provided by the PDP Wrapper;
- **/notifyRegistration,** provided by the Discovery Wrapper.

The blockchain peer notifies the corresponding BCa every time a new block is added to the ledger, and the BCa forwards the notification to the corresponding PDP or Discovery wrapper. Thus, the notifyPolicy and notifyRegistration are invoked only when a consensus is reached into the federation and the same set of registrations and policies are, then, updated locally in each domain, resulting to a shared status between the local fedDs and fedPDPs. The synchronization process, with respect to the fedPDP case, is shown in figure 4.6.

It is interesting to note that the component requesting a write operation to the fedPDP (or fedD) receives a response only when the write operation is added to the

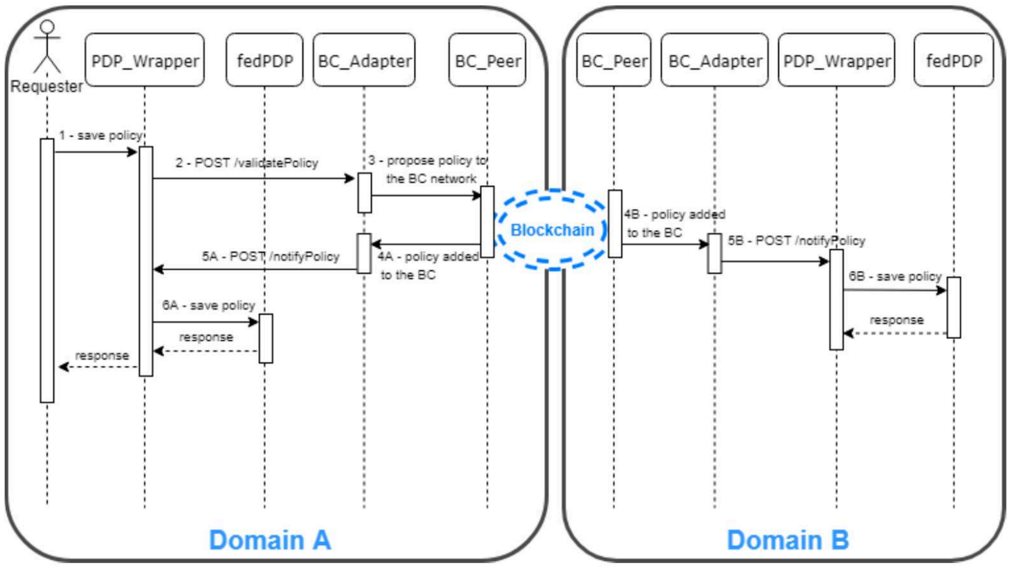


Figure 4.6: Federated Policy Decision Points (fedPDPs) through blockchain

blockchain ledger. In order to do so, the wrapper keeps track of all the pending requests, and respond to them when the corresponding notification arrives from the BC Adapter and the policy (or registration) request is forwarded to the fedPDP (or fedD).

4.6.5 Application example: Marketplace

We develop an IoT application on top of our system that allows automated data sharing among parties through a marketplace. The latter discovers data available and allows the access to data after a purchase. The transaction is stored into blockchain, and, once it is verified, triggers a creation of a new policy again stored in the blockchain.

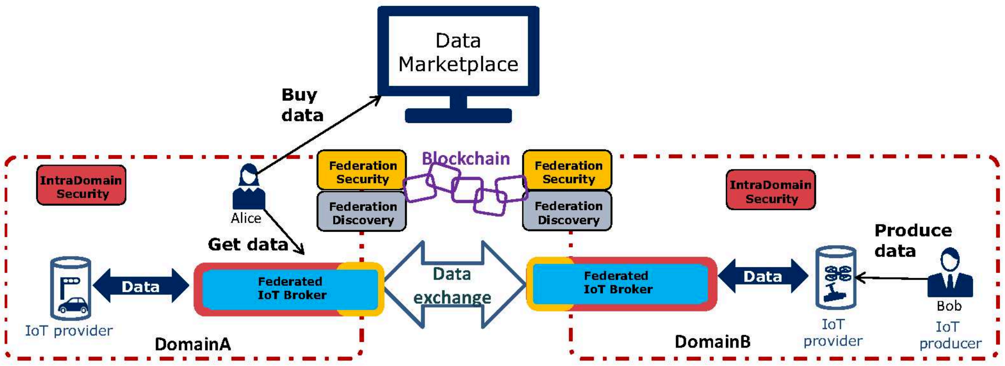


Figure 4.7: IoT marketplace application for federated systems.

The application scenario is the one depicted in Fig. 4.7. Bob is a data producer of domainB owning an IoT deployment and pushing data to an IoT provider. Within the domainA, Alice is a data consumer that needs data for her application. The IoT Registrar of DomainB is configured to aggregate registrations at type level, while the IoT Registrar of the IoT providers registers data availability at entity level.

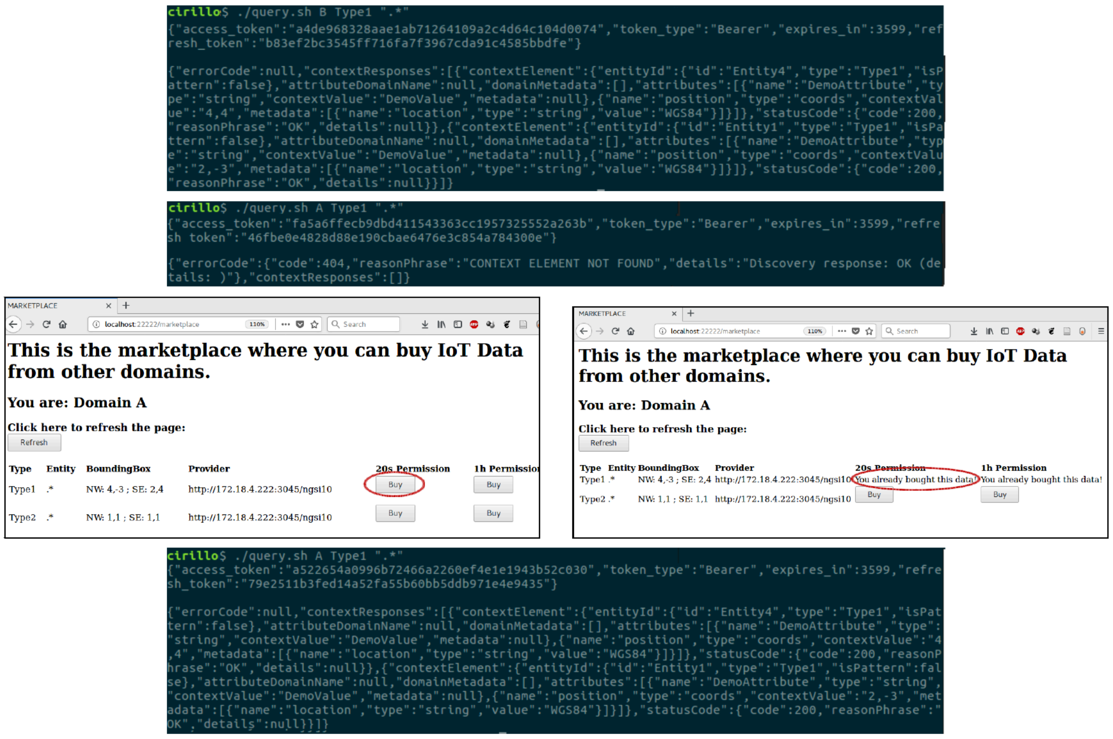
Table 4.4: Data produced by Bob and registrations available from within the domain (made by the IoT registrar of the IoT provider) and in the federation (made by the IoT registrar of the domain).

Pushed Data (by Bob)	Intra-domain Registrations (aggregation: entity level)	Federated Registrations (aggregation: type level)
entityId: {id: ”Entity1”, type: ”Type1”}, attributes:[{name: ”attr1”,contextValue: ”value1”}, {”name”: ”position”, contextValue: ”2,-3”]}	entities: [{id: ”Entity1”, type: ”Type1”}], contextMetadata: [{ {name: ”providerClass”, value: ”protectedData”} {name: ”SimpleGeoLocation”, value: {nwCorner: ”2,-3”, seCorner: ”2,-3”}}}], providingApplication: ”http://172.18.2.41:3045/v1”	entities: [{id: ”.*”, type: ”Type1”, isPattern: ”True”}] contextMetadata: [{ {name: ”SimpleGeoLocation”, value: {nwCorner: ”2,-3”, seCorner: ”2,-3”}}}], providingApplication: ”http://172.18.4.222:3045/ngsi10”
entityId: {id: ”Entity4”, type: ”Type1”}, attributes:[{name: ”attr1”,contextValue: ”value2”}, {”name”: ”position”, contextValue: ”4,4”]}	entities: [{id: ”Entity4”, type: ”Type1”}], contextMetadata: [{ {name: ”providerClass”, value: ”protectedData”} {name: ”SimpleGeoLocation”, value: {nwCorner: ”4,4”, seCorner: ”4,4”}}}], providingApplication: ”http://172.18.2.41:3045/v1”	entities: [{id: ”.*”, type: ”Type1”, isPattern: ”True”}] contextMetadata: [{ {name: ”SimpleGeoLocation”, value: {nwCorner: ”2,4”, seCorner: ”2,-3”}}}], providingApplication: ”http://172.18.4.222:3045/ngsi10”

Bob classifies his data as ”protectedData” since he wishes to get money on return of sharing data. Table 4.4 shows the data pushed by Bob related to two entities (first columns), and the produced registrations available from within the domain and into the federation, made by the IoT provider’s IoT registrar and by the intra-domain IoT registrar respectively. There are two intra-domain registrations, each refers to a pushed entities. There is only one federated registration (third column of Table 4.4), with two states that changes during time. The second federated registration overwrites the first one after the push of the second entity since the two registrations refer to the same set of entities (i.e., same entity type and a generic id declared by the regular expression wildcard $.*$). The second registration has an updated bounding box covering both entities. The class of the provider is ”protectedData”, thus, the intra-domain IoT registrar generates no policy, resulting into data discoverable (because of the registration) but not accessible (because of the default conservative policy) among the federation. The *providingApplications* refers to the IoT provider’s Context Management and to the $inFedB_B$ (or their shielding PEP components) for the intra-domain and federated registrations respectively.

At this point the data is accessible by Bob but not by Alice (see Fig. 4.8.a-b). At the beginning of each request, an access token is given by the fedIdM which is, then, carried in the request. The PEP proxy of the federated broker is rejecting every request coming outside domainB. Alice can discovery on the marketplace what data she can buy. When she buys data by clicking the button in the marketplace (Fig. 4.8.c) a transactions is recorded in the blockchain and a new policy is stored into the federated PDP. At this point, Alice can receive the data for type *Type1* (Fig. 4.8.d) provided by and IoT Provider of a different domain. After 20 seconds the purchase expires, the policy is deleted from the PDP, and, consequently, the Alice’s right to access Type1 data withdrawn.

Being our system designed and implemented following open principles, standards and software, more complex marketplace can be used on top such as the full-fledged



a)

```
cirillo$ ./query.sh B Type1 ".*"
{"access_token":"a4de968328aae1ab71264109a2c4d64c104d0074","token_type":"Bearer","expires_in":3599,"ref
resh_token":"b83ef2bc3545ff716fa7f3967cda91c4585bbdfe"}

{"errorCode":null,"contextResponses":[{"contextElement":{"entityId":{"id":"Entity4","type":"Type1","isP
attern":false},"attributeDomainName":null,"domainMetadata":[],"attributes":[{"name":"DemoAttribute","ty
pe":"string","contextValue":"DemoValue","metadata":null},{"name":"position","type":"coords","contextVal
ue":"4,4","metadata":[{"name":"location","type":"string","value":"WGS84"}]}]},"statusCode":{"code":200,
"reasonPhrase":"OK","details":null}},{"contextElement":{"entityId":{"id":"Entity1","type":"Type1","isPa
ttern":false},"attributeDomainName":null,"domainMetadata":[],"attributes":[{"name":"DemoAttribute","typ
e":"string","contextValue":"DemoValue","metadata":null},{"name":"position","type":"coords","contextValu
e":"2,-3","metadata":[{"name":"location","type":"string","value":"WGS84"}]}]},"statusCode":{"code":200,
"reasonPhrase":"OK","details":null}}]}
```

b)

```
cirillo$ ./query.sh A Type1 ".*"
{"access_token":"fa5a6ffecb9dbd411543363cc1957325552a263b","token_type":"Bearer","expires_in":3599,"refre
sh token":"46fbe0e4828d88e190cbae6476e3c854a784300e"}

{"errorCode":{"code":404,"reasonPhrase":"CONTEXT ELEMENT NOT FOUND","details":"Discovery response: OK (de
tails: )"},"contextResponses":[]}
```

c)

MARKETPLACE

localhost:22222/marketplace

This is the marketplace where you can buy IoT Data from other domains.

You are: Domain A

Click here to refresh the page:

Refresh

Type	Entity	BoundingBox	Provider	20s Permission	1h Permission
Type1	.*	NW: 4,-3 ; SE: 2,4	http://172.18.4.222:3045/ngsi10	Buy	Buy
Type2	.*	NW: 1,1 ; SE: 1,1	http://172.18.4.222:3045/ngsi10	Buy	Buy

MARKETPLACE

localhost:22222/marketplace

This is the marketplace where you can buy IoT Data from other domains.

You are: Domain A

Click here to refresh the page:

Refresh

Type	Entity	BoundingBox	Provider	20s Permission	1h Permission
Type1	.*	NW: 4,-3 ; SE: 2,4	http://172.18.4.222:3045/ngsi10	You already bought this data!	You already bought this data!
Type2	.*	NW: 1,1 ; SE: 1,1	http://172.18.4.222:3045/ngsi10	Buy	Buy

d)

```
cirillo$ ./query.sh A Type1 ".*"
{"access_token":"a522654a0996b72466a2260ef4e1e1943b52c030","token_type":"Bearer","expires_in":3599,"refre
sh_token":"79e2511b3fed14a52fa55b60bb5ddb971e4e9435"}

{"errorCode":null,"contextResponses":[{"contextElement":{"entityId":{"id":"Entity4","type":"Type1","isPat
tern":false},"attributeDomainName":null,"domainMetadata":[],"attributes":[{"name":"DemoAttribute","type":
"string","contextValue":"DemoValue","metadata":null},{"name":"position","type":"coords","contextValue":"4
,4","metadata":[{"name":"location","type":"string","value":"WGS84"}]}]},"statusCode":{"code":200,"reasonP
hrase":"OK","details":null}},{"contextElement":{"entityId":{"id":"Entity1","type":"Type1","isPattern":fal
se},"attributeDomainName":null,"domainMetadata":[],"attributes":[{"name":"DemoAttribute","type":"string",
"contextValue":"DemoValue","metadata":null},{"name":"position","type":"coords","contextValue":"2,-3","met
adata":[{"name":"location","type":"string","value":"WGS84"}]}]},"statusCode":{"code":200,"reasonPhrase":"
OK","details":null}}]}
```

Figure 4.8: Buy data through the IoT Marketplace: a) Bob requests and receives data from his own domain, b) Alice requests but receives no data from domainB, c) Alice browse the marketplace and buys data of type *Type1*, d) Alice has access to data for 20 seconds

SynchroniCity IoT Marketplace[8] or the FIWARE Biz Framework[9]

4.7 Evaluation

Tab. 4.5 shows the numbers of messages exchanged for query, establishing a subscription channel, and return a notification, for centralized approach, federated approach but not secured, and federation with security. We assume that the user making the request is not protected by any PEP. Both asynchronous (i.e., requests, notifications) and synchronous (i.e., response to requests) messages are counted. The row regarding the secured federation is reporting a lower bound of messages, which is the case when all the components already possess a valid access token. As we might expect, passing through security layers (PEPs, IdMs and PDPs) increases significantly the number of total messages exchanged. Nevertheless, we will see in emulation experiments that the performance is affected only in certain scenarios.

Our experiments use docker containers and networks for testing the distributed query paradigms on secured and unsecured (lacking the security layer) federated architectures

[8]https://gitlab.com/synchronicity-iot/synchronicitydatamarketplace
[9]https://business-api-ecosystem.readthedocs.io

Table 4.5: Message exchanged for each function

	Query	Initialize Subscription	Notification
Centralized	1	1	1
Federation	7	10	4
Secured Federation	⩾28	⩾40	⩾13

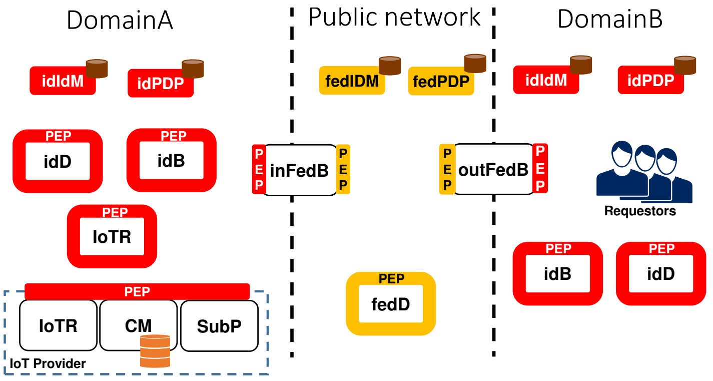


Figure 4.9: Testing architecture

compared with an unsecured publish-query paradigm (representing a centralized architecture). Fig. 4.9 illustrates the tested secure federated architecture comprehensive of the security layer (IdMs, PEPs and PDPs). For simplicity of experimentation, we omits the components not involved in the tests. Further, we deploy a centralized federation security system and federation discovery.

4.7.1 Publish-Query scenario

We test the scenario where requestors within domain$_\mathrm{B}$ request data residing into IoT provider(s) into domain$_\mathrm{A}$. For testing we use from the FIWARE framework, Orion as the Context Management (CM), Aeron IoT Broker as Broker, NEConfMan as Discovery, AuthZForce as PDP, KeyRock as IdM, and Wilma as PEP. In addition, we uses our prototype of IoT Registrar and Subscription Proxy. We use Apache JMeter to perform query requests for a random number of randomly chosen entities. The test is carried out varying number of total entities handled by the CM (100, 1000 and 10000) that represents the size of the IoT deployment, by considering each entity a thing. We vary also the number of concurrent requesting clients (threads) between 20 and 100. The number of attributes per stored entity and the number of attributes queried per entity is fixed to 100 and 20 respectively. We have taken 10000 as the top size of an IoT deployment taking into consideration the real Smart City of SmartSantander [95] where more than 20000 entities are handled. We ran the experiments on a single server with a 8 core Intel Xeon CPU E5620@2.40GHz and 24 GBs of RAM.

In Fig. 4.10 the latency experienced during tests is shown on the Y-axis against the number of queried entities on the X-axis, both in logarithm scale. The colours of the

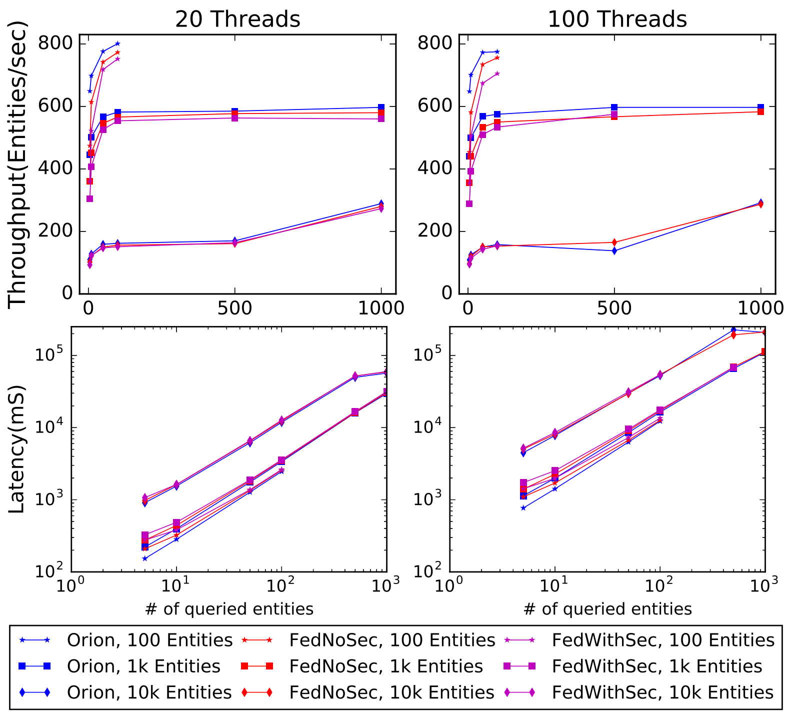


Figure 4.10: Latency and throughput for query scenarios

lines represent the architecture tested, while the different marker shapes represent the number of entities provided by the CM. For fair comparison, test point with a non-zero error rate are omitted from the plots. The throughput is normalized by considering the amount of information represented by entity contexts returned in each query response. For example, if the throughput of a testing scenario where each request queries 100 entities achieves 20 requests/sec, it is normalized to 2000 entities/sec. We can note that the different colour lines are getting closer in latency as the number of queried entities increases. This means that the overhead introduced by the federated architectures becomes increasingly negligible as the amount of requested data per query increases. In addition, as the dimension of the deployment increases (total entities up to 10000) the three lines are getting closer to each other indicating that the bottleneck becomes the IoT provider. The throughput presents similar behaviour, viz. the performances of the federated architectures get closer to those of simple IoT provider as the amount of information exchanged in the queries as well as the dimension of single platform increases.

We performed one more test to investigate the benefit of load balancing implicit in a distributed architecture. We compared the federated architectures (unsecured and secured) comprising 10 CMs, each handling 1000 entities, with a single unsecured centralized CM with 10000 entities. Each of the small CM handles disjoint set of entities whilst the big CM handles them all. We have then performed randomized queries, and, therefore, for each query one or more CMs need to be contacted by the Broker. The test results (Fig.4.11) show that a federated architecture performs much better than a

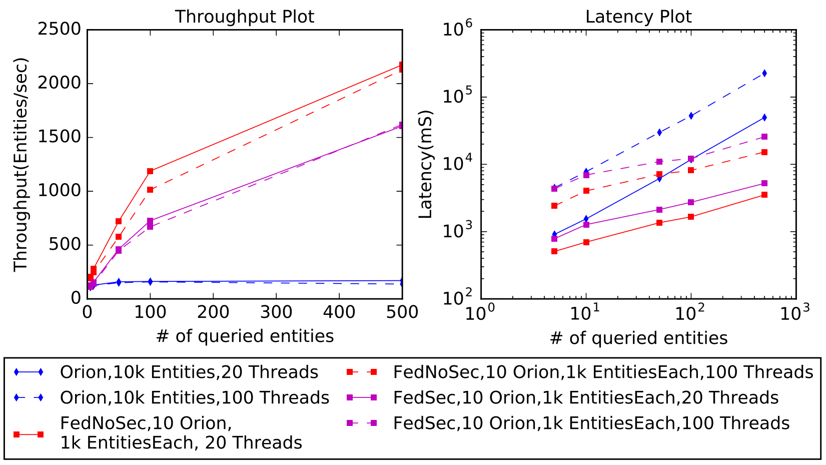


Figure 4.11: Results for single- and multi-provider scenarios

centralized approach, in terms of both throughput and latency.

4.7.2 Publish-Notify scenario

The evaluation method for the Publish-Subscribe paradigm is different from the Publish-Query case, since the data flow is asynchronous. To setup the evaluation we populate the tested system with a varying number of subscriptions for a varying number of entities. For the centralized scenario, the subscriptions were sent directly to the CM. For the federated scenario, the subscriptions (NGSI-10 subscribeContext) are issued to the idB_B for entities residing in $DomainA$, whose availabilities are already registered in the fedD component. The messages reach the correct IoT provider within $DomainA$ depending on the requested entities, and, thus, the subscription channels are established throughout the federation.

We use a *hit notification rate (HR)* as a system reliability metric. This indicates the ratio between the actual number of notification over the expected number of notifications:

$$HR = \frac{notificationsCount}{expectedNotifications} \tag{4.1}$$

where *notificationCount* is the actual number of notifications received on the notification server, while *expectedNotifications* is the expected number of notifications to be received if none of them is lost.

In order for *expectedNotifications* to be deterministic, the subscriptions are issued with a fixed number of total entities, such that the *number of subscriptions issued for each entity (NSE)* is:

$$NSE = \frac{totalNumberOfSubscriptions \times numberOfEntitiesEachSubscription}{numberOfEntities} \tag{4.2}$$

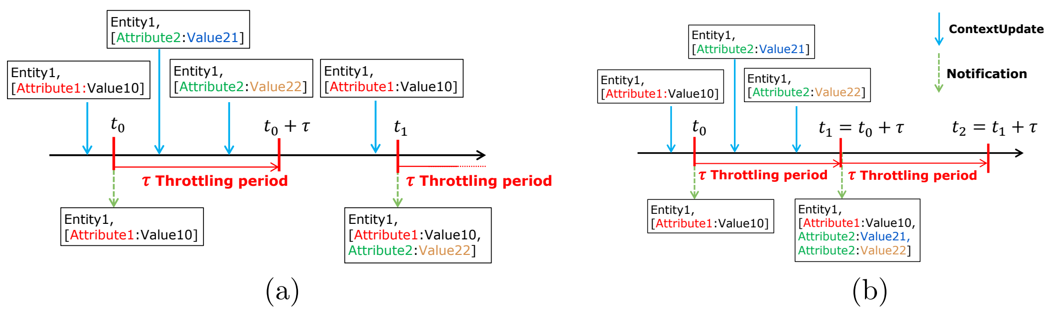

Entity1, [Attribute2:Value21]
Entity1, [Attribute1:Value10]
Entity1, [Attribute2:Value22]
Entity1, [Attribute1:Value10]
t_0
$t_0 + \tau$
t_1
τ Throttling period
τ Throttling period
Entity1, [Attribute1:Value10]
Entity1, [Attribute1:Value10, Attribute2:Value22]
(a)
Entity1, [Attribute2:Value21]
Entity1, [Attribute1:Value10]
Entity1, [Attribute2:Value22]
ContextUpdate
Notification
t_0
$t_1 = t_0 + \tau$
$t_2 = t_1 + \tau$
τ Throttling period
τ Throttling period
Entity1, [Attribute1:Value10]
Entity1, [Attribute1:Value10, Attribute2:Value21, Attribute2:Value22]
(b)

Figure 4.12: Throttling behaviour: a) discard approach, b) cumulative approach

The second step of the evaluation process is to generate data for the entities. We issue NGSI-10 updateContext requests for a single random entity at random intervals to the IoT Provider of Domain A in order to raise notifications. Each messages carry as attributes value the timestamp of the request. Being the whole system, composed of two domains, hosted on the same physical machine, we can compute the experienced latency by checking the notification arrival timestamp at the requestors within *domainB*.

Being NSE the number of subscription issued for each entity, the *expectedNotifications* value of HR (4.1) is:

$$expectedNotifications = NSE \times numberOfUpdates \tag{4.3}$$

for simplicity, we keep NSE as an integer.

A parameter to consider in the Publish-Subscribe scenario is the *throttling* of the subscription defined in the NGSI standard as the "proposed minimum interval between notifications" [58]. However, the NGSI standard does not specify exactly how the throttling should be implemented. Different implementation heavily influences the federation behaviour in the Publish-Subscribe case. Fig. 4.12 shows two approaches to throttling. The Orion CM implementation discards all the new data related to entities matching a subscription if the throttling time since the previous notification for such subscription is not passed. In addition, in the Orion CM implementation of notifications, only one single entity is contained in each notification, leading to many notification massages of very small dimensions. In facts, in case of a non-null throttling, the Orion CM sends less notifications badly affecting the HR, whereas, in case of a null throttling time, the federation is flooded by huge number of very small notification messages. This actual implementation penalizes the federation performance since, as also shown in the Publish-Query scenario, the impact of the overhead imposed by the federation layer decreases as the dimension of the messages exchanged increases.

To address this problem, for the Publish-Subscribe scenario we use the Aeron IoT Broker as a CM, instead of Orion. The Aeron component handles the throttling differently. It stocks all the updates received during the throttling period, and it sends a unique cumulative notification at the end of it. To benefit of the advantage of the cumulative throttling behaviour but also to affect as less as possible the latency of each notification, the throttling time during the tests is set to the minimum possible for Aeron

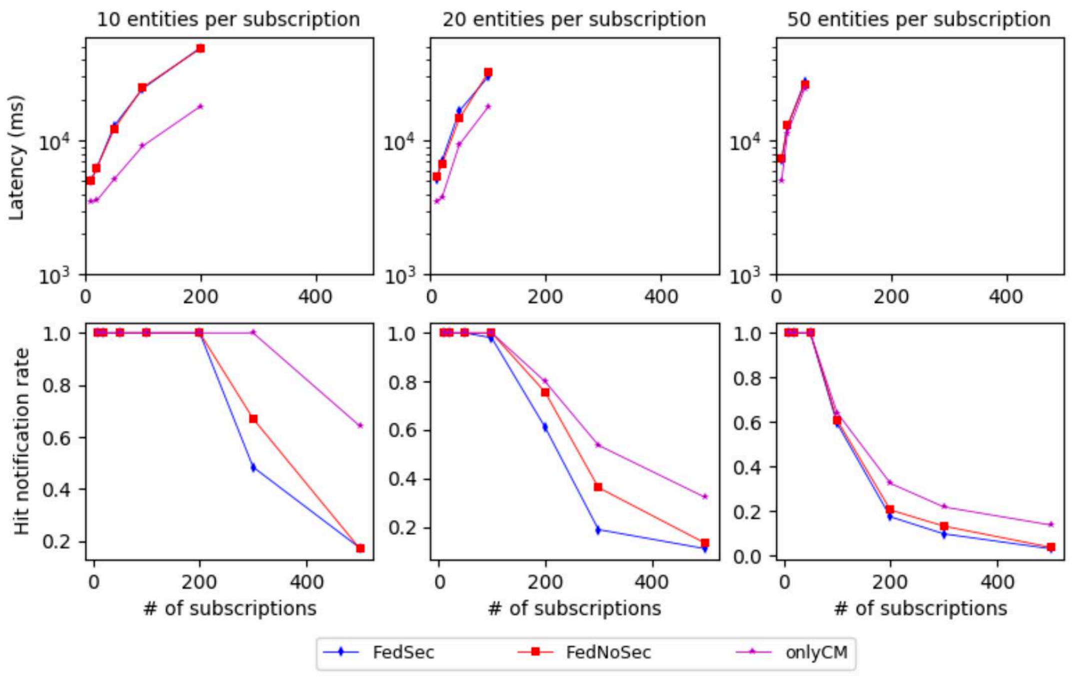


Figure 4.13: Publish-Subscribe Plots with 10, 20 and 50 entities for each subscription

IoT Broker(i.e. 100ms).

Figures 4.13 shows experienced latency and HR for different number of subscriptions and entities in each subscription.

As expected, the experienced latency in the federated scenarios is closer to that of the single Aeron CM when the number of entities for each subscription is higher. Indeed, for a given number of issued subscriptions, the increase of the number of entities for each subscription leads to bigger dimension of notifications flowing throughout the federation, while it does not affect the number of exchanged messages. Thus, the difference on the experienced latency into the federation compared with the centralized approach, on a log scale, decreases as the number of entities for each subscription increases. The same applies with respect to the HR, since the lines representing the federated architectures are closer to that of the centralized approach as the number of entities for each subscription increases. This behaviour is explained by the fact that the increase in the number of entities in each subscription mainly affects the CM component, which matches every received update with the corresponding relevant subscriptions and build bigger notification messages.

However, it slightly affects also all the IoT Brokers involved in the notification flow, since they have to match the incoming notifications with the corresponding existing subscriptions, build new notification messages and send them after the throttling time, potentially merging one or more inbound notifications into a single outbound one. This processing can also be affected by the dimension of the notifications, leading the performance difference of the federation with the centralized case to be lower for higher number of entities in each subscription, but never negligible. However, the overhead imposed by the security layer is extremely negligible, since the lines representing the secured federated architecture is always almost overlapping to that of the federation with no security layer.

Differently from the Publish-Query scenario, the performance of the federated architecture do not get closer to that of the centralized approach as the number of subscription increases. This is because the increase in the number of subscriptions (X-axis) causes a consistent rise of the number of notification flowing throughout the federation compared with the simple centralized CM case. Referring to the static evaluation of Tab. 4.5 (3rd column), if *numberOfSubscriptions* is the number of subscriptions issued, we have the total number of notifications flowing throughout the system as reported in Table 4.6.

Table 4.6: Message exchanged for each function

	Number of Notifications
Centralized	$numberOfSubscriptions$
Federation	$4 \times numberOfSubscriptions$
Secured Federation	$\geqslant 13 \times numberOfSubscriptions$

The increase in the number of subscription causes a much bigger load on the federated system leading to a faster degradation of the performance. This is very different from what happens in the Publish-Query case where the increase in the number of queried entities does not affect the number of messages but only their dimension.

4.8 Conclusions

In this chapter we have presented a distributed federation architecture scalable by design. Privacy and security are taken into consideration as they play a big role in the overall system. The evaluation shows that overhead introduced in big scale deployments is negligible and the federation approach is even better performing in multi provider scenarios. Since the adoption of standard-based communications protocol, our system will automatically inherit future advancements in the standard such as the semantics and linked data concepts.

Data usage control is the next step for a secured IoT data exchange and, in the next chapter, we start from the proposed architecture to design a distributed data usage control system.

Chapter 5

Data Usage Control

Data usage control is of utmost importance for federated data analytics across multiple business domains. However, the existing data usage control approaches are limited due to their complexity and inefficiency. This chapter proposes an *intent-oriented* data usage control system for federated data analytics, called *IntentKeeper*. The system allows users to specify intents for data usage policies and services easily. Thus, it reduces the data sharing complexity for data providers and consumers. Moreover, IntentKeeper enforces preventive and proactive data usage control for better security and efficiency through joint decisions based on policy enforcement and service orchestration. The use case validations for the automotive industry scenario show that IntentKeeper significantly reduces the complexity of policy specification (up to 75% for moderately complex scenarios) compared to the state-of-the-art flow-based approach. Lastly, the experimental results show that the IntentKeeper system provides sufficiently short response times (less than 40ms) with minimal overhead (less than 10ms).

5.1 IntentKeeper: Intent-oriented Data Usage Control for Federated Data Analytics

In the past decade, big data analytics has been often carried out by companies within their own individual centralized data infrastructure using existing open-source data processing frameworks such as Hadoop [110], Storm [111], Spark [112], and Flink [113]. However, in the new business domains like smart cities, industry 4.0, and eHealth, data is often generated and managed by different organizations and there is a strong demand for data sharing across different organizations' domains in order to create new businesses or improve the efficiency of existing ones. This demand triggers a fundamental change to the underlying data sharing infrastructure for supporting federated and trusted data analytics across domains. For example, Europe promotes a federated data infrastructure for joint businesses via two co-related initiatives, namely IDSA [16] and GAIA-X [14].

Due to the data protection and privacy regulation GDPR [114], one of the biggest open challenges to provide such a federated data sharing infrastructure is to enforce proper data usage control across different management domains regarding user-defined

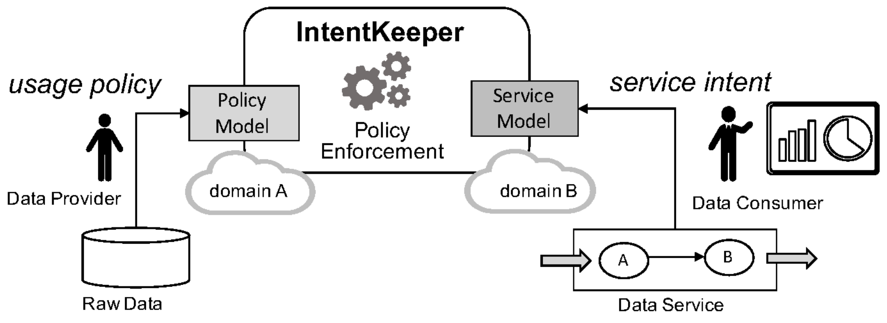


Figure 5.1: Basic idea behind IntentKeeper with three key elements: *Policy model*, *Service model*, and *Policy enforcement* to directly translate a service intent and relevant usage policies into the actual data processing flows

data usage control policies. Traditional data access control mechanisms (e.g., role-based [115] or attribute-based [116]) can limit data access rights by restricting who is allowed to access which data. Still, these mechanisms *do not provide data providers control to regulate how data consumers can utilize their data after having access.* To overcome such a limitation, data usage control [117, 118] has been proposed as a complementary extension to traditional data access control by enforcing what must or must not happen to data even after the data is shared. However, to realize data usage control for federated data analytics across domains, we face the following challenges.

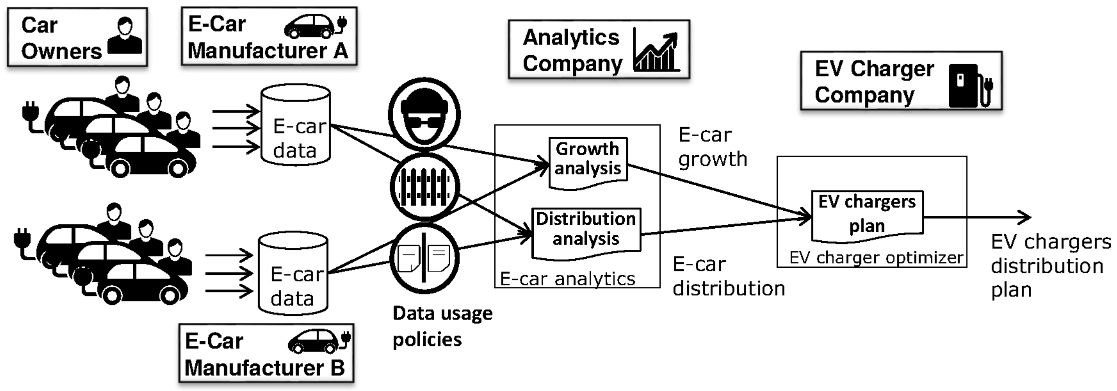


Figure 5.2: Automotive scenario

1) *Service orchestration*, representing the mechanism to orchestrate the data processing pipelines of upper layer data analytics services [3]. For traditional data analytics in a cluster environment, data providers and consumers are usually the same users from the same organization. However, for federated data analytics across multiple domains, data providers and consumers are often from different organizations. In order to enable trusted data usage across domains transparently, the processing pipelines of data usage services must be made clear to both consumers and providers for federated data analytics. Therefore, it is highly desirable to have a common service model to define such data usage pipelines on a higher level. With such a common service model, we can

outsource the complexity of programming data services from data providers/consumers to third-party service developers. Unfortunately, such a service model is still missing in state of the art.

2) *Policy modelling*, meaning how to express data usage policies from the perspective of data providers [119, 120]. Several policy models define data usage constraints flexibly and formally. Still, they generally represent the fine-grained low-level data usage constraints and usually require significant effort from data providers. For example, LUCON [121] requires its users to specify data usage constraints of a data processing pipeline per flow, which is feasible only when the logic of the data processing pipeline is pre-defined and static. As pointed out by the survey in [122], existing policy models are limited to express data usage control behaviors for dynamic data processing flows and also face the problem of balancing the trade-off between expressiveness and ease-of-use.

3) *Policy enforcement*, referring to the mechanism to continuously monitor and enforce pre-defined data usage constraints in the actual data usage process, which consists of three phases: before usage, ongoing usage, and after usage [123]. As reported in [122], most existing data usage enforcement mechanisms [90, 121, 124] can only support detective enforcement for static and pre-defined data processing flows, mainly due to their designs that fully decouple policy enforcement from service orchestration. They can monitor and check whether the orchestrated data processing flows are compliant with the defined usage policies after the data processing flows have been established; however, they cannot enforce data usage control preventively and proactively.

To overcome the limitations of existing data usage control approaches, in this chapter we introduce an intent-oriented design principle to ease the modeling of both data usage control and service orchestration and, then, present a new data usage control approach called *IntentKeeper*[1] to enable preventive data usage enforcement for federated data analytics in a proactive manner. As illustrated in Fig. 5.1, the high level idea of IntentKeeper is to take two types of inputs, *usage policy* from the data provider and *service intent* from data consumer, and then translates both inputs directly into a set of orchestration actions that can set up the data processing flows compliant with the defined data usage policies.

Overall, we make the following contributions in this chapter.

- We present a new intent-oriented approach of modelling both usage policy and service intent to reduce the complexity for data providers and data consumers. Its effectiveness and efficiency are validated via a detailed use case study. The validation result show that our approach can reduce the specification complexity by 75% of moderately complex scenarios.

- We present a new mechanism to enable preventive and proactive enforcement for data usage control by combining policy enforcement and service orchestration to make joint decisions. This new mechanism is realized in a decentralized way by leveraging permissioned blockchain to provide cross-domain synchronization and traceability.

[1]The name is inspired by ZooKeeper [125].

- We introduce the detailed design and implementation of the IntentKeeper system and also report its performance evaluation results. The experimental results show that the IntentKeeper system provides sufficiently short response times.

5.2 Background and Challenges

5.2.1 Data Usage Control

Data sharing between companies is a common practice in various industrial verticals, such as smart manufacturing, supply chain management, and automotive. In the past, data consumers and data providers/owners could negotiate agreements to allow sharing of data under certain limitations on its usage. Consumers had to respect the agreements after the data was shared. In the end, this approach relies on legal enforcement that reacts after the rules are broken. However, for the upcoming data-driven economy, the efficiency of such an approach is questionable because data providers simply have no technical mean to trace and enforce the actual usage of their data once their data is shared.

Recently, as an alternative approach, data usage control has become more and more important to enable efficient and trusted data sharing across multiple domains. However, it is facing new challenges as we start to scale up data analytics for more domains and better openness. Let us take a concrete scenario to analyze these challenges.

5.2.2 Automotive Scenario

The targeted use case is an automotive scenario in a urban environment (see Fig. 5.2). Electric cars from different manufacturers produce data of various sorts, such as GPS location, battery status, mileage counting, and speed. The produced data may contain personal data belonging to car owners. Often, car owners share data with their car manufacturer for the visualization of statistics, trends, and suggestions through a portal. The car owner, before sharing the data, might sign a pre-formatted “terms and conditions” contract with the manufacturer that states that the usage of their personal data might happen only after anonymization. Many organizations might be interested in using the data collected by the car manufactures in order to derive mobility insights. Let us assume that a market analytics company wishes to analyze the trends of electric cars distribution, usage and growth in urban environments. The trend analysis might be based, for instance, on statistical methods or on machine learning. This analysis is valuable for commercial purposes, e.g., for companies deploying electric vehicle (EV) charger stations. The target of such companies might be to minimize the number of installed stations (thus, minimizing the costs) and to maximize the quality of service for e-car owners by intelligently distributing the stations where the cars are usually parked for longer periods (e.g., near offices and residential areas) and where there is a higher concentration of EVs. The insight analysis of the analytics company might be very sensitive for car manufacturers since it might disclose internal market trends to the competitors. Furthermore, different car manufacturers might not want that their own data get merged (separation of duties), to avoid comparative studies. In the absence of consortium agreements within a vertical, such a scenario, that is beneficial for both urban and environmental sustainability, while also representing an interesting business opportunity, simply cannot take place.

5.2.3 Motivation

According to our analysis, the presented automotive scenario raises the following challenges that go beyond the capability of existing data usage control approaches.

Challenge 1: high complexity when dealing with dynamic data processing flows

The data processing flows of the required data analytics can change over time as data sources join and leave. For example, a car manufacturer manages the reported data streams from a large number of e-cars and might want to specify the same data usage policy for all of the car owners. In this case, the number of data processing flows to be enforced by a single data usage control policy will be changing over time as the e-cars join and leave. This is challenging for the existing flow-based approaches like LUCON [121] since they require the data processing flows to be defined in advance.

Challenge 2: policy enforcement to be considered not only "after", but also "during" the construction of data processing flows

Data processing flows are established to allow the permitted data usage for data consumers. However, in many cases it might be too late to consider the policy enforcement of data usage control after the establishment of the required data processing flows. For example, the car owners might allow their e-car data to be utilized by an analytics company for the e-car growth analysis only within the car manufacturer's domain and after their data are anonymized. In this case, a data anonymization function needs to be inserted since the beginning of the requested data processing flows. Also, the e-car growth analysis processing has to be deployed into the car manufacturer's domain. These two behaviors are part of the policy enforcement, but they have to be taken into account by service orchestration during the construction process of the requested data processing flows.

5.3 Intent-oriented Data Usage Control with IntentKeeper

To address the aforementioned challenges, we design an intent-oriented data usage control system called *IntentKeeper* with the following features:

- *Simple usage policy specification*: IntentKeeper simplifies the specification of usage control policies for data providers via an intent-oriented data usage policy model.
- *Preventive and proactive data usage control*: Service orchestration works together with policy enforcement to make joint decisions so that the orchestrated workload can already respect relevant data usage control policies.

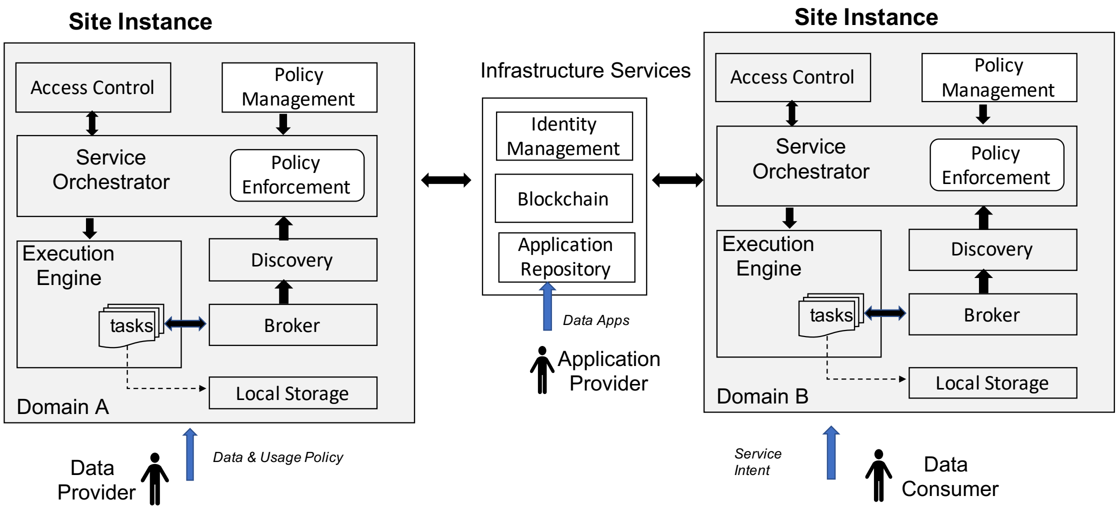


Figure 5.3: Overview of the IntentKeeper system

- *Decentralized policy enforcement*: Joint decisions between service orchestration and policy enforcement across domains are made in a decentralized manner for better scalability and reliability.

In this section, we introduce the major design aspects of IntentKeeper, starting with a system overview.

5.3.1 System Overview

As shown in Fig. 5.3, the IntentKeeper is a decentralized system that includes a few infrastructure services and a set of site instances, each of which is deployed and managed within a separated organization domain.

The infrastructure services include: **1) Identity Management** service that identifies and authenticates all site instances, each representing a certified domain user; **2) Blockchain** service that propagates, synchronizes, and records some important global information across all sites, such as user-defined data usage policies, the domain name of each site, and also the policy enforcement decisions made within each site; **3) Application Repository** that saves the docker images and metadata of all data applications developed and registered by application providers. The infrastructure services are the basis for the development of a marketplace for IntentKeeper. A marketplace matches data providers with data consumers, handling data exchange agreements and policies consequently. The state-of-the-art approach adopts blockchain technologies [126] due to the possibility of handling smart contracts for the generation of policies triggered by transactions. We embrace this approach and use the blockchain for both data availability and policies. In addition, immutable storage such as blockchain permits to have a log of all policies, services intents, and data availability. Thus, blockchain allows the traceability of data usage that is important for auditing the system's behavior and to trace which consumer uses which data.

Each site instance represents a specific domain and has the same set of system components as explained below.

- **Policy Management** provides a graphical user interface (GUI) for data providers to specify their data usage control policies and also maintain the user-defined data usage policies from all domains to keep them synchronized via the Blockchain service.
- **Access Control** module manages traditional data access control to enforce which user can access which data. In our design, we try to reduce the complexity for data providers to specify and manage all kinds of low-level access control rules, because these rules could be derived by policy enforcement on the fly during service orchestration according to the high-level data usage control policies.
- **Discovery** module is managing the availability of data entities with their registered metadata and also maintaining a domain-based routing table to enable cross-domain data exchange.
- **Broker** module manages all entity data published by data providers, and then establishes the data flows between different data services via a content-based pub-/sub interface.
- **Service Orchestrator** handles the high-level service requests issued by data consumers, and then translates them into low-level orchestration actions, which can establish a cross-domain data processing pipeline that is compliant with the data usage control policies of data providers.
- **Execution Engine** module carries out the actual orchestration actions generated by the service orchestrator, such as launching and configuring a dockerized task and establishing its input and output flows.

There exist three user types in IntentKeeper: *data provider*, *service provider*, and *data consumer*. Data providers use two interfaces, one for publishing their data and the other for specifying the data usage control policies associated with their data. Data consumers issue a service request to trigger a data processing pipeline that can utilize the data from data providers to perform data analytics. To simplify the complexity of defining and programming data analytics logics for both data providers and consumers, we outsource such complexity to application developers who can develop and publish all kinds of reusable cloud-native applications based on our service programming model described in Section 5.3.4.

5.3.2 Trust Management

The first issue for IntentKeeper is to define a trusted ground for the entire system and to manage its update and delegation in a decentralized environment. In the state-of-the-art, a few studies such as PrivacyGuard [127] and FairAccess [128] consider removing any central trust point using smart contracts in a blockchain system (e.g., HyperLedger, Ethereum), but mainly for enabling authentication or access control. In terms of data usage control, the same approach of using smart contracts to enforce all control points of the entire data usage control logic is limited due to its low efficiency and significant overhead. To have a more practical approach, in our design, we follow a *semi-trusted model*, where each domain holds a certified enforcement component that is deemed to be honest while carrying out the tasks it is designed for.

5.3.3 Policy Model

In our simple policy model design, when defining a policy, the data provider does not need to know the details of service description, service deployment, or underlying computing infrastructure. The policy specifies high-level targets (see Fig. 5.4) such as: *what:* data the policy covers; *who:* the data provider; *to whom:* the data consumer (it can be any); *purpose:* the data analytics service identifier (it can be any); *constraints:* a set of rules defining *actions* to be taken before using the data, when the processing is ongoing, and after processing the data. The first parameters (who, what, to whom, purpose) specify the targets to be matched to apply the constraints. IntentKeeper, then, enforces the constraints to prevent data misuse. The constraints' action may affect the service lifecycle altering the orchestration of service execution, or the data lifecycle (e.g., pre- and postprocessing) with an operator that processes data, which is called policy operator. Different policy operators are provided as atomic orchestration actions to enforce various macro behaviors of data usage control, such as anonymization, encryption before saving, delete after use. Since these policy operators are semantically annotated and ready for direct use, we can largely simplify what data providers have to define.

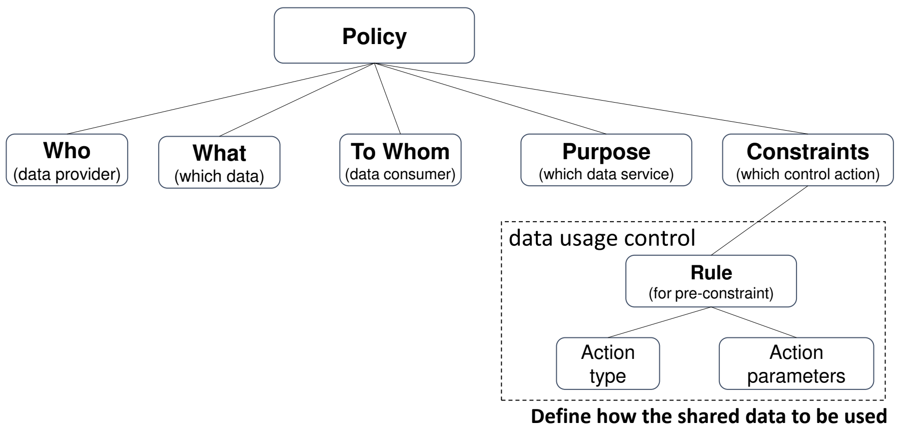


Figure 5.4: Model of intent-oriented usage policy

Table 5.1 shows examples of policies based on the automotive scenario we introduced earlier. The anonymize policy targets the car data to be preprocessed before being used. In this example, it does not matter who the car consumer is and for which purpose. The fenced policy targets the deployment orchestration of the service. That policy specifies that if the analytics company wants to use car data from the car manufacturer, the data will not leave the car manufacturer's infrastructure. On the other hand, the data provider will host the analytics execution locally with the data. The third policy defines the maximum time window of car data (time series) that can be used for e-car analytics. This time-to-live (TTL) example specifies the time window size parameter as 2 hours.

Table 5.1: Policy examples

Policy	Who	What	To Whom	Purpose	Constraint
Anonymize	Car manufacturer	car	any	any	preprocessing: anonymizer
Fenced	Car manufacturer	car	Analytics company	any	fencedData
Time-To-Live	Car manufacturer	car	any	e-car analytics	TTL: 2 hours

Our policy model can be considered as a simplification of the Open Digital Rights Language (ODRL) [129]. Our policy model can inherit the ODRL ontology and can be mapped with ODRL language as in Listing 5.1.

Listing 5.1: Anonymize policy encoded in ODRL

```
{"uid": "http://example.com/anonymize",
    "permission": [{
        ...,
        "target": "entityType:car",
        "action": "use",
        "duty": [{
           "action": [{
              "rdf:value": { "@id": "preprocess" },
              "refinement": [{
                 "leftOperand": "useFunction",
                 "operator": "eq",
                 "rightOperand": "anonymizer"}]}]
... }
```

5.3.4 Service Model

In the IntentKeeper system, a data service is represented by a service topology and a set of user-defined intents. The service topology is a graph of tasks, where each task performs some sort of data processing. Tasks in the same topology are linked with each other based on the dependency of their data inputs and outputs. Service designers annotate each task via a graphical editor, to define their input and output data and to define a granularity feature that determines how input data should be divided into task instances, for parallelization of computation. Each task instance runs within a docker container. By design, a service topology only defines the data processing logic of a data service.

To trigger the service topology, service consumers need to define an intent to express their high-level goals of using such service. More specifically, as illustrated by Figure 5.5, an intent can be customized to cover the following goals:

1. *Service topology* that defines which service logic to be triggered, and its identification name is matched with the purpose field of the policy;
2. *Scope* that defines the scope to select the input data for applying the selected service topology: locally into the domain, globally into the whole federation, or to a specific domain;
3. *Service level objective (SLO)* that defines latency requirements, bandwidth saving, or privacy/security needs;
4. *Priority* that defines how the triggered service deployment could utilize the shared computing resources with the other existing services.

With such an intent-based programming model, IntentKeeper can dynamically orchestrate concrete service deployment plans to meet any user-defined intents more flexibly, even for the same service topology.

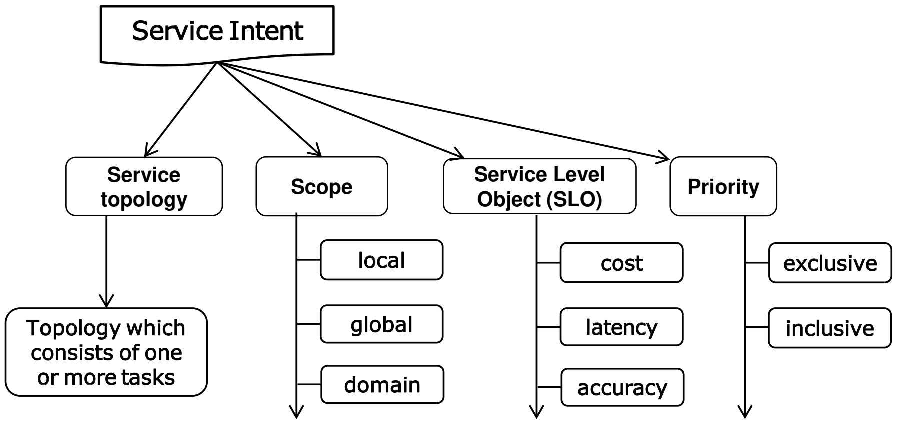


Figure 5.5: Model of service intent

5.3.5 Federated Service Orchestration

IntentKeeper orchestrates the service requests across the site instances to deploy services into computing nodes (see Fig. 5.6). A service intent triggers the orchestration process that is based on the data availability information stored in the discovery component. Discovery keeps the registry of all the IntentKeeper sites and the data available on each site. We design the discovery to have a partial registry to be shared between other discovery instances. The shared registry refers only to the data visible in the federation while the private part of the registry consists of data to be used only locally in the domain. Sharing the registry is a fundamental operation to enable a data marketplace where data consumers discover data.

The actual data follows a separate channel. Data producers push data to a broker component that dispatches data locally to the computing nodes and cross-domains to other IntentKeeper site instances. Computing nodes are isolated, and they communicate only through the broker. Also, the deployed service tasks cannot communicate with each other but must go through the broker layer. This design choice allows full visibility to IntentKeeper of the data exchange happening within the site instance and across instances. The following actions are designed to dynamically orchestrate tasks of the service, both in the local computing nodes and remotely.

- *ADD_TASK*: To launch a new task in a local computing node with the initial input streams. When launching a new task, the computing node fetches the Docker image for this task, and, then, launches it within a dedicated Docker container. After that, it subscribes the input entity to the broker on behalf of the running task by establishing a direct data channel ending into the running task. Finally, the newly created task is reported back to the orchestrator.
- *REMOVE_TASK*: To terminate an existing running task. The termination forces a stop and removal of the corresponding running container, as well as the dismantlement of the established data flows with unsubscribe messages.
- *ADD_INPUT*: To subscribe to a new input stream on behalf of a running task so that the new input stream can flow into the running task.
- *REMOVE_INPUT*: To unsubscribe from some existing input stream on behalf of a running task so that the task stops receiving entity updates from this input stream.
- *SEND_TASK*: To send a single-task intent to a service orchestrator within a different IntentKeeper instance. The expected outcome of this command is the migration of the orchestrations procedure to a remote site.

5.3.6 Policy Enforcement

IntentKeeper takes advantage of the federated data management and the service intent management to orchestrate the service deployment with policy enforcement. The Data Usage Control Policy Enforcement Point (DUC PEP) takes responsibility for this process. After a data consumer submits a service intent to IntentKeeper, the service orchestrator subscribes (see Alg.2) for data availability (line 4) of input data (line 3) for each function of the submitted service (line 2) . When the local data management notifies data availability for that service, the DUC PEP applies a policy control procedure to determine a set of atomic actions that execute the orchestration compliant with usage control.

Alg. 3 shows the policy enforcement logic. The discovery notifies the DUC PEP of the availability of a provider for data linked to an availability subscription and, therefore, unambiguously to a service intent. The DUC PEP retrieves all the matching policies owned by the provider targeting the notified data, the service, and the consumer. If

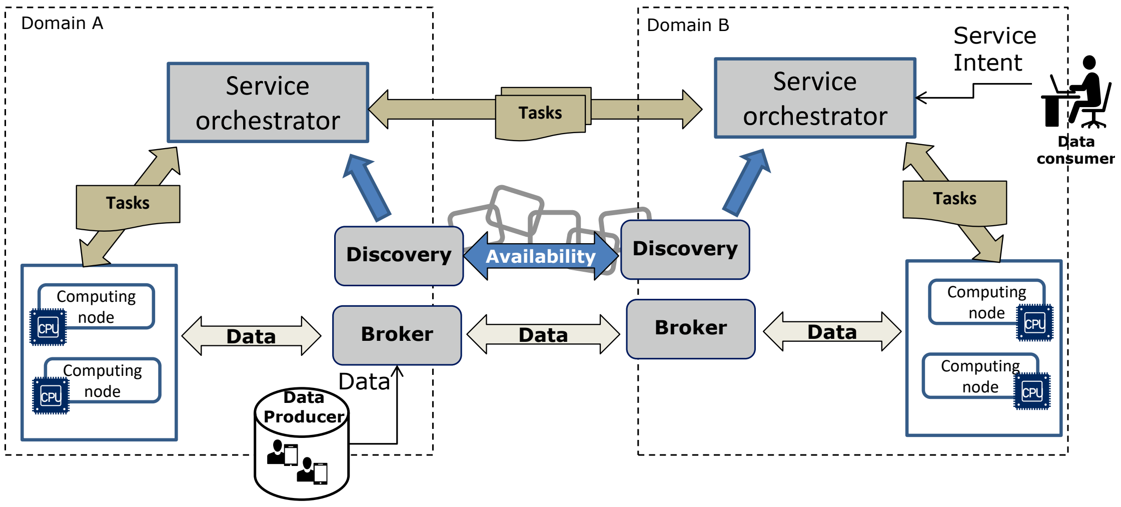


Figure 5.6: Federated service orchestration and data management.

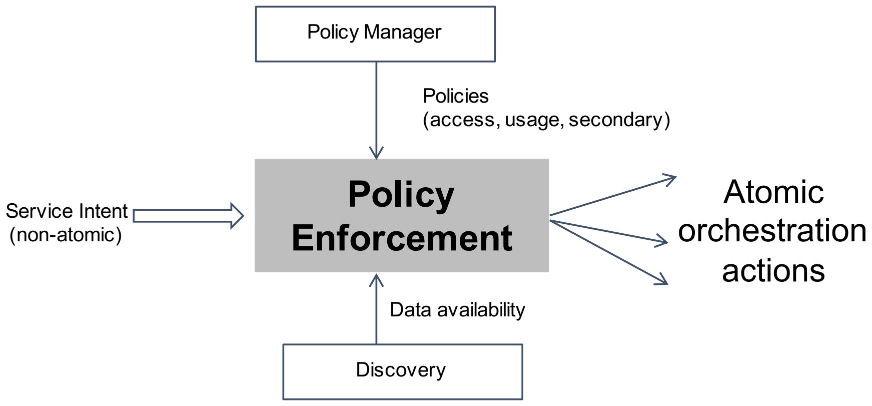


Figure 5.7: Usage control policy enforcements.

there are no matching policies, the procedure adopts a conservative approach and assumes that the data cannot be used by the consumer (lines 4-5). Alg. 3, then, checks the matched policies to determine the atomic actions such as prepending the preprocessing functions (lines 11-12), executing the input functions on the provider side (lines 13-14), generating a runtime routine (lines 15-17) that alters the function execution lifecycle. When all the service inputs are available (line 18), the orchestrator instantiates all the functions in local workers. The *prependFunction*(F, f_p) replaces every input of f_p found in the function set F with the output of f_p. Then, it adds the f_p to the set F. The *executeRemotely(F,input)* procedure sends all the functions having *input* contained in their input set to the data provider by calling the service orchestration function *SEND_TASK*. The procedure, then, subscribes for every output of the sent functions and removes the sent functions from the F set.

The anonymize policy in Table 5.1 expresses a preprocessing function. Once Algo-

Algorithm 2: ReceiveIntent(s)

```
Input  : $s \leftarrow (F = \{f_1, f_2, ..., f_n\}, consumer)$, set of
             functions submitted by $consumer$;
         $f_i = \{op, I_i, O_i\}$, operator, inputs, and
             outputs;
         $I_i = \{input_{i1}, input_{i2}, ..., input_{im}\}$;
Output: $M_{subId} = (s, SubID)$, map $s$ to subscriptions;
         $SubID = \{subId_{11}, ..., subId_{nm}\}$;
         $M_{input} = (s, I)$, map $s$ to required input set;
         $M_{routine} = (s, R)$, map $s$ to runtime routines;
1 begin
2     foreach $f_i \in F$ do
3         foreach $input_{ij} \in I_i$ do
4             $subId \leftarrow$ subscribe for availability of $input_{ij}$;
5             $SubID \leftarrow SubID \cup \{subId\}$;
6             $M_{subId} \leftarrow M_{subId} \cup \{s, subId\}$;
7             $M_{input} \leftarrow M_{input} \cup \{s, I_i\}$;
8     $M_{routine} \leftarrow M_{routine} \cup \{s, \emptyset\}$
```

rithm 3 retrieves the policy, the *prependFunction* selects the functions having car as input, and replaces the original input of each function with the anonymizer function output. The procedure adds anonymizer function to the functions set of the service. A similar approach can be followed for appending postprocessing functions. The fenced policy of Table 5.1 implies that the analytics company can use original data, but the data cannot leave the provider's cloud. Therefore, *executeRemotely* sends to the input provider an intent for each function that uses the input. The IntentKeeper residing in the provider site receives the intent and proceeds with Alg. 2 and 3. The federated data management, then, routes the output of the functions to the consumer site. Finally, DUC PEP translates the runtime constraints given by the TTL policy into a commands routine to be executed on the locally managed computing nodes. For instance, the TTL policy translates to periodic REMOVE_TASK-ADD_TASK instructions.

To handle data availability when the service is running, a conservative approach is to stop the service and analyze all the policies again, while, an incremental approach is to add inputs and instantiate new functions. Though the incremental approach can have a positive impact in a very dynamic scenario since the analytics never stops, it might result in inconsistent status or conflicts.

Algorithm 3: DataAvailabilityNotification($subId, input$)

```
Input  : $d$, local domain;
         $M_{subId} = (s, SubID)$, map $s$ to subscriptions;
         $SubID = \{subId_{11}, ..., subId_{nm}\}$;
         $M_{input} = (s, I)$, map $s$ to required input set;
         $M_{routine} = (s, R)$, map $s$ to runtime routines;
         $E = \{s_1, s_2, .., \}$, services in execution
Output: $M_{subId} = (s, SubID)$, map $s$ to subscriptions;
         $M_{input} = (s, I)$, map $s$ to required input set;
         $M_{routine} = (s, R)$, map $s$ to runtime routines;
1  begin
2      $s \leftarrow M.get(subId)$;
3      $P \leftarrow$ retrieve policies matching $input$, $d$, $s$, provider;
       /* check if policies allows usage */
4      if $P = \emptyset$ then
5          return
       /* remove $input$ from required inputs */
6      $M_{input} \leftarrow M_{input} - \{s, input\}$;
7      $F \leftarrow s.F$ /* set of functions of $s$ */
8      if $s \in E$ then
9          addInput($F$,$input$,$P$);
10         return;
       /* prepend preprocessing to $s$ */
11     foreach $p \in P$ with $f_p$ in preprocessing rule do
12         prependFunction($F$,$f_p$)
       /* is $input$ fenced? */
13     if $input$ may not be used within $d$ then
14         executeRemotely($F$,$input$);
       /* create runtime routines */
15     foreach $p \in P$ with a runtime rule do
16         $routine \leftarrow$ generate routine;
17         $M_{routine} \leftarrow M_{routine} \cup \{s, routine\}$;
       /* if inputs are available run $s$ */
18     if $M_{input}.get(s) = \emptyset$ then
19         foreach $f_i \in F$ do
20             ADD_TASK($f_i$);
21             $E \leftarrow E \cup s$;
22         foreach $routine \in M_{routine}.get(s)$ do
23             schedule $routine$;
```

5.4 Implementation

5.4.1 Implementation with FogFlow

We implemented the IntentKeeper system by extending our open source edge computing framework FogFlow [3]. The original FogFlow framework is designed to enable edge computing with a dataflow-based programming model. It provides a distributed data management layer and a centralized service orchestrator. In order to realize the design of IntentKeeper, we have the following changes.

First, we changed the distributed data management layer in FogFlow to support the decentralized discovery of entities across domains. The new discovery component implemented for IntentKeeper is able to share and synchronize a domain-based routing table with all the other domains via the Blockchain service. Moreover, it is able to save certain entities like user-defined data usage policies into the blockchain persistently.

Second, we changed FogFlow's service orchestrator to realize the policy enforcement logic proposed in Alg. 3 and also implemented a decentralized orchestration mechanism across multiple domains so that the service orchestrators inside different domains can collaborate with each other in response to a user-defined service intent.

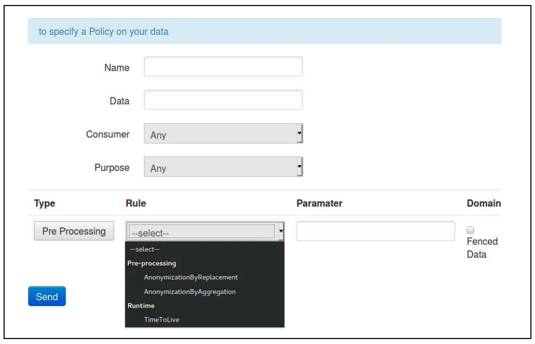
Lastly, we leveraged off-the-shelf software for the other missing components. More specifically, Docker Hub is used as the application repository, Hyperledger is deployed to provide the Blockchain service, and Vault is used to provide the identification management service for domain users and also the access control within each site.

5.4.2 Policy Editor

We implemented a GUI for adding policies to the system (see Fig. 5.8-a). The form fields easily map with our policy model. We further simplified the policy definition with the possibility to combine the fenced rule with data handling (preprocessing, runtime) rules by selecting a box. The GUI pre-fills the list of rules with the registered policy operators. When a user clicks on the *Send* button, the front-end sends the policy to the wrapper of the local policy manager that propagates it, through blockchain, to the other policy managers.

Fig. 5.8-b shows the registration form of a new operator to alter the data lifecycle of services. The fields of this form are:

- *operator* name used to pre-fill the policy editor rule list;
- *output* is used during the policy enforcement to alter the service topology (by the *prependFunction* procedure), and it might indicate a variation of the original input type (by prefixing or a suffixing) or an output type replacement;
- *image*, *tag*, *HardwareType* and *OSType* are technical properties of the Docker container that execution engine instantiates.


to specify a Policy on your data

Name

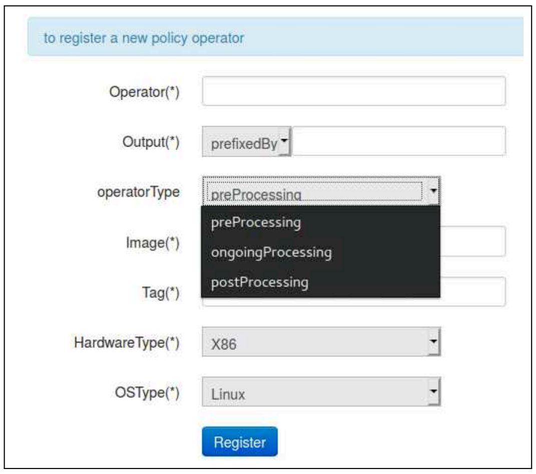
Data

Consumer: Any

Purpose: Any

Type	Rule	Paramater	Domain
Pre Processing	--select--		Fenced Data

(a)


to register a new policy operator

Operator(*)

Output(*): prefixedBy

operatorType: preProcessing

Image(*)

Tag(*)

HardwareType(*): X86

OSType(*): Linux

(b)

Figure 5.8: GUI for editing a policy (a) and to register a new policy operator (b)

5.4.3 Blockchain Integration

We implement the cross-domain synchronization between discovery components (for the IntentKeeper site registry and their provided data) and between policy management components using the Hyperledger fabric framework [109].

In every IntentKeeper site, a wrapper component, that interacts with the blockchain, intercepts every modification to the stored entries. Each site has a blockchain client that stores the ledger data locally and engages with the clients of the other sites for reaching consensus to alter the ledger. The ledger data is accessible with a key-value store interface implemented by CouchDB [109] allowing the insertion, modification and deletion of a key-value. The entries are stored by their unique identifiers (registration ID and policy ID). When the wrapper intercepts a modification to the policy set or the data registration set, it requests the blockchain client to validate the modification. The blockchain client, then, starts a consensus procedure with the other IntentKeeper sites to validate the modification. If the blockchain clients reach consensus on the modification, each of them sends a notification message to the local wrapper component. The notification encapsulates the modification method (e.g., “registerContext”, “registerPolicy”, “deleteRegistration”) with the original message body. The wrapper forwards the message body to the local policy management or discovery components depending on the modification method. Discovery and policy manager components receive the message transparently through their API.

A new IntentKeeper site may join the federation at any moment. When the site boots, the wrapper retrieves from the blockchain the current state of IntentKeeper sites set, registrations set, and policies set. The wrapper pushes the retrieved information to the local discovery and policy manager. Finally, the broker sends to the discovery a new site registration that blockchain propagates. At this point, the new site is ready to cooperate in the federation of IntentKeepers.

5.5 Use Case Validation

We validate the IntentKeeper system by applying the policy examples shown in Table 5.1 to cover three different use cases of the automotive scenario.

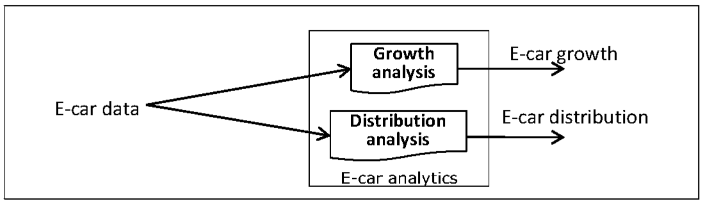
Use Case 1 — Anoymize Before Use: a privacy preserving agreement can be signed by the car owners before they publish their data to the car manufacturer's cloud. Such an agreement is realized by a data usage policy that constrains the car owner data to be anonymized before any use by anybody. This constraint specifies the name of the anonymizer function to be used as the preprocessing function. After the policy correctly propagates among the domains, the service orchestrator modifies every service that requests such data. The orchestrator prepends the anonymizer function before the other tasks of the triggered service topology.

Use Case 2 — Fenced Data: the car manufacturer sets the policy to enforce the original data to never leave provider premises. IntentKeeper orchestrates all the associated data analytics functions to be executed remotely within the car manufacturer domain. The data consumers can only receive the generated outputs of these functions.

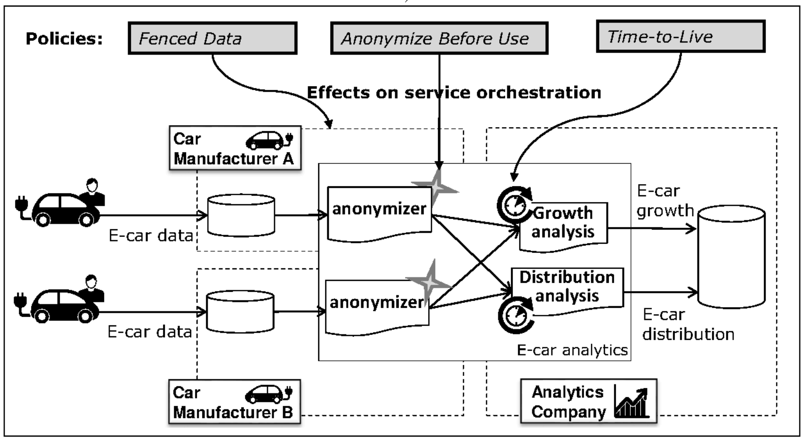
Use Case 3 — Time-to-Live: this policy aims to avoid profiling of users by forcing the deletion of EV data every two hours. IntentKeeper schedules a periodic routine that instructs the computing nodes to destruct (e.g., REMOVE_TASK) and re-deploy (e.g., ADD_TASK) the functions. Destructing a function instance consequently destructs all in-memory data of the executing function.

We modelled the *e-car analytics* service as depicted in Fig. 5.9-a. The two functions *growth analysis* and *distribution analysis* are of arbitrary nature, they might apply statistical functions or machine learning models. The execution of the atomic actions in the involved IntentKeeper site instances results into the deployment of two anonymizer preprocessing functions, one in each car manufacturer IntentKeeper, and the scheduling of a routine to destroy and re-deploy the e-car analytics service functions. Fig. 5.9-b illustrates the actual service deployment generated by the federated service orchestration with respect to the defined data usage policies.

Table 5.2 compares our intent-oriented usage control system with a flow-oriented one (e.g., LUCON [121]). We analyze the number of elements that users (data owner, data provider and data consumer) need to configure in each system in order to enforce the usage control described in Fig. 5.9, including policies, functions, and links between functions. For the flow-oriented approach, the number of functions and links increases with the number of data providers and the number of involved policies, while, there is no such effect for our intent-oriented approach. The number of flow-oriented policies may vary as these may refer to a single endpoint (viz., one policy per analytics function) or to a regular expression (viz., one policy for both functions). While IntentKeeper transparently manages to meet the compliance of the data consumer intent with data owner and data provider policies, a flow-oriented approach controls only the communications channels by dropping or admitting a message. The flow-oriented approach burdens the data consumers with the deployment of a service graph, composed of functions and links, that respects the policies, and, in some cases, it burdens also the data providers with service deployment (e.g., the fenced policy constrains to execute functions within the provider



a)



b)

Figure 5.9: Effects of the policies on the service orchestration. The three involved IntentKeeper sites (car manufacturer A and B, and analytics company) collaborate to modify the original service topology (a) by preprending functions that they deploy within car manufacturers' premises while scheduling to periodically destroy and re-deploy the analytics functions (b).

premises). We consider an automotive scenario having three policies, two data providers and two data consumers (for the same service). This scenario is moderately complex compared to a simple scenario which includes only one single policy and single provider. When using a flow-oriented approach in this use case, the users need to configure up to 44 elements, whereas when using our intent-oriented approach the elements to configure are 11. Overall, our approach reduces the complexity up to 75%.

Finally, in the flow-oriented approach the data providers need to be aware of all data consuming service instances and create a set of policies for each of them. The last two rows of Table 5.2 show the case of two different data consumers deploying two e-car analytics service. While the flow-oriented approach doubles the configuration effort, for our intent-oriented usage control the same service descriptions and policies are valid.

Fig. 5.10, then, scales up the scenario complexity of the Table 5.2 analytically. The graphs report the number of elements to configure (policies, functions, and links) when increasing the number of data providers (up to 20) and the number of e-car analytics service (Fig. 5.9.a) instances. For our approach, the number of policies is the only value increasing with the number of data providers since every data provider needs to set their own policies (three in our scenario), whereas all the other elements are constant. For the flow-oriented approach, all the elements are always directly increasing with the scenarios' complexity.

Table 5.2: Operational configurations needed for our intent-oriented usage control system compared with a flow-oriented usage control system [121]

	data providers	**Policies**		**Functions**		**Links**	
		Intent Keeper	LUCON	Intent Keeper	LUCON	Intent Keeper	LUCON
Anonymize	1	1	1-2	3	3	2	5
	2	2	2-4	3	4	2	8
Fenced	1	1	1-2	2	2	2	4
	2	2	2-4	2	4	2	8
TTL	1	1	1-2	2	3	2	5
	2	2	2-4	2	4	2	8
Anonymize + fenced + TTL	1	3	2-3	3	4	2	6
	2	6	4-6	3	6	2	10
Anon.+fen.+TTL; 2 services	1	3	4-6	3	8	2	12
	2	6	8-12	3	12	2	20

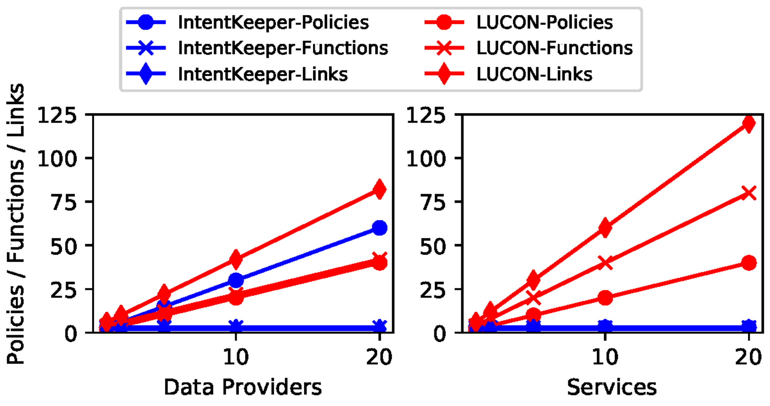


Figure 5.10: Analytical study of the complexity for the automotive scenario varying number of involved data providers and number of triggered services.

5.6 Performance Evaluation

We evaluate the impact of using IntentKeeper for executing an analytics service. IntentKeeper might introduce overhead with the orchestration of data analytics on federated environments, the enforcement of data usage control, and management of federated data flows at service runtime. We evaluate IntentKeeper by assessing the different delays that might affect the service execution. We observe end-to-end delays which can be due to the data propagation through the federated data management, the policies evaluation by the service orchestrator, the service instantiation by the computing nodes, and the traversing of the data through the pipeline in federated analytics configuration. Further, we assess the impact of the blockchain on the overall performance for the decentralized approach. The evaluation scenario includes two domains, domain A and domain B.

There are two actors, a provider of car data and a data consumer requesting the e-car analytics service. We test the impact of having the data consumer in domain B while the data provider is in domain A compared to the case of both data consumer and provider being in the same domain (A).

We use two identical 8 core Intel Xeon CPU E5620@2.40GHz machines with 24 GBs of RAM each. We used the Docker framework v.18.09.0, and dockerized Hyperledger fabric v.1.4.7. The first machine hosts the two IntentKeeper site instances, one per domain, the second machine hosts either the blockchain for the decentralized approach or the centralized components (i.e., policy manager and discovery) depending on the experiment. With this setup, the communication for data availability and policy between site instances always goes through the second machine.

The results highlight that the overhead of the policy evaluation and federated orchestration of the intents is negligible. Blockchain brings the major part of latency (up to 10 seconds of latency to launch a federated analytics service). We demonstrate that once the system launches the service, a data point needs less than 0,01s to traverse a federated analytics pipeline.

5.6.1 Propagation Delay

First, we study the delay introduced to propagate control information, such as policies, data availability registrations, service descriptions, and service intents. We also analyze two configurations: i) centralized policy manager and discovery; ii) a policy manager and a discovery within each site instances synchronized through blockchain.

Table 5.3: Propagation delay (ms) of different types of messages for two configurations of policy manager and discovery components: centralized and decentralized with blockchain

	Registrations	**Data Pushes**	**Policies**
Centralized	1.73	4.25	2.05
Decentralized	2476.59	2365.57	2578.66

Table 5.3 shows the propagation delay (ms) for data availability registrations, data pushes, and policies messages. Each experiment lasts two minutes, during which 1 request per second is sent. We measure the delay from the time a message is sent from within domain A to the time a message is notified to a subscriber in domain B. Decentralized configuration shows higher latency due to the blockchain ($\sim$2.4 sec) compared to the centralized approach (less than 5 ms).

Moreover, we observe that high workload heavily affects blockchain performance. Fig. 5.11 shows the performance on logarithmic scale of the federated system when we increase the number of concurrent clients, from 1 to 20, sending registrations to the discovery. The lines in Fig. 5.11 are clustered between centralized and decentralized approaches. The decentralized system with blockchain responds with higher delays that worsen when increasing the number of concurrent clients. The latency increases to a critical situation (up to 2 min delay) when there are 10 or 20 concurrent clients.

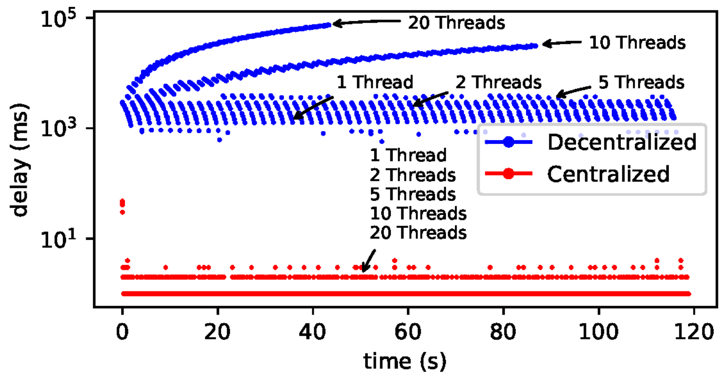


Figure 5.11: Propagation delay, through experimentation time, for data availability registrations through centralized discovery and decentralize (with blockchain) for varying number of concurrent clients.

5.6.2 Service Orchestration Delay

Second, we evaluate the service orchestration delay introduced by the usage policy control. The goal is to send a service intent and measure the delay till the service is running. We implement the e-car analytics service and test the system with a policy that constrains the data usage after the preprocessing for anonymization and a policy that forces the original data processing to happen inside the data provider domain.

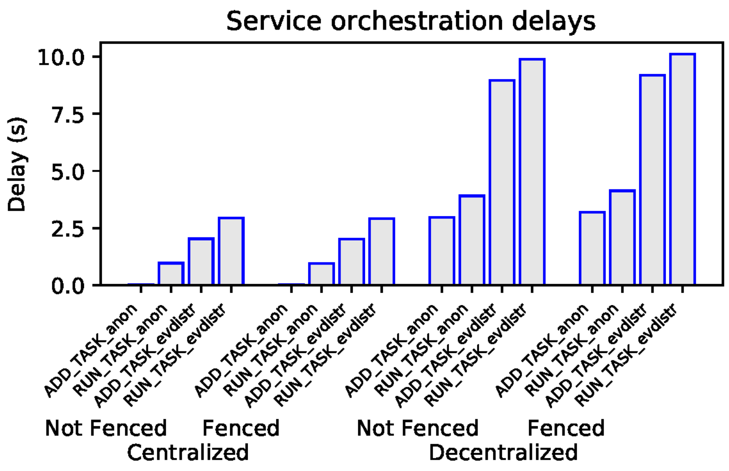


Figure 5.12: Service orchestration delay: From the submitted service intent by the data consumer till service runtime.

Fig. 5.12 shows the service execution delay for the case of preprocessing policy combined with the fenced policy. We pinpoint four time instants: when IntentKeeper sends the ADD_TASK command to the computing node to instantiate the preprocessing task, when the preprocessing task is running, when IntentKeeper sends ADD_TASK for the e-car analytics functions, and when the whole service is running. IntentKeeper of domain B needs less than 0.04s to receive the service intent, retrieve the policies, evaluate the policies, decide and apply atomic actions, such as modifying the service topology to include the preprocessing function, and sending the SEND_TASK remotely (in presence, also, of fenced policy) to domain A. The local computing node takes ∼1s to boot up a docker container. The data consumer IntentKeeper site instance deploys the rest of the analytics service after the first anonymized data point is available in the system. The difference between a single node analytics (not fenced) and a federated analytics (fenced) is negligible. The higher latency values for the decentralized system are due to the latency delay introduced by blockchain.

Table 5.4: Service orchestration delays (in seconds) for different policies and for data consumer residing in the data provider domain (A) or another domain (B).

System	**Centralized**						**Decentralized**					
Policies	**No Rule**		**Not Fenced**		**Fenced**		**No Rule**		**Not Fenced**		**Fenced**	
Consumer Domain	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
Policies	**No Rule**		**Not Fenced**		**Fenced**		**No Rule**		**Not Fenced**		**Fenced**	
Consumer Domain	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
ADD_TASK_anon	-	-	-	0.03	-	0.04	-	-	-	2.97	-	3.19
RUN_TASK_anon	-	-	-	0.97	-	0.95	-	-	-	3.91	-	4.13
ADD_TASK_evdistr	0.03	0.03	0.03	2.03	0.03	2.02	3.69	3.47	3.54	8.97	2.92	9.18
RUN_TASK_evdistr	0.97	0.98	0.96	2.94	0.97	2.92	4.63	4.42	4.49	9.88	3.88	10.11

Table 5.4 reports all the experimental results considering also the case of a policy allowing data to be used without preprocessing (no rule), and the case of both data provider and data consumer residing on the same domain A such that the policies have no effect. The experiments are repeated 5 times. Fig. 5.12 and Table 5.4 contain the averaged results.

5.6.3 Service Latency

Lastly, we observe the service latency. In this experiment, after the system runs the service, we evaluate the performance of IntentKeeper for streaming federated analytics. We measure the time a data point needs to traverse the established service pipeline, both in case of a single node environment and federated environment (two IntentKeeper site instances). We repeat each experiment 10 times.

Fig. 5.13 shows that the latency introduced by the federated configuration for data analytics is negligible (less than 0.4s). The latency is considerably higher (∼7s) when the control information flow is decentralized. This happens when the data entering the pipeline refers to a yet unseen device such as a new car. In this case, the new car needs to be first announced through the discovery, thus, the blockchain. When a data point refers to a device already seen by the pipeline, the latency is negligible as shown by the last three measurements in Fig. 5.13 (less than 10ms). When not affected by the blockchain, IntentKeeper's performance is inline with the NGINX guideline for real time web service [130].

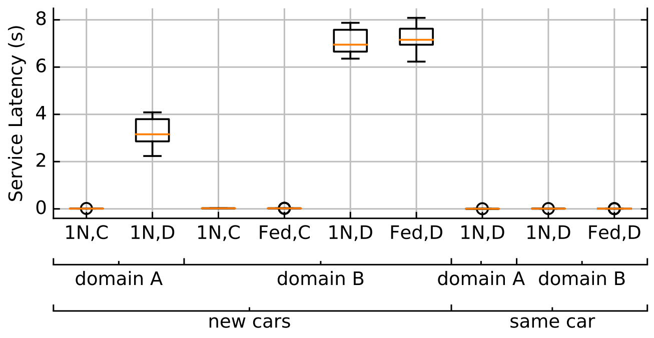


Figure 5.13: Latency of the data processing pipeline after being established: Centralized (C) and Decentralized with blockchain (D) approaches; single node (1N) and federated (Fed) analytics. We push data either always from new cars or always from the same car.

5.7 Related Work

5.7.1 Federated Data Analytics

In terms of big data analytics, there has been a new trend of moving from single-domain based data analytics to federated data analytics across multiple domains. Initially, Hadoop was the dominating framework to do scalable data analytics within a cluster. Later, Apache Spark and Flink became more popular due to their capability of enabling efficient data processing for both historical data and stream data. Recently, as more data is generated and managed by different business partners inside different management domains, federated data analytics is becoming an attractive approach for service providers to leverage raw data across different data silos. The most common example of federated analytics is federated learning [131, 132], which can train an AI model over decentralized datasets by performing the model training locally on each data set. The data usage pattern of transfer learning is similar to the fenced data supported by IntentKeeper. According to the International Data Space Association (IDSA) report [16], although federated analytics is highly desirable by enterprises, its adoption is still limited due to the shortage of data usage control and common service model.

5.7.2 Data Usage Control

In the state-of-the-art, a few generic low-level policy models are proposed for data usage control. For example, UCON$_{ABC}$ [133] has a data usage model to restrict the usage decisions between subjects and objects in terms of authorizations, obligations, and conditions, which can be further defined with subjects and objects' attributes. An extension of UCON is proposed in [134]. LUCON [121] is a policy language for controlling data flows between endpoints. However, users need to write detailed data usage control rules per flow. The Open Digital Rights Language (ODRL) [129] is a policy expression language that provides a flexible and interoperable information model, vocabulary, and encoding mechanisms for representing statements about the usage of content and services. Overall, those policy models have the difficulty to express data usage control policies for dynamic data processing flows. As compared to those generic models, our intent-oriented policy model is more straightforward for data providers to define, since it can refer to the same service model as data consumers to specify data usage logic and directly use high-level policy operators to realize usage control behaviors.

A few systems provide data usage control in distributed environments. Kelbert [135] introduces a data usage control mechanism for distributed systems based on cross-system data flow tracking and policy propagation. This mechanism does not provide any service model, and it needs to hook system calls to monitor/control data usage. Therefore, it cannot influence data usage pipelines for making them compliant with user-defined usage policies. MYDATA [136] enables fine-grained masking and filtering of data flow based on various plugins. Recently, a FIWARE-based data usage control solution is introduced by Alonso et al. [90]. In all these systems, the orchestration of data usage services is handled before policy enforcement in a separated process. Therefore, they can only monitor the generated data usage flows, and, then, decide to terminate them or not in a reactive manner. Unlike these systems, IntentKeeper can make joint decisions between service orchestration and policy enforcement to generate data usage flows that are already compliant with the defined usage control policies. Furthermore, IntentKeeper enables preventive data usage control in a proactive manner, which provides better security and higher efficiency.

5.8 Conclusions and Future Work

This chapter presents a novel data usage control system, called IntentKeeper, to support federated and trusted data analytics across multiple organization domains. IntentKeeper takes an intent-oriented approach to model both usage policies and data services, thereby reducing the complexity for both data providers and data consumers. By combining service orchestration and policy enforcement to make joint decisions, it enforces data usage control preventively and proactively for better security and efficiency. We introduce the detailed design and implementation of IntentKeeper and also report its use case validation and performance evaluation results.

In the future, we plan to improve the adoption of IntentKeeper by the following: 1) providing a list of policy templates and adding more policy operators for more usage control patterns, 2) taking the ownership management of processed data into account, 3) expressing and enforcing usage control policies of the processed data.

Chapter 6

IoT data services ecosystem

Smart city solutions are often monolithically implemented from sensors data handling through to the provided services. The same challenges are regularly faced by different developers, for every new solution in a new city. This is not sustainable in the mid- and long-term. Instead, expertise and know-how can be reused and the effort shared when embracing the hyperconnected IoT vision, where the services are decoupled from the underlying IoT infrastructure and they rely on common interfaces and data models. In this chapter, we present the SynchroniCity EU project's experience of coordinating the efforts to build 35 smart city solutions in 27 cities between Europe and South Korea by maximizing the sharing of components. The final target is to have a live technical community of smart city application developers. The results of this activity comes from the involvement of tens of heterogeneous stakeholders among academia, industry, small and medium enterprises (SMEs), and cities' governance. All the involved smart cities expose the same IoT interfaces and data models relying on the FIWARE-based hyperconnected IoT implementation by SynchroniCity (see chapter 3). To share efforts, we encourage developers to devise applications using a modular approach. Single-function components that are re-usable by other city services are packaged and published as standalone components, named *Atomic Services*. We identify 15 atomic services addressing smart city challenges in data analytics, data evaluation, data integration, data validation, and visualization. 38 instances of the atomic services are already operational in several smart city pilots. We detail in this chapter, as atomic service examples, data predictor components. Furthermore, we describe real world atomic services usage in the scenarios of Santander (Spain) and Vejle, Aarhus and Odense (Denmark).

6.1 SynchroniCity and the shared ecosystem

Internet-of-Things (IoT) benefits in the smart city scenario is extensively acknowledged by countless real application deployments [137]. The developments of these applications usually happen as standalone activities resulting in city-wide IoT segments fragmentation, or even domain-wide within the same city [138]. The SynchroniCity project aims to "synchronize" [5] IoT infrastructures among cities in order to overcome vendor lock-in, by adopting open solutions, and city lock-in, by adopting common interfaces [81]. The approach of SynchroniCity is to deploy an overlay on top of existing smart city infrastructures. The SynchroniCity overlay is based on FIWARE, an IoT framework of open source components implementing open API and standards. This gives the ground to city services developers to develop applications relying on standardized interfaces and data models. The advantage is porting IoT solutions from an environment to another with minimal effort, thus, enabling an IoT services market ecosystem.

The initial cities, named Reference Zones (RZs), are: Antwerp, Carouge, Eindhoven, Helsinki, Manchester, Milan, Porto, and Santander. Those cities homogenize their IoT data with the same data models [27] and formats. This data are exposed with standard interfaces for context management (Next Generation Service Interface - NGSI [58]), historical time-series, and open data. Using these common interfaces, service developers implemented several smart city applications addressing use cases defined by local municipalities [12].

This chapter reports the collaboration activities among the applications developers (including academia, SMEs and industries). The approach is to build the applications by composing small services, namely *atomic services*, each implementing a single functional block, towards Service-Oriented Architecture (SOA) [139]. The atomic services are exchanged between city application developers and published in a one-stop-shop repository[1] following a one-page documentation scheme. The advantage of this approach is multi-fold: 1) small companies can leverage others' know-how to speed-up applications implementation, breaching the barrier of monolithic vertical developments only sustainable for big companies [140]; 2) it enables an IoT services market [141, 142]; 3) operational smart city applications can quickly adapt to IoT evolution by having the atomic services up to date or integrating new ones.

In this chapter we go through the process of giving rise to 15 atomic services during 22 months (from Jan 2018 till Oct 2019) in parallel with the implementation of 35 smart city services. A bootstrap phase comprises a static analysis of use case requirements from RZs [8] and of available off-the-shelf services. During a second phase, 12 RZs' city services architectures are analyzed to identify common challenges and building blocks. The still ongoing third phase gives the developers the possibility to spontaneously offer their IoT services as atomic services. The latter phase involves 23 city services piloted in 27 cities. In one case an application team, rather than implementing an atomic service, leveraged the know-how of the developer of situation prediction atomic services (i.e., parking and traffic flow estimator) by requesting a new one for outdoor noise estimation. This shows the marketability of developers' know-how and IoT services. In other cases, i.e. for 3 cities in Denmark (Aarhus, Vejle and Odense), the proposed atomic services aim to keep pace of the IoT evolution (e.g., the new NGSI-LD standard [89]) and to smooth integration for escaping city lock-in. These atomic services address data integration [143], data validation, and data visualization [144]. This chapter will show:

- **Analysis of cities commonalities**. We evaluate data input and targeting challenges of 12 services in 9 cities. The results motivate us to create a smart city

[1] https://gitlab.com/synchronicity-iot?filter=atomic+service

technical community.

- **Design a collaborative approach** for starting and maintaining the community alive.
- **Atomic services examples and usage in real scenario**. We describe parking/traffic/noise estimators service as examples of atomic services. In addition, we report the experience of the real city scenarios and how they benefited by using atomic services: implement a new multi-modal transportation city service in Santander; bring legacy city services of Vejle, Aarhus and Odense to open market.
- **Evaluate the atomic service approach** through a validation process and the assessment of the community engagement.

6.2 Related Work

Building interoperable services on top of IoT systems is an open and timely research challenge, as demonstrated by the presence of recent works in the literature. Experimentation-as-a-Service (EaaS) [92] is a paradigm to execute processing routines over a centralized platform that offers data. Re-usable and portable experiments process data regardless its origin. The architecture presented in [92] has the technical potential to offer experiments as re-usable services but this is not explored by the authors. Several aspects are already tackled to enable IoT services ecosystem, such as architectures [145, 146] and procedures to acquire IoT services [146], or how to generate IoT services business model [147, 148]. The authors of [141] present a proof-of-concept of IoT services marketplace. None of the mentioned works show to be embodied in reality. The reason is that a community of such kind is not easy to self-blossom but needs to be guided.

Big cloud providers, such as Amazon Web Services [149] and Microsoft Azure [150], offer service marketplaces, but mainly for industrial IoT projects. This is due to the reluctance from city governance to fall in vendor lock-in trap [151, 5]. FIWARE, instead, is a growing open alternative to proprietary platforms [5]. In particular, FIWARE domain-specific enablers (DSEs) [152] are a collection of IoT applications and services for different domains, such as manufacturing, media, e-health, agrifood or energy. The catalogue is formed by re-usable components, similar to our atomic services, and monolithic applications. The FIWARE DSEs methodology is to simply share applications' software built upon the FIWARE framework. However, FIWARE DSEs lacks: a) a methodology to systematically identify re-usable components as services, b) a large number of IoT services and involved parties, c) the attempt to keep the community alive. Hence, our work is complementary to the FIWARE DSEs.

6.3 Smart city services

The SynchroniCity project brings together several stakeholders with the aim of building a technical smart city community. In the following section we go through the work

Table 6.1: SynchroniCity smart city services

Smart City Theme		City Service Topic	N. of services	Involved Cities	N. of pilots
Mobility	Encouraging non-motorized transport	insightful (clean air, crowdsourced, safe) bicycle path recommendation; secure bike parking; stolen bike recovery; electric bike usage monitoring	8	Eindhoven (NL), Milan (IT), Antwerp (BE), Santander, La Nucía(ES), Manchester (UK), Dublin, Donegal (IE), Faro (PT), Helsinki (FI), Aarhus (DK)	18
	Multi-modal transportation	commuter assistant; park & ride; public transportation usage maximization; zero emission journey planner; barrier-free planner for disabled people;	6	Porto (PT), Santander (ES), Helsinki (FI), Milan (IT), Carouge (CH), Seongnam (KR)	6
	Enabling Mobility as a Service (MaaS)	adaptive lighting; traffic optimization; bus stops crowd and air monitoring; smart parking	5	Porto (PT), Santander, Torrelavega (ES), Milan (IT), Antwerp (BE), Seongnam (KR)	10
Sustainability	Climate Change Adaptation	green roof management; building energy management	3	Carouge (CH), Milan (IT), Eindhoven (NL), Porto (PT), Antwerp (BE), Vejle, Odense (DK)	8
	Reducing Air and Noise Pollution	indoor/outdoor air quality management; clean air around schools; noise pollution planning; urbanization impact monitoring	5	Helsinki, Tampere (FI), Santander, Bilbao, Onda (ES), Antwerp (BE), Carouge (CH), Edinburgh (UK), Novi Sad (RS), Eindhoven (NL)	13
	Waste Management	waste collection optimization; waste collection monitoring	2	Porto (PT), Carouge (CH), Catalayud (ES), Aarhus (DK)	4
Governance	Community Policy Suite	agile governance; data visualization; public spaces air and noise monitoring; insights for cycling infrastructure; smart city business intelligence; disable people accessibility monitoring; traffic insights; environment monitoring; bus stops crowd and air monitoring; elderly care service monitoring	13	Manchester, Edinburgh(UK), Porto (PT), Carouge (CH), Cabildo de la Palma, Cartagena, Santander, Bilbao, Torrelaveda, Onda (ES), Eindhoven (NL), Antwerp (BE), Milan (IT), Helsinki(FI), Aarhus, Vejle, Odense (DK)	23
	Increasing citizen engagement in decision making	ease open data accessibility; open data visualization; citizens engagements on urbanization	3	Porto (PT), Manchester (UK), Santander (ES), Milan (IT), Herning (DK), Antwerp (BE), Carouge (CH), Novi Sad (RS), Helsinki (FI)	11
Data Mining		data lake value extraction	1	Carouge (CH), Bordeaux (FR), Seongnam (KR)	3
Privacy		citizens awareness of IoT	2	Antwerp (BE), Manchester (UK), Dublin (IE), Carouge (CH)	4

done towards establishing a community for smart city services. We encompass a total of 35 smart city services[2] that involve 27 different cities distributed in Europe and South Korea, reaching 72 running pilots. The themes includes (see Table 6.1): 1) mobility; 2) sustainability; 3) governance; 4) data mining; 5) privacy.

6.3.1 Cities and Pilots commonalities

Cities have latent commonalities regarding city challenges to target and available datasets to handle and process. In this section we analyse the initial 12 smart city services planned by the 8 RZs plus Seongnam (Korea). The analysis outcomes demonstrates that cities face similar challenges when developing smart services even if of different themes. The city services are grouped by three application themes [12]: 1) human-centric mobility, 2) multi-modal transportation; 3) community policy support. Fig. 6.1 shows the associations between data sources to city services, and, then, to applications challenges. The data source types are grouped following the FIWARE data models [27]. The challenges are:

- *C.1 - Enabling Mobility-as-a-Service (MaaS).* Cities seek to shift from transportation ownership models to a service model. Targets are to enable multimodal trans-

[2]https://synchronicity-iot.eu/cities-pilots/

portations, to open real time data on transport modes, to improve efficiency of existing infrastructure, and to redistribute stakeholder roles in the ecosystem.

- *C.2 - Encouraging non-motorised (active) transport.* Air pollution in urban areas is a known problem and cities are implementing different solutions to address it. An approach is to encourage citizens to use zero emission alternatives for short distance urban trips.
- *C.3 - Increasing citizen engagement in decision making.* Often the decision on urban policies are left to governance without great involvement of the real beneficiaries: the citizens. Thus, even if cities are spending effort to become smart, the citizens do not perceive the benefits. Citizens involvement in the process is a new form of democracy [153].
- *C.4 - Increasing diversity in political engagement.* Engagement to the decision making process is often viable only to whom can physically attend. This is not possible to citizens lacking time (e.g., those with family responsibilities), or who cannot easily move (e.g., elderly persons). Digitalizing the process encourages participation from whom is often silent.
- *C.5 - Climate Change Adaptation.* Extreme urban climate shifts are recurrent all around the globe. Cities are studying different solutions to mitigate dangerous situations, such as flash flooding or extreme urban heat.
- *C.6 - Reducing Air and Noise Pollution* Urban environment pollution puts citizens at different risks, such as respiratory issues (due to air pollution) or stress (due to noise pollution [154]). Reducing or managing in an optimal way these two factors can increase life quality.

Fig.6.1 demonstrates that cities are not alone in their problems, even for different application themes. Thus, a technical community is desirable.

6.4 Collaborative approach

To cope with a big number of data producers, service providers and cities, SynchroniCity designed a reference architecture [5]. It has as founding principles the avoidance of *city lock-in* and *vendor lock-in*: the first is to ease city service replication among cities; the second is to keep the market open. The strategy is to follow the Open and Agile Smart Cities (OASC) Minimum Interoperability Mechanisms (MIMs) [81], to aim at a shared ecosystem of data and services. Operational smart city deployments are homogenized with the overlay SynchroniCity middleware, based on the open source FIWARE framework [60], and data are exposed with common interfaces and data models [27]. The data is then consumed by city services. Figure 6.2 depicts the overall concept of the shared ecosystem To boost service providers collaboration and ignite the community, we steer the city services development around the concept of *atomic service*. Developer teams

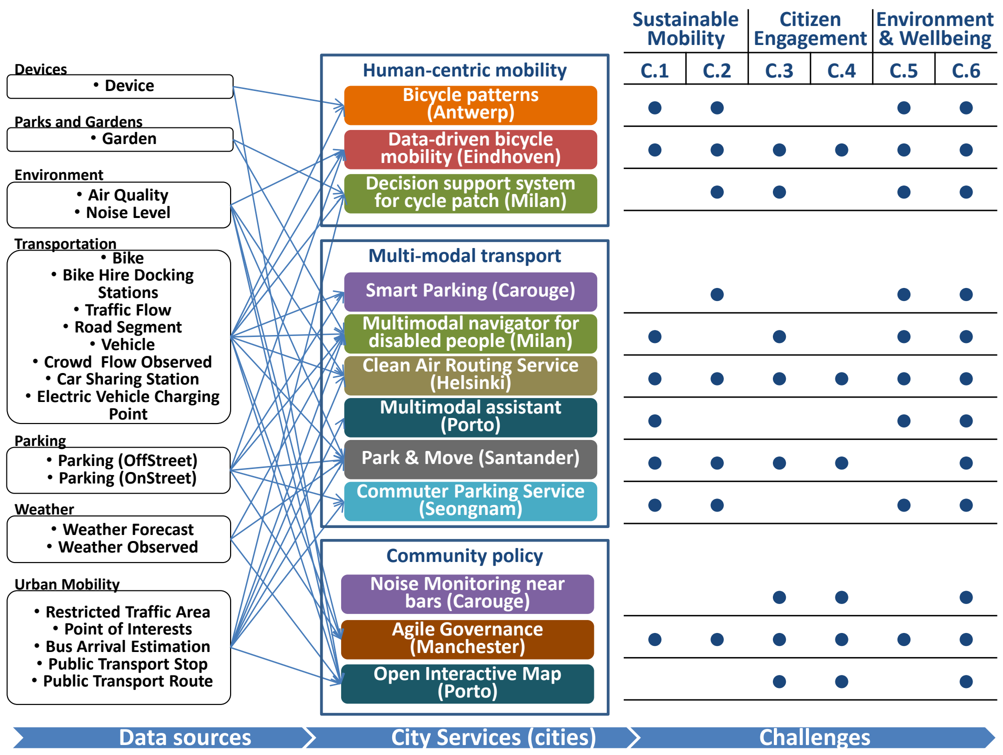


Figure 6.1: Associations between data sources to city services, and, then, to applications challenges. Services for the same city are in boxes of same color.

are encouraged to build the city services following a modular paradigm, embracing the concepts of Service Oriented Architecture (SOA). If a service sub-component is generic enough to be re-usable by another city service, then it is proposed as atomic service: a single functional block consuming data and implementing any kind of feature, such as managing, enriching, joining or filtering the input. Atomic service has similarity with the concept of microservice in the fact of being a self-contained piece of software targeting a specific task. Nevertheless, an important characteristic is that atomic services must be re-usable. Therefore, while an atomic service instance can be a microservice, vice versa is not always true.

Atomic services are identified during three phases (see Fig. 6.3): bottom-up, a supervised top-down, and a top-down phase. The *bottom-up phase* is a preamble phase to identify atomic services before city services are designed. During this period, initial city service themes (i.e., multi-modal transportation, human-centric mobility, and community policy suite) are described with use-cases and requirements with a shared effort among multiple stakeholders (i.e., municipalities, data providers, service providers, technology providers) [12]. The requirements are, then, defined by means of a questionnaire. The synthesis from all the answered questionnaires [8] brings the identification of an initial set of atomic services. The questionnaire aims to: 1) prioritize application requirements for each city; 2) identify available re-usable software components; 3) identify available know-how by developer teams. The prioritization of requirements gave us the target

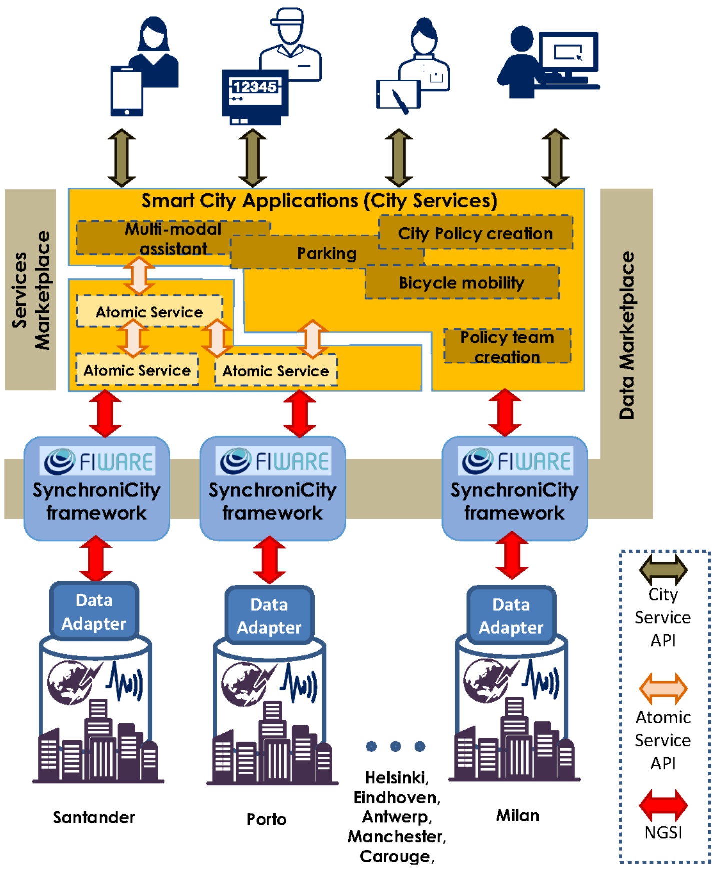


Figure 6.2: City services and atomic services shared ecosystem.

to be addressed collaboratively and, thus, by an atomic service. Whereas, the listing of available software components and know-how identified re-usable software and developer teams. During the following phase, *supervised top-down phase*, developer teams are left to design their city service taking into account the identified atomic services. The resulting city services' architectures are, then, jointly analyzed and compared. Common modules are selected as atomic services. The *top-down phase* gives the field to the city services owner to freely design their own service and identify possible components that might be good candidates to become atomic services. The community, then, decides whether accepting those by checking if compliant with at least one of the following requirements: 1) to be part of two or more city services; 2) to be generic enough to have the potential to be used by multiple parties; 3) to have the potential to become part of a greater ecosystem (e.g., a Grafana plugin). The last phase is continuous and followed by any new city service project. The first two phases lasted 13 months (from January 2018 to January 2019) and involved 12 city services in 9 cities. The last phase is still ongoing and covers other 23 city services piloted in all the 27 cities.

Once an atomic services is acknowledged, it shall comply to technical principles to ensure quality: open source code access (e.g., Gitlab), complete (and "templated") documentation (e.g., Apiary and ReadTheDocs), and easy to deploy (i.e., Docker).

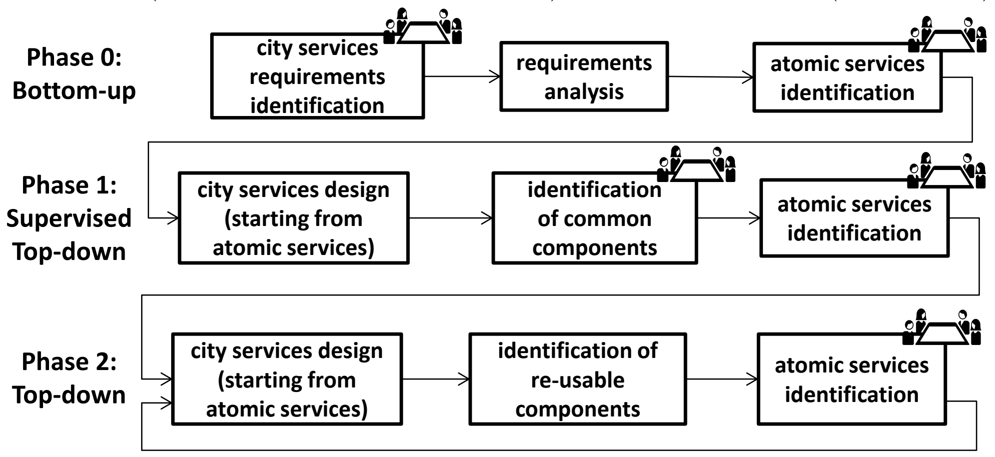

Phase 0:
Bottom-up
city services requirements identification
requirements analysis
atomic services identification
Phase 1:
Supervised
Top-down
city services design (starting from atomic services)
identification of common components
atomic services identification
Phase 2:
Top-down
city services design (starting from atomic services)
identification of re-usable components
atomic services identification

Figure 6.3: Collaborative approach towards the implementation of 35 city services. Some of the steps (with meeting icon) involve interaction with the community.

6.4.1 Questionnaire

The design of the city services started with the identification of stakeholders and functional requirements (FRs) for each of the initial application themes [12]: multi-modal transportation, encouraging non-motorized transport, and community policy suite. As methodology to define atomic services from FRs we grouped them in small subsets, and for each of the subset we assign an atomic service to address the related FRs.

The identified FRs cover all the aspects of an application theme, and typically not all are required by a specific city service. Moreover cities and project partners have previous experiences with smart cities applications, thus already existing components might be re-used simply off-the-shelf, or with minimal integration effort.

For these motivations we create a questionnaire (see Table 6.2) with the following targets: 1) identify previous experiences with the application themes, 2) identify re-usable software components, 3) identify possibility to interact with the SynchroniCity infrastructure (read-only or read-write), 4) identify which packaging tool is preferred for sharing components, 5) identify license foreseen for the atomic/city services to be implemented, 6) prioritize application themes FRs for each city.

6.4.2 Functional Requirements (FRs) analysis.

We distribute the questionnaire to the 8 cities plus additional partners that collaborate with other cities (some of the questions were not applicable to the latter group). The questionnaire was answered by all the involved cities for each application themes, with additional answers by other cities and partners of the project. Table 6.3 shows the cities that answered the questionnaires; between brackets are the additional cities not directly involved in the related application themes. In total 9 partners participate, categorized in 5 municipalities, 2 SMEs, and 2 academia/research foundations.

Table 6.2: Atomic services and city services questionnaire

Question				Answer
• Have you have already implemented an application satisfying the proposed use-cases [12]? • If yes at the above question, can you please provide an architecture overview (sub-components, atomic services)?				
• Have you any related atomic services already implemented? • If yes at the above question, are you willing to share them (OpenSource, Freeware, Licensing)? • If yes at the above question, can you provide an architecture overview for each atomic service you possess?				
• Do you envision any additional atomic service to share across application of the same theme and/or across application of different themes? Please give a brief description/suggestion.				
• Is the NGSI data from the IoT infrastructure read-only towards the atomic services or the atomic services will be capable of writing new data? (e.g., new entities or commands like in the case of traffic light controlling)				
• Which methods and related tools are desired for atomic service deployment (e.g. docker, debian package)?				
• Will the new code you implement for SynchroniCity be reusable? (Open Source, Freeware,)?				
• Which FR your application shall satisfy? (Priority 0 to 5: 0 for FR not needed, 5 for FR of utmost importance) In case you have implemented atomic services, which FR do they fully or partially satisfy/support?	FR-1	Priority (0-5)	Already satisfied; Partially satisfied; not implemented; n.a.	
	FR-2	Priority (0-5)	Satisfaction level	
	...	Priority (0-5)	Satisfaction level	
	FR-n	Priority (0-5)	Satisfaction level	

From the answers we filter and prioritize the functional requirements with the following criteria: 1) threshold to filter out FRs not to be addressed because not of common interest: more than 2 cities with priority equal or higher than 3; 2) filter out FRs already supported by the data producers and IoT framework; 3) rank by priorities average.

Table 6.3: Functional Requirements (FRs) of common interest

Application Theme	**Cities answering the questionnaire**	**Total FRs**	**Selecte FRs**	**FRs to address**
Human-centric traffic mgmt.	Ant, Ein, Mil; (Car, San)	12	8	7
Multi-modal transportation	Hel, Mil, Por, San; (Car)	33	28	19
Community Policy Suite	Car, Man, Por; (San)	9	8	8

In particular, for point 2), we evaluate the SynchroniCity framework [155] and the legacy IoT systems involved. For instance, the security layer of the IoT framework fully addresses three FRs (plus partially a fourth one). In fact, the Single Sign-On (SSO) approach is envisioned for all the city services that are developed following the SynchroniCity philosophy of a shared ecosystem. This, on the one hand, relieves the application developer from the burden of managing credentials, and on the other hand, the city service users (e.g. citizens) to create a new user for every new city service. Another FR was addressed by the geographical data query and subscription methods offered by the infrastructure. Finally the cities IoT providers together with the adoption of a common standard, such as OMA NGSI [58], and common data models, such as FIWARE data models [27], were already addressing six data related requirements [3]. Table 6.3 summarizes the filtered FRs, a more detailed list of selected FRs can be viewed in [156].

6.5 Atomic Services

By the end of SynchroniCity (Dec 2019) the roster accounts 15 atomic services. Table 6.4 summarizes the services by category and by the phase of selection. The first three services entering the roster are the results of the bottom-up phase. During this phase, partners with previous experience in smart city projects bring their expertise and previous results. The *Smart Cities Dashboard* and the *Grafana Dashboard* are visualization tools used in past pilots. The *Routing Service (Open Trip Planner - OTP)* is a data evaluation service, which is adopted by many current city services concerning a journey. Even if Grafana and OTP are third party software, they are still accounted as atomic services. Our approach is not to forcefully create new components, but to share best practices and expertise to city services developers. Indeed the final goal is to create a self-sustained community. In the case of those two services, what is provided is the support, tutorial, and ready-to-use packaging for smart city context.

During the supervised top-down phase, different smart city services' architectures are collaboratively analyzed and compared. This resulted in five atomic services: *Parking* and *Traffic Estimator* are data analytics services that use artificial intelligence (AI) to predict, respectively, parking and traffic situation; *GTFS-RT Loader*, *NGSI Urban Mobility to GTFS Adapter* and *GTFS Fetcher* are data integration services necessary to integrate the routing service, that digests General Transit Feed Specification (GTFS) files, with the underlying Next Generation Service Interface (NGSI) protocol adopted by the FIWARE-based framework.

During the top-down approach, other seven atomic services entered the community. The *(Outdoor) Noise Estimator* is a data analytics service based on the same system of the other two estimators. This predictor resulted important for the Carouge's community policy suite service, and we are asked to tailor a new atomic service for their needs. The *Bike Data Visualiser* and *Grafana NGSI Map plugin* are two visualization atomic services. The latter is a plugin for Grafana (also published in the Grafana community) to readily use NGSI data. The *Transformer GPS to NGSI Traffic Flow Observed (TFO)*, and *Legacy to NGSI Transformer* adapt simple JSON data to the FIWARE data models, enabling the usage of NGSI-based atomic services. The *NGSI to NGSI-LD* is meant to keep the pace of the NGSI evolution towards the linked data [89]. Finally the *NGSI*

[3]As of end of 2020, the OMA NGSI has evolved to ETSI CIM NGSI-LD [89] and FIWARE data models are evolved into Smart Data Models [80]

Validation checks the correctness of NGSI message against the FIWARE data models JSON schemas, increasing city services reliability.

Table 6.4: Atomic Services selected with different approaches: phase 0) bottom-up; phase 1) supervised top-down; phase 2) top-down.

Category	data analytics			data eval.	data integration						data val.	visualization			
Atomic Service	parking estimator	traffic estimator	noise estimator	routing	gtfs-rt loader	gtfs fetcher	ngsi to gtfs	gps to gtfs	ngsi to ngsi-ld	json to ngsi	ngsi validation	dashboard	grafana	bike data visualizer	ngsi grafana plugin
Phase 0				✓								✓	✓		
Phase 1	✓	✓			✓	✓	✓								
Phase 2			✓					✓	✓	✓	✓			✓	✓

6.5.1 Data analytics atomic services

In [8], we introduced the legacy solution for predicting parking area availability and traffic flow based on time-series data. These services expose HTTP interfaces for providing data prediction based on machine learning (ML). From these two atomic services we assemble a third estimator for outdoor noise level. Namely, we leverage data from outdoor deployed noise sensors (e.g. Carouge, Santander) to predict the noise level in the next time window (e.g. 1 hour).

Fig. 6.4 shows the current architecture that is easier to configure/deploy/use from the previous version [8] and, at the same time, more scalable, flexible and stable. Following the data flow, underlying IoT infrastructures (i.e., FIWARE-based Synchronicity Framework) feed the entry point of the atomic service, so called *IoT Data Manager*. This component gathers data from the various nodes/IoT devices in three ways: last values-context information, historical data, and event-based subscription. Past, present and future data is forwarded to the *Data Storage Cluster*, a decentralized datastore based on Elasticsearch. Optionally, a FIWARE Context Broker[4] is used to store the output of the estimators as new attributes for each sensor, enabling these services to be used as standalone solutions when combined with other components (e.g., Grafana for visualization, Quantum Leap[5] for timeseries storage).

The AI operations are performed by two components based on Keras, TensorFlow and Scikit-Learn as ML engine:

[4]https://fiware-orion.readthedocs.io/

[5]`https://smartsdk.github.io/ngsi-timeseries-api/`

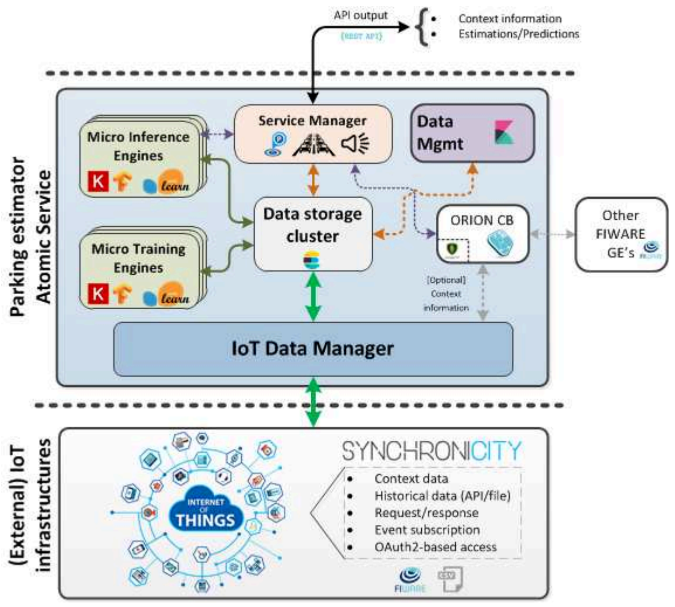


Figure 6.4: Estimator atomic service(s) architecture

- **Micro Training Engine(s)** generate the models upon the data (i.e. historical time series) collected at the Data Storage Cluster. A model is generated for every context entity (i.e., a thing) having a minimum number of samples (configurable; by default, 1000). Since new data is continuously streamed onto the system, the trainer is scheduled to periodically re-train the models (by default, in a daily basis). The trainers can be configured with different algorithms, training window size, and train/test ratio.

- **Micro Inference Engine(s)**. For each sensor, given the model generated by a training engine and a sample chunk containing the latest observations, the inference engine calculates the predictions/estimations for the next time-window. There is one micro inference engine per micro training engine. The inference is performed, by default, every 15 minutes. The prediction values are saved into the Data Storage Cluster and, optionally, in the context broker.

Original sensor observations and estimations are exposed by two interfaces. *Data Management* is a Kibana-based dashboard that visualizes data available in the Data Cluster Storage. The *Service Manager* provides an HTTP API where users can: 1) query all the information stored in the Data Cluster Storage; 2) call a prediction from a Micro Inference Engine and retrieve the output. The service manager is the sole component actually different between estimators (i.e. parking, traffic and outdoor noise), since each of the Service Managers is tailored according to its respective data API and output format.

6.5.2 Atomic services as building blocks for city services: multi-modal transportation city service in Santander

In this section we describe the combination of atomic services to provide a routing service in a multi-modal transportation scenario based on Open Trip Planner (OTP). We focus on the routing service deployed in the city of Santander that aims to provide routes within the city and from the city to nearby villages. The transportation and mobility alternatives include public bus service managed by a private company, mobility facilities within the city, which include mechanical stairs and funicular, and a private ferry service that connects the city with two villages through the bay.

The resulting service uses the FIWARE Context Broker as the central piece, which stores heterogeneous urban data according to the defined data models (see Fig. 6.5). In addition, the routing engine relies on the GTFS and GTFS Real Time (GTFS-RT) data models to generate routes. Different data providers (city services or utilities) generate NGSI entities, that are consumed and transformed by the atomic services, so that it can be consumed by the routing engine. It is worth noting that the public services (i.e., buses, escalators, funiculars) create entities linking to regular GTFS files using the *GtfsTransitFeedFile* data model and periodically provide arrival estimation information by updating *ArrivalEstimation* entities (e.g., bus arrival at a stop). On the other hand, the private service (i.e., ferries) lacks standard data models, thus, it uses a set of NGSI entities to store scheduling information.

The atomic services deployed in this scenario are:

- **NGSI Urban Mobility to GTFS** consumes NGSI urban mobility entities and generates GTFS feeds (i.e., *.zip* files) storing them locally. Then, it creates Gtfs-TransitFeedFile entities in the context broker pointing at GTFS files.
- **GTFS Fetcher** feeds OTP with GTFS files. This atomic service tracks the modifications in the GtfsTransitFeedFile entities to update and reload the OTP databases and maps.
- **GTFS-RT Loader** consumes ArrivalEstimation entities from the NGSI context broker, to generate GTFS-RT file. The service subscribes to ArrivalEstimation updates in the context broker, to propagate them to GTFS-RT feeds. The service also exposes this real time information through a REST interface.
- **Routing Service**, based on Open Trip Planner, consumes GTFS and GTFS-RT data and provides multi-modal routes, supporting customization options. The GTFS Fetcher pushes GTFS feeds to the routing service. The routing service can also pro-actively consume GTFS-RT from an endpoint.

6.5.3 Danish smart cities experiences

Atomic services come also from the adaptation of old solution to new technologies and standards. Danish cities is working to establish successful smart city solutions for at

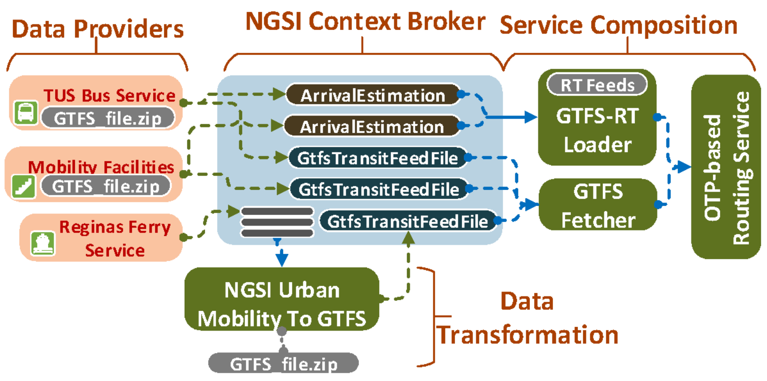


Figure 6.5: Deployment and composition of atomic service to provide multi-modal transportation routing service in the city of Santander

least a decade. Early attempts by larger municipalities are made by launching open data platforms and working with the concept of digital marketplaces where public data would be made available to foster innovation with local businesses. Later attempts include working on living labs and establishing a technical foundation for deploying IoT solutions. However, the truth is that none of this has yet made real impact, and it has not been able to scale above limited trials.

Danish cities are increasingly joining forces through networks (e.g., OS2, Gate21), to share knowledge and ensure that future smart city solutions are interoperable. Many cities have experienced pilots that have not delivered what was promised, and so their business cases have not been realized. With the market moving ever faster and global players like Microsoft and Amazon also pushing their solutions to cities, some cities feel there are too many standards, not enough national guidance and some are close to giving up entirely or capitulating to global vendors with full stack solutions.

Some danish cities like Vejle, Aarhus and Odense are working hard to break this deadlock, and these cities are using an agile approach that allows them move forward and innovate at a higher speed, and with a lower cost. Both Vejle and Odense have started to build their own simple IoT solutions from the ground up, based on their own concrete needs. Without over engineering their solutions, they have been able to build very simple “platforms” that can ingest data from multiple heterogeneous legacy systems into a modern ICT infrastructure. The problem is that these solutions are not based on standards, they cannot scale to meet new demands and each city is working independently of each other. Fortunately we could integrate their architecture with SynchroniCity, and all the effort they have spent understanding and parsing raw source data is not wasted. SynchroniCity project foresees that providing an excellent technical user experience for developers is key to adoption, especially in smaller cities. Individual software engineers often choose the component that enables them to deliver functionality with least effort, hence why Vejle choose to use Node-RED, influxdb and SQLserver.

We shift Aarhus custom platform to a standard compliant (i.e., NGSI) platform using the *Legacy to NGSI Transformer* atomic service that translates generic JSON format to NGSI. On the city service we replace: custom visualizations and webportals with

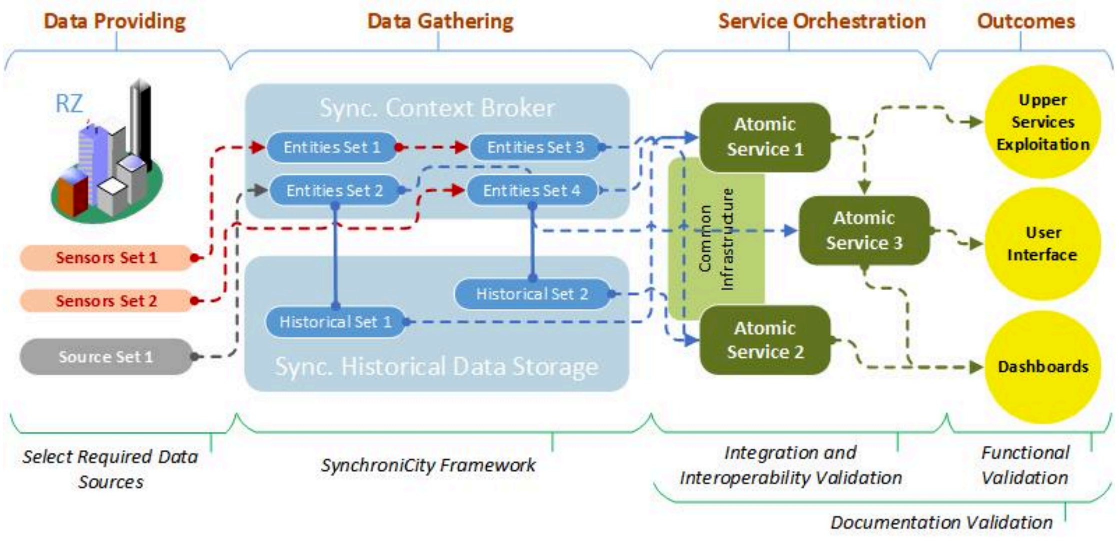


Figure 6.6: Validation layout

the *NGSI Grafana plugin*, and custom Redis data store with Quantum Leap. We also enhance reliability by the adopting the *NGSI Validation* atomic service. In addition, we already implemented the *NGSI to NGSI-LD* atomic service as preparation for the NGSI transition to linked data [89].

6.6 Evaluation

While the process of identification and publication of new atomic services is still ongoing, we validated the quality of the eight bottom-up and supervised top-down services. In the process, we also analyze the community engagement.

6.6.1 Validation

Synchronicity atomic services are distributed with documentation regarding the deployment process, provided functionalities (e.g. the consumed/produced datasets) and their APIs. In this sense, the validation focus is set on three main aspects: 1) documentation regarding requirements (e.g., software and hardware), installation process and usability; 2) deployment and integration with the SynchroniCity framework; 3) the features covering city service themes functional requirements. The latter point makes sense only for atomic services chosen under supervised phases, while designing city services per theme. For atomic services chosen during the top-down phase, this validation step cannot be verified since the features relate to single city services and not to city service themes.

We define a set of compositions, called scenarios, that relate atomic services based on consumed data sources and provided outcomes. These scenarios require an integration with a SynchroniCity core infrastructure to collect data. Each of these compositions evaluate the provided documentation that describes the involved atomic services, the integration with SynchroniCity framework, and their interoperability capabilities. We set up three scenarios:

- The **Routing** scenario combines GTFS information feeds with the SynchroniCity OTP-based routing service to provide multi-modal routes. It validates the capabilities of the atomic services framework to support GTFS files management and GTFS-RT feeds generation to build multi-modal routes within the city. The layout of this scenario corresponds to the Santander application (see §6.5) and puts together NGSI, GTFS and OTP services on top of the Santander SynchroniCity framework.
- The **Estimation** scenario exploits data from ParkingSpot, OnStreetParking and TrafficFlowObserved entities provided by the Santander deployment, both last value and historical data, to feed the Parking and Traffic estimators services. Both services combined offer an overview of the current and incoming traffic situation on a city, providing also valuable information to upper mobility services or applications.
- The third scenario, **City Data Visualization**, is composed by the Smart City Dashboard and Grafana Dashboard atomic services that provide views of the available IoT data and their impact on different city indicators. These two services consume data from the NGSI context broker.

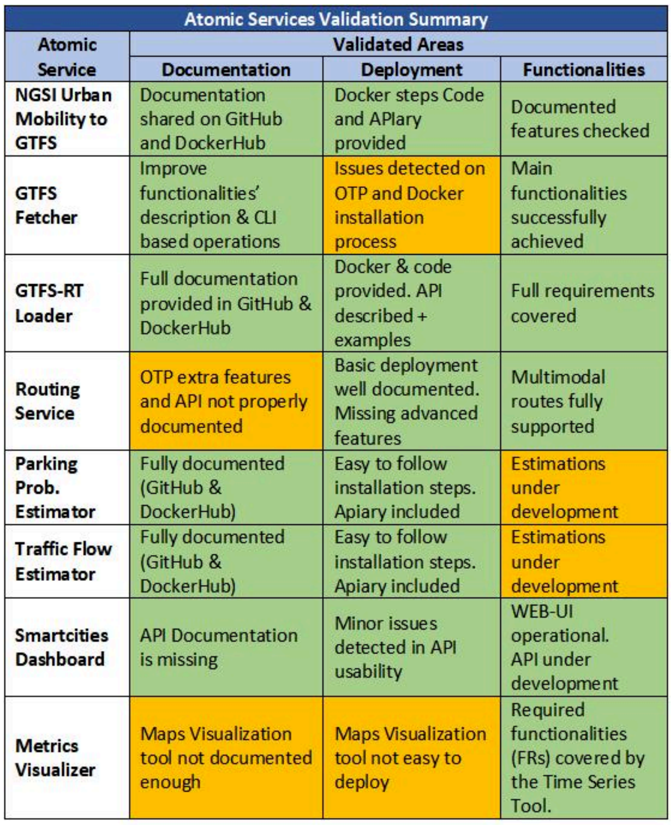
We emulate a SynchroniCity core framework with a testing environment (see Figure 6.6) that includes components and data sources. Following their documentation, we deploy the atomic services that directly consume SynchroniCity resources. Then, we deploy and check the rest of atomic services that consume output from the first ones (and if applicable, also from the SynchroniCity components). Finally, we validate the documented functionalities of each atomic service. This is an iterative process, thus, each detected issue in any of the validation steps, including provided documentation, is reported to the service developer to allow improvements in the next iteration.

Figure 6.7 shows a status summary shared with atomic services developers or, in case of external third party components (e.g., Grafana and OTP), the designed responsible.

6.6.2 Community Engagement

The number of atomic services increased consistently throughout time, reaching a total of 15 atomic services. Fig. 6.8 shows also the trend of their adoption by city services, reaching a total of 38 “success stories”. Each integration of an atomic service in a city service is counted (e.g., the multi-modal transportation city service of Santander described in §6.5.2 counts 4 adopted atomic services).

Atomic service providers are fairly distributed among the three categories of academia, industry, and SMEs (see Fig. 6.9). Among the atomic services users (i.e., parties involved in the development of city services that integrate an atomic service), also the governance category have similar share with the others. Typically, governance parties have not much



Atomic Services Validation Summary			
Atomic Service	Validated Areas		
	Documentation	Deployment	Functionalities
NGSI Urban Mobility to GTFS	Documentation shared on GitHub and DockerHub	Docker steps Code and APIary provided	Documented features checked
GTFS Fetcher	Improve functionalities' description & CLI based operations	Issues detected on OTP and Docker installation process	Main functionalities successfully achieved
GTFS-RT Loader	Full documentation provided in GitHub & DockerHub	Docker & code provided. API described + examples	Full requirements covered
Routing Service	OTP extra features and API not properly documented	Basic deployment well documented. Missing advanced features	Multimodal routes fully supported
Parking Prob. Estimator	Fully documented (GitHub & DockerHub)	Easy to follow installation steps. Apiary included	Estimations under development
Traffic Flow Estimator	Fully documented (GitHub & DockerHub)	Easy to follow installation steps. Apiary included	Estimations under development
Smartcities Dashboard	API Documentation is missing	Minor issues detected in API usability	WEB-UI operational. API under development
Metrics Visualizer	Maps Visualization tool not documented enough	Maps Visualization tool not easy to deploy	Required functionalities (FRs) covered by the Time Series Tool.

Figure 6.7: Validation summary of SynchroniCity atomic services (May 2019)

resources for development. This explains why their contribution to atomic services is limited while the adoption is fostered. If we count the parties as many times as they are involved in a city service (third set of bars in Fig. 6.9), we can observe that SMEs and governance take big advantage to this shared effort. Being academia usually the most willing to share and to experiment, this category of parties result to be the biggest in terms of providers and users. Industry also helped with 4 atomic services, but the low number of adoption is due to their involvement into a limited number of city services.

Table 6.5: Third party services adoption

	Academia	**Governance**	**Industry**	**SME**	**tot**
Own Adoption	6	1	1	10	18
External Adoption	7	6	3	4	20

Atomic services are adopted almost evenly by the same developer party and by third parties. Table 6.5 shows these numbers together with the detailed by stakeholder category. It is interesting to note that while academia group is open to share and adopt ready solutions, governance and SMEs have opposite attitudes. In particular governance re-uses third parties software whereas SMEs are keen to implement their own service

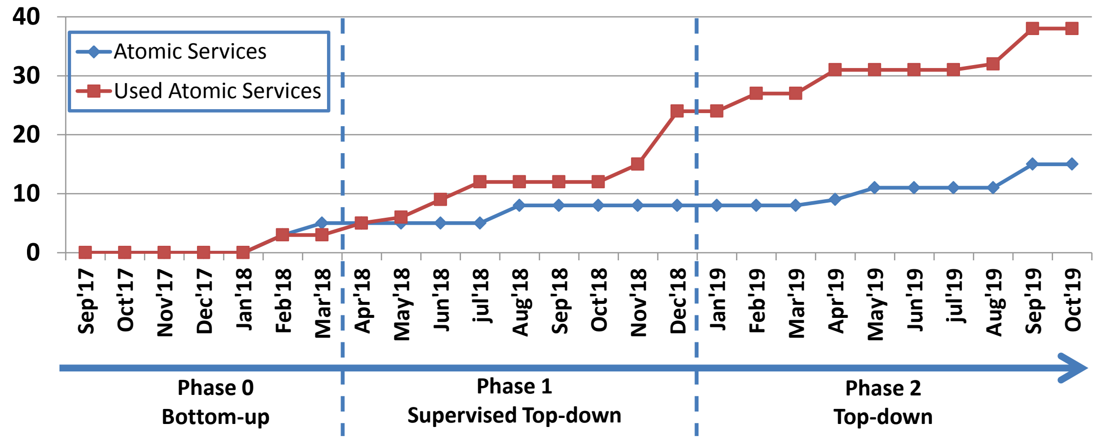


Figure 6.8: Trend of atomic services and their adoption by city services

Table 6.6: City services categorizations

	City Services	Adopted AS
Public governance projects	12	26
6-months pilots	16	0
Individual exploitations	8	12

and offer it in the market. This is explained by the fact that startups develop their core technology to build product differentiation. This is demonstrated also in Table 6.6 where the city services of Table 6.1 are categorized in public governance projects, 6-months pilots driven by startups, and individual exploitation of a single stakeholder. We note that startups project do not adopt any atomic service, mostly due to the short time project spent to develop their core technology. Cities projects, instead, are confirmed to will to re-use technologies. These two attitudes are complementary and the atomic services community helps providers to meet customers.

If we analyze the distribution of atomic services categories by city services categories we can notice that the multi-modal transportation is the category with the biggest number of atomic services adopted. That is because the city services in this category were the most deeply designed with common effort [8] across the bottom-up and the supervised top-down phases. Furthermore this kind of services involved advanced data manipulation (i.e., parking and traffic prediction analytics) and evaluation (i.e., routing service), with necessary data integration (i.e., to and from GTFS). Community policy suite services take advantages from atomic services, mainly from the visualization typology (especially Grafana) being this kind of applications oriented to aid human decisions.

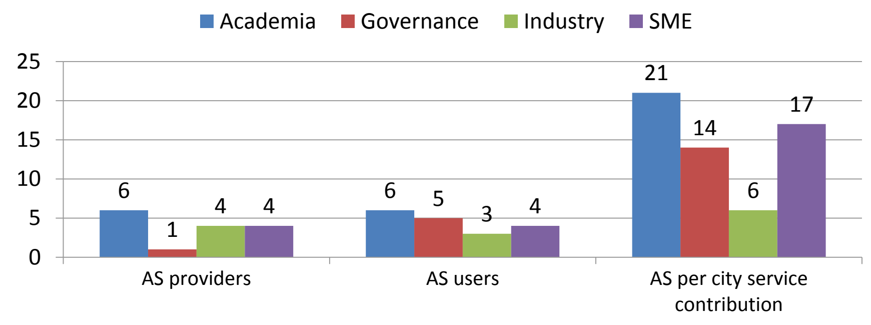


Figure 6.9: Distribution of atomic service contribution and usage per stakeholder

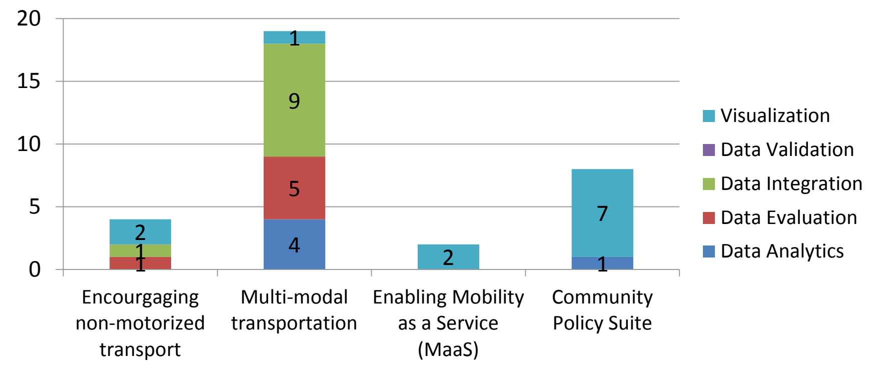


Figure 6.10: Distribution of atomic service categories among city service categories

6.7 Conclusions

In this chapter we presented the collaboration among many stakeholders for the implementation of 35 city services piloted in 27 cities. The approach is to encourage developers to have modular design of their applications in order to identify re-usable IoT services, named atomic services. The identification of atomic services went through three phases, with one still open since it envisions contribution from the open community of smart city developers. We identified 15 atomic services addressing IoT challenges in data analytics, data evaluation, data integration, data validation, and visualization. We present examples of atomic services (related to data analytics) and their usage in real application in Santander and Denmark. We performed validation on the quality of the atomic services and an assessment of the adoption by smart city application. It resulted that a total of 38 instances of different atomic services are operating in several city services.

Chapter 7

HyperConnected Smart City Services

Understanding crowd mobility behaviors would be a key enabler for crowd management in smart cities, benefiting various sectors such as public safety, tourism and transportation. This chapter discusses the existing challenges and the recent advances to overcome them and allow sharing information across stakeholders of crowd management through Internet of Things (IoT) technologies. The chapter proposes the usage of the new *federated interoperable semantic IoT platform* (FIESTA-IoT), which is considered as "a system of systems". The platform can support various IoT applications for crowd management in smart cities. In particular, the chapter discusses two integrated IoT systems for crowd mobility: 1) *Crowd Mobility Analytics System*, 2) *Crowd Counting and Location System* (from the SmartSantander testbed). Pilot studies are conducted in Gold Coast, Australia and Santander, Spain to fulfill various requirements such as providing online and offline crowd mobility analyses with various sensors in different regions. The analyses provided by these systems are shared across applications in order to provide insights and support crowd management in smart city environments.

In addition, we leverage the information transparency through semantic interoperability and present another example application, namely Smart City Magnifier (SCM). We take advantage of a semantic approach to resource and context management, showing a seamless and transparent federation of different heterogeneous systems towards a holistic smart city monitoring service. SCM encompasses: semantic mediation, linking of data to entities and of entities to entities, and data augmenting through situation awareness.

7.1 Crowd Mobility

Sustainable development of cities is a major global challenge as more than half of the world population is living in urban areas. The smart city concept allows optimizing services for urban areas because or as a result of the advancement of the new technologies ranging from very small devices to big data centers. These technologies can be considered

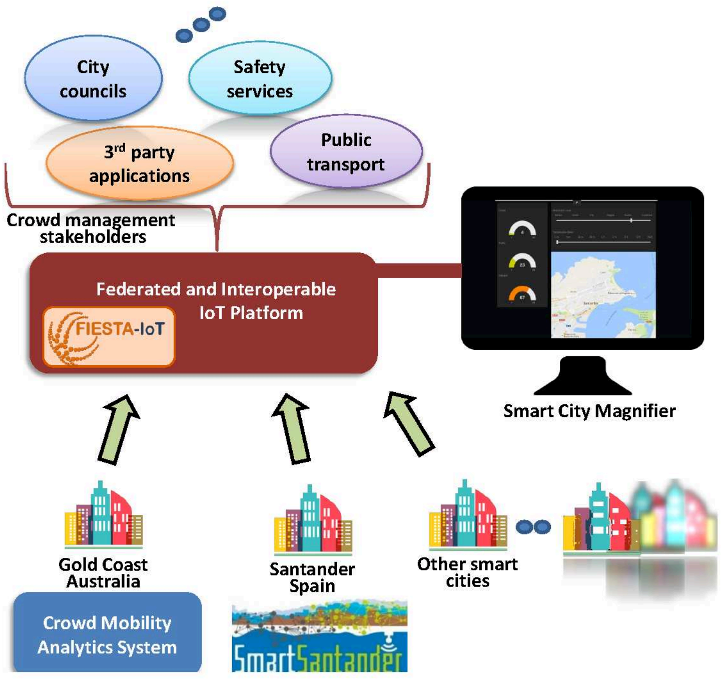


Figure 7.1: Federated and interoperable IoT platform supporting crowd management stakeholders of smart cities.

in the context of IoT, where many objects, devices, machines, and data centers are connected. The usage of IoT technologies for *crowd management* in urban environments is promising for the future of smart cities.

IoT technologies can enable many improvements for crowd management, which spans sectors such as transportation services (e.g., operating public transport or directing pedestrian traffic), public safety (e.g., detection of fighting incidents), and tourism (e.g., event management for enhanced visitor experience). For instance, movement behaviors of crowds may indicate situations such as traffic congestion, emergency incidents, and panic situations during certain events such as large gatherings in city squares.

While cities aim to achieve smart urban services, new challenges arise due to the limitations and deficiencies of the current systems and technologies in terms of *scalability* of the connected systems, *information transparency* between different systems (i.e., semantic interoperability) or stakeholders, *data federation*, and *information privacy*. When mobility information must be shared across multiple stakeholders, a proprietary infrastructure cannot fulfill all the different requirements that they impose. For example, some of the stakeholders expect real-time mobility monitoring service for event detection while others require historical mobility data analytics to analyze efficiency of services in different urban environments (e.g., train station, stadium, city square). It is very difficult to re-design a one-size-fit-all IoT system when new requirements arise for various environments or different time periods. A better solution is to provide “a system of systems” in which a new service can be easily developed or setup to handle any new requirement by leveraging existing technologies and infrastructure. To make such a system of systems useful, semantic models based on an appropriate ontology are needed for transparently exchanging data, analytics results, and allowing to share new insights from different crowd management applications. The federation of data, results, and learned insights is the key technical enabler to understand the crowd mobility behaviors in a smart city. Finally, privacy preservation is a problem of utmost importance for smart cities. While various data from vast deployment of sensors travel through the IoT systems, preserving privacy at a level closer to the data contributors (providers) is an important challenge.

This chapter describes the recent advances in IoT for understanding crowd mobility in smart cities. The *federated and interoperable semantic IoT* (FIESTA-IoT) platform for smart cities is introduced for the specific perspective of crowd management applications. Fig. 7.1 illustrates the outlook of the smart city applications leveraging the smart city platform for sharing information across various stakeholders. While the platform is currently in use for several smart city testbeds, the chapter focuses on two IoT systems for crowd mobility, namely *Crowd Mobility Analytics System* (CMAS) and *Crowd Counting and Location System* (CCLS) and discusses the aspects related to the aforementioned limitations.

Two pilot studies are conducted in Gold Coast, Australia and Santander, Spain, where various sensors are deployed in urban areas. The first pilot study uses CMAS in Gold Coast for a medium-scale smart city deployment. The requirements of the pilot include analyzing heavy or light pedestrian traffic at streets with or without vehicles. The second pilot study uses CCLS in an indoor market in Santander. The requirements include detecting people (crowd size) and locating their positions at public buildings of a city and other critical infrastructures. In both pilots, data anonymisation limits tracking devices for long time periods. On the other hand, online and offline analytics information needs to be shared across various stakeholders such as city councils and visualized in several interfaces using IoT technologies and infrastructure to provide insights for crowd management in smart cities.

7.1.1 Crowd Mobility Analytics using the Smart City Platform

Federated and Interoperable IoT Platform

Smart city data is often gathered by solutions where dedicated networks of sensors or data sources produce observations to be consumed by specific applications. The systems usually differ from each other, serving for distinct purposes, and they are mostly not interoperable [69, 157]. In this regard, creating crowd management services that harness the abundant data from a smart city (e.g., environmental data, road traffic information) would require either ad-hoc integration or creation of new systems. This situation raises a new requirement of an integrated “system of systems” or “container of systems”.

To overcome this challenge, we propose a crowd mobility-based instantiation of the FIESTA-IoT platform [92] and provide semantic interoperability from IoT deployments to the services (shown in Figure 7.2). The heterogeneous IoT deployments on the *IoT Devices and Systems* (bottom layer) are integrated to the Cloud and data is anonymised

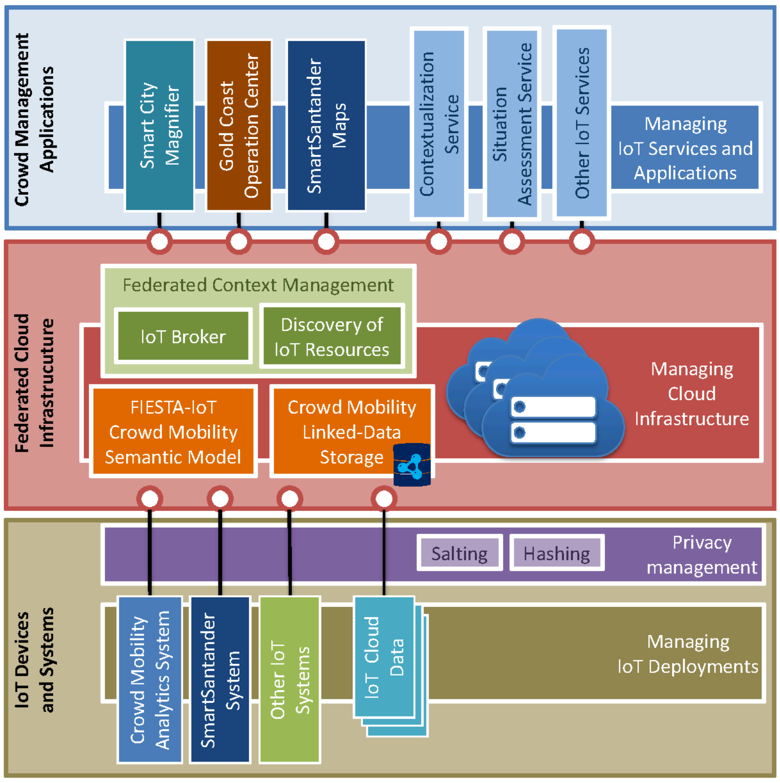
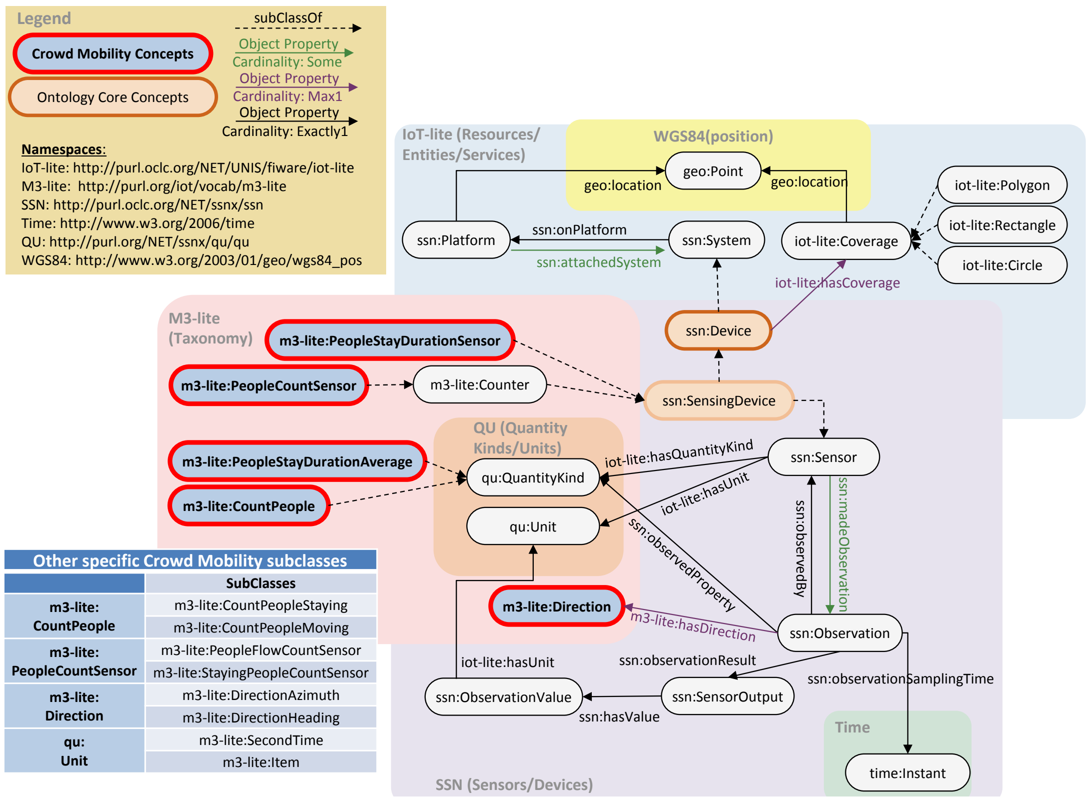

Crowd Management Applications
Smart City Magnifier
Gold Coast Operation Center
SmartSantander Maps
Contextualization Service
Situation Assessment Service
Other IoT Services
Managing IoT Services and Applications
Federated Cloud Infrastructure
Federated Context Management
IoT Broker
Discovery of IoT Resources
FIESTA-IoT Crowd Mobility Semantic Model
Crowd Mobility Linked-Data Storage
Managing Cloud Infrastructure
IoT Devices and Systems
Salting
Hashing
Privacy management
Crowd Mobility Analytics System
SmartSantander System
Other IoT Systems
IoT Cloud Data
Managing IoT Deployments

Figure 7.2: Crowd mobility-based instantiation of the federated and interoperable IoT platform.

with salting and hashing. In this layer, in addition to the two crowd mobility systems (CMAS and CCLS), IoT Cloud Data from external platforms can be connected. Furthermore, there exist other IoT FIESTA-IoT systems that can be leveraged. Currently more than 5000 sensors (from 11 integrated testbeds [158]) report environmental data (e.g., temperature, humidity, illuminance, noise level), road traffic information (e.g., vehicle speed, traffic intensity), car and bike parking spots, estimated arrival times of buses, and smart building information (e.g., human occupancy, power consumption).

At the *Federated Cloud Infrastructure* (middle layer), the data from the bottom layer is modelled using the FIESTA-IoT Semantic Model and stored in the Linked-Data Storage. In particular, the semantic model for crowd mobility data is described in Section 7.1.1. The data in the Cloud infrastructure is accessible through the Federated Context Management which exposes NGSI and SPARQL interfaces. Our open source IoT Broker (Aeron Broker) component provides scalable federation for the context management, whereas IoT Discovery (NEConfMan) enables easy registration and discovery of resources with features such as geo-discovery.

The crowd management-related IoT data is harnessed by *Crowd Management Applications* (top layer) which contain IoT services provided by the platform and crowd mobility applications. These services enhance the crowd mobility data through reason-



Other specific Crowd Mobility subclasses

	SubClasses
m3-lite: CountPeople	m3-lite:CountPeopleStaying
	m3-lite:CountPeopleMoving
m3-lite: PeopleCountSensor	m3-lite:PeopleFlowCountSensor
	m3-lite:StayingPeopleCountSensor
m3-lite: Direction	m3-lite:DirectionAzimuth
	m3-lite:DirectionHeading
qu: Unit	m3-lite:SecondTime
	m3-lite:Item

Figure 7.3: Modeling crowd mobility information based on FIESTA-IoT ontology.

ing by aggregating the semantic data and assessing the situations related to physical objects (i.e., Contextualization Service) at different levels of abstraction such as buildings level or street level. Assessment of the situations can be performed through; a) pre-defined thresholds, b) anomaly detection, c) time-series analysis, d) artificial intelligence. The obtained situations are displayed on the dashboard in Figure 7.1, named *Smart City Magnifier*, which reports alerts regarding traffic status, crowd flows, critical events (e.g., fire bursting), and so on. Moreover, crowd mobility applications such as *Gold Coast Operation Center* and *SmartSantander Maps* receive the results (generated by CMAS, CCLS or other IoT services) from the Cloud and provide visualizations.

Crowd Mobility Semantic Model

In order to provide seamless interoperability and information transparency from IoT systems to the crowd management applications, the crowd mobility outcomes are semantically annotated following the FIESTA-IoT ontology [26] as shown in Fig. 7.3 (with a stress on the specific taxonomy of M3-lite for crowd mobility).

Rich and complex knowledge is represented with an ontology as things are connected to each other through relationships. Things are not identified as individuals, but as classes of individuals. Moreover, a class might have sub-classes. For example, people-

CounterX is an instance of *PeopleCountSensor* class which is a subclass of *Counter* (see Fig. 7.3). The classes can be defined and described in taxonomies and an ontology may use classes from different ontologies or taxonomies. Relationships between classes are known as properties and it is possible to define properties' cardinality. Each class and property used in an ontology is uniquely identified by a namespace prefix and the class or property specific name. For example, *m3-lite:PeopleCountSensor* is a class defined in the M3-lite ontology. For the sake of readability, in this paragraph we are omitting the namespace prefix while they are shown with prefix in Fig. 7.3.

The core concept is the *SensingDevice*, representing a sensor that produces *Observation*, which is a measurement (or computation) of a phenomenon related to an object happened at a specific *Instant.* For example, a crowd mobility detector can be seen as a *Device* composed of multiple *SensingDevices.* In this sense, such a detector can have one *PeopleFlowCountSensor* and one *StayingPeopleCountSensor*, which are subclasses of *PeopleCountSensor.* The *Observation*(s) is expressed with a *QuantityKind* having a *Unit.* Following our example, the *QuantityKind* associated to the data generated by the *PeopleFlowCountSensor* is *CountPeopleMoving* (subclass of *QuantityKind*) with *Item* as its *Unit* and with the *Direction* property expressed either in geodetic *DirectionAzimuth* or as a generic *DirectionHeading.* The directions start from the *Point* that is the *location* of the physical *Platform. Platform* is meant as the supporting dock to which the *Device* is attached. The *StayingPeopleCountSensor* generates *CountPeopleStaying* values expressed in *Item.* The system also consists of *PeopleStayDurationSensor* that generates *PeopleStayDurationAverage* values measured in *SecondTime.* Each *SensingDevice* might have a *Coverage*, specified either as *Polygon/Rectangle/Circle* or as a simple *Point.* This indicates the geographic extent of the *Observation.*

Integrated IoT Systems

Mobility Analytics System The CMAS (extended from our system in [96]) is integrated with the platform via semantic annotation of the outcome. The developed system consists of Wi-Fi sniffers, stereoscopic cameras, IoT gateways, and data analytics modules. The Wi-Fi sniffers are capable of capturing wireless probes broadcasted by mobile devices. Based on the captured Wi-Fi probes, the system can count the mobile devices in these sensing areas. The cameras are co-located with specific Wi-Fi sniffers deployed at the dedicated *calibration choke points.* A built-in people counting software runs in the cameras. Both Wi-Fi device detection and people counting results are reported to to the Cloud, where data analytics modules reside, through the IoT gateways. Three analytics modules are developed: *crowd estimation, people flows, and stay duration.* The *crowd estimation* module outputs number of people by correlating the stereoscopic camera counts and the number of Wi-Fi enabled devices at the calibration points. Based on the correlation between the two data modalities, the calibration of the data analytical results are applied in other sensing areas without cameras. The module monitoring *people flows* infers crowd movement in these areas. Finally, the *stay duration* module estimates the waiting times and the number of waiting people. All analytics results are exported to the Federated Cloud Infrastructure so the crowd analytics results are discoverable and available for applications in the smart city platform.

Crowd Counting and Location System Different from CMAS, CCLS aims at analysing crowd behaviour in public buildings of a city, as well as critical infrastructures. The system relies in the analyses of IEEE802.11 frames to discover devices in the surroundings of the deployment, normally within the monitored areas. Similar to CMAS, the deployed nodes capture “Probe Request” frames sent by smartphones, which include a Wi-Fi interface in “active search” mode, incorporated in most of them. However, CCLS does not only aim at detecting people, but also aims at locating them. For this, the system stores the RSSI and sequence number from the captured frames. It is possible to locate people by processing this information using RSSI-based algorithms. All the post-processing is performed in an edge server, where all the measurements are sent after the corresponding anonymisation techniques are applied. Once the anonymised raw measurements are analyzed and the counting and location analytics applied over them (i.e., the estimated crowd size and positions are obtained), these observations are semantically annotated and pushed to the Federated Cloud Infrastructure. For the semantic modelling, each crowd estimator is modelled as an *PeopleCountSensor*, with a specific *Coverage* (representing the area to which the estimations apply), that generates *CountPeople* observations expressed in *Item*.

Privacy Considerations

One of the essential requirements is dealing with tracked devices’ privacy. Nowadays, privacy is one of the major public concerns. In this sense, data protection laws have to be observed when handling data that could be personal. Quite restrictive rules apply in most countries of the world, being the countries from the European Union (EU) some of the most restrictive ones. These rules are recently updated through the General Data Protection Regulation (GDPR) [159] enforcement.

The Wi-Fi sensors in CMAS and CCLS deal with MAC addresses, which are considered personal data under the new EU regulation. As it is stated in the GDPR [159], “The principles of data protection should apply to any information concerning an identified or identifiable natural person”. Therefore, Wi-Fi-based tracking services in public or private spaces can be performed only if the service obtains the user’s opted-in permission, or data is anonymised in such manner that the user is no longer identifiable, as mentioned in the 26th article from the aforementioned regulation. The Article 29 Working Party, recently replaced by the European Data Protection Board (EDPB), is in charge of analysing the compliance of the privacy rules. In a document released to analyze the ePrivacy regulation compliance with GDPR [160], the Data Protection Working Party states that Wi-Fi tracking can only be performed either if there is consent or the personal data is anonymised. Within the same document, four conditions are mentioned for the latter case to be compliant with the GDPR:

- The purpose of the data collection from terminal equipment is restricted to mere statistical counting.

- The tracking is limited in time and space to the extent strictly necessary for this purpose.
- The data will be deleted or anonymised immediately afterwards.
- There exist effective opt-out possibilities.

Considering that user's permission request is impossible to obtain in normal conditions within the subject of the experimentation, the only option is to anonymise data regarding to MAC addresses. Thus, experimentation security measures must be undertaken to address both, data integrity and anonymisation. Therefore, any type of experimentation or service provision must take into account this concern, which is usually underestimated by system developers.

CCLS in Santander is based on the Spanish Personal Data Protection Laws and the Spanish Law Protection Office recommendations for data anonymisation [161]. The recommendation consists on the use of a cryptographic hash function with randomly generated hash keys. More precisely, the HMAC protocol, which provides such mechanisms, is recommended. In the SmartSantander deployment, we implement the HMAC algorithm along with the SHA256 hashing function, with a 12-bytes randomly generated key. Finally, in order to ensure a non-reversible process, this implementation also comprises a procedure to destroy and renew the key during specific session periods. For CMAS in Gold Coast, the hashed and salted Wi-Fi probe data is sent to the Cloud. The stereoscopic cameras do not record video or perform face detection. The cameras simply count the passage of people through predefined lines at the choke points. The outputs of the camera are people count-in and -out values. The main drawback of this procedure is the limitation of tracking devices throughout long periods (as in [162]) or longer travels within the city, but it is the price that must be paid to meet the privacy requirements.

7.1.2 Pilot Studies in Australia and Spain

Pilot Deployment in Gold Coast

Pilot Setup The deployments in Gold Coast include 17 Wi-Fi sensors and 2 stereoscopic cameras. The Wi-Fi sensors are custom-built devices for outdoor deployments. Two cameras are used at the calibration choke points, where there is a camera and a Wi-Fi sensor deployed together. The cameras are the Hella Advanced People Sensor APS-90E deployed at a height about 3.6 meters. Each camera is configured to capture the entire choke point for accurate counting.

The deployments target two regions. These sensors deployed in these areas are considered as *Cluster 1* for (expected) heavy pedestrian traffic and *Cluster 2* for light traffic places. Each cluster has a stereoscopic camera for the calibration. The collected data is sent to the Cloud where two virtual machines are created for the clusters. Clustering the areas allows applying CMAS to city-scale by sharing the raw data load.

Pilot Operation The pilot study activities started in September 2017 and CMAS has been in use starting from November 2017. Various types of pilot tests are conducted on the field during the operation of the pilot. Manual counting is performed using video footages taken from different deployment areas. In comparison to manual counting, the cameras provide an accuracy between 88% and 98%, which mainly depends on the weather and lightning conditions. Furthermore, field tests for heavy and light traffic areas resulted in 93% and 89% crowd size accuracy compared to manual counting. The results obtained from outside the choke points give further confidence to treat stereoscopic camera results as *near ground truth* as proposed in [96].

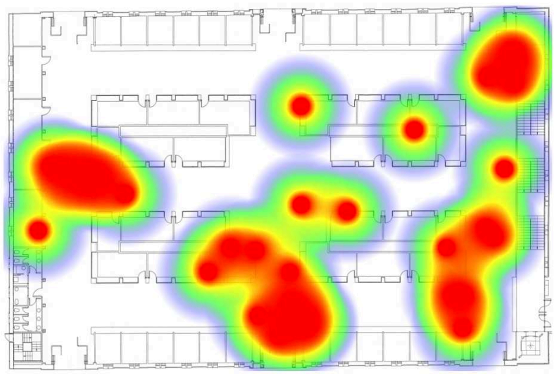


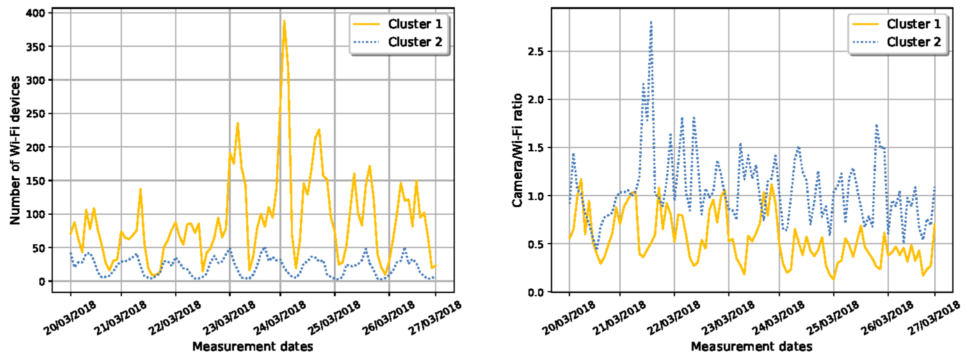
Figure 7.4: Heatmap from Mercado del Este in Santander.

Gold Coast pilot successfully tests the crowd mobility analytics services by leveraging federation of clusters and interoperability using the semantic model to share the results with stakeholders. This shows that similar systems can be developed and leveraged by future crowd management applications using the smart city platform.

Pilot Deployment in Santander

Pilot Setup CCLS is deployed in the "Mercado del Este" market, a restored symmetric building that contains shops, restaurants, a regional tourist office, and a museum. This building is particularly interesting as it usually receives significant numbers of visitors due to its central location, with exceptionally crowded periods.

The system is composed of 8 devices installed within the market building. These devices include a Wi-Fi interface aimed at detecting surrounding Wi-Fi enabled visitors' devices. Internet connectivity is provided through the Municipality Network, and the devices are powered using Power over Ethernet connected to the market's electrical grid. In addition to the wireless interfaces, half of the devices also include environmental sensors measuring temperature and humidity. Device deployment is carried out with the collaboration and supervision of the municipality and the market managers. Considering the main goal of monitoring people within the market, two parameters are considered in order to get market status snapshots over the time. First, the number of visitors within the market in different time frames and second, the location of the visitors in the different areas of the market.



(a) The number of unique Wi-Fi devices detected (hourly).

(b) Hourly changes of the camera/Wi-Fi ratios.

Figure 7.5: Weekly measurements from Gold Coast. Cluster 1: Pedestrian ways with heavy pedestrian traffic, Cluster 2: Roads including vehicles and light pedestrian traffic.

Pilot Operation Firstly, in order to monitor the visitors within the market, we follow a deterministic approach, in which we consider that a device is inside the building if a minimum of 6 sensor nodes detect it with a certain level of RSSI. In our deployment, this solution is feasible considering the particular symmetric distribution of the building and the location of the sensor nodes, covering the external wall of the building. Secondly, device locations are estimated using the Weighted Centroid Algorithm [163], which provides a reasonable approximation of 5 meters to the ground-truth measurements without any ad-hoc calibration. For the cases that require more precision, these positioning methods are able to introduce less than 2 meters error if the system is calibrated in advance. Synthesized information including real-time visitor location and detected number of visitors per unit of time is provided through a web portal to the market managers and municipality responsibles. Fig. 7.4 shows the heat map of the market in a specific moment. Other parameters, such as the visitors' dwell time in different long-term periods, are not analysed due to the privacy safeguards that have to be addressed.

7.1.3 City-Scale Experiments

This section discusses some of the experimental observations from the Gold Coast pilot with CMAS. Specifically, it includes the variance in the crowd estimation for the Wi-Fi sensors and cameras. Our focus in the experimental study is to observe the dynamic changes in the *number of unique Wi-Fi devices detected* and the *correlation coefficient* (or simply camera/Wi-Fi ratio), which is a dynamic parameter that is computed by the *Adaptive Linear Calibration Algorithm* [96]. The coefficient basically indicates the proportion of the number of people (count-in and count-out events) detected by the camera to the number of devices detected by the Wi-Fi sensors every time interval. We analyze the hourly results for the two clusters, where 5 minute time intervals are aggregated and averaged for 1 hour.

Figure 7.5-a shows the average number of Wi-Fi devices detected for one-week period. There exists an increased activity in Cluster 1 region especially during Friday (23/03/2018) and the following weekend. This can be due to crowdedness in the shopping street and the beach area contained in this region. Moreover, there is a peak in Saturday that can be due to an event or gathering. Figure 7.5-b shows the change of the coefficients (ratios). The ratios are computed at the calibration choke points (providing near-ground truth for the measurements). The hourly ratio is computed such that number of people count-in and count-out events are divided to the number of Wi-Fi probes. First, for Cluster 2 with light traffic, correlation coefficient is mostly (almost all days) higher compared to Cluster 1. Second, correlation coefficient values lie mostly in the range of (0.2, 2), whereas the peak value is about 2.8. This indicates that the results based on Wi-Fi-only measurements are likely to have less accuracy most of the times of the days and the correlation changes throughout the days. Lastly, there exists certain regularity in the correlation from one day to another, which can be learned through a time period and then applied to other time periods where camera is temporarily inactive or removed. On the other hand, as seen in the peak hours of Cluster 2, the ratios do not lie within a narrow range. One reason can be events affecting the volume of pedestrians. Lastly, Fig. 7.5-b shows relatively higher variance of the coefficient for areas with light pedestrian traffic. Calibration could be necessary for shorter time intervals.

Overall, it is observed that effective use of Wi-Fi sensing and combining them with sensing by stereoscopic cameras produce accurate sensing in large scale for both the heavy and light pedestrian traffic areas. Moreover, the variance between heavy and light traffic shows the usefulness of the clustering approach which treats these regions separately.

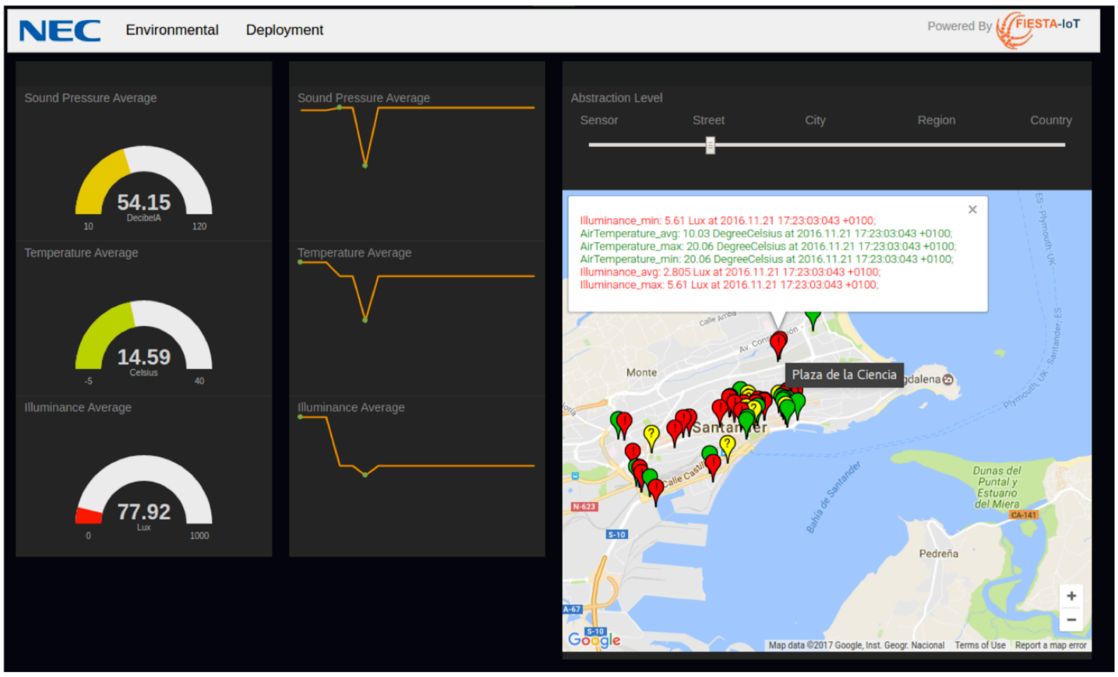
7.1.4 Related Work

There are recent studies that focus on understanding of human mobility through IoT devices such as wireless sensors. Jara et al. [164] observed the relation between traffic behavior and temperature conditions as a smart city application through deployment of IoT devices in Santander. Tong et al. [165] propose usage of Wi-Fi sensors to understand passenger flows. Evaluation through simulation results shows high accuracy. Zhao et al. [166] survey the recent advances in understanding human mobility in urban environments. The study lists some of the existing urban human mobility datasets collected such as GPS, GSM, Wi-Fi, and Bluetooth traces. Similarly, Zhou et al. [167] discuss the topic of human mobility in urban environments and present a taxonomy of crowdsensed input data types and application outcomes such as crowd density and flows within building, and people transportation mode identification (cycling, running, bus riding). Lastly, Montjoye et al. [162] focus on the privacy aspect by analyzing long period Wi-Fi traces and show that 95% of the individuals can be uniquely identified using spatiotemporal datasets.

7.2 Smart City Magnifier: a portable application on hyperconnected IoT

As an example of hyperconnected IoT service that makes use of cross-cloud interoperation, we have implemented a smart city application, named Smart City Magnifier (SCM). The application purpose is to analyse the situation of generic geographic areas and compute indicators representing the health of the urban and natural environments. The outcome of the indicators are then showed into a dashboard that highlights the critical situations such as emergencies (e.g., fire breaks), high crowd densities, and urban and rural pollution. The data is harvested from data sources belonging to different IoT domains hosted by separated IoT clouds and going through a process of associating observations to real-world things (e.g., a city or a neighbourhood) and then analyse the situations related to those things. The presented implementation processes data from multiple and heterogeneous IoT systems across 16 cities in the world.

7.2.1 Application Scenario



Illuminance_min: 5.61 Lux at 2016.11.21 17:23:03:043 +0100;
AirTemperature_avg: 10.03 DegreeCelsius at 2016.11.21 17:23:03:043 +0100;
AirTemperature_max: 20.06 DegreeCelsius at 2016.11.21 17:23:03:043 +0100;
AirTemperature_min: 20.06 DegreeCelsius at 2016.11.21 17:23:03:043 +0100;
Illuminance_avg: 2.805 Lux at 2016.11.21 17:23:03:043 +0100;
Illuminance_max: 5.61 Lux at 2016.11.21 17:23:03:043 +0100;

Figure 7.6: Smart City Magnifier application dashboard

The envisioned scenario is a city applications analyzing and reporting situations of generic urban area with different level of details . The situations are related to multiple degrees of freedom such as: i) arbitrary geographic scope and, ii) entities abstraction. In the first case, situations classification of a given geographic bounding box are computed discovering meaningful data items on the knowledge graph. The geographic scope is selected by the user, for instance, by focusing a graphical map widget. In Figure 7.6 the geographical scope situations are visualized by the leftmost columns of gauge widgets and the related timeseries plots. In the second case, situations are made by iteratively abstracting geographic things (i.e. entities). The lowest is the sensor level where the situations are referred to items making observations, that for simplicity we call sensors. Examples of sensors situations might be temperature (i.e. too low, too high) or correctness (e.g. reporting the exact same value in the past 2 days). Higher levels of abstraction might be building, street, neighbourhood, city, region and country level. These higher level things might geographically encompass sensors or observations (e.g. social media posts) to which were not originally associated. For example sensors observations might be linked at the same time to a building, a street and so on. City situations analytics tasks infer environmental situations in different part of a urban area (e.g., per street, per neighbourhood, per city), or emergency situations such as fire breaks or flood. The knowledge graph, in this case, is identifying for which entities a situations can be computed and a the same time which are the related observation to take into consideration. The situations are then symbolized with a traffic-light schema (red for alert, yellow for a warning and green for a good status) and drawn in the map (see Figure 7.6). The different abstraction levels is then selected by the top sliding bar which will then changes the pins in the map widget. Different views of situations (e.g. environmental, deployment) are selectable by tabs. The whole applications is conceived to work in an agnostic manner regarding the IoT data sources and leverage the linked data knowledge.

7.2.2 Semantic Interoperability for Information Transparency

An example of a semantic approach to IoT interoperability is the combination of open standards such as OMA NGSI with the semantic concept of ontology (see Figure 7.7) such as the one developed within the FIESTA-IoT EU project[1] [26] which is a synthesis of known ontologies and taxonomies (such as Semantic Sensor Network-SSN[2], M3-lite[3], IoT-lite[4]) with the aim of covering the contexts generated by multiple heterogeneous IoT deployments. The ontology is conceived to deal with resources description and observations.

In an ontology, rich and complex knowledge is represented as a graphs of nodes, expressing things, and links between nodes that express the relationship between two things. Things are categorized in classes and every thing is an instance of a class. A class may have sub-classes, and an instance of a class is also an instance of all the super-classes of such class. Classes and relationships are specified into taxonomies and an ontology can use classes and relationships from one or more taxonomies. Each class and property used

[1]Federated Interoperable Semantic IoT Testbeds and Applications: FIESTA-IoT. Available online: `http://fiesta-iot.eu/` (Accessed on the 15th of December 2020)

[2]Semantic Sensor Network - SSN. Available online: `https://www.w3.org/TR/vocab-ssn/` (Accessed on the 15th of December 2020)

[3]M3-lite. Available online: `http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.html` (Accessed on the 15th of December 2018)

[4]IoT-lite. Available online: `https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/` (Accessed on the 15th of December 2020)

into an ontology is uniquely identified by a namespace prefix and the class or property specific name. For example, *ssn:Sensor* is a class defined in the SSN ontology. For sake of readability, in this paragraph, we are omitting the namespace prefix, but they are all represented in Figure 7.7. In the FIESTA-IoT ontology, the *Observation* is the core element that bridges the physical phenomena sensed in the M3-lite taxonomy[5] via the *QuantityKind* (e.g., *Temperature*, *CountPeopleMoving*). This taxonomy is fundamental to cover the many different sensors of various IoT deployments. Each *Observation* has a timestamped (*Instant*), a location (*Point*), a unit *Unit*, observed by a sensor (*Sensor*) that is the superclass for all kind of sensors as defined in the M3-lite taxonomy (e.g., *Thermometer*, *PeopleFlowCountSensor*).

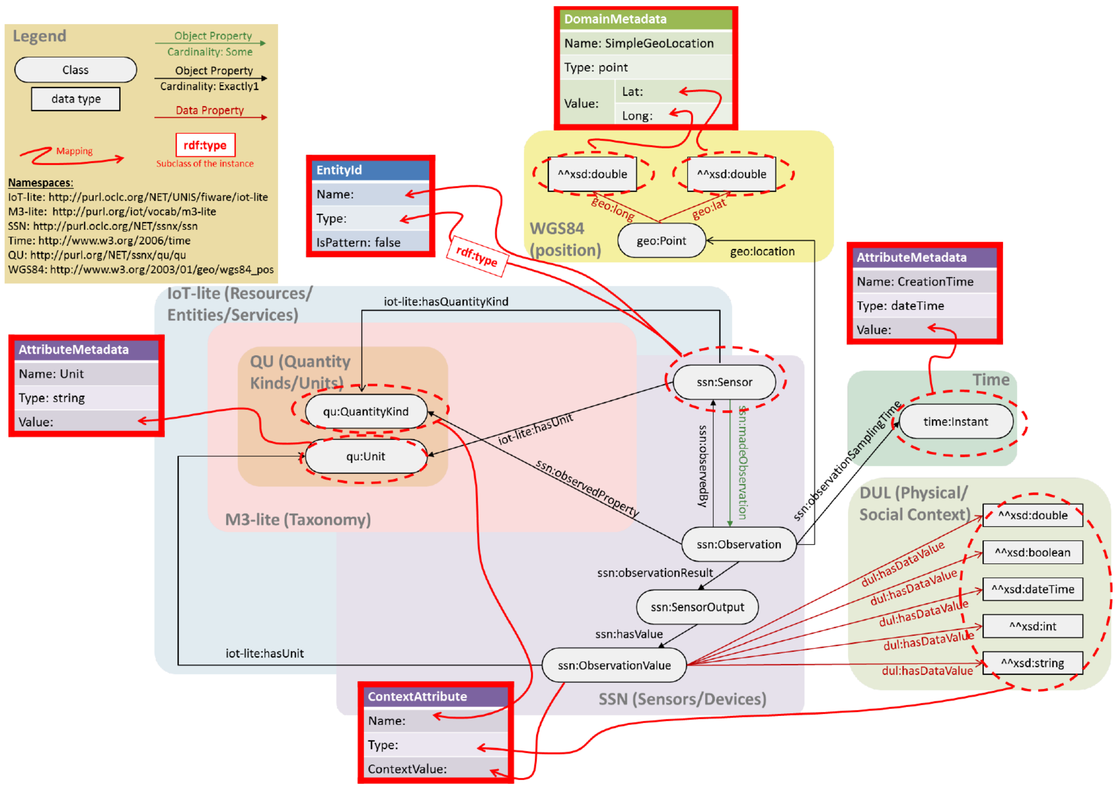


Figure 7.7: Combining the context management interface standard OMA NGSI with semantics offered by the FIESTA-IoT ontology.

In the NGSI standard, each piece of NGSI context data refers to an identifier of the entity, namely *EntityId* and reports several *ContextAttributes*. EntityIds might have one or more *DomainMetadata* which specify metadata in common to all the attributes (e.g., the location of the sensed attributes). In addition, each *ContextAttributes* might have multiple *AttributeMetadata* that are specific to such attribute.

[5]M3-lite taxonomy. Available online: `http://purl.org/iot/vocab/m3-lite` (Accessed on the 15th of December 2020)

The mapping of the FIESTA-IoT ontology with the NGSI format is shown in Figure 7.7. The instance of the ssn:Sensor is used as the Name of the EntityId, whereas the specific class of such instance is set as its Type, the *isPattern* (that is used to indicate whether EntityId identifies a range of entities) is set always to false. Each ssn:Observation is uniquely linked to a ContextAttribute that takes the qu:QuantityKind as Name, the class of dul:hasDataValue as Type and the ssn:ObservationValue as ContextValue. Furthermore, each ContextAttribute has one AttributeMetadata for the qu:Unit and another one for the time:Instant. The location of the sensor is stored as a DomainMetadata for all the attributes.

OMA NGSI is currently under study of the Context Information Management (CIM) working group of the European Telecommunications Standards Institute (ETSI) for evolving in a new version that incorporates linked data concepts is and formatted in NGSI-Linked Data (NGSI-LD) [89]. Therefore, it is allowed, if not even recommended, the usage of an ontology in combination with the context management.
The new ETCI CIM standard suggests a porting from legacy NGSI to the new NGSI-LD. Further, the FIWARE data models are already ported as an ontology[6]. We have used such directions to port the Smart City Magnifier on NGSI-LD.

7.2.3 Smart Cities Enabled by Future Hyperconnected IoT

Figure 7.8 shows the concrete implementation architecture of the Smart-City Magnifier application. Data are acquired by many Semantic Mediation Gateways (SMGs) that have the purpose of semantically translating from a format (protocol and data model) to another format. In this application, SMGs transform data to NGSI format modelled with FIESTA-IoT ontology. The integrated IoT deployments are either coming from the FIESTA-IoT system [168, 92], from crowd mobility IoT deployments in New Zealand, from the smart city of Murcia offered by the SMARTIE EU project [169], or from the cities complying with SynchroniCity (section 3.2). The deployments behind the FIESTA-IoT framework are exposing the IoT data either through the FIESTA-IoT historical triple store accessible via SPARQL Protocol and RDF Query Language (SPARQL) queries or via exposed IoT endpoints, discoverable again through a SPARQL query, for every available sensors. In both cases, the data format is plain JSON-Linked Data (JSON-LD) annotated with the FIESTA-IoT ontology. Through the FIESTA-IoT platform, more than 7500 sensors or resources could be accessed from 10 testbeds reporting environmental data (e.g., outdoor temperature, humidity, particles concentration, luminosity, noise level), road traffic monitoring (e.g., vehicle speed, traffic intensity), car and bike parking spots, public transportation status (e.g., bus estimated arrival times, vehicle localization), garbage management, soil and trees monitoring (urban parks and garden, rural areas), pedestrian presence detector, electromagnetic outdoor exposure, smart building/office information (e.g., human presence, power consumption, heating ventilation and air conditioning-HVAC system, solar panels), signal power and power consumption of wireless sensors, and sea water quality (e.g., pH, Ammonium). The

[6]https://fiware.github.io/data-models/context.jsonld

crowd mobility deployments from New Zealand are instead exposing the data via an ad hoc JSON format. The Murcia smart city data, comprehensive of 430 city wide urban mobility sensors (e.g., traffic sensors, bike and car parking spot sensors, public transportation status sensors) and 25 smart building sensors in the university campus, can be accessed via the NGSI interface, which is following an ad-hoc data format defined by the specific IoT silo owner. Finally, SynchroniCity cities exposes data as NGSI modelled with FIWARE data models [27].

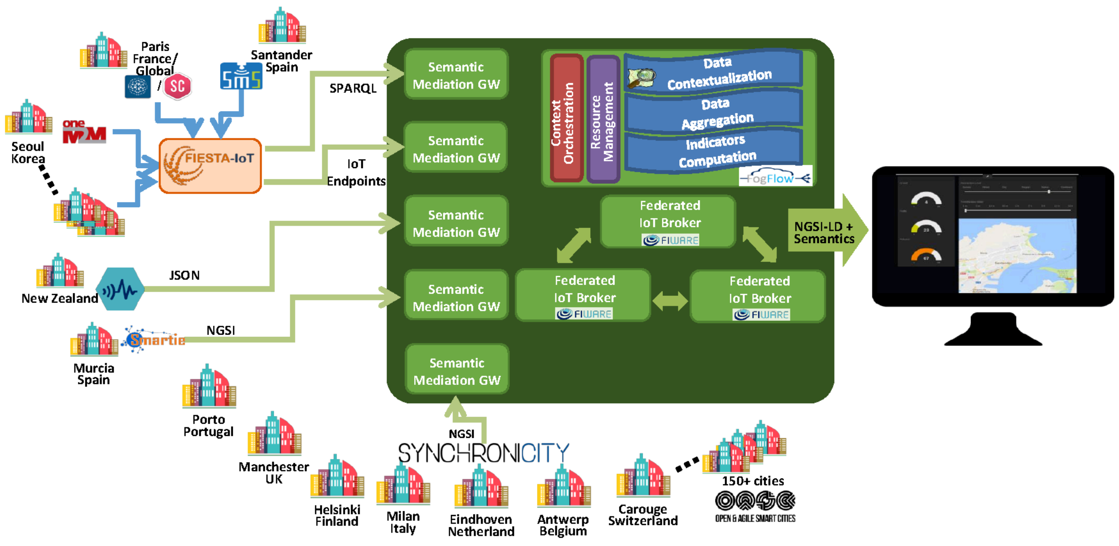


Figure 7.8: Realization of the Smart-City Magnifier application leveraging *semantic mediation*, *semantic interoperability*, *resource orchestration* and *federation of IoT platforms*.

The interoperable data is then, on the one hand, handled on multiple IoT Broker instances federated in a topology of peers, and, on the other hand, orchestrated by the *FogFlow* [3] framework which offers a service-defined analytics layer. The federation of brokers is meant both to enhance the scalability of the whole system, since the data coming from the IoT deployments are sharded among them, and to differentiate IoT domains. Any of the brokers is able to provide all the available data in the federation through a single NGSI-LD request, necessitating sometimes of a multi-hop connection, transparently executed, if data are not directly provided by the contacted broker. The data analytics instead is made of atomic tasks handled and instantiated by the FogFlow framework which manages the available computation resources and orchestrates the data stream between tasks. Since the future smart cities take place in a digital real-time world, real-time information fusion through the hyperconnected IoT framework are required to provide contextualized information to the different levels for executing necessary actuations. A first task performs *Data Contextualization* that associates virtual entities to the incoming observations. Contextualizing, in this scope, is the act of inferring the real-world things (e.g., a building, a street, a square, a suburb, a city etc.) to which each geotagged observation belongs, making usage of the knowledge storage offered by external knowledge bases such as OpenStreetMap[7]. One observation might be associated to one or more real-world things, for instance, an observation of outdoor temperature sensor attached to a building wall can be associated to such building, to the street where the building is located, to the neighbourhood and to the city. At the same time, one thing can have more than one observations associated, for example, if two temperature sensors are located on two buildings in the same street, such street has two sources of temperature observations. Since the data contextualization does not need raw data from other sources, it can be executed on the edge or on the cloud specific of the data source domain without accessing the federation. The *Data Aggregation* groups the incoming observations by the inferred things. The aggregation makes usage, for instance, of statistics means. The *Indicators Computation* is calculating smart city Key Performance Indicators (KPIs) [170] assessing the status of city and identifying critical situations. The latter two data analytics tasks necessitates cross-domain data and therefore they make data requests trough the federation.

The output of the analytics is handled again by the federated IoT Brokers. The dashboard is then acquiring the information and visualizing it on a map presenting the alerts and warnings with the means of several visualization widget tools. The same visualization dashboard is used for visualizing all the situations inferred from the IoT silos integrated in Europe, Korea and New Zealand, with a total of 16 city areas (Guildford in the United Kingdom, Santander and Murcia in Spain, Seoul in South Korea, Toulouse and Paris in France, Heraklion and Volos in Greece, Waterford in Ireland, Milan and Minervino Murge in Italy, Wellington and Christchurch in New Zealand, Porto in Portugal, Helsinki in Finland, Carouge in Switzerland). Since the analytics processing tasks are working on homogeneous semantically annotated data, the geographic scope of the dashboard map can simply be navigated to focus on a different city.

7.2.4 Cloud-Edge stream processing

FogFlow (as seen in chapter 2.2) is a distributed execution framework to support dynamic processing flows over cloud and edges. It can dynamically and automatically composite multiple NGSI-based data processing tasks to form high level IoT services, and, then, orchestrate and optimize the deployment of those services within a shared cloud-edge environment, with regards to the availability, locality, and mobility of IoT devices. Processing tasks are packed in docker[8] container and are automatically pulled by the system from a shared Docker repository (e.g. DockerHub[9]).

[7]OpenStreetMap project. Available online: `https://www.openstreetmap.org/` (accessed on 2nd of December 2020)

[8]https://www.docker.com/

[9]https://hub.docker.com/

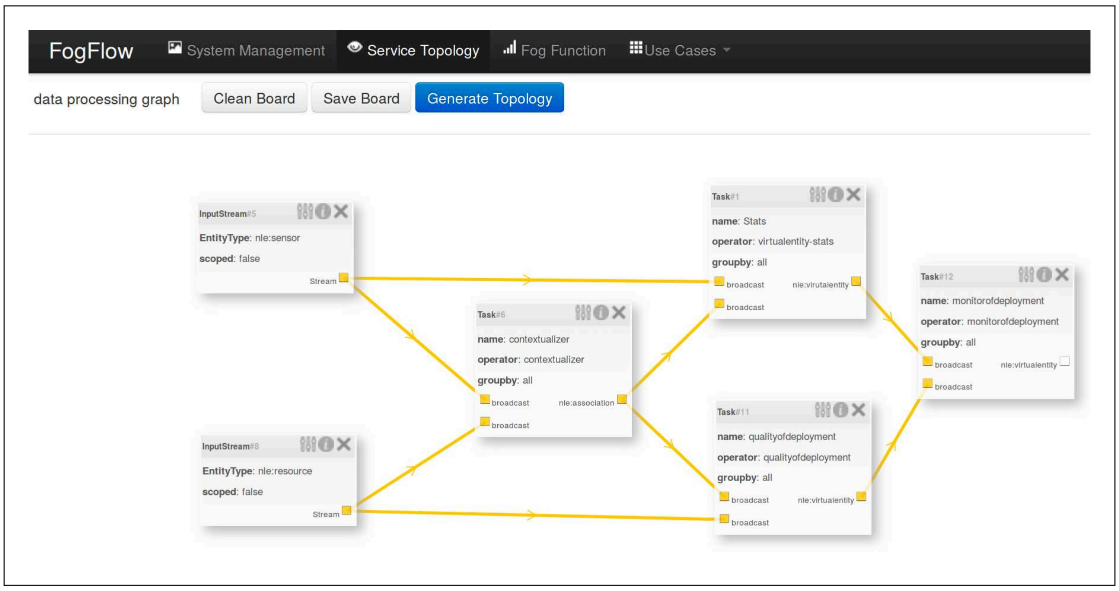


Figure 7.9: SCM service topology designed through the FogFlow graphic user interface.

The processing tasks are placed in a topology similar to the one shown in Fig. 7.9, where:

- *Two input streams* enable the data flow of resources' observations and resources' status information in the analytics tasks

- *Contextualizer task* associates virtual entities, making usage of the NGSI Nominatim service, to the incoming observations flowing. Contextualizing, in this scope, is the act of inferring the location context (e.g. a building, a street, a square, a suburb, a city etc.) to which each geotagged observation belongs. We implemented the *NGSI Nominatim* an NGSI service that exposes an NGSI interface to request the associations of a geographic point (formed by a latitude and longitude) to a set of real geographic items touched by such point. The system is using the OpenStreetMap Nominatim[10] we service as remote service.

- *Aggregator task* (named Stats) aggregates the incoming observations by the virtual entities. The aggregation makes usage of simple statistics means, such maximum, minimum, average.

- *IoT quality of deployment task* (named qualityofdeployment) calculates the quality of deployment (in the form of number of resources, geographical density of resource, etc.,) for each of the virtual entities contextualized.

- **IoT deployment monitor task** monitors the amount of observation coming from resources from each the virtual entities contextualized. It takes as input statistics

[10]https://nominatim.openstreetmap.org

of the observations for each of the virtual entities (output of the stats task) and the quality of deployment parameters (output of the qualityofdeplotment task) and monitor the activity of the sensor through the time.

7.3 Conclusions

The presented crowd mobility work focuses on finding insights behind crowd mobility such as detecting crowdedness. However, understanding more complex crowd mobility behaviour in a large-scale city area such as movements of groups [171] (e.g., family) could be helpful for crowd management and enhancing smart mobility in the cities. The collected mobility information can serve as input of human mobility simulations to further study how city dynamics are affected by crowd mobility patterns. With the combination of real mobility dataset in a simulated environment, learning new mobility insights opens up new opportunities for new crowd management strategies (e.g., congestion avoidance, evacuation planning, demand management) that can further improve the public service and safety in smart cities. In future developments, the semantic interoperability through ontologies can be leveraged more extensively for cross-infrastructure communication and knowledge sharing. The new advancements of the NGSI-LD protocol are centered around the concepts of linked data. This opens a new horizon where knowledge graphs are shared among various infrastructures and, while their administrators own the produced data, it is still accessible seamlessly and transparently by all actors in the multi-infrastructure federation.

The Smart City Magnifier application shows the possibility to develop data analytics services on top of an information transparent system where data is seamlessly available through a common interface. We make usage of the software-defined services approach and design analytics part as a topology of atomic processing tasks. We also integrate external knowledge base, such as OpenStreetMap, through the same NGSI data interface. The application relies on the FogFlow orchestration to deploy analytics task when needed. The data coming from the diverse data providers are handled on different context managements that are federated among them. In future developments, we will move more towards a context-based serverless approach [172]. For instance, the conditions check to color the indicators green, yellow, or red happens at the visualization time. This checks might run on the back-end continuously and triggered on the occurrence by the service orchestration.

Chapter 8

Conclusions and Future Work

In this thesis, we have studied the evolution of Internet-of-Things towards a global interconnection and interworking of heterogeneous devices, systems and services. We have provided key technical solutions to enable the hyperconnected IoT vision and we have reported experiences to apply them on real world multi-party scenarios. We have also discussed standards and open sources aspects on both 5G and IoT that we have adopted and expanded throughout the thesis as basis to achieve a viable data sharing.

Overall, in this thesis we have focussed on four main topics: i) decentralization of an IoT deployment across the whole service computing stack such as edge, fog and cloud; ii) federation of (decentralized) IoT systems managed by diverse parties; iii) enforcing data usage control throughout the federation of decentralized IoT systems; iv) realizing real hyperconnected IoT services and an IoT services ecosystem among 27 cities across Europe, Asia and Oceania. We horizontally leveraged several technical enablers such as service-defined data analytics, comprehensive of elastic edge-cloud orchestration, information transparency, data contextualization, and semantic discovery.

The experiments conducted on the several proposed solutions show that the path towards an hyperconnected IoT is not only beneficial for the economy [32] but also viable and, even, advantageous in terms of performances, scalability, and reduced services management complexity. Further, in this thesis we extensively demonstrate that a context-based data management approach for IoT is very beneficial for IoT services development, deployment and management. The proposed mechanisms are well fitting with ETSI MEC and the FIWARE ecosystem, and readily to be applied and exploited.

We can summarize the contribution of this thesis as follows:

- In chapter 2 we present a novel approach to optimally handle IoT deployments on a 5G infrastructure. The basic idea is the synergistic cooperation of the IoT tenant and the 5G operator to trade-off resources among slices and avoid service degradation. In addition, we explore the correlated fog computing topic and propose a programming model to decouple the IoT service development to the orchestration among cloud and edges. Further, the services are also decoupled to the context management and the data discovery and provisioning is performed transparently by the proposed system, namely FogFlow.

- In chapter 3 we analyze the benefits that a Multi-access Edge Computing (MEC) platform might bring to a well developed smart city deployment such as Smart Santander. To address the identified requirements, we propose an ETSI MEC-based architecture that facilitates IoT networks and services deployments but also encompasses an IoT gateway middleware for handling high-computation applications with low-latency requirements. We also investigate FIWARE, one of the most prominent standard-based open source IoT platforms, presenting how it is used in three different real world scenarios reporting lessons learnt from these experiences. We, then, compare FIWARE with commercial and other open sources solutions assessing its strengths in terms of information transparency, semantic mediation, and services handling, and its readiness towards commercial exploitations.

- In chapter 4 we propose an architecture to federate decentralized IoT platforms managed by multiple parties. The proposed architecture is scalable by design allowing federation of federations, thanks also to the two-levels approach for the security and context management. The data managed through a context-oriented approach, and data flows establishment happens transparently from the data consumers and data providers perspectives. The experiments show that the overhead for enabling a hyperconnection of systems is much lower in case of big amount of data exchanges. In addition, the results demonstrate that the federation allows the performances scalability of IoT systems when the size of IoT scenarios increases.

- In chapter 5 we introduce a novel data usage control system, namely IntentKeeper, adopting a service-defined data analytics approach. The proposed system simplifies the data usage policies definition for the data providers and policies-compliant data analytics services definition for the data consumers into highly dynamic scenario such as hyperconnected IoT. By combining service orchestration, context management, and policy enforcement, IntentKeeper enforces data usage control preventively and proactively.

- In chapter 6 we present the methodology to develop a shared ecosystem for IoT services involving different stakeholders. The methodology relies on a FIWARE framework bringing information transparency due to the semantic interoperability of the FIWARE data models. We applied the methodologies among 35 city services piloted into 27 different cities.

- In chapter 7 we shows examples of smart city services that leverage the global interconnection of IoT resources. We develop new concepts of crowd mobility that uses data seamlessly from different providers. In addition, we present a smart city generic monitoring application that transparently uses data from 16 city deployments to infer situations of the real world and associate them to real things. We show also how to implement such service using the FogFlow programming model relying on it for the service orchestration.

With this thesis we have addressed several aspects towards an hyperconnected data sharing, however there are future works that the author of this thesis plan to follow as next steps.

Data sharing is beneficial for the data analytics ecosystem and our intent-oriented data usage control approach greatly facilitates to handle the complexity of multi-party and distributed systems. However, we have assumed that the actual data owner and the data provider are the same role. This is not always the case and might keep exposed the problem of personal data handling for companies (e.g., data covered by the GDPR). Another open challenges is the *transitivity* of usage control: how to transfer the original data provider's (data owner's) rights onto the processed data and how to determine when to stop the rights transferring on following data processing. The data consumers, that spend efforts and costs to process data, need to have a clear view on the carry over restrictions on the processed data.

Another open challenge for the data sharing is the data *traceability.* A retrospective on how the data has been used and from where the processed data comes is important for both the data producers and data consumers. Data producers wants to have transparency on where the their data has been used in order to auditing the actual enforcements of their policy, or even to apply policies not yet specified at the time the data was shared (e.g., GDPR principle of the "right to be forgotten") [173]. Data consumers desire to have full understanding on the impact on their business of every usage of data and the possibility to check "what-if" conditions in case of changes of policies. In this thesis we have moved some steps towards the traceability direction with the usage of distributed ledger technologies, but a deep study on the traceability model integrity is needed.

Data sharing promises to have positive impact on the economy if adopted in a shared ecosystem (e.g., possibility to sell data, reduction of costs for service development) but this has not yet happened in the reality. A known obstacle is the effort to integrate systems. In this thesis we have approached the semantic mediation between systems mostly with ad-hoc Semantic Mediation Gateways (SMGs). Such an approach is clearly not scalable for a global IoT . However, a good burst towards a realization of an hyperconnected IoT might be automatic means of *ontology matching* [174] perhaps taking advantages of AI techniques.

References

[1] Flavio Cirillo, Fang-Jing Wu, Gürkan Solmaz, and Ernö Kovacs. Embracing the future internet of things. *Sensors*, 19(2), 2019.

[2] V. Sciancalepore, F. Cirillo, and X. Costa-Perez. Slice as a Service (SlaaS) optimal IoT slice resources orchestration. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–7, Dec 2017.

[3] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa. Fogflow: Easy programming of iot services over cloud and edges for smart cities. *IEEE Internet of Things Journal*, 5(2):696–707, April 2018.

[4] Lanfranco Zanzi, Flavio Cirillo, Vincenzo Sciancalepore, Fabio Giust, Xavier Costa-Perez, Simone Mangiante, and Guenter Klas. Evolving multi-access edge computing to support enhanced iot deployments. *IEEE Communications Standards Magazine*, 3(2):26–34, 2019.

[5] F. Cirillo, G. Solmaz, E. L. Berz, M. Bauer, B. Cheng, and E. Kovacs. A standard-based open source iot platform: Fiware. *IEEE Internet of Things Magazine*, 2(3), Sep. 2019.

[6] F. Cirillo, N. Capuano, S. P. Romano, and E. Kovacs. Liots: League of iot sovereignties. a scalable approach for a transparent privacy-safe federation of secured iot platforms. In *2019 IEEE 44th Conference on Local Computer Networks (LCN)*, pages 226–229, 2019.

[7] Flavio Cirillo, Bin Cheng, Raffaele Porcellana, Marco Russo, Gurkan Solmaz, Hisashi Sakamoto, and Simon Pietro Romano. IntentKeeper: Intent-oriented Data Usage Control for Federated Data Analytics. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, Nov. 2020.

[8] F. Cirillo, D. Straeten, D. Gómez, J. Gato, L. Diez, I. E. Maestro, and R. Akhavan. Atomic services: sustainable ecosystem of smart city services through pan-european collaboration. In *2019 Global IoT Summit (GIoTS)*, pages 1–7, June 2019.

[9] Flavio Cirillo, David Gómez, Luis Diez, Ignacio Elicegui Maestro, Thomas Barrie Juel Gilbert, and Reza Akhavan. Smart city iot services creation through large scale collaboration. *IEEE Internet of Things Journal*, 2020.

[10] Gurkan Solmaz, Fang-Jing Wu, Flavio Cirillo, Erno Kovacs, Juan Ramón Santana, Luis Sánchez, Pablo Sotres, and Luis Munoz. Toward understanding crowd mobility in smart cities through the internet of things. *IEEE Communications Magazine*, 57(4):40–46, 2019.

[11] Didier Helal, Samuel Dubouloz, Jorgen Mortensen, Paolo Baldi, and Paul Royo. Real time 3d environmental noise monitoring and mapping using large-scale internet of things. In *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, volume 259, pages 7937–7948. Institute of Noise Control Engineering, 2019.

[12] M. Hultermans et al. SynchroniCity D3.1 - Specification and design of initial IoT applications. Technical report, SynchroniCity, Jan. 2018.

[13] Panu Maijala, Zhao Shuyang, Toni Heittola, and Tuomas Virtanen. Environmental noise monitoring using source classification in sensors. *Applied Acoustics*, 129:258–267, 2018.

[14] GAIA-X project. GAIA-X: Driver of digital innovation in Europe. Featuring the next generation of data infrastructure. Technical report, 2020.

[15] F. Wu, G. Solmaz, and E. Kovacs. Toward the future world of internet-of-things. In *2018 Global Internet of Things Summit (GIoTS)*, pages 1–6, June 2018.

[16] International Data Spaces Association (IDSA). IDS Reference Architecture Model - v2.0. Technical report, 2018.

[17] Bin Cheng, Apostolos Papageorgiou, Flavio Cirillo, and Martin Bauer. Geelytics: Enabling on-demand edge analytics over scoped data sources. In *IEEE International Congress on Big Data*, June 2016.

[18] 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 Hans-W. Gellersen, editor, *Handheld and Ubiquitous Computing*, pages 304–307, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[19] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[20] Ovidiu Vermesan and Peter Friess. *Building the hyperconnected society: Internet of things research and innovation value chains, ecosystems and markets*, volume 43. River Publishers, 2015.

[21] Karolj Skala, Davor Davidovic, Enis Afgan, Ivan Sovic, and Zorislav Sojat. Scalable distributed computing hierarchy: Cloud, fog and dew computing. *Open Journal of Cloud Computing (OJCC)*, 2(1):16–24, 2015.

[22] Salvatore Longo and Bin Cheng. Privacy preserving crowd estimation for safer cities. In *UbiComp workshop on Smart Cities: People, Technology and Data*, pages 1543–1550, September 2015.

[23] oneM2M. oneM2M Technical Specification (Document number: TS-0022-V2_3_1). Technical report, 3 2018.

[24] Open Mobile Alliance. Open Mobile Alliance, Next Generation Service Interfaces (NGSI) - OMA TS NGSI Context Management - V1.0-20120529-A. http://www.openmobilealliance.org/release/NGSI/V1_0-20120529-A/ OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf.

[25] E. Kovacs, M. Bauer, J. Kim, J. Yun, F. Le Gall, and M. Zhao. Standards-based worldwide semantic interoperability for iot. *IEEE Communications Magazine*, 54(12):40–46, December 2016.

[26] R. Agarwal, D. G. Fernandez, T. Elsaleh, A. Gyrard, J. Lanza, L. Sanchez, N. Georgantas, and V. Issarny. Unified iot ontology to enable interoperability and federation of testbeds. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 70–75, Dec 2016.

[27] FIWARE Found. FIWARE data models.

[28] Raphaël Ventura, Vivien Mallet, Valérie Issarny, Pierre-Guillaume Raverdy, and Fadwa Rebhi. Estimation of urban noise with the assimilation of observations crowdsensed by the mobile application ambiciti. pages 5444–5451, 2017.

[29] Denise Lund, Carrie MacGillivray, Vernon Turner, and Mario Morales. Worldwide and regional Internet of Things (IoT) 2014-2020 forecast: A virtuous circle of proven value and demand. Technical report, International Data Corporation (IDC), May 2014.

[30] Rob van der Meulen and Viveca Woods. Gartner says smart cities will use 1.6 billion connected things in 2016. Technical report, Gartner, Inc., December 2015.

[31] Microsoft. What's new with the data culture at Microsoft. Technical report, Microsoft Corporation, August 2015.

[32] PWC. Data exchange as a first step towards data economy. 2018.

[33] Fatih Kilic and Claudia Eckert. iDeFEND: Intrusion detection framework for encrypted network data. In *Proceedings of the 14th International Conference on Cryptology and Network Security (CANS'15)*, volume 9476 of *Lecture Notes in Computer Science*, pages 111–118. November 2015.

[34] Watts N. et al. The 2018 report of the lancet countdown on health and climate change: shaping the health of nations for centuries to come. *The Lancet*, 392(10163):2479 – 2514, 2018.

[35] Gurkan Solmaz and Damla Turgut. Pedestrian mobility in theme park disasters. *Communications Magazine, IEEE*, 53(7):172–177, July 2015.

[36] Daniel Iland and Elizabeth Belding. EmergeNet: Robust, rapidly deployable cellular networks. *Communications Magazine, IEEE*, 52(12):74–80, December 2014.

[37] M. Deruyck, J. Wyckmans, L. Martens, and W. Joseph. Emergency ad-hoc networks by using drone mounted base stations for a disaster scenario. In *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)(WIMOB)*, volume 00, pages 1–7, Oct. 2016.

[38] Ling Li. China's manufacturing locus in 2025: With a comparison of "made-in-china 2025" and "industry 4.0". *Technological Forecasting and Social Change*, 135:66 – 74, 2018.

[39] NGMN Alliance. Description of the Network Slicing Concept. *NGMN 5G P1*, Jan. 2016.

[40] K. Samdanis, X. Costa-Perez, and V. Sciancalepore. From Network Sharing to Multi-tenancy: The 5G Network Slice Broker. *IEEE Communications Magazine*, 54(7):32–39, July 2016.

[41] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs. Mobile Traffic Forecasting for Maximizing 5G Network Slicing Resource Utilization. In *INFOCOM'17*, May 2017.

[42] C. Pereira, A. Pinto, D. Ferreira, and A. Aguiar. Experimental Characterisation of Mobile IoT Application Latency. *IEEE Internet of Things Journal*, 2017.

[43] L. Toka et al. A Resource-Aware and Time-Critical IoT Framework. In *INFOCOM'17*, May 2017.

[44] J. S. Leu et al. Improving Heterogeneous SOA-Based IoT Message Stability by Shortest Processing Time Scheduling. *IEEE Transactions on Services Computing*, 7(4):575–585, Oct 2014.

[45] M. A. Nef et al. Enabling QoS in the Internet of Things. In *CTRQ 2012: The Fifth International Conference on Communication Theory, Reliability, and Quality of Service*, 2012.

[46] S. Oh et al. A Scheme to Smooth Aggregated Traffic from Sensors with Periodic Reports. *Sensors*, 17(3), 2017.

[47] N. C. Luong, P. Wang, D. Niyato, Y. Wen, and Z. Han. Resource Management in Cloud Networking Using Economic Analysis and Pricing Models: A Survey. *IEEE Communications Surveys Tutorials*, 2017.

[48] C. A. Gizelis and D. D. Vergados. A Survey of Pricing Schemes in Wireless Networks. *IEEE Communications Surveys Tutorials*, 13(1):126–145, 2011.

[49] L. Ramaswamy et al. Towards a Quality-centric Big Data Architecture for Federated Sensor Services. In *2013 IEEE International Congress on Big Data*, pages 86–93, June 2013.

[50] T. Jacobs et al. *D.14.2.2: FIWARE GE Open Specifications (IoT Chapter) - IoT Broker Release 5.* Future Internet Core, 2016.

[51] M. Richart, J. Baliosian, J. Serrat, and J. L. Gorricho. Resource Slicing in Virtual Wireless Networks: A Survey. *IEEE Transactions on Network and Service Management*, 13(3):462–476, Sept 2016.

[52] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker. Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Communications Magazine*, May 2017.

[53] C. Papadimitriou et al. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., 1982.

[54] CityPulse, 2017.

[55] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs. Building a Big Data Platform for Smart Cities: Experience and Lessons from Santander. In *IEEE International Congress on Big Data*, pages 592–599, Jun. 2015.

[56] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pages 13–16, 2012.

[57] Enrique Saurez, Kirak Hong, Dave Lillethun, Umakishore Ramachandran, and Beate Ottenwälder. Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pages 258–269. ACM, 2016.

[58] Open Mobile Alliance. NGSI Context Management. OMA-TS-NGSI Context_Management-V1_0, OMA, May 2012.

[59] M. Bauer, E. Kovacs, A. Schülke, N. Ito, C. Criminisi, L. W. Goix, and M. Valla. The context api in the oma next generation service interface. In *2010 14th International Conference on Intelligence in Next Generation Networks*, pages 1–5, Oct 2010.

[60] FIWARE Found. Fiware foundation.

[61] WISE-IoT Consortium. European WISE-IoT project. http://wise-iot.eu/en/home/.

[62] Open & agile smart cities, 2017.

[63] Aeron broker, 2017.

[64] Orion broker, 2017.

[65] B. Cheng, A. Papageorgiou, and M. Bauer. Geelytics: Enabling on-demand edge analytics over scoped data sources. In *Proceedings of IEEE International Congress on Big Data*, pages 101–108, June 2016.

[66] Openfog consortium, 2017.

[67] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. Mcep: A mobility-aware complex event processing system. *ACM Trans. Internet Technol.*, 14(1):6:1–6:24, August 2014.

[68] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog: A programming model for large–scale applications on the internet of things. *Network*, 12(F13):F14, 2013.

[69] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, Feb 2014.

[70] Maria A. Lema et al. Business Case and Technology Analysis for 5G Low Latency Applications. *IEEE Access*, 5:5917–5935, 2017.

[71] A. Reznik et al. Cloud RAN and MEC: a perfect pairing, Feb. 2018.

[72] A. Martinez-Balleste, P. A. Perez-martinez, and A. Solanas. The pursuit of citizens' privacy: a privacy-aware smart city is possible. *IEEE Communications Magazine*, 51(6):136–141, Jun. 2013.

[73] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang. A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications. *IEEE Access*, 5:6757–6779, 2017.

[74] OpenFog Consortium Architecture Working Group. Openfog reference architecture for fog computing. Technical report, Feb 2017.

[75] X. Sun and N. Ansari. EdgeIoT: Mobile Edge Computing for the Internet of Things. *IEEE Communications Magazine*, 54(12):22–29, Dec. 2016.

[76] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust. Mobile-Edge Computing Architecture: The role of MEC in the Internet of Things. *IEEE Consumer Electronics Magazine*, 5(4):84–91, Oct. 2016.

[77] ETSI MEC ISG. Mobile Edge Computing (MEC); Framework and reference architecture. DGS MEC 003, ETSI, Apr. 2016.

[78] ETSI MEC ISG. Mobile Edge Computing (MEC); General principles for Mobile Edge Service APIs. DGS MEC 009, ETSI, Jul. 2017.

[79] ETSI MEC ISG. Mobile Edge Computing (MEC); Mobile Edge Platform Application Enablement. DGS MEC 011, ETSI, Jul. 2017.

[80] FIWARE Found. and TMForum. Smart Data Models.

[81] Open and Agile Smart Cities (OASC). Minimal interoperability mechanisms (mims). Technical Report V1_16.01.2019, OASC, January 2019.

[82] N. Koshizuka, S. Haller, and K. Sakamura. Cpaas.io: An eu-japan collaboration on open smart-city platforms. *Computer*, 51(12):50–58, Dec 2018.

[83] Ken Sakamura. Open IoT platform and IoT engine, Jun 2016.

[84] D. C. G. Valadares, M. S. L. da Silva, A. E. M. Brito, and E. M. Salvador. Achieving data dissemination with security using fiware and intel software guard extensions (sgx). In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7, 2018.

[85] Quyet H. Cao, Madhusudan Giyyarpuram, Reza Farahbakhsh, and Noel Crespi. Policy-based usage control for a trustworthy data sharing platform in smart cities. *Future Generation Computer Systems*, June 2017.

[86] Abayomi Otebolaku and Young-Gab Kim. D2.3 - end-to-end security and trust framework. Technical report, Wise-IoT, October 2017.

[87] John Soldatos, Nikos Kefalakis, Manfred Hauswirth, Martin Serrano, Jean-Paul Calbimonte, Mehdi Riahi, Karl Aberer, Prem Prakash Jayaraman, Arkady Zaslavsky, Ivana Podnar Žarko, Lea Skorin-Kapov, and Reinhard Herzog. Openiot: Open source internet-of-things in the cloud. In Ivana Podnar Žarko, Krešimir Pripužić, and Martin Serrano, editors, *Interoperability and Open-Source Solutions for the Internet of Things*, pages 13–25, Cham, 2015. Springer International Publishing.

[88] S. K. Datta and C. Bonnet. A lightweight framework for efficient m2m device management in oneM2M architecture. In *2015 International Conference on Recent Advances in Internet of Things (RIoT)*, April 2015.

[89] ETSI. Context Information Management (CIM); Application Programming Interface (API). Technical Report ETSI GS CIM 009 V1.1.1, ETSI ISG CIM, January 2019.

[90] Álvaro Alonso, Alejandro Pozo, José Manuel Cantera, Francisco de la Vega, and Juan José Hierro. Industrial data space architecture implementation using FIWARE. *Sensors*, 18(7), 2018.

[91] P.P. Ray. A survey on internet of things architectures. *King Saud U. J. - Comp. and Info. Sciences*, 30(3), 2018.

[92] J. Lanza, L. Sánchez, J. R. Santana, R. Agarwal, N. Kefalakis, P. Grace, T. Elsaleh, M. Zhao, E. Tragos, H. Nguyen, F. Cirillo, R. Steinke, and J. Soldatos. Experimentation as a service over semantically interoperable internet of things testbeds. *IEEE Access*, 6:51607–51625, 2018.

[93] R. Lea and M. Blackstock. City hub: A cloud-based iot platform for smart cities. In *IEEE CLOUDCOM 2014*, Dec.

[94] Tachmazidis I. *et al.* A hypercat-enabled semantic internet of things data hub. In *The Semantic Web*. Springer, 2017.

[95] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R. Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, and Dennis Pfisterer. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217 – 238, 2014.

[96] Fang-Jing Wu and Gürkan Solmaz. Crowdestimator: Approximating crowd sizes with multi-modal data for internet-of-things services. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, pages 337–349, New York, NY, USA, 2018. ACM.

[97] B. Carballido Villaverde *et al.* Service discovery protocols for constrained machine-to-machine communications. *IEEE Comm. S. Tut.*, 16(1), 2014.

[98] J. Mineraud *et al.* A gap analysis of internet-of-things platforms. *Computer Communications*, 89-90, 2016.

[99] S. Nastic *et al.* Patricia – a novel programming model for iot applications on cloud platforms. In *2013 IEEE SOCA*, 2013.

[100] Z. Yan *et al.* A survey on trust management for internet of things. *Journal of Network and Computer Applications*, 42, 2014.

[101] R. Roman *et al.* On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 2013.

[102] Idra - Open Data Federation Platform. https://idra.readthedocs.io (accessed 17/05/2019).

[103] Alireza Hassani, Alexey Medvedev, Pari Delir Haghighi, Sea Ling, Arkady Zaslavsky, and Prem Prakash Jayaraman. Context definition and query language: Conceptual specification, implementation, and evaluation. *Sensors*, 19(6), 2019.

[104] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015.

[105] Z. Zhu *et al.* Wearable sensor systems for infants. *Sensors*, 2015.

[106] M. Abomhara and G. M. Køien. Security and privacy in the internet of things: Current status and open issues. In *2014 PRISMS*, May 2014.

[107] K. R. Özyilmaz, M. Doğan, and A. Yurdakul. Idmob: Iot data marketplace on blockchain. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 11–19, 2018.

[108] M. Wu, K. Wang, X. Cai, S. Guo, M. Guo, and C. Rong. A comprehensive survey of blockchain: From theory to iot applications and beyond. *IEEE Internet of Things Journal*, 6(5):8114–8154, 2019.

[109] E. Androulaki *et al.* Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *ACM Proceedings of the 13th EuroSys Conference*, 2018.

[110] Apache Hadoop. https://hadoop.apache.org/.

[111] Apache Storm. https://storm.apache.org/.

[112] Apache Spark. https://spark.apache.org/.

[113] Apache Flink. https://flink.apache.org/.

[114] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide.* Springer Publishing Company, Incorporated, 1st edition, 2017.

[115] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security*, 9(4):391–420, November 2006.

[116] Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

[117] Usage Control In The Interation Data Spaces. `https://www.internationaldataspaces.org/wp-content/uploads/2019/11/Usage-Control-in-IDS-V2.0_final.pdf`.

[118] Ravi Sandhu and Jaehong Park. Usage control: A vision for next generation access control. In *International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 17–31. Springer, 2003.

[119] Jaehong Park and Ravi Sandhu. Towards usage control models: Beyond traditional access control. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, 2002.

[120] Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal model and policy specification of usage control. *ACM Transactions on Information and System Security*, Nov 2005.

[121] Julian Schütte and Gerd Stefan Brost. LUCON: Data flow control for message-based iot systems. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 289–299. IEEE, 2018.

[122] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Survey: Usage control in computer security: A survey. *Computer Science Review*, 2010.

[123] Alexander Pretschner, Manuel Hilty, Florian Schütz, Christian Schaefer, and Thomas Walter. Usage control enforcement: Present and future. *IEEE Security & Privacy*, 6(4):44–53, 2008.

[124] Florian Kelbert and Alexander Pretschner. Data usage control for distributed systems. *ACM Transactions on Privacy and Security*, April 2018.

[125] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, number 9, 2010.

[126] K. R. Özyilmaz, M. Doğan, and A. Yurdakul. IDMoB: IoT Data Marketplace on Blockchain. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 11–19, 2018.

[127] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. Enforcing private data usage control with blockchain and attested off-chain contract execution. *arXiv preprint arXiv:1904.07275*, 2019.

[128] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. Fairaccess: a new blockchain-based access control framework for the internet of things. *Security and Communication Networks*, 9(18), 2016.

[129] ODRL Information Model 2.2. `https://www.w3.org/TR/odrl-model`.

[130] NGINX. Is your API real time? `https://www.nginx.com/blog/api-real-time-test-latency-responsiveness-nginx-rtapi-tool/`.

[131] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[132] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[133] Jaehong Park and Ravi Sandhu. The UCONABC usage control model. *ACM Trans. on Information and System Security (TISSEC)*, 7(1):128–174, feb 2004.

[134] Lili Sun and Hua Wang. A purpose based usage access control model. *International Journal of Computer and Information Engineering*, 2010.

[135] Alexander Pretschner, Manuel Hilty, and David Basin. Distributed usage control. *Communications of the ACM*, 49(9):39–44, September 2006.

[136] MYDATA. `https://www.mydata-control.de`.

[137] Tai h. Kim, Carlos Ramos, and Sabah Mohammed. Smart City and IoT. *Future Generation Computer Systems*, 76:159 – 162, 2017.

[138] Riccardo Petrolo, Valeria Loscrì, and Nathalie Mitton. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Transactions on Emerging Telecommunications Technologies*, 28(1):e2931, 2017. e2931 ett.2931.

[139] M. Abu-Matar. Towards a software defined reference architecture for smart city ecosystems. In *2016 IEEE International Smart Cities Conference (ISC2)*, pages 1–6, Sep. 2016.

[140] AIOTI WG02. Report on innovation ecosystems. Technical report, Alliance Internet of Things Innovation, 2015.

[141] S. Kubler, J. Robert, A. Hefnawy, K. Främling, C. Cherifi, and A. Bouras. Open iot ecosystem for sporting event management. *IEEE Access*, 5:7064–7079, 2017.

[142] Mika Westerlund, Seppo Leminen, and Mervi Rajahonka. Designing business models for the internet of things. *Technology Innovation Management Review*, 4:5–14, 07/2014 2014.

[143] P. Desai, A. Sheth, and P. Anantharam. Semantic gateway as a service architecture for iot interoperability. In *2015 IEEE International Conference on Mobile Services*, pages 313–319, June 2015.

[144] M. Y. Khalid, P. H. H. Then, and V. Raman. Exploratory study for data visualization in internet of things. In *IEEE 42nd Annual Computer Sw and Applications Conf. (COMPSAC)*, volume 02, pages 517–521, July 2018.

[145] A. Bröring and et al. Enabling iot ecosystems through platform interoperability. *IEEE Software*, 34(1):54–61, Jan 2017.

[146] Aparna Saisree Thuluva and et al. Recipes for iot applications. In *Proceedings of the Seventh International Conference on the Internet of Things*, IoT '17, NY, USA, 2017. Association for Computing Machinery.

[147] Werner Schladofsky and et at. Business models for interoperable iot ecosystems. In *Interoperability and Open-Source Solutions for the Internet of Things*, pages 91–106. Springer International Publishing, 2017.

[148] Xenia Ziouvelou and Frank McGroarty. A business model framework for crowd-driven iot ecosystems. pages 262–284, 2019.

[149] Amazon Web Services. Aws marktplace.

[150] Microsoft Azure. Microsoft azure marketplace.

[151] B. Ahlgren, M. Hidell, and E. C. . Ngai. Internet of things for smart cities: Interoperability and open data. *IEEE Internet Computing*, 20(6):52–56, Nov 2016.

[152] FIWARE Found. FIWARE Domain Specific Enablers (DSEs).

[153] Sehl Mellouli, Luis . Luna-Reyes, and Jing Zhang. Smart government, citizen participation and open data. *Information Polity*, 19:1–4, 2014.

[154] Lisa Goines and Leda Hagler. Noise pollution: A modern plague. In *Noise Pollution: A Modern Plague*, 2007.

[155] M. Maggio et al. SynchroniCity D2.10 - Reference Architecture for IoT Enabled Smart Cities, Update. Technical report, SynchroniCity, Aug. 2018.

[156] F. Cirillo et al. SynchroniCity D3.2 - Suite of baseline implementations - basic. Technical report, SynchroniCity, Jun. 2018.

[157] M. Yannuzzi et al. A new era for cities with fog computing. *IEEE Internet Computing*, 21(2):54–67, Mar. 2017.

[158] L. Sánchez et al. Federation of Internet of Things Testbeds for the Realization of a Semantically-Enabled Multi-Domain Data Marketplace. *Sensors*, 18(10):3375, Oct. 2018.

[159] General Data Protection Regulation. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46. *Official Journal of the European Union (OJ)*, 59(1-88):294, Apr. 2016.

[160] Article 29 Working Party. Opinion 01/2017 on the proposed regulation for the ePrivacy regulation (2002/58/EC), Apr. 2017. Accessed on Jan. 3, 2019.

[161] Agencia Española de Protección de Datos. Orientaciones y garantías en los procedimientos de anonimización de datos personales, October 2016. Accessed on Jan. 3, 2019.

[162] Y.-A. De Montjoye et al. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3:1376, Mar. 2013.

[163] I Nižetić Kosović and Tomislav Jagušt. Enhanced weighted centroid localization algorithm for indoor environments. *International Journal of Computer, Control, Quantum and Information Engineering*, 8(7), July 2014.

[164] Antonio J Jara, Dominique Genoud, and Yann Bocchi. Big data for smart cities with KNIME a real experience in the SmartSantander testbed. *Software: Practice and Experience*, 45(8):1145–1160, May 2015.

[165] H. Tong et al. Modeling large passenger flow safety by simulation and testing. In *Proceedings of IEEE ICCC'17*, pages 235–239, Dec. 2017.

[166] K. Zhao et al. Urban human mobility data mining: An overview. In *Proceedings of IEEE Big Data'16*, pages 1911–1920, Dec. 2016.

[167] Y. Zhou et al. Understanding urban human mobility through crowdsensed data. *IEEE Communications Magazine*, 56(11):52–59, Nov. 2018.

[168] Luis Sánchez, Jorge Lanza, Juan Ramón Santana, Rachit Agarwal, Pierre Guillaume Raverdy, Tarek Elsaleh, Yasmin Fathy, SeungMyeong Jeong, Aris Dadoukis, Thanasis Korakis, Stratos Keranidis, Philip O'Brien, Jerry Horgan, Antonio Sacchetti, Giuseppe Mastandrea, Alexandros Fragkiadakis, Pavlos Charalampidis, Nicolas Seydoux, Christelle Ecrepont, and Mengxuan Zhao. Federation of internet of things testbeds for the realization of a semantically-enabled multi-domain data marketplace. *Sensors*, 18(10), 2018.

[169] J. Bohli, A. Skarmeta, M. Victoria Moreno, D. García, and P. Langendörfer. Smartie project: Secure iot data management for smart cities. In *2015 International Conference on Recent Advances in Internet of Things (RIoT)*, pages 1–6, April 2015.

[170] Minako Hara, Tomomi Nagao, Shinsuke Hannoe, and Jiro Nakamura. New key performance indicators for a smart sustainable city. *Sustainability*, 8(3), 2016.

[171] Gürkan Solmaz, Jonathan Fürst, Samet Aytaç, and Fang-Jing Wu. Group-in: Group inference from wireless traces of mobile devices. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 157–168. IEEE, 2020.

[172] Bin Cheng, Jonathan Fuerst, Gurkan Solmaz, and Takuya Sanada. Fog function: Serverless fog computing for data intensive iot services. In *2019 IEEE International Conference on Services Computing (SCC)*, pages 28–35. IEEE, 2019.

[173] Quyet H Cao, Madhusudan Giyyarpuram, Reza Farahbakhsh, and Noel Crespi. Policy-based usage control for a trustworthy data sharing platform in smart cities. *Future Generation Computer Systems*, 107:998–1010, 2020.

[174] Peter Ochieng and Swaib Kyanda. Large-scale ontology matching: state-of-the-art analysis. *ACM Computing Surveys (CSUR)*, 51(4):1–35, 2018.